

SQL

CREATE DATABASE "USUARIOS"

Añadir .db

Tabla: estructura de datos que se distribuye en filas y en columnas.

Campo: nombre/número de la columna

Registro: cada fila.

Valor: lo que contiene cada cuadro.

Consulta: pedirle información a la base de datos, hacer una búsqueda en una base de datos, modificar una base de datos.

Identificador:

Tipo de campo:

```
CREATE TABLE "TURNOS_MEDICOS" (  
  "id_turno" INTEGER,  
  "profesional" TEXT,  
  "id_usuario" INTEGER,  
  "motivo" TEXT,  
  "horario" TEXT,  
  PRIMARY KEY("id_turno" AUTOINCREMENT)  
);
```

Integer: número

Text: texto

Blob: almacena datos binarios, archivos, imágenes, fotos, videos.

Real: valores con coma, equivalente a float. Almacena menos información y es más rápido.

Numeric: trabaja con números con

precisión matemática muy alta. Es más lento.

Nombre	Tipo	NN	PK	AI	U	Por defecto	Check
nombre	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Por defecto indicará un mismo valor siempre que se crea, por ejemplo si creamos un registro y no ponemos un nombre aparecerá el valor por defecto.

Para introducir datos: ejecutar SQL.

Select:

Permite hacer consultas a una base de datos.

SELECT * FROM users -> selecciona todo de la tabla "users".

Insert:

Para insertar datos.

INSERT INTO users (nombre,apellido,edad)

```
INSERT INTO users (nombre,apellido,edad)  
VALUES ("IGNACIO", "MACAYA", "34")
```

Insertar dentro de la tabla users los valores de nombre, apellido y edad.

```
INSERT INTO users (nombre,apellido,edad)  
VALUES ('GONZALO', 'MACAYA', '23'),  
      ('MARC', 'PUIG', '46')
```

Insertar varios registros.

SELECT NOMBRE FROM USERS -> solo nos devolverá la columna nombres, si queremos más columnas podemos ponerlas entre comas.



SQL desde cero

En hoja de datos si seleccionamos esta opción aparecerá un registro con NULLS. Podemos modificarlos y ponerles el valor que queramos.

Identifiers: campo para identificar un registro entero.

Clave primaria: para crearla vuelvo a estructura, click derecho sobre la tabla, modificar, creamos un nuevo campo que se llame id usuario, para crear la clave primaria tenemos que eliminar todos los registros.

Para eliminar todos los registros:

DELETE FROM users

Nombre	Tipo	NN	PK	AI	U
id usuario	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Para crear la clave primaria selecciono PK (primary key) y AI (autoincrement)

Creo una nueva tabla que se llame turnos_medicos

Con los campos: id_turno, profesional, id_usuario y horario

Creo un registro:

```
1 INSERT INTO TURNOS_MEDICOS (profesional, id_usuario, motivo, horario)
2 VALUES ("Dr.Ramirez", 6, "Dolor de garganta", "14:00");
3
4 SELECT * FROM TURNOS_MEDICOS;
5 SELECT * FROM users
```

	id usuario	nombre	apellido	edad
1	5	IGNACIO	MACAYA	34
2	6	ALVARO	MACAYA	31
3	7	GONZALO	MACAYA	23
4	8	JUAN	POLIT	32
5	9	NICOLÁS	POLIT	30
6	10	BORJA	POLIT	28
7	11	CARLOS	MACAYA	22

Aquí se superponen las tablas, muestra únicamente la de usuarios.

Para que muestre las dos juntas tenemos que hacer:

Clave foránea: hace referencia a una **clave primaria de otra tabla**, en este caso sería **id_usuario de la tabla turnos_medicos de la clave primaria de users**. Las claves foráneas son menos eficientes pero son necesarias.

Relacionar tablas:

las tablas se relacionan por las foreign keys o claves foráneas.

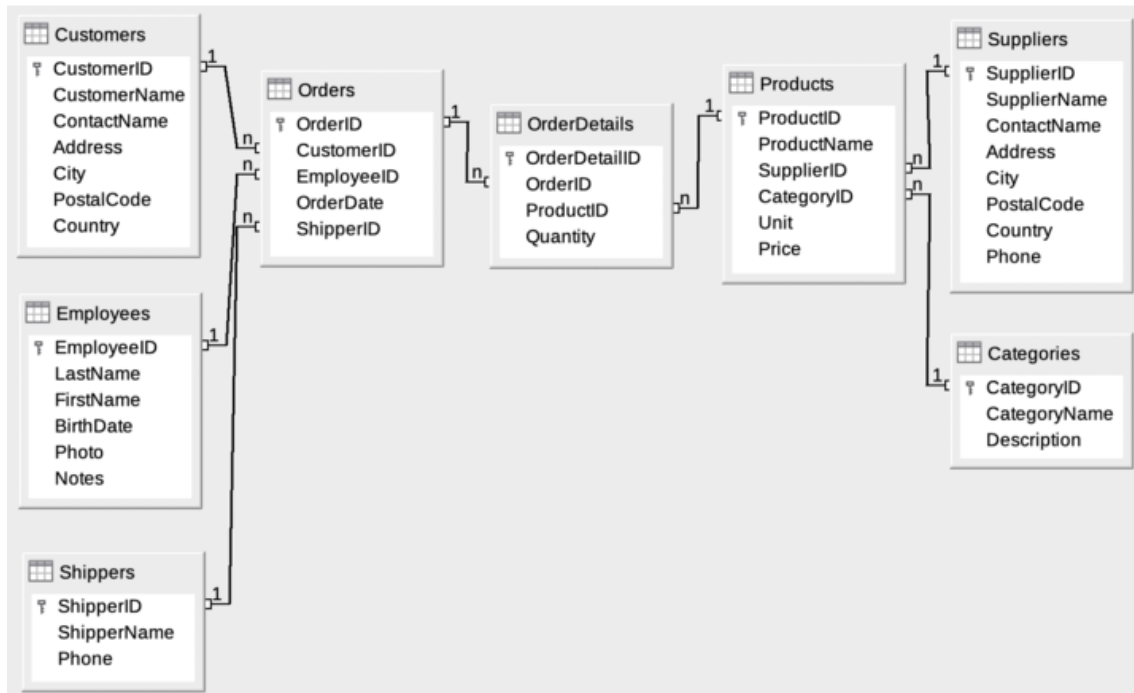
La id usuario de turnos médicos estará relacionada con la id usuario de usuarios, que será la primary key.

SQL desde cero

Base de datos Northwind:

https://en.wikiiversity.org/wiki/Database_Examples/Northwind/SQLite

Para saber como se relacionan las diferentes tablas tenemos un diagrama.



La tabla principal es customers.

La llave denota las primary key.

Shipper: empresa que se hace cargo del envío.

As y order by:

As:

```
SELECT LastName AS apellido FROM Employees
```

Devolverá los apellidos pero la columna se llamará apellidos, la tabla employees no variará porque cundo usamos select nos devuelve una nueva tabla, podemos asignar un nuevo nombre al campo

```
SELECT LastName AS apellido, FirstName AS nombre FROM Employees
```

Usamos as para agregarle un nombre más descriptivo a la tabla

Order by:

```
SELECT * FROM Products  
ORDER BY Price DESC
```

Por defecto la tabla está ordenada por orden de registro (id)

Ordeno la tabla de productos ordenado de más caro a más barato.

```
SELECT * FROM Products  
ORDER BY ProductName ASC
```

Ordenará de forma alfabética, si hay números, letras, en primer lugar irán los null, son los que valen menos, después van los números, después los caracteres especiales, y lo que más rango tiene son las letras. Y si ordenamos de forma descendente es al revés.

```
SELECT * FROM Products  
ORDER BY ProductName ASC NULLS LAST
```

Nulls last pondrá los nulos al final de la lista.

```
SELECT * FROM Products
ORDER BY ProductName, SupplierID DESC
```

Primero ordenará por product name y después por Supplier ID.

```
SELECT DISTINCT ProductName FROM Products
```

eliminando los repetidos.

Distinct: se coloca después del select y sirve para devolver los valores únicos,

```
SELECT DISTINCT ProductName
FROM Products ORDER BY random()
```

Seleccionará todos los productos y los ordenará de forma aleatoria

Condiciones, cláusula Where

Si quiero buscar un producto por un id o filtrar por nombre podemos filtrar por condiciones. En Where se tiene que indicar la condición, si la condición es True devolverá algo.

```
SELECT ProductName FROM Products
WHERE ProductID = 14
```

Selecciona el nombre del producto de la tabla productos donde el product id = 14.

```
SELECT * FROM Products
WHERE ProductName = "Tofu"
```

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
14	Tofu	6	7	40 - 100 g pkgs.	23.25

Buscamos el registro donde product name = Tofu

```
SELECT * FROM Products
WHERE Price < 40
```

Si queremos saber qué productos tiene un precio inferior a 40

```
SELECT * FROM Products
WHERE Price > 40
```

Si queremos saber qué productos valen más que 40

Con el asterisco seleccionamos campos (columnas), si es asterisco seleccionamos todos los campos, el where nos selecciona registros (filas).

Update:

Si queremos actualizar algún valor:

```
UPDATE TURNOS_MEDICOS SET horario = "10:30"
WHERE id_turno = 2
```

Actualizaremos en la tabla de turnos médicos el horario a las 10:30 siempre que el id sea 2.

```
UPDATE TURNOS_MEDICOS
SET horario = "10:30",
SET motivo = "dolor de muelas"
WHERE id_turno = 2
```

Si queremos cambiar más de un elemento del registro, cambio tanto el horario como el motivo de la visita.

Operadores lógicos AND OR NOT:

Queremos indicar que de los clientes 50 al 55 han ganado un premio:

```
SELECT * FROM Customers
WHERE CustomerID >=50 AND CustomerID < 55
```

Or:

Queremos seleccionar 2 empleadas, Anne y Nancy:

SQL desde cero

```
SELECT * FROM Employees  
WHERE FirstName = "Anne" OR FirstName = "Nancy"
```

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA in psychology from ...
9	Dodsworth	Anne	1969-07-02	EmpID9.pic	Anne has a BA degree in English from St. ...

Podemos unir el or y el and.

```
SELECT * FROM Products  
WHERE Price < 20 OR CategoryID = 6
```

Queremos que nos seleccione todos los registros de producto cuyo precio sea inferior a 20 o el id de la categoría sea 6.

```
SELECT * FROM Products  
WHERE (Price < 20 OR CategoryID = 6) AND SupplierID = 7
```

Ahora queremos que seleccione todos los registros cuyo precio sea

inferior a 20 o el id de la categoría sea 6, pero lo que sí queremos que se cumpla es que el SupplierID = 7. Si no ponemos paréntesis primero se ejecutará la primera condición OR y después la AND toda junta.

Not:

Queremos que no devuelva los productos cuyo precio es mayor a 40.

```
SELECT * FROM Products  
WHERE NOT Price > 40
```

Te niega una condición.

Ahora sabemos que un cliente nos ha hackeado y lo que sabemos es que el cliente proviene de Estados Unidos, vamos a mandar un mensaje a todos los clientes que no sean de Estados Unidos.

```
SELECT * FROM Customers  
WHERE NOT Country = "USA"
```

Ahora sabemos que los que nos hackearon son de USA o de Francia.

```
SELECT * FROM Customers  
WHERE NOT (Country = "USA" OR Country = "France")
```

```
SELECT * FROM Customers  
WHERE NOT Country = "USA" AND NOT Country = "France"
```

Ahora queremos seleccionar los 5 ganadores del premio pero el premio no está disponible en Alemania:

```
SELECT * FROM Customers  
WHERE CustomerID >= 50  
AND  
CustomerID < 55  
AND NOT Country = "Germany"
```

Limit:

Cuando queremos limitar el número de registros que nos devuelva.

```
SELECT * FROM Customers  
WHERE CustomerID >= 50  
AND NOT Country = "Germany"  
LIMIT 5
```

```
SELECT * FROM Products
WHERE NOT CategoryID = 6
AND NOT SupplierID = 1
AND Price <= 30
ORDER BY RANDOM()
LIMIT 3
```

Un cliente viene a preguntarnos por nuestros productos y no quiere carne de ave, carne de ave es la categoría 6 y quiere los productos de mayor calidad, sabemos que el proveedor 1 tiene productos de muy mala calidad, por lo que lo excluimos, también quiere que el precio sea menor que 30 y quiere que le muestre 3 opciones y quiere que se muestren aleatoriamente.

Distinto VS Not

```
SELECT * FROM Customers
WHERE Country != "USA"
```

Queremos seleccionar todos los clientes que no provengan de Estados Unidos, ¿qué diferencia hay entre NOT y !=, devuelven lo mismo pero distinto de es un operador de comparación, sirve para comparar valores. La comparación compara un valor con otro valor.

Between:

Operador que sirve para seleccionar valores entre un rango específico.

```
SELECT * FROM Products
WHERE Price BETWEEN "20" AND "40"

SELECT * FROM Products
WHERE Price NOT BETWEEN "20" AND "40"
AND CategoryID = 6
```

Queremos que se nos muestren todos los productos que cuesten más de 20 y menos que 40.

También nos puede servir el operador not between que selecciona todos los productos que cuestan menos de 20 y más de 40.

También nos sirve para seleccionar fechas.

```
SELECT * FROM Employees
WHERE BirthDate BETWEEN "1960-0-01" AND "1970-0-1"
```

queremos seleccionar los trabajadores que han nacido entre 1960 y 1970.

Between incluye los elementos límite.

Like:

Operador de comparación que sirve para filtrar en función de ciertos patrones de cadenas de texto. Se parece a las expresiones regulares pero mucho más simple y limitado.

```
SELECT * FROM Employees
WHERE "LastName" LIKE "Fuller"
```

Queremos que nos devuelva el registro donde el apellido sea Fuller.

```
SELECT * FROM Employees
WHERE "LastName" LIKE "%r"
```

Significa que antes de la r hay alguna cosa, que empieza por r "r%" o que contiene r, "%r%".

También tenemos el "_"

```
SELECT * FROM Employees
WHERE "LastName" LIKE "f____r"
```

Si queremos que nos devuelva el trabajador que tiene un apellido que empieza por f, tiene 4 letras entre medio y acaba con r.

También se puede juntar con not.

Is null, is not null:

```
SELECT * FROM Products
WHERE ProductName IS NULL
```

Devuelve los valores que son nulos, de igual manera podemos decir "not null" y nos devolverá todos los valores que no son nulos.

In:

```
SELECT * FROM Products
WHERE SupplierID = 3
OR SupplierID = 4
OR SupplierID = 5
OR SupplierID = 6
```

Operador lógico, introducción a las subconsultas. Tenemos que seleccionar los productos de los proveedores 3,4,5,6.

```
SELECT * FROM Products
WHERE SupplierID IN (3,4,5,6)
```

Devolverá el mismo resultado de una forma más abreviada. El supplier ID va mirando cada registro y si el resultado coincide con el valor de la tupla lo devuelve. Se puede utilizar en select, update y delete.

```
SELECT * FROM Employees
WHERE LastName IN ("Fuller","King")
```

Ahora quiero seleccionar a los trabajadores Fuller y King.

También podemos usar el not, por ejemplo en el caso de que no queramos que nos devuelvan estos dos registros.

```
SELECT * FROM Employees
WHERE LastName NOT IN ("Fuller","King")
```

Con IN podemos hacer una subconsulta dentro de una consulta, no es la mejor forma.

Funciones de agregación (subconsultas):

Permite agrupar datos y resumirlos, se utilizan con la función select.

```
SELECT COUNT(FirstName) FROM Employees
```

Quiero contar el número de valores en una columna, por ejemplo quiero que nos diga cuantos apellidos hay en la columna apellidos de trabajadores, devolverá 10.

```
SELECT COUNT(FirstName) AS Cantidad_de_Nombres FROM Employees
```

Count es una función de agregación que nos sirve para contar la cantidad de valores en un campo.

Cantidad_de_Nombres
10

Si queremos sumar todos los precios de la columna de precios de la tabla de productos.

```
SELECT sum(Price) FROM Products
```

Si queremos que nos devuelva el promedio del precio de los productos.

```
SELECT avg(Price) FROM Products
```

Si queremos redondear un valor:

```
SELECT round(avg(Price)) FROM Products
```

Si queremos que nos redondee a dos decimales:

```
SELECT round(avg(Price),2) FROM Products
```

```
SELECT min(price) FROM Products
```

Para obtener el precio mínimo

```
SELECT ProductName, min(price) FROM Products
WHERE ProductName IS NOT NULL
```

Ahora seleccionará el nombre y el precio mínimo de la tabla de productos siempre que el nombre de producto no sea nulo.

```
SELECT ProductName, max(price) FROM Products
WHERE ProductName IS NOT NULL
```

Ahora selecciono el nombre del producto y el precio máximo de la tabla de productos siempre que el producto no sea nulo.

Group by y having:

Group By sirve para agrupar registros en función de una o varias columnas.

VENTAS REALIZADAS		
PRODUCTO	CLIENTE	CANTIDAD
BANANA	XXXXX	4
MANZANA		3
BANANA		3
MANZANA		4
CIRUELA		5
MANZANA		2
BANANA		4
CIRUELA		3
MANZANA		2

GROUP BY PRODUCTO

PRODUCTO	CANTIDAD
BANANA	13
MANZANA	10
CIRUELA	8

Si quiero agrupar por producto, y los campos con el mismo valor se agrupan. Si queremos saber cuantas frutas se han vendido.

```
SELECT SupplierID, ROUND(AVG(Price),2) FROM Products
GROUP BY SupplierID
```

Esto nos indicará el promedio del precio de lo que cuestan los productos de los

proveedores.

```
SELECT SupplierID, ROUND(AVG(Price)) as promedio FROM Products
GROUP BY SupplierID
ORDER BY promedio DESC
```

ahora los ordeno en función del promedio y el más alto estará en

primer lugar.

```
SELECT CategoryID, ROUND(AVG(Price)) as promedio FROM Products
WHERE CategoryID IS NOT NULL
GROUP BY CategoryID
ORDER BY promedio DESC
```

primero va la condición, después va el group by y luego podemos

ordenar. Si ahora además le pedimos el nombre nos devolverá el valor más bajo.

```
SELECT ProductName, CategoryID, ROUND(AVG(Price)) as promedio FROM Products
WHERE CategoryID IS NOT NULL
GROUP BY CategoryID
ORDER BY promedio DESC
```

El producto que mostrará será el de valor más bajo.

Ahora quiero saber cual es el proveedor que tiene los productos más altos.

```
SELECT SupplierID, ROUND(AVG(Price)) as promedio FROM Products
GROUP BY SupplierID
HAVING promedio > 40
```

Con where no podemos hacer referencia a la función de

agrupación, Group by nos permite crear grupos, having nos permite hacer filtrado de grupos. En grupos en vez de usar where usamos having.

```
SELECT ProductID, SUM(Quantity) as total FROM OrderDetails
GROUP BY ProductID
HAVING total < 50
ORDER BY total DESC
```

Como podemos saber cual es el producto que más se vende. Primero sumamos las ventas por el id de los

productos, ahora queremos desterrar los productos que han vendido menos de 50.

Si ahora queremos saber el producto más vendido

```
SELECT ProductID, SUM(Quantity) as total FROM OrderDetails
GROUP BY ProductID
ORDER BY total DESC
LIMIT 1
```

Y el que menos se ha vendido:


```
SELECT ProductID, SUM(Quantity) as total FROM OrderDetails
GROUP BY ProductID
ORDER BY total ASC
LIMIT 1
```

No podemos agregar una función de agregación a otra función de agregación.

select | where | group by | having (no se puede usar sin el group by antes) | order by | limit

Subconsultas:

Nos permite unir dos tablas. No suele ser la mejor forma de unir tablas. Primero se ejecuta una subconsulta, esos resultados sirven para generar la consulta principal. Las subconsultas solamente recuperan datos.

La subconsulta tiene que ser sí o sí un select porque necesitamos que nos devuelvan datos.

Seleccionamos el product Id y la cantidad de Order Details, pero quiero saber cual es el nombre del producto, que no está en order details, sino que está en products. La subconsulta se pone entre paréntesis.

```
SELECT ProductID,
       Quantity,
       (SELECT ProductName FROM Products WHERE OrderDetails.ProductID = ProductID) as Nombre
FROM OrderDetails
```

En una subconsulta no podemos poner un alias.

```
SELECT ProductID,
       Quantity,
       (SELECT ProductName FROM Products WHERE OD.ProductID = ProductID) as Nombre,
       (SELECT Price FROM Products WHERE OD.ProductID = ProductID) as Precio
FROM OrderDetails as OD
```

Si ahora no queremos devolver solo el nombre sino que también queremos devolver el precio.

Si ahora queremos que nos devuelva el total de la cantidad del precio multiplicado por la cantidad vendida:

```
SELECT ProductID, SUM(Quantity) as total_vendido,
       (SELECT Price FROM Products WHERE ProductID = OD.ProductID) as Price,
       SUM(Quantity) * (SELECT Price FROM Products WHERE ProductID = OD.ProductID) as Cantidad_total
FROM OrderDetails as OD
GROUP BY ProductID
```

```
SELECT ProductID, SUM(Quantity) as total_vendido,
       (SELECT Price FROM Products WHERE ProductID = OD.ProductID) as Price,
       (SELECT ProductName FROM Products WHERE ProductID = OD.ProductID) as Name,
       (SUM(Quantity) * (SELECT Price FROM Products WHERE ProductID = OD.ProductID)) as Cantidad_total
FROM OrderDetails as OD
GROUP BY ProductID
ORDER BY Cantidad_total asc
```

Si ahora queremos ordenar por lo que más ingresos han generado.

```
SELECT ProductID, SUM(Quantity) as total_vendido,
       (SELECT ProductName FROM Products WHERE ProductID = OD.ProductID) as Name,
       (SUM(Quantity) * (SELECT Price FROM Products WHERE ProductID = OD.ProductID)) as Cantidad_total
FROM OrderDetails as OD
WHERE (SELECT Price FROM Products WHERE ProductID = OD.ProductID) > 40
GROUP BY ProductID
ORDER BY Cantidad_total DESC
```

Ahora no queremos que salga precio pero queremos aplicar un filtro donde aparezcan los productos que cuesten más de 40.

Si queremos ver la cantidad de ventas que tiene un empleado

SQL desde cero

```
-- vamos a intentar relacionar 3 tablas, obtener a los empleados que consiguieron vender más cantidades promedio--
SELECT LastName, FirstName,
(
    SELECT SUM(od.Quantity) FROM [Orders] o, [OrderDetails] od
    WHERE o.EmployeeID = e.EmployeeID AND od.OrderID = o.OrderID
) as unidades_totales_vendidas
FROM [Employees] e
WHERE unidades_totales_vendidas > (SELECT avg(unidades_totales_vendidas) FROM (
    SELECT (SELECT SUM(od.Quantity) FROM [Orders] o, [OrderDetails] od
    WHERE o.EmployeeID = e2.EmployeeID AND od.OrderID = o.OrderID) as unidades_totales_vendidas FROM [Employees] e2
    GROUP by e2.EmployeeID
) )
```

Ejercicio donde queremos saber cuales son los vendedores que han vendido por encima del promedio.

Joins:

Los joins usan índices. Se usan para combinar dos o más tablas de una base de datos y obtener una tabla nueva.

4 tipos de Join:

Inner Join

devuelve la coincidencia entre ambos.

```
SELECT * FROM Employees e, Orders o
WHERE e.EmployeeID = o.EmployeeID
```

No utilizamos inner Join pero podemos hacerlo igual porque cruzamos dos tablas siempre que las employee ID coincidan.

```
SELECT * FROM Employees e
INNER JOIN Orders o
ON e.EmployeeID = o.EmployeeID
```

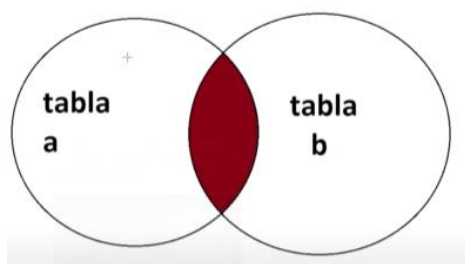
El on sirve como condición, entonces queremos cruzar la tabla employees con la tabla orders siempre y cuando los employee id coincidan.

Ahora vamos a crear una tabla con premios de 5 meses.

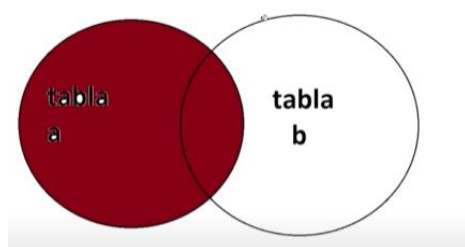
```
INSERT INTO REWARDS (EmployeeID, Reward, MONTH) VALUES
(3,200,"January"),
(2,180,"February"),
(5,250,"March"),
(1,280,"April"),
(8,160,"May"),
(NULL,NULL,"June")
```

```
SELECT * FROM Employees e
INNER JOIN Rewards r ON e.EmployeeID = r.EmployeeID
```

Nos devolverá la información de todos los empleados que tienen recompensas.



Left Join

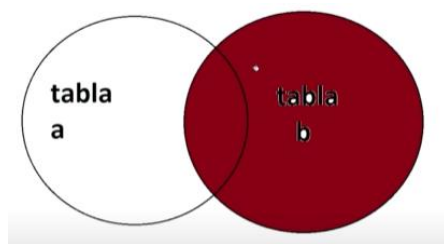


Devuelve una tabla y parte de la información de la otra tabla. Por ejemplo devolverá la tabla de empleados completa y si ha recibido un premio mostrará si ha recibido una recompensa. De lo contrario lo rellena con campos nulos.

```
SELECT FirstName, LastName, Reward, MONTH FROM Employees e
LEFT JOIN Rewards r ON e.EmployeeID = r.EmployeeID
```

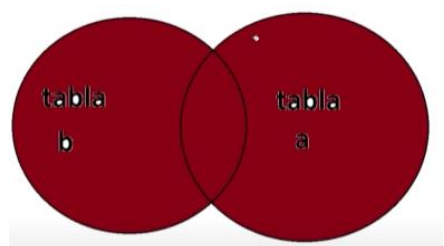
	FirstName	LastName	Reward	MONTH
1	Nancy	Davolio	280	April
2	Andrew	Fuller	180	February
3	Janet	Leverling	200	January
4	Margaret	Peacock	NULL	NULL
5	Steven	Buchanan	250	March
6	Michael	Suyama	NULL	NULL
7	Robert	King	NULL	NULL
8	Laura	Callahan	160	May
9	Anne	Dodsworth	NULL	NULL
10	Adam	West	NULL	NULL

Right Join



Devuelve toda la tabla b y aquello que corresponde con la tabla A, indicará la información de los empleados que han obtenido una recompensa.

Full Join



Devuelve todo, full outer join, es posible que no devuelva un producto cartesiano, devolverá una multiplicación. Devolverá todos los empleados aunque no tengan recompensa y todas las recompensas aunque no tengan trabajadores.

Cross Join

COLORES		TAMAÑO
ROJO		CHICO
VERDE		MEDIANO
AZUL		GRANDE

CROSS JOIN

COLORES	TAMAÑO
ROJO	CHICO
ROJO	MEDIANO
ROJO	GRANDE
VERDE	CHICO
VERDE	MEDIANO
VERDE	GRANDE
AZUL	CHICO
AZUL	MEDIANO
AZUL	GRANDE

Devuelve todas las posibilidades que pueden darse si juntamos una columna con otra.

SQL desde cero

Union: Si se repiten filas union las agrupa y las hace una sola. Tienen que tener la misma cantidad de columnas con el mismo tipo de dato.

ALUMNOS CON ALTO IQ	
ALUMNO	GRADO
JUAN	2
JAIME	3
LUCAS	1
PEDRO	2

ALUMNOS ESPECIALES	
ALUMNO	GRADO
MATEO	3
JOTA	2
LUCAS	1
HENDI	2

UNION ALL

ALUMNO	GRADO
1 JUAN	2
2 JAIME	3
3 LUCAS	1
4 PEDRO	2
5 MATEO	3
6 JOTA	2
7 LUCAS	1
8 HENDI	2

UNION

ALUMNO	GRADO
1 JUAN	2
2 JAIME	3
3 LUCAS	1
4 PEDRO	2
5 MATEO	3
6 JOTA	2
7 HENDI	2

Union sin all devuelve lo mismo excepto las columnas repetidas, elimina los datos redundantes.

Cardinalidad

Determina como va a ser la relación entre dos entidades (tablas)

Autor ID	Nombre	Apellido	Edad
1	Edgar	Poe	58
2	Lucas	Dalto	21

LibroID	AutorID	Género	Año
1	2	Comedia	2021
2	1	Drama / Terror	1845
3	1	Thriller	1849

Aquí tenemos una base de datos con autores y libros, un autor puede escribir varios libros. La cardinalidad nos ayuda a entender esta relación.



4 tipos de relaciones o cardinalidades:

- Uno a uno: un registro en una tabla se relaciona exactamente con un registro de otra tabla (1:1), por ejemplo una persona y documento de identidad.
- Uno a muchos o muchos a uno (1:n) (n:1) cuando un registro de una tabla se relaciona con muchos registros de otra tabla.



Autores			
AutorID	Nombre	Apellido	Edad
1	Edgar	Poe	58
2	Lucas	Dalto	21
3	Julio	Cortázar	69
4	Juan	Pérez	41


Libros			
LibroID	AutorID	Género	Año
1	2	Comedia	2021
2	1	Drama / Terror	1845
3	1	Thriller	1849
4	1	Terror	1844

- n:m: muchos a muchos: varios registros una tabla se relacionan con varios registros de otra tabla.

 Estudiantes					 Cursos			
EstudianteID	Nombre	Apellido	Edad		Cursoid	Estudiantes	Hs.	Día
1	Helen	Chufe	22	$n (n : m) m$	1	7	2	Jueves
2	Aitor	Tilla	34		2	21	4	Lunes
3	Susana	Oria	19		3	24	1	Martes
4	Lola	Mento	21		4	8	6	Viernes

Un curso puede ser tomado por varios estudiantes y un estudiante puede tomar varios cursos. Como esta cardinalidad es complicada de crear en tipo código se crea una tabla intermedia que se llama código que inscripciones que relaciona las claves primarias.

 Estudiantes					 Cursos			
EstudianteID	Nombre	Apellido	Edad		Cursoid	Estudiantes	Hs.	Día
1	Helen	Chufe	22	$n (n : m) m$	1	7	2	Jueves
2	Aitor	Tilla	34		2	21	4	Lunes
3	Susana	Oria	19		3	24	1	Martes
4	Lola	Mento	21		4	8	6	Viernes

 Inscripciones			
InscripciónID	Cursoid	EstudianteID	Fecha
1	1	1	06-202
2	2	1	03-2023
3	3	2	01-2023
4	4	2	12-2022
5	5	2	11-2022

Si se tomara de referencia
 de tablas inscripciones, es lo sería (1,n)

1 (1,n) n

1 (1,n) n

1:n -> es la forma más común.

Normalización de base de datos

Proceso que nos sirve para eliminar anomalías en la base de datos, hay cinco niveles que llamamos forma normal.

1NF -> Garantizar que cada atributo (columna) contenga un valor único atómico. Por ejemplo que en una tabla de empleados en una columna haya el nombre, en la segunda los apellidos y en la tercera el teléfono, pero no incluir Nombre y teléfono en una sola.

2NF -> Cada atributo que no sea una clave debe depender completamente de toda la clave primaria. Por ejemplo una tabla con ID pedido, ID cliente, Fecha del Pedido, ID producto, Nombre Producto, Precio del producto, Cantidad del producto no cumple la segunda NF porque nombre del producto precio del producto y cantidad del producto dependen de ID del pedido que es la clave primaria, no de ID producto. Intenta eliminar la dependencia parcial.

PedidoID	ClienteID	Fecha del pedido	ProductoID	Nombre del producto	Precio del producto	Cantidad del producto

No dependerían de ID pedido, dependerían de ID producto, que es otro atributo a la clave primaria. Se debería crear una tabla de productos.

ProductoID	Nombre del producto	Precio del producto

De esta manera se evita redundancia de información

3NF -> cada atributo debe depender de la clave primaria y no de atributos que no son claves primarias. Dependencia transitiva: cuando un atributo depende de otro atributo y este otro depende de la clave primaria.

ID_Cliente	Nombre	Código_Postal	Ciudad	Estado
1	Juan	0303456	Lima	Lima
2	Luis	505011	Arequipa	Arequipa

La ciudad depende de estado. Se tendría que crear otra tabla.

4NF -> intentamos eliminar la redundancia. Intenta eliminar una dependencia multivaluada. Los valores están relacionados con múltiples valores de otra columna.

ID_Producto	Nombre	Categoría	Subcategoría
1	TV	Electrónica	Pantallas
2	Mouse	Computadoras	Accesorios
3	Teclado	Computadoras	Accesorios
4	Celular	Electrónica	Teléfonos

Aquí categoría y subcategoría están relacionadas. Una subcategoría pertenece a una única categoría pero una categoría puede tener muchas subcategorías.

Categorías principales

Subcategorías

ID_Principal	Nombre_Principal
1	TV
2	Mouse
3	Teclado
4	Celular

ID_Subcategoria	ID_Principal	Nombre_Subcategoria
1	Electrónica	Pantallas
2	Computadoras	Accesorios
3	Computadoras	Accesorios
4	Electrónica	Teléfonos

5NF -> Se asegura que no haya dependencias de unión entre los atributos. Un atributo no debe depender de la unión de los atributos de dos tablas diferentes.

La clave es encontrar el equilibrio entre la normalización y la practicidad.

- Hay que analizar tablas y atributos, identificar claves primarias, después identificar las dependencias funcionales.

 Cientes						
CienteID	Nombre	Dirección	Teléfono	Correo electrónico	Ciudad	Estado
1	Homero	Av Siempreviva 742	5550354	homero@mail.com	Springfield	Massachusetts
2	Walter	308 de Negra Arroyo Lane	1245678	walter@mail.com	Albuquerque	Nuevo México
3	Gale	6353 Juan Tabo	2424589	gale@mail.com	Albuquerque	Nuevo México
4	Chandler	Yemen 123	9985563	chandler@mail.com	Yemen	Yemen

Un ejemplo, tenemos esta tabla con los siguientes atributos: identificamos la clave primaria, sería cliente ID, las dependencias funcionales, todos los datos dependen de la clave primaria, no hay dependencia funcional. Cumpliría las tres primeras formas normales.

Cómo normalizar una tabla:

Tenemos la siguiente tabla:

 Ventas										
VentaID	Fecha	CienteID	Nombre del cliente	Dirección del cliente	Ciudad del cliente	Estado del cliente	Producto ID	Nombre del producto	Precio del producto	Cantidad
1	21-03	1	Juan	Piamonte 123	La Plata	Buenos Aires	1	Vaso	\$4000	4
2	20-08	2	Leo	San Martín 41	Córdoba	Córdoba	2	Lavarropas	\$170.000	1
4	10-10	3	Manuel	Salta 1012	Salta	Salta	3	Televisor	\$60.000	2

- Identificamos las claves primarias: Venta ID, identificamos las dependencias funcionales: ID del cliente determina el nombre, la dirección, la ciudad y el estado. Producto ID determina el nombre del producto y el precio del producto, haya dependencias funcionales parciales. Cumple la primera forma normal.

Ventas	VentaID	Fecha	ClienteID	ProductoID	Precio del producto	Cantidad
	1	21-03	1	1	\$4000	4
	2	20-08	2	2	\$170.000	1
	4	10-10	3	3	\$60.000	2

Clientes	ClienteID	Nombre del cliente	Dirección del cliente	Ciudad del cliente	Estado del cliente
	1	Juan	Delgado 148	Córdoba	Córdoba
	2	Marcelo	Jujuy 779	Maipú	Mendoza
	4	Ernesto	9 de Julio 38	Rio Cuarto	Córdoba

2. Para normalizar a la segunda forma normal hay que dividir en dos tablas. Y cumpliría también la tercera forma normalizada por lo que la tabla estaría formalizada.

Índices

Nos sirve para hacer búsquedas más rápidas.

Índices únicos: como primary key, no permite tener valores nulos, nos permite diferenciar cada fila.

```
CREATE INDEX nombre on Products (ProductName)
```

```
SELECT * FROM Products WHERE ProductName = "Chais"
```

Esto sirve por ejemplo para mejorar búsquedas que pueden ser muy largas. De esta manera el procedimiento es menor. Este índice permite campos nulos y campos duplicados. Si por ejemplo queremos que en la tabla empleados no se repita nombre y apellido podemos hacer:

```
CREATE UNIQUE INDEX name ON Employees (FirstName, LastName)
```

Si pones un nuevo registro con alguno de los nombres y apellidos ya creados saltará un mensaje de error porque ya existe el nombre. De esta manera la búsqueda será mucho más eficiente.

índices no únicos: puede tener campos duplicados, valores nulos. Pero sirve para optimizar la búsqueda de las consultas.

Los índices consumen mucho espacio en disco. A más índices más complejo es mantenerlo. Utilizar índices en columnas donde realmente lo necesiten. Campos para condiciones de búsqueda o de filtrado, donde los usamos en cláusulas where o join, porque se usan constantemente, cuando tienen una alta cardinalidad.

A veces aunque agreguemos índices no nos mejora.

Vistas

Una vista se crea sobre una consulta select que se crea sobre una o varias tablas, no almacenan datos, es una referencia a una consulta que nos devuelve una vista.

Para crear una vista:

```
CREATE VIEW Productos_simplificados AS
SELECT ProductID, ProductName, Price FROM Products
WHERE ProductID > 50
ORDER BY ProductID DESC
```

Y en la pestaña estructura podremos acceder a ella. También tienen un impacto sobre el rendimiento de la tabla.

SQL desde cero

Bloqueos y transacciones

los bloqueos son un mecanismo para regular los accesos concurrentes, cuando dos usuarios entran al mismo tiempo para modificar un campo. Cuando se realiza un bloqueo se bloquea toda la base de datos.

- Bloqueo compartido: nadie puede escribir pero se puede leer. Cuando nosotros estamos leyendo la base de datos.
- Bloqueo reservado: nadie puede escribir, pero se puede leer. Se aplica al escribir en la base de datos.
- Bloqueo exclusivo: cuando estamos escribiendo en una base de datos y no queremos que otra persona lea o escriba.

Begin, Roll back y commit.

Begin: empieza la transacción

Roll back : no asienta el cambio.

Commit: asienta el cambio.

```
BEGIN TRANSACTION;  
  
UPDATE Products SET ProductName = "pezon" WHERE ProductName = "Chais";  
  
COMMIT
```

```
BEGIN TRANSACTION;  
  
UPDATE Products SET ProductName = "pezon" WHERE ProductName = "Chais";  
  
ROLLBACK
```

Procedimientos almacenados

Es un conjunto de instrucciones o comandos que se guardan en la base de datos y que podemos ejecutar en cualquier momento, es como un recetario. Si tenemos que hacer siempre los mismos pasos podemos crear varias consultas, mejora la seguridad, simplifica la programación y aumenta el rendimiento porque los ejecutamos en el servidor de la base de datos. Se ejecuta como si fuera una biblioteca. Las funciones definidas por el usuario se puede ejecutar desde el lenguaje madre (por ejemplo Python).

Funciones definidas por el usuario

Es una función creada para ser ejecutada durante una consulta, toma estos valores, los procesa y devuelve una respuesta, tiene que estar definida en SQLite pero está en lenguaje anfitrión, como Python.