

# SQL

## Objetivo

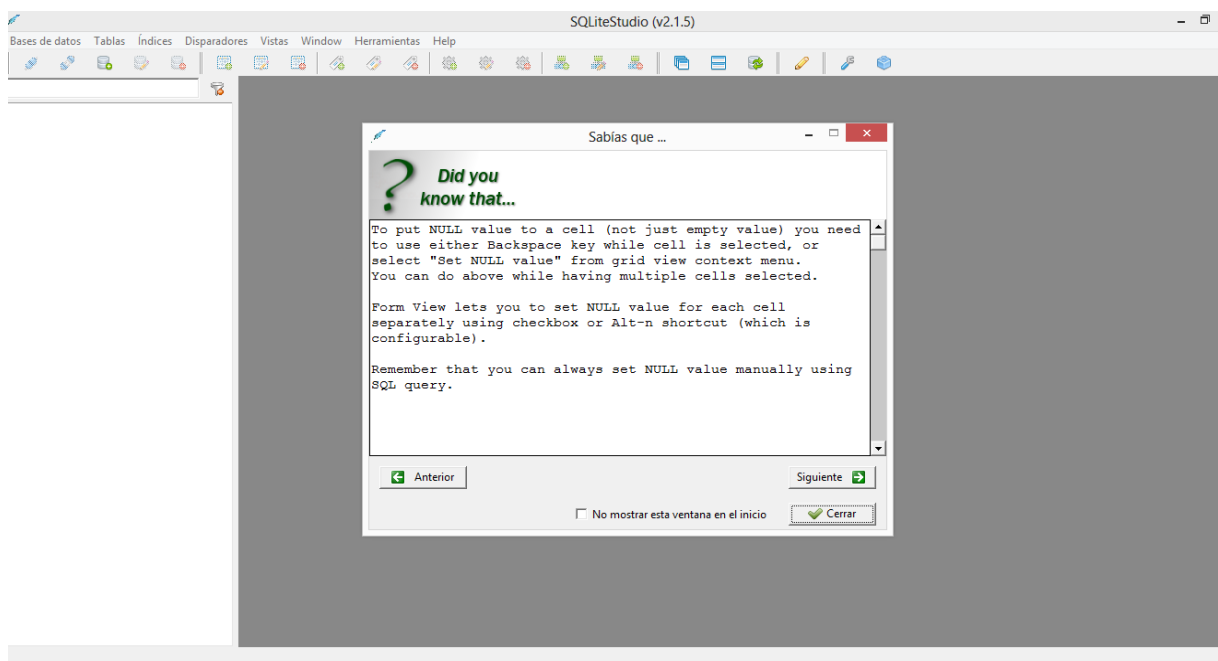
Introducir SQL usando el sistema gestor de bases de datos relacionales SQLite.

## SQLiteStudio

Para poner en práctica SQL se va a utilizar SQLiteStudio un sistema de gestión de bases de datos relacionales que puede descargarse en: <http://sqlitestudio.pl/?act=download>

SQLiteStudio se basa en SQLite, un sistema de gestión de bases de datos relaciones que implementa un subconjunto del estándar SQL. Se puede consultar el subconjunto de SQL soportado por SQLite en la siguiente dirección: <http://www.sqlite.org/lang.html>

La descarga del enlace anterior es un fichero ejecutable. Si se pulsa sobre el ejecutable, aparece la interface principal.



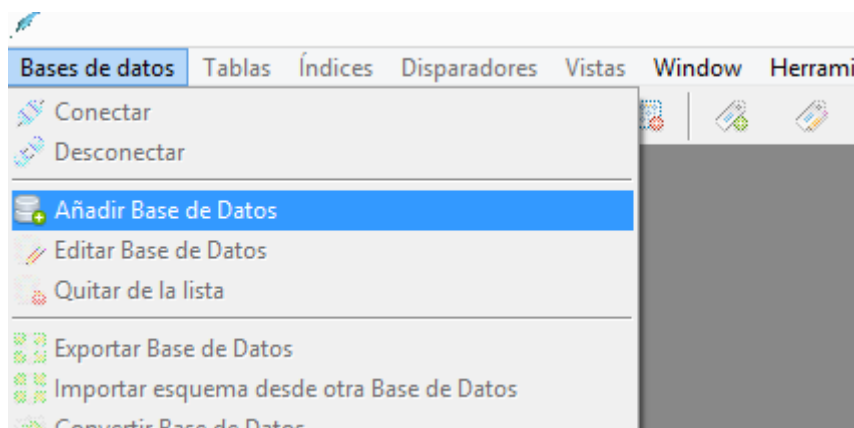
## Ejemplo 1

Se va a considerar una base de datos para gestionar una empresa de arquitectura que tiene sedes en Madrid, Zaragoza y Granada. La información que se desea almacenar es:

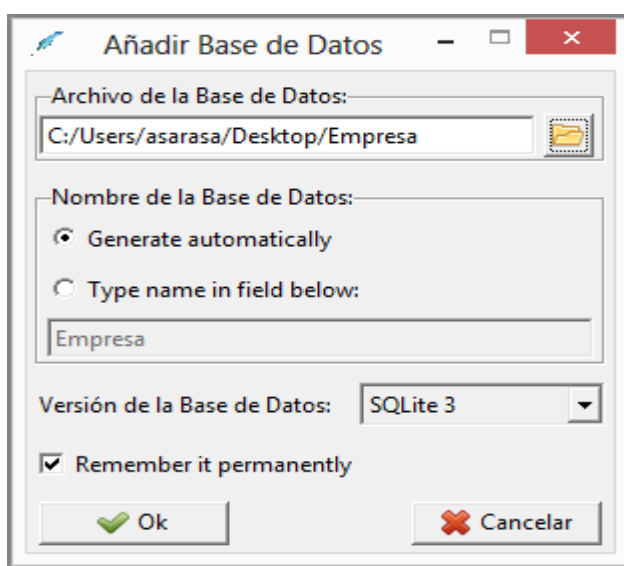
- Sobre los empleados: código de empleado, nombre y apellido, sueldo, nombre y la ciudad de su departamento y el número de proyecto al que están asignados.
- Sobre cada departamento: nombre, ciudad donde se encuentran y teléfono. Un departamento con el mismo nombre puede estar en ciudades diferentes, y en una misma ciudad puede haber departamentos con nombres diferentes.
- Sobre los proyectos: código, nombre, precio, fecha de inicio, fecha prevista de finalización, fecha real de finalización y el código de cliente.
- Sobre los clientes: código de cliente, el nombre, el NIF, la dirección, la ciudad y el teléfono.

En primer lugar hay que crear la base de datos que contendrá las tablas que almacenarán la información:

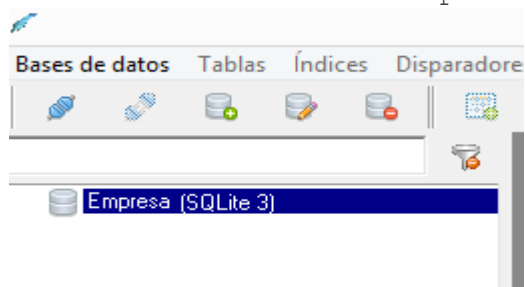
- Se pulsa sobre "Bases de datos".
- En el desplegable que aparece se selecciona "Añadir Bases de Datos".



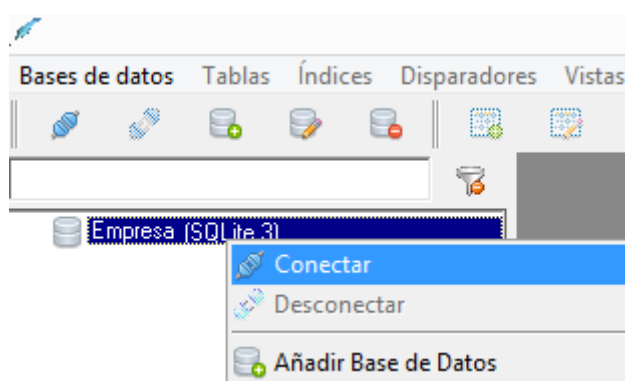
- Aparece un formulario en el que se debe indicar dónde se ubicará la base de datos usando el navegador "Archivo de la Base de Datos". Así mismo se puede elegir el nombre de la base de datos o bien dejar que lo genere automáticamente, y elegir el tipo de base de datos.



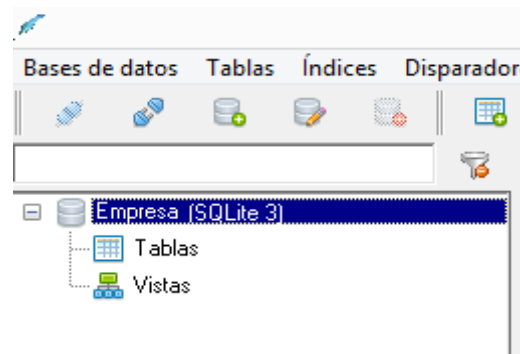
- Cuando se pulsa sobre "OK" la nueva base de datos aparece en el marco izquierdo de la interface principal.



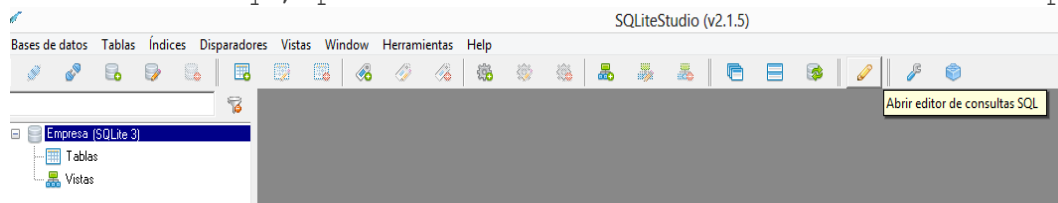
- Para poder usar la base de datos se selecciona con el ratón, y con botón derecho aparece un desplegable en el que se elige "Conectar".



- Una vez que se ha conectado con la base de datos, en el marco izquierdo aparece listadas las tablas y vistas asociadas a la base de datos. En el ejemplo aparecen vacías.




- Para interactuar con la base de datos, se debe abrir el editor de sql, pulsando sobre el icono en forma de lápiz.



- El editor de SQL se muestra en la parte derecha de la interface principal.

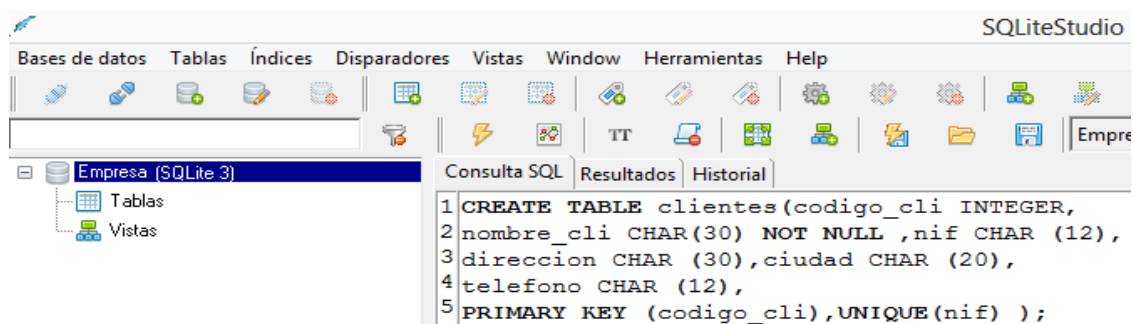


- Vamos a crear las diferentes tablas. Para crear una tabla se puede usar un formulario gráfico (icono ) o bien el editor de SQL. Para ilustrar el uso de SQL, se va a utilizar el editor dónde habrá que introducir las sentencias en la sintaxis de SQL.
- Se crea una tabla clientes para almacenar la información sobre los mismos. Para crear una tabla se utiliza la sentencia **CREATE TABLE**:

```
CREATE      TABLE      nombre_tabla(      definición_columna[,
definición_columna...][, restricciones_tabla]);
```

En este caso se introduce la siguiente sentencia en el editor:

```
CREATE TABLE clientes(codigo_cli INTEGER,
nombre_cli CHAR(30) NOT NULL ,nif CHAR (12), direccion
CHAR (30),ciudad CHAR (20), telefono CHAR (12),PRIMARY
KEY (codigo_cli),UNIQUE(nif) );
```









En el ejemplo se puede ver que las columnas tienen asociado un tipo de datos:

Tipos de datos predefinidos	
Tipos de datos	Descripción
CHARACTER (longitud)	Cadenas de caracteres de longitud fija.
CHARACTER VARYING (longitud)	Cadenas de caracteres de longitud variable.
BIT (longitud)	Cadenas de bits de longitud fija.
BIT VARYING (longitud)	Cadenas de bits de longitud variables.
NUMERIC (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
DECIMAL (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
INTEGER	Números enteros.
SMALLINT	Números enteros pequeños.
REAL	Números con coma flotante con precisión predefinida.
FLOAT (precisión)	Números con coma flotante con la precisión especificada.
DOUBLE PRECISION	Números con coma flotante con más precisión predefinida que la del tipo REAL.
DATE	Fechas. Están compuestas de: YEAR año, MONTH mes, DAY día.
TIME	Horas. Están compuestas de HOUR hora, MINUT minutos, SECOND segundos.
TIMESTAMP	Fechas y horas. Están compuestas de YEAR año, MONTH mes, DAY día, HOUR hora, MINUT minutos, SECOND segundos.

Se pueden definir restricciones a nivel de columna:

Restricciones de columna	
Restricción	Descripción
NOT NULL	La columna no puede tener valores nulos.
UNIQUE	La columna no puede tener valores repetidos. Es una clave alternativa.
PRIMARY KEY	La columna no puede tener valores repetidos ni nulos. Es la clave primaria.
REFERENCES tabla [(columna)]	La columna es la clave foránea de la columna de la tabla especificada.
CHECK (condiciones)	La columna debe cumplir las condiciones especificadas.



- Podéis navegar por las diferentes pestañas (Estructura, Datos, ...) y sobre los iconos inferiores (       ) para investigar su utilidad.
- A continuación se pueden crear el resto de tablas de la base de datos de ejemplo de la misma forma:
  - o Tabla departamentos

```
CREATE TABLE departamentos
(nombre_dep CHAR(20),
ciudad_dep CHAR(20),
telefono INTEGER DEFAULT NULL,
PRIMARY KEY (nombre_dep, ciudad_dep));
```

- o Tabla proyectos

```
CREATE TABLE proyectos
(codigo_proyec INTEGER, nombre_proyec CHAR(20),
precio REAL, fecha_inicio DATE, fecha_prev_fin DATE,
fecha_fin DATE DEFAULT NULL, codigo_cliente INTEGER,
PRIMARY KEY (codigo_proyec),
CHECK (fecha_inicio < fecha_prev_fin),
CHECK (fecha_inicio < fecha_fin),
FOREIGN KEY (codigo_cliente) REFERENCES clientes (
codigo_cli));
```

- o Tabla empleados

```
CREATE TABLE empleados
(codigo_empl INTEGER, nombre_empl CHAR (20),
apellido_empl CHAR(20), sueldo REAL CHECK (sueldo >
7000),
nombre_dep CHAR(20), ciudad_dep CHAR(20),
num_proyec INTEGER, PRIMARY KEY (codigo_empl),
FOREIGN KEY (nombre_dep, ciudad_dep) REFERENCES
departamentos (nombre_dep, ciudad_dep),
FOREIGN KEY (num_proyec) REFERENCES proyectos
(codigo_proyec) );
```

- El resultado final debería ser similar al siguiente:



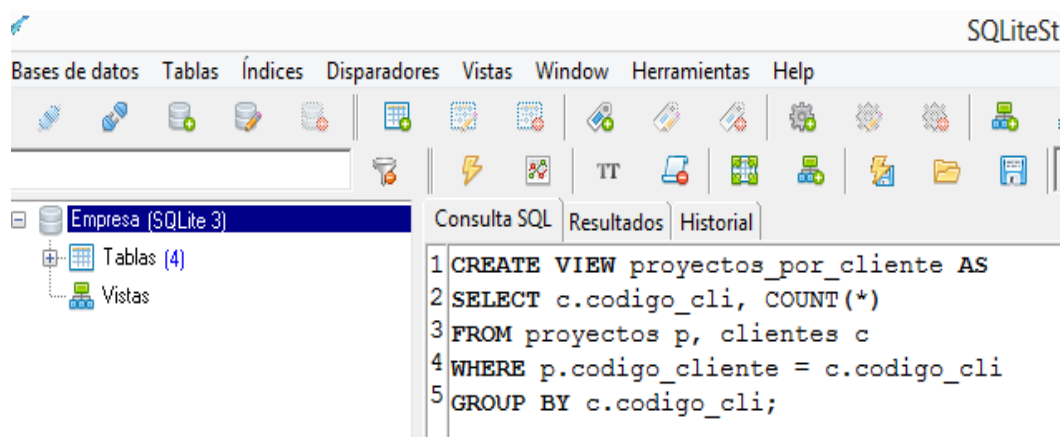
- Observar que al crear una tabla, las restricciones se pueden definir a nivel de columna o bien a nivel de tabla. Así en el caso de que la restricción haga referencia a una sola columna, se puede optar por una opción u otra. Sin embargo si hace referencia a más de una columna debemos definir la restricción a nivel de tabla.
- A continuación vamos a crear una vista utilizando las tablas proyectos y clientes. Para ello se usa el editor de SQL de la misma forma que en el caso de la creación de tablas.
- Una vista es una tabla ficticia(no existen como un conjunto de valores almacenados en la base de datos) que se construye a partir de una consulta a una tabla real. Para definir una vista se usa la siguiente sintaxis:

```
CREATE VIEW nombre_vista [(lista_columnas)] AS (consulta)
[WITH CHECK OPTION];
```

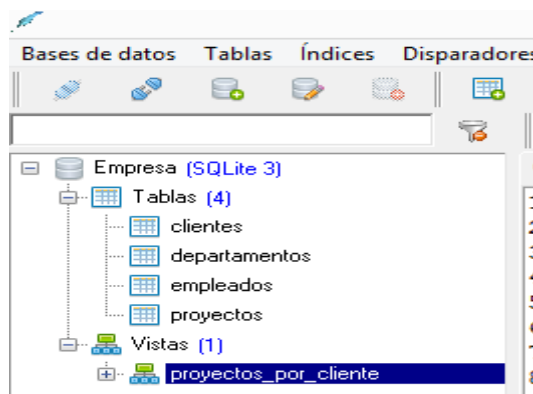
donde se indica el nombre de la vista, a continuación se pueden especificar los nombres de las columnas de la vista, se define la consulta que construirá la vista, y se puede añadir la cláusula "with check option" para evitar inserciones o actualizaciones excepto en los registros en que la cláusula WHERE de la consulta se evalúe como true.

```
CREATE VIEW proyectos_por_cliente AS
SELECT c.codigo_cli, COUNT(*)
FROM proyectos p, clientes c
WHERE p.codigo_cliente = c.codigo_cli
GROUP BY c.codigo_cli;
```





- El resultado de la ejecución debería ser similar al siguiente:



- Por último se puede probar lo siguiente:
  - Eliminar la vista proyectos\_por\_cliente usando la sentencia: `DROP VIEW proyectos_por_cliente`
  - Para borrar una vista se utiliza la sentencia `DROP VIEW`:  
`DROP VIEW nombre_vista (RESTRICT|CASCADE);`  
 donde:
    - La opción `RESTRICT` indica que la vista no se borrará si está referenciada,
    - La opción `CASCADE` indica que todo lo que referencie a la vista se borrará con ésta.
  - Eliminar la tabla clientes usando la sentencia: `DROP TABLE clientes`
  - Para borrar una tabla se utiliza la sentencia `DROP TABLE`:  
`DROP TABLE nombre_tabla {RESTRICT|CASCADE};`  
 donde:

- La opción RESTRICT indica que la tabla no se borrará si está referenciada,
  - La opción CASCADE indica que todo lo que referencie a la tabla se borrará con ésta.
- o Añadir una nueva columna denominada "país" de tipo CHAR(20) a la tabla clientes:  
 ALTER TABLE clientes ADD COLUMN pais CHAR(20)
- o Para modificar una tabla se utiliza la sentencia **ALTER TABLE:**  
**ALTER TABLE** nombre\_tabla {acción\_modificar\_columna|acción\_modif\_restricción\_tabla};  
 donde:
- **acción\_modificar\_columna** puede ser:{ADD [COLUMN] columna def\_columna |ALTER [COLUMN] columna {SET def\_defecto|DROP DEFAULT}|DROP [COLUMN] columna {RESTRICT|CASCADE}}
  - **acción\_modif\_restricción\_tabla** puede ser:{ADD restricción|DROP CONSTRAINT restricción {RESTRICT|CASCADE}}

## Ejemplo 2

Se va a considerar una base de datos para gestionar las solicitudes de acceso de estudiantes a las universidades. La información que se desea almacenar es:

- Sobre las universidades: nombre, comunidad autónoma en la que se encuentra, y número máximo de plazas.
- Sobre los estudiantes: identificador de estudiante, nombre, nota obtenida y un valor de corrección.
- Sobre las solicitudes: identificador de estudiante, nombre de universidad, carrera solicitada y decisión sobre la solicitud.

Se van a crear 3 tablas: universidades, estudiantes y solicitudes.

```
CREATE TABLE Universidades(Nombre_Univ CHAR(35),
Comunidad CHAR(35), Plazas INTEGER,
PRIMARY KEY ( Nombre_Univ ),
UNIQUE ( Nombre_Univ, Comunidad ));
```

```
CREATE TABLE Estudiantes(ID INTEGER, Nombre_Est
CHAR(35), Nota REAL, Valor INTEGER,
PRIMARY KEY ( ID ),
UNIQUE ( ID ) );
```

```
CREATE TABLE Solicitudes(ID INTEGER, Nombre_Univ
CHAR(35), Carrera CHAR(35), Decision CHAR(35),
PRIMARY KEY ( ID, Nombre_Univ, Carrera ),
FOREIGN KEY ( ID ) REFERENCES Estudiantes ( ID ),
FOREIGN KEY ( Nombre_Univ ) REFERENCES
Universidades ( Nombre_Univ ),
UNIQUE ( ID, Nombre_Univ, Carrera ) );
```

A continuación se rellenan las tablas con los siguientes datos:

Tabla Universidades


Nombre_Univ	Comunidad	Plazas
Universidad Complutense de Madrid	Madrid	15000
Universidad de Barcelona	Barcelona	36000
Universidad de Valencia	Valencia	10000
UPM	Madrid	21000

Tabla Estudiantes

ID	Nombre_Est	Nota	Valor
123	Antonio	8.9	1000
234	Juan	8.6	1500
345	Isabel	8.5	500
456	Doris	7.9	1000
543	Pedro	5.4	2000
567	Eduardo	6.9	2000
654	Alfonso	7.9	1000
678	Carmen	5.8	200
765	Javier	7.9	1500
789	Isidro	8.4	800
876	Irene	6.9	400
987	Elena	6.7	800

Tabla Solicitudes

ID	Nombre_Univ	Carrera	Decision
123	Universidad Complutense de Madrid	Informatica	Si
123	Universidad Complutense de Madrid	Economia	No
123	Universidad de Barcelona	Informatica	Si
123	UPM	Economia	Si
234	Universidad de Barcelona	Biologia	No
345	Universidad de Valencia	Bioingenieria	Si
345	UPM	Bioingenieria	No
345	UPM	Informatica	Si
345	UPM	Economia	No
678	Universidad Complutense de Madrid	Historia	Si
987	Universidad Complutense de Madrid	Informatica	Si
987	Universidad de Barcelona	Informatica	Si
876	Universidad Complutense de Madrid	Informatica	No
876	Universidad de Valencia	Biologia	Si
876	Universidad de Valencia	Biologia Marina	No
765	Universidad Complutense de Madrid	Historia	Si
765	UPM	Historia	No
765	UPM	Psicologia	Si
543	Universidad de Valencia	Informatica	No

En el archivo creación.sql se encuentran las sentencias SQL para generar y rellenar las tablas. Para ello se usa la opción de ejecutar SQL desde archivo pulsando sobre el icono , el cual muestra un explorador de archivos que permite seleccionar el archivo, y ejecutarlo.

A continuación se van a ilustrar diferentes casos de uso de SQL.

### **CONSULTAS EN SQL**

```
SELECT [ALL | DISTINCT ]
        <nombre_campo> [{,<nombre_campo>}]
FROM <nombre_tabla>|<nombre_vista>
        [{,<nombre_tabla>|<nombre_vista>}]
[WHERE <condicion> [{ AND|OR <condicion>}]]
[GROUP BY <nombre_campo> [{,<nombre_campo> }]]
[HAVING <condicion>[{ AND|OR <condicion>}]]
[ORDER BY <nombre_campo>|<indice_campo> [ASC | DESC]
        [{,<nombre_campo>|<indice_campo> [ASC | DESC
]]]]
```

**SELECT** Palabra clave que indica que la sentencia de SQL que queremos ejecutar es de selección.

**ALL** Indica que queremos seleccionar todos los valores. Es el valor por defecto y no suele especificarse casi nunca.

**DISTINCT** Indica que queremos seleccionar sólo los valores distintos.

**FROM** Indica la tabla (o tablas) desde la que queremos recuperar los datos. En el caso de que exista más de una tabla se denomina a la consulta "consulta combinada" o "join". En las consultas combinadas es necesario aplicar una condición de combinación a través de una cláusula WHERE.

**WHERE** Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Admite los operadores lógicos AND y OR.

**GROUP BY** Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.

**HAVING** Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de WHERE pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a GROUP BY y la condición debe

estar referida a los campos contenidos en ella.

Presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con ASC (orden ascendente) y DESC (orden descendente). El valor predeterminado es ASC.

### Cláusula WHERE

La cláusula WHERE es la instrucción que nos permite filtrar el resultado de una sentencia SELECT. Habitualmente no deseamos obtener toda la información existente en la tabla, sino que queremos obtener sólo la información que nos resulte útil en ese momento. La cláusula WHERE filtra los datos antes de ser devueltos por la consulta. Cuando en la Cláusula WHERE queremos incluir un tipo texto, debemos incluir el valor entre comillas simples.

### Cláusula ORDER BY

La cláusula ORDER BY es la instrucción que nos permite especificar el orden en el que serán devueltos los datos. Podemos especificar la ordenación ascendente o descendente a través de las palabras clave ASC y DESC. La ordenación depende del tipo de datos que este definido en la columna, de forma que un campo numérico será ordenado como tal, y un alfanumérico se ordenará de la A a la Z, aunque su contenido sea numérico. El valor predeterminado es ASC si no se especifica al hacer la consulta.

### Cláusula GROUP BY

La cláusula GROUP BY permite agrupar las filas según las columnas indicadas, excepto aquellas afectadas por funciones de agregación

Las funciones de agregación son funciones que permiten realizar operaciones sobre los datos de una columna. Algunas funciones son las siguientes:

Funciones de agregación	
Función	Descripción
COUNT	Nos da el número total de filas seleccionadas
SUM	Suma los valores de una columna
MIN	Nos da el valor mínimo de una columna
MAX	Nos da el valor máximo de una columna
AVG	Calcula el valor medio de una columna

En general, las funciones de agregación se aplican a una columna, excepto COUNT que se aplica a todas las columnas de las tablas seleccionadas. Se indica como COUNT (\*).

Sin embargo si se especifica `COUNT(distinct columna)`, entonces sólo contará los valores que no nulos ni repetidos, y se especifica `COUNT (columna)`, sólo contaría los valores que no nulos.

### Cláusula HAVING

La cláusula HAVING especifica las condiciones para recuperar grupos de filas.

### CONSULTAS GENERALES

- Obtener los identificadores, nombres y notas de los estudiantes con una nota mayor de 7.  

```
SELECT ID, Nombre_Est, Nota FROM ESTUDIANTES WHERE Nota > 7
```
- Obtener los nombres de los estudiantes y las carreras que han solicitado.  

```
SELECT DISTINCT Nombre_Est, Carrera FROM Estudiantes, Solicitudes
WHERE Estudiantes.ID = Solicitudes.ID;
```
- Obtener los nombres y notas de los estudiantes así como el resultado de su solicitud de manera que tengan un valor de corrección menor que 1000 y hayan solicitado la carrera de Informática en la Universidad Complutense de Madrid.  

```
SELECT Nombre_Est, Nota, Decision FROM Estudiantes,Solicitudes
WHERE Estudiantes.ID = Solicitudes.ID
AND Valor < 1000 AND Carrera = 'Informatica' AND Nombre_Univ =
'Universidad Complutense de Madrid';
```
- Obtener todas las universidades que tienen más de 20000 plazas e imparten la carrera de Informática.  

```
SELECT DISTINCT Universidades. Nombre_Univ
FROM Universidades, Solicitudes
WHERE Universidades.Nombre_Univ = Solicitudes.Nombre_Univ
AND Plazas > 20000 AND Carrera = 'Informatica';
```
- Obtener la información sobre todas las solicitudes: ID y nombre del estudiante, nombre de la universidad, nota y plazas, ordenadas de forma decreciente por las notas y en orden creciente de plazas.  

```
SELECT Estudiantes.ID, Nombre_Est, Nota, Solicitudes.Nombre_Univ,
Plazas FROM Estudiantes, Universidades, Solicitudes
WHERE Solicitudes.ID = Estudiantes.ID AND Solicitudes.Nombre_Univ =
Universidades.Nombre_Univ
ORDER BY Nota DESC, Plazas;
```
- Obtener todas las solicitudes a carreras que tengan relación con la biología.  

```
SELECT * FROM Solicitudes WHERE Carrera like '%bio%';
```

- Obtener la información del producto escalar de las tablas Estudiantes y Universidades.  
SELECT \* FROM Estudiantes, Universidades;
- Obtener la información del estudiante junto a la nota ponderada usando la columna valor.  
SELECT ID, Nombre\_Est, Nota, Valor, Nota\*(Valor/1000.0) AS Ponderacion FROM Estudiantes;
- Contar el número de estudiantes con nota distinto de nulo.  
SELECT COUNT(\*) FROM Estudiantes WHERE Nota IS NOT NULL;
- Contar el número de notas diferentes entre los estudiantes.  
SELECT COUNT(DISTINCT Nota) FROM Estudiantes WHERE Nota IS NOT NULL;
- Obtener los estudiantes cuya nota ponderada cambia en más de un punto respecto a la nota original.  
SELECT ID, Nombre\_Est, Nota, Nota\*Valor/1000.0 as Ponderada FROM Estudiantes WHERE ABS(Nota\*(Valor/1000.0) - Nota) > 1.0;
- Obtener los pares nombre de universidad, comunidad autónoma y nota de las solicitudes con mayor nota.  
SELECT DISTINCT Universidades.Nombre\_Univ, Comunidad, Nota FROM Universidades, Solicitudes, Estudiantes  
WHERE Universidades.Nombre\_Univ = Solicitudes.Nombre\_Univ AND Solicitudes.ID = Estudiantes.ID  
AND NOTA >= (SELECT NOTA FROM Estudiantes, Solicitudes WHERE Estudiantes.ID = Solicitudes.ID and Solicitudes.Nombre\_Univ = Universidades.Nombre\_Univ);
- Obtener toda la información sobre las solicitudes.  
SELECT E.ID, E.Nombre\_Est, E.Nota, S.Nombre\_Univ, U.Plazas FROM Estudiantes E, Universidades U, Solicitudes S  
WHERE S.ID = E.ID and S.Nombre\_Univ = U.Nombre\_Univ;
- Estudiantes con la misma nota.  
SELECT E1.ID, E1.Nombre\_Est, E1.Nota, E2.ID, E2.Nombre\_Est, E2.Nota FROM Estudiantes E1, Estudiantes E2  
WHERE E1.Nota = E2.Nota and E1.ID < E2.ID;
- Lista de nombres de universidades y estudiantes.  
SELECT Nombre\_Univ AS Nombre FROM Universidades  
UNION ALL  
SELECT Nombre\_Est AS Nombre FROM Estudiantes  
ORDER BY Nombre;



- IDS de estudiantes que solicitaron Informática y Económicas.  

```
SELECT ID FROM Solicitudes WHERE Carrera = 'Informatica'
INTERSECT
SELECT ID FROM Solicitudes WHERE Carrera = 'Economicas';
```
- IDS de estudiantes que solicitaron Informática pero no Económicas.  

```
SELECT ID FROM Solicitudes WHERE Carrera = 'Informatica'
EXCEPT
SELECT ID FROM Solicitudes WHERE Carrera = 'Economicas';
```

## **MODIFICACIONES EN SQL**

### **INSERTAR FILAS EN UNA TABLA**

Una sentencia INSERT de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

```
INSERT INTO 'tablatura' ('columna1', ['columna2,... ']) VALUES
('valor1', ['valor2,...'])
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia INSERT deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

### **MODIFICAR UNA FILA DE UNA TABLA**

Una sentencia UPDATE de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

```
UPDATE nombre_tabla SET columna = {expresión|DEFAULT|NULL}
[, columna = {expr|DEFAULT|NULL} ...] WHERE condiciones;
```

La cláusula SET indica qué columna modificar y los valores que puede recibir, y la cláusula WHERE especifica qué filas deben actualizarse.

### **ELIMINAR UNA FILA DE UNA TABLA**

Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla.

```
DELETE FROM tabla WHERE columna1 = 'valor1'
```

## MODIFICACIÓN DE TABLAS

- Insertar las siguientes filas en la tabla Estudiantes:  
(432,"José",null,1500) y (321,"María",null,2500)  
`INSERT INTO Estudiantes VALUES (432, 'José', null, 1500);`  
`INSERT INTO Estudiantes VALUES (321, 'Maria', null, 2500);`
- Insertar una nueva universidad: ("Universidad de Jaen",  
"Andalucia",11500)  
`INSERT INTO Universidades VALUES ('Universidad de Jaen', 'Andalucia', 11500);`
- Modificar la tabla solicitudes de forma que aquellos estudiantes que no solicitaron ninguna universidad, soliciten "Informática" en la "Universidad de Jaen".  
`INSERT INTO Solicitudes SELECT ID, 'Universidad de Jaen', 'Informatica', null FROM Estudiantes WHERE ID NOT IN (SELECT ID FROM Solicitudes);`
- Admitir en la "Universidad de Jaen" a todos los estudiantes de Económicas quienes no fueron admitidos en dicha carrera en otras universidades.  
`INSERT INTO Solicitudes SELECT ID, 'Universidad de Jaen', 'Economicas', 'Si' FROM Estudiantes WHERE ID IN (SELECT ID FROM Solicitudes WHERE Carrera='Economicas' AND Decision='No');`
- Borrar a todos los estudiantes que solicitaron más de 2 carreras diferentes.  
`DELETE FROM Solicitudes WHERE ID IN (SELECT ID FROM Solicitudes GROUP BY ID HAVING COUNT(DISTINCT Carrera) >2);`
- Borrar las universidades que no tengan solicitudes de "Informatica".  
`DELETE FROM Universidades WHERE Nombre_Univ NOT IN (SELECT Nombre_Univ FROM Solicitudes WHERE Carrera = 'Informatica');`
- Aceptar las solicitudes a la Universidad de Jaén con una nota menor de 7, cambiando la carrera solicitada a Economicas.  
`UPDATE Solicitudes SET Decision= 'Y', Carrera= 'Economicas' WHERE Nombre_Univ= 'Universidad de Jaen' AND ID IN (SELECT ID FROM Estudiantes WHERE Nota < 7);`
- Convertir la solicitud para la carrera de Economicas con la mayor nota en una solicitud para la carrera de Informatica.  
`UPDATE Solicitudes SET Carrera='Informatica' WHERE Carrera= 'Economicas' AND ID IN (SELECT ID FROM Estudiantes WHERE Nota >= (SELECT Nota FROM Estudiantes WHERE ID in (SELECT ID FROM Solicitudes WHERE Carrera='Economicas')));`

- Dar a todo el mundo la máxima nota y el más pequeño valor de ponderación.  
`UPDATE Estudiantes SET Nota=(SELECT MAX(Nota) FROM Estudiantes),  
 Valor=(SELECT MIN(Valor) FROM Estudiantes);`
- Aceptar todas las solicitudes.  
`UPDATE Solicitudes SET Decision= 'Si';`

### **FUNCIONES DE AGREGACIÓN**

- Hallar la nota media de todos los estudiantes.  
`SELECT AVG(Nota) FROM Estudiantes`
- Obtener la nota más pequeña de los estudiantes que han solicitado la carrera de Informatica.  
`SELECT AVG(Nota) FROM Estudiantes WHERE ID IN (SELECT ID FROM Solicitudes WHERE Carrera= 'Informatica');`
- Obtener el número de Universidades que tienen más de 15000 plazas.  
`SELECT COUNT(*) FROM Universidades WHERE Plazas> 15000;`
- Obtener el número de estudiantes que han solicitado plaza en la "Universidad Complutense de Madrid".  
`SELECT COUNT(DISTINCT ID) FROM Solicitudes WHERE Nombre_Univ= 'Universidad Complutense de Madrid';`
- Obtener el número de solicitudes a cada universidad.  
`SELECT Nombre_Univ, COUNT(*) FROM Solicitudes GROUP BY Nombre_Univ;`
- Obtener las carreras en las que la nota máxima de las solicitudes está por debajo de la media.  
`SELECT Carrera FROM Estudiantes, Solicitudes WHERE Estudiantes.ID= Solicitudes.ID GROUP BY Carrera  
 HAVING MAX(Nota) < (Select AVG(Nota) FROM Estudiantes);`
- Obtener las universidades con menos de 5 solicitudes.  
`SELECT Nombre_Univ FROM Solicitudes GROUP BY Nombre_Univ HAVING  
 COUNT(DISTINCT ID) < 5;`
- Obtener el número de universidades solicitadas por cada estudiante, incluyendo 0 en el caso de no haberse solicitado ninguna.  
`SELECT Estudiantes.ID, COUNT(DISTINCT Nombre_Univ) FROM Estudiantes,  
 Solicitudes  
 WHERE Estudiantes.ID = Solicitudes.ID GROUP BY Estudiantes.ID  
 UNION  
 SELECT ID,0  
 FROM Estudiantes  
 WHERE ID NOT IN (SELECT ID FROM Solicitudes);`

- Obtener los estudiantes tales que el número de los otros estudiantes con la misma nota es igual al número de los otros estudiantes con el mismo valor de ponderación.  

```
SELECT * FROM Estudiantes E1 WHERE
(SELECT COUNT(*) FROM Estudiantes E2 WHERE E2.ID <> E1.ID AND E2.Nota
= E1.Nota)=
(SELECT COUNT(*) FROM Estudiantes E2 WHERE E2.ID <> E1.ID AND
E2.Valor = E1.Valor)
```
- Obtener el número de universidades solicitadas por cada estudiante.  

```
SELECT Estudiantes.ID, Nombre_Est, COUNT(DISTINCT Nombre_Univ),
Nombre_Univ FROM Estudiantes, Solicitudes
WHERE Estudiantes.ID = Solicitudes.ID GROUP BY Estudiantes.ID
```
- Obtener las plazas de universidad por comunidad autónoma.  

```
SELECT Comunidad, SUM(Plazas) FROM Universidades GROUP BY Comunidad;
```
- Obtener la máxima y mínima nota de las solicitudes por universidad y carrera.  

```
SELECT Nombre_Univ, Carrera, MIN(Nota), MAX(Nota) FROM Estudiantes,
Solicitudes
WHERE Estudiantes.ID = Solicitudes.ID GROUP BY Nombre_Univ, Carrera;
```
- Obtener la diferencia entre la media de las notas de los estudiantes que solicitaron Informática y la media de las notas de los estudiantes que no la solicitaron.  

```
SELECT DISTINCT (SELECT AVG(Nota) AS NotaMedia FROM Estudiantes WHERE
ID IN
(SELECT ID FROM Solicitudes WHERE Carrera='Informatica'))-
(SELECT AVG(Nota) AS NotaMedia FROM Estudiantes WHERE ID NOT IN (
SELECT ID FROM Solicitudes WHERE Carrera= ' Informatica')) AS d FROM
Estudiantes;
```

## OPERACIONES ENTRE TABLAS

- Obtener los nombres de los estudiantes y las carreras que han solicitado.  

```
SELECT DISTINCT Nombre_Est, Carrera FROM Estudiantes JOIN Solicitudes
ON Estudiantes.ID = Solicitudes.ID;
```
- Obtener el nombre de las notas de los estudiantes con valor de ponderación menor de 1000 que hayan solicitado Informática en la “Universidad de Valencia”.  

```
SELECT Nombre_Est, Nota FROM Estudiantes JOIN Solicitudes ON
Estudiantes.ID = Solicitudes.ID
AND Valor < 1000 AND Carrera='Informatica' AND Nombre_Univ=
'Universidad de Valencia';
```
- Obtener la siguiente información de una solicitud: ID, nombre del estudiante, nota, nombre de la universidad y plazas de la universidad.

```
SELECT Solicitudes.ID, Nombre_Est, Nota, Solicitudes.Nombre_Univ,
Plazas
FROM Solicitudes JOIN Estudiantes JOIN Universidades ON Solicitudes.
ID= Estudiantes.ID
AND Solicitudes.Nombre_Univ = Universidades.Nombre_Univ;
```

- Obtener los nombres de los estudiantes y las carreras que han solicitado.

```
SELECT DISTINCT Nombre_Est, Carrera FROM Estudiantes INNER JOIN
Solicitudes ON Estudiantes.ID = Solicitudes.ID;
```

O bien

```
SELECT DISTINCT Nombre_Est, Carrera FROM Estudiantes NATURAL JOIN
Solicitudes;
```

- Obtener el nombre de las notas de los estudiantes con valor de ponderación menor de 1000 que hayan solicitado Informatica en la "Universidad de Valencia".

```
SELECT Nombre_Est, Nota FROM Estudiantes JOIN Solicitudes USING(ID)
WHERE Valor<1000 AND Carrera='Informatica'
AND Nombre_Univ='Universidad de Valencia';
```

- Obtener los pares de estudiantes con la misma nota.

```
SELECT E1.ID, E1.Nombre_Est, E1.Nota, E2.ID,E2.Nombre_Est, E2.Nota
FROM Estudiantes E1 JOIN Estudiantes E2 USING (Nota)
WHERE E1.ID < E2.ID;
```

- Obtener la siguiente información de la solicitud de un estudiante:Nombre, ID, nombre de la universidad,y carrera.

```
SELECT Nombre_Est, ID, Nombre_Univ, Carrera FROM Estudiantes LEFT
JOIN Solicitudes USING(ID);
```