SQL

**KHAN-apuntes**

<https://es.khanacademy.org/computing/computer-programming/sql#relational-queries-in-sql>

Píldoras Informáticas

Contenido

[Contenido 2](#_Toc22755122)

[ALTER 7](#_Toc22755123)

[AND / OR / NOT 8](#_Toc22755124)

[AS 9](#_Toc22755125)

[BETWEN 10](#_Toc22755126)

[CASE 11](#_Toc22755127)

[COUNT 13](#_Toc22755128)

[CREATE TABLE 14](#_Toc22755129)

[CRUD (modelo) 15](#_Toc22755130)

[DELETE 16](#_Toc22755131)

[DROP 17](#_Toc22755132)

[GROUP BY 18](#_Toc22755133)

[HAVING 19](#_Toc22755134)

[IN / NOT IN 20](#_Toc22755135)

[INSERT INTO \_\_ VALUES / 21](#_Toc22755136)

[JOIN 22](#_Toc22755137)

[Join;LEFT / RIGHT/ FULL OUTER JOIN 26](#_Toc22755138)

[Join:SELF JOIN 29](#_Toc22755139)

[LIKE 31](#_Toc22755140)

[ORDER BY (asc / des ) 33](#_Toc22755141)

[SELECT \_\_ FROM 34](#_Toc22755142)

[SUBCONSULTAS 35](#_Toc22755143)

[UPDATE 38](#_Toc22755144)

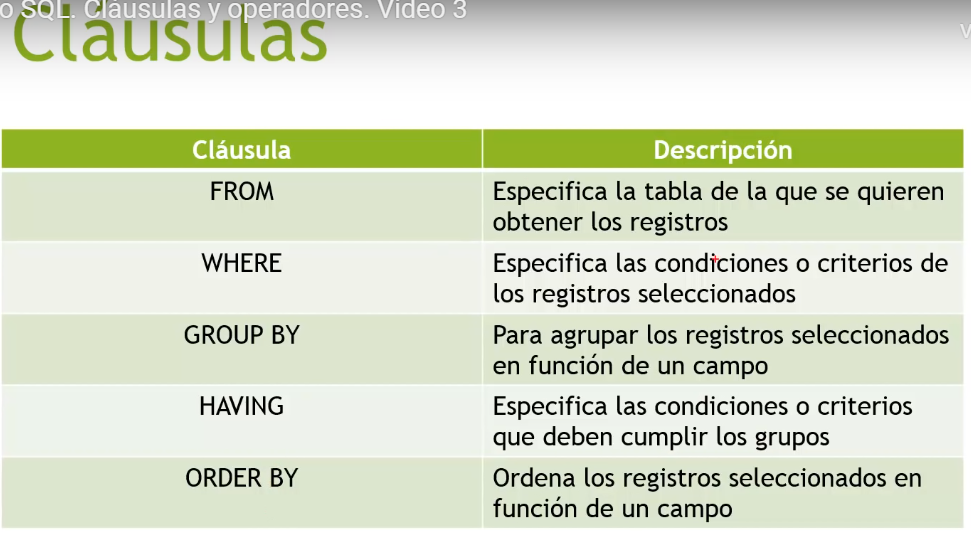
[WHERE 39](#_Toc22755145)

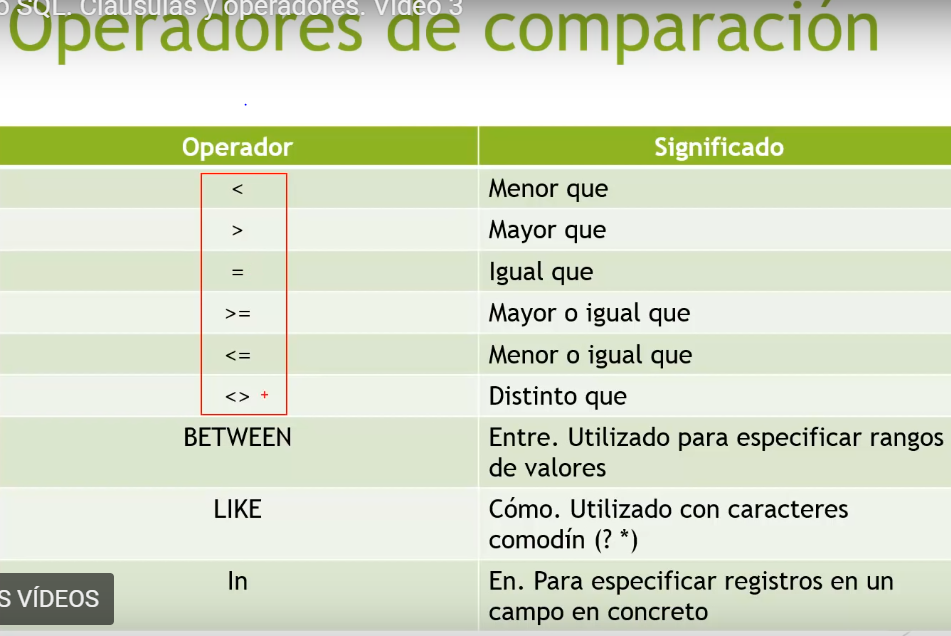
[SUM , MAX, MIN,AVG( 39](#_Toc22755146)

[Separar datos en tablas relacionadas 40](#_Toc22755147)



1. **Comando**
2. **FROM**
3. **WHERE**
4. **GROUP BY**
5. **HAVING**
6. **ORDER BY**







**COMILLAS**

**En SQL estricto se usan comillas SIMPLES ‘ ‘**

**Algunas aplicaciones admiten DOBLES “ “**

**En SQL se acaban las sentencias con ; aunque muchas aplicaciones no lo requieran**

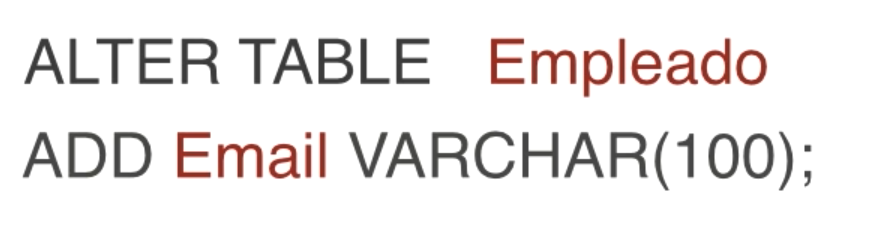


**ACID**

**Las transacciones en bases de dados tienen que ser**



## ALTER

**/\*\* Permite modificar una tabla. Añadimos un nuevo campo para email con 100 caracteres como máximo \*\*/** 

## AND / OR / NOT

**/\*\* AND tiene prioridad sobre OR, pero con paréntesis podemos lograr la expresión deseada \*\*/**

**/\*\* AND selecciona calorias >50 y más de 30 minutos \*\*/**

**SELECT \* FROM exercise\_logs WHERE calories > 50 AND minutes < 30;**

**/\* \*OR selcciona >50calorias y ritmo cardíaco>100 \* \*/**

**SELECT \* FROM exercise\_logs WHERE calories > 50 OR**

**heart\_rate > 100;**

## AS

**/\*\* crea un nombre para la columna (resultado de sumar calorías) \*\*/**

**SELECT type, SUM(calories) AS total\_calories FROM exercise\_logs GROUP BY type;**

## BETWEN

**/\*\* consulta entre dos parámetros. En el ejemplo son fechas \*\*/**

**SELECT \* FROM productos WHERE fecha**

**BETWEEN ‘300-03-01’ AND ‘200-04-03’**

**/\*\* equivale a: \*\*/**

**SELECT \* FROM productos WHERE**

**fecha>=’200-03-01’ AND fecha <=’200-04-03’**

## CASE

**/\* CASE funciona como un switch o iF para seleccionar opciones \*/**

**SELECT type, heart\_rate,**

**CASE**

**WHEN heart\_rate > 220-30 THEN "above max"**

**WHEN heart\_rate > ROUND(0.90 \* (220-30)) THEN "above target"**

**WHEN heart\_rate > ROUND(0.50 \* (220-30)) THEN "within target"**

**ELSE "below target"**

**END as "hr\_zone"**

**FROM exercise\_logs;**

**/\*\* para agrupar esas consultas por zonas \*\*/**

**SELECT COUNT(\*),**

**CASE**

**WHEN heart\_rate > 220-30 THEN "above max"**

**WHEN heart\_rate > ROUND(0.90 \* (220-30)) THEN "above target"**

**WHEN heart\_rate > ROUND(0.50 \* (220-30)) THEN "within target"**

**ELSE "below target"**

**END as "hr\_zone"**

**FROM exercise\_logs**

**GROUP BY hr\_zone;**

## COUNT

**/\*\* cuenta todos los elementos de la tabla \*\*/**

**SELECT COUNT (\*) FROM personas**

**/\*\* cuenta todos los elementos de la tabla con Salrio>5000 \*\*/**

**SELECT COUNT (\*) FROM personas**

**WHERE Salario>5000;**

**/\*\* NO SE PUEDEN MEZCLAR COLUMNAS AL USAR COUNT \*\*/**

**SELECT COUNT (\*), color FROM producto ;**

**/\*\*daría error. Necesitamos poner ORDER BY\*\*/**

**SELECT COUNT (\*), color FROM producto GROUP BY color;**

## CREATE TABLE

**/\*\*crear tabla identificando columnas) \*\*/**

**CREATE TABLE groceries (**

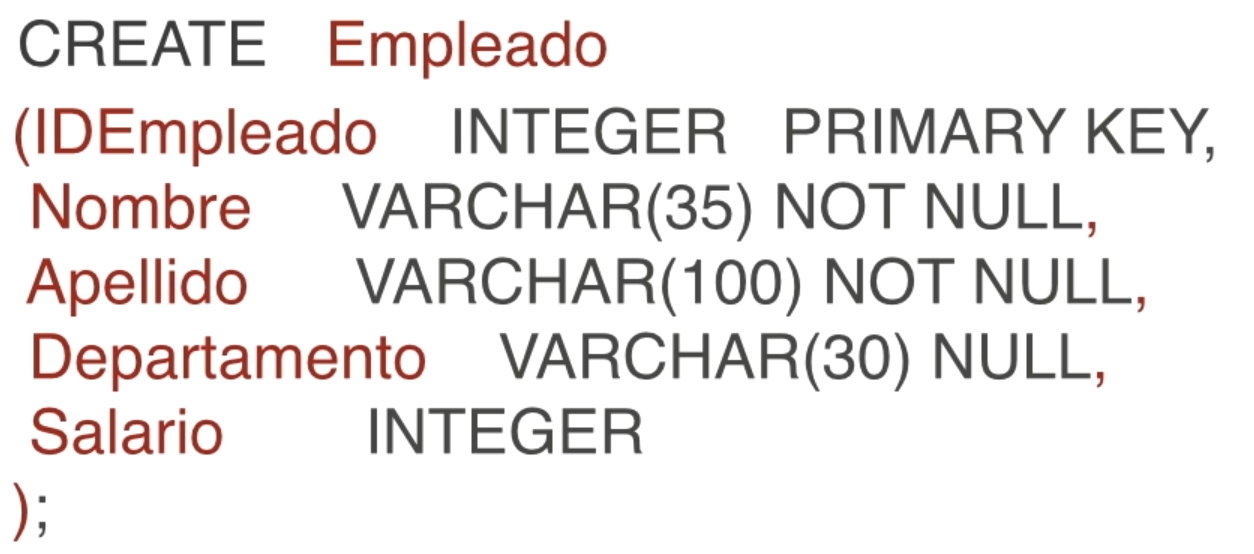
**id INTEGER PRIMARY KEY,**

**name TEXT, quantity INTEGER );**

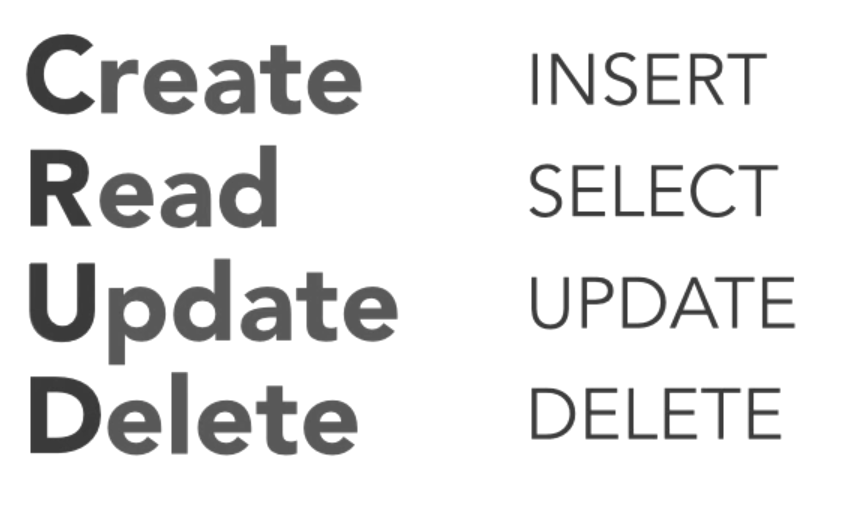
**/\*\* crear tabla configurando id autogenerada\*\*/**

**CREATE TABLE exercise\_logs**

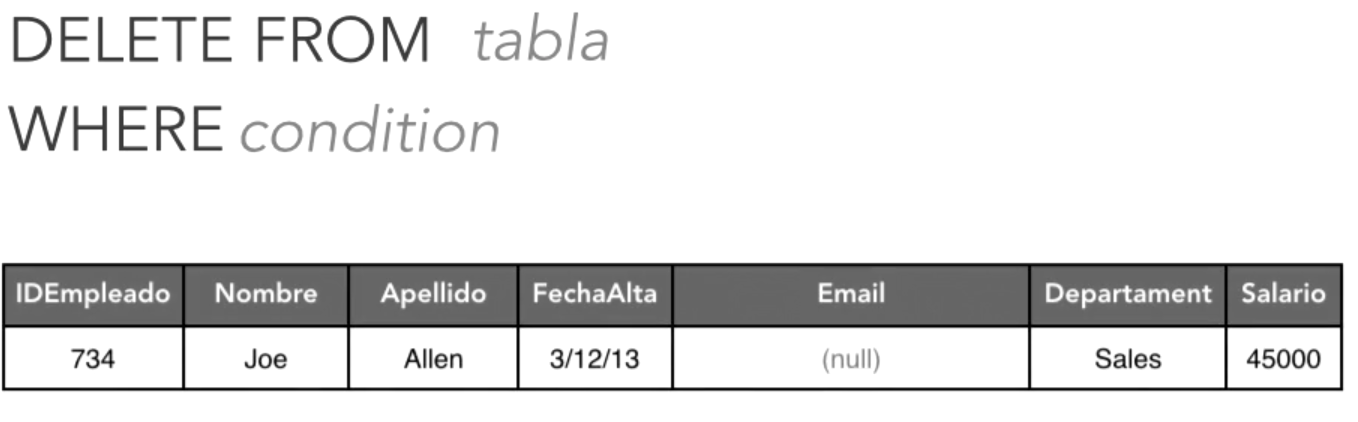
**(id INTEGER PRIMARY KEY AUTOINCREMENT,**

**type TEXT);** 

## CRUD (modelo)



## DELETE



**/\*\* Poner condición para evitar borrar tabla \*\*/**



## DROP

**/\*\* elimina una tabla !!!! \*\*/**

**DROP TABLE empleado**

## GROUP BY

**/\*\* agrupa pasillos y luego muestra la suma de los elementos**

**de cada pasillo sin especificar pasillo \*\*/**

**SELECT SUM(quantity) FROM groceries GROUP BY aisle;**

**/\*\* muestra pasillo y la suma de los elementos de ese pasillo (ailse). \*\*/**

**SELECT aisle, SUM(quantity) FROM groceries GROUP BY aisle;**

## HAVING

**/\*\* filtra valores para un resultado AGRUPADO, no para cada valor individual de la tabla . La suma de calorías la ponemos en una columna llamada total\_calories i luego con HAVING pedimos que el resultado total de esas sumas >150. (Es fácil confundir HAVING con WHERE \*\*/**

**SELECT type, SUM(calories) AS total\_calories FROM exercise\_logs**

**GROUP BY type HAVING total\_calories > 150**

**HAVING COUNT**

**/\*\* COUNT: mostrará type que tenga 2 o más valores \*\*/**

**SELECT type FROM exercise\_logs GROUP BY type HAVING COUNT(\*) >= 2;**

## IN / NOT IN

**/\*\* para seleccionar varios valores . Primero sin IN \*\*/**

**SELECT \* FROM exercise\_logs WHERE type = "biking" OR type = "hiking" OR type = "tree climbing" OR type = "rowing";**

**/\* Con IN \*/**

**SELECT \* FROM exercise\_logs WHERE type IN ("biking", "hiking", "tree climbing", "rowing");**

**/\*\* también podemos poner los valores que no cumplen con NOT IN \*\*/**

**SELECT \* FROM exercise\_logs WHERE type NOT IN ("biking", "hiking", "tree climbing", "rowing");**

## INSERT INTO \_\_ VALUES /

**/\*\*añadir valores para todos los campos\*\*/**

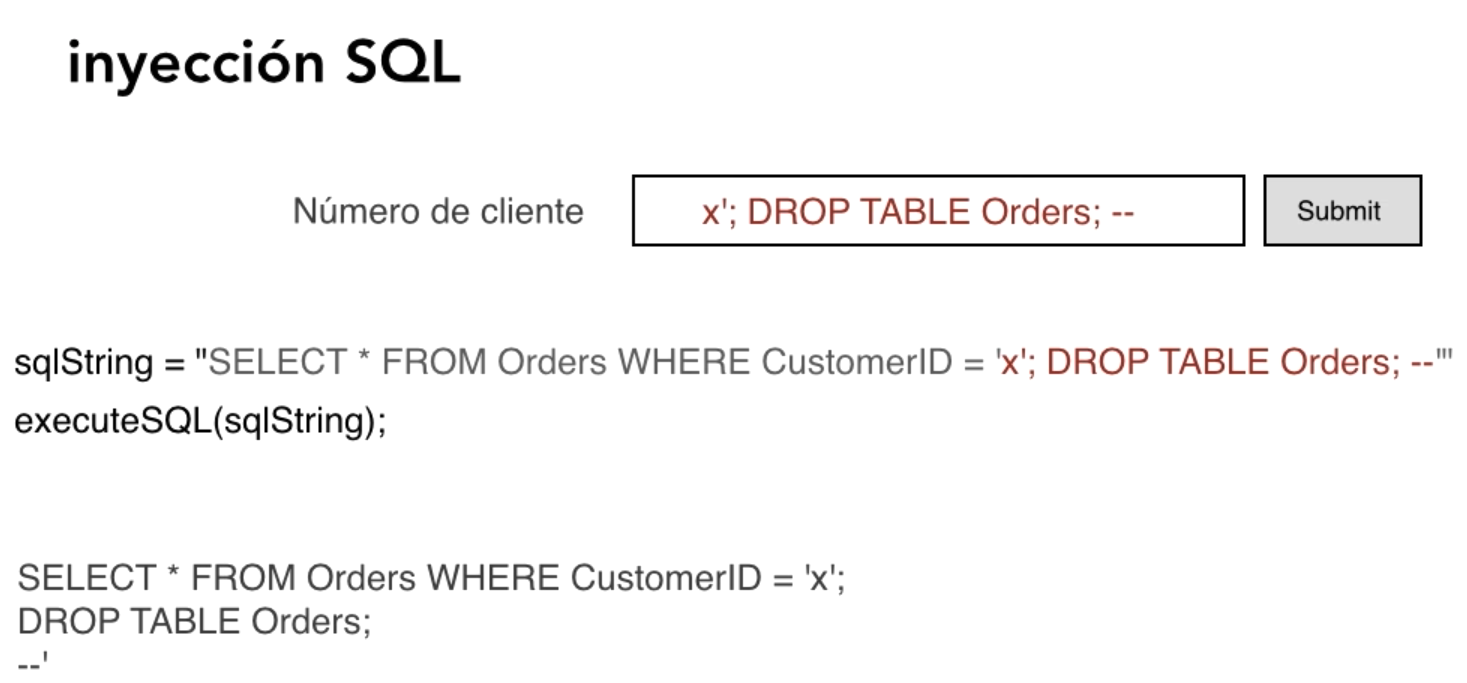
**INSERT INTO groceries VALUES (1, "Bananas", 4);**

**/\*\* añadir solo en campos concretos , poniendo id autogenerada\*\*/**

**INSERT INTO exercise\_logs(type, minutes, calories, heart\_rate) VALUES ("biking", 30, 100, 110);**

## INYECCIÓN SQL

*\*\* Hay mucho riesgo de que se inyecte código malicioso que borra la base de datos, como: \*\*/*



**/\*\* Por protección es mejor utilizar códigos alamcenados \*\*\*/**

## JOIN

*\*\* Para unir tablas relacionadas \*\**

CREATE TABLE students (id INTEGER PRIMARY KEY,

first\_name TEXT, last\_name TEXT, email TEXT, phone TEXT, birthdate TEXT);

INSERT INTO students (first\_name, last\_name, email, phone, birthdate)

VALUES ("Peter", "Rabbit", "peter@rabbit.com", "555-6666", "2002-06-24");

INSERT INTO students (first\_name, last\_name, email, phone, birthdate)

VALUES ("Alice", "Wonderland", "alice@wonderland.com", "555-4444", "2002-07-04");

CREATE TABLE student\_grades (id INTEGER PRIMARY KEY,

student\_id INTEGER, test TEXT, grade INTEGER);

INSERT INTO student\_grades (student\_id, test, grade)

VALUES (1, "Nutrition", 95);

INSERT INTO student\_grades (student\_id, test, grade)

VA LUES (2, "Nutrition", 92);

INSERT INTO student\_grades (student\_id, test, grade)

VALUES (1, "Chemistry", 85);

INSERT INTO student\_grades (student\_id, test, grade)

VALUES (2, "Chemistry", 95);

SELECT \* FROM student\_grades;

/\* cross join \*/

SELECT \* FROM student\_grades, students;

/\* implicit **inner** join \*/

SELECT \* FROM student\_grades, students

WHERE student\_grades.student\_id = students.id;

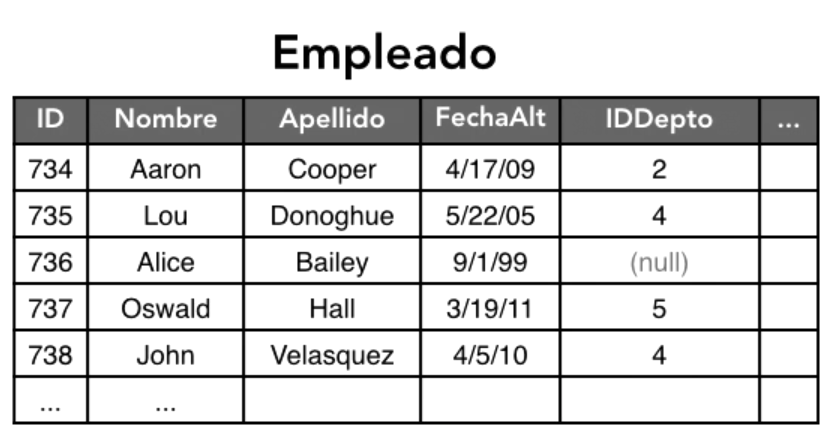
/\* explicit inner join - JOIN \*/

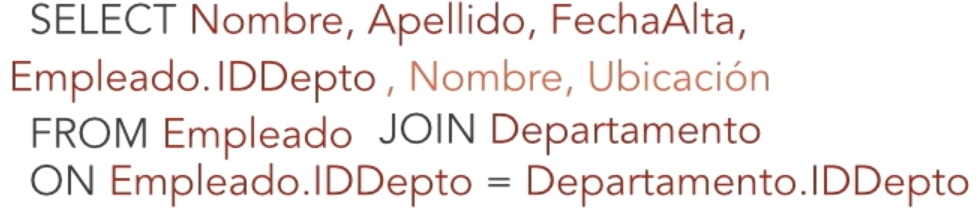
**SELECT students.first\_name, students.last\_name, students.email, student\_grades.test, student\_grades.grade FROM students**

**JOIN student\_grades**

**ON students.id = student\_grades.student\_id**

**WHERE grade > 90;**

## Join;LEFT / RIGHT/ FULL OUTER JOIN

*\*\* Muestra valores vacíos de la tabla de la izquierda cuando hacemos consulta. Existe también RIGHT OUTER JOIN però es innecesario si ponemos la primera tabla con las filas que queremos mostrar. También FULL OUTER JOIN que mira las dos tablas \*\**

/\* outer join \*/

SELECT students.first\_name, students.last\_name, student\_projects.title

FROM students

LEFT OUTER JOIN student\_projects

ON students.id = student\_projects.student\_id;



Ahora muestra fila 3 (Alice valores Null) porque manda la tabla de izquierda.



**Ahora manda la tabla derecha, la qual tiene el departamento de Producción sin correspondencia en la tabla de Empleado:**

## Join:SELF JOIN

CREATE TABLE students (id INTEGER PRIMARY KEY AUTOINCREMENT,

first\_name TEXT, last\_name TEXT, email TEXT, phone TEXT, birthdate TEXT, buddy\_id INTEGER);

INSERT INTO students

VALUES (1, "Peter", "Rabbit", "peter@rabbit.com", "555-6666", "2002-06-24", 2);

/\*\* ...INTO varios valores... self join : relacionameos una tabla consigo misma. La tabla «students» se relacionará con la propia tabla students con un alias «students buddies». También reombramos la columna correu electrònic para clarificar que es el correu electrònic del amigo \*/

**SELECT students.first\_name, students.last\_name, buddies.email as buddy\_email**

**FROM students**

**JOIN students buddies**

**ON students.buddy\_id = buddies.id;**

MÚLTIPLES JoiN

[**https://es.khanacademy.org/computing/computer-programming/sql/relational-queries-in-sql/pt/combining-multiple-joins**](https://es.khanacademy.org/computing/computer-programming/sql/relational-queries-in-sql/pt/combining-multiple-joins)

**CREATE TABLE students** (id INTEGER PRIMARY KEY, first\_name TEXT, last\_name TEXT, email TEXT,

phone TEXT, birthdate TEXT);

INSERT INTO students (first\_name, last\_name, email, phone, birthdate)

VALUES ("Peter", "Rabbit", "peter@rabbit.com", "555-6666", "2002-06-24");

INSERT INTO students (first\_name, last\_name, email, phone, birthdate)

VALUES ("Alice", "Wonderland", "alice@wonderland.com", "555-4444", "2002-07-04");

INSERT INTO students (first\_name, last\_name, email, phone, birthdate)

VALUES ("Aladdin", "Lampland", "aladdin@lampland.com", "555-3333", "2001-05-10");

INSERT INTO students (first\_name, last\_name, email, phone, birthdate)

VALUES ("Simba", "Kingston", "simba@kingston.com", "555-1111", "2001-12-24");

**CREATE TABLE student\_projects** (id INTEGER PRIMARY KEY, student\_id INTEGER, title TEXT);

INSERT INTO student\_projects (student\_id, title) VALUES (1, "Carrotapault");

INSERT INTO student\_projects (student\_id, title) VALUES (2, "Mad Hattery");

INSERT INTO student\_projects (student\_id, title) VALUES (3, "Carpet Physics");

INSERT INTO student\_projects (student\_id, title) VALUES (4, "Hyena Habitats");

**CREATE TABLE project\_pairs** (id INTEGER PRIMARY KEY, project1\_id INTEGER, project2\_id INTEGER);

INSERT INTO project\_pairs (project1\_id, project2\_id) VALUES(1, 2);

INSERT INTO project\_pairs (project1\_id, project2\_id) VALUES(3, 4);

**SELECT a.title, b.title FROM project\_pairs**

**JOIN student\_projects a**

**ON project\_pairs.project1\_id = a.id**

**JOIN student\_projects b**

**ON project\_pairs.project2\_id = b.id;**

## LIKE

**/\*\* LIKE buscará un valor (entre % %) dentro de una frase \*\*/**

**SELECT \* FROM exercise\_logs WHERE type IN (**

**SELECT type FROM drs\_favorites WHERE reason LIKE "%cardiovascular%");**

**LIKE**

**/\*\* compara de modo flexible buscando solo elementos deseados dentro de una frase \*\*/**

**SELECT \* FROM exercise\_logs WHERE type IN (**

**SELECT type FROM drs\_favorites WHERE reason = "Increases cardiovascular health");**

**/\* LIKE \*/**

**SELECT \* FROM exercise\_logs WHERE type IN (**

**SELECT type FROM drs\_favorites WHERE reason LIKE "%cardiovascular%");**

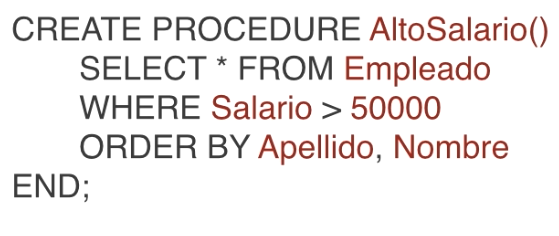
**/\*\* parámetros \*\*/**

**Green% /\*\* empiezan por Green \*\*/**

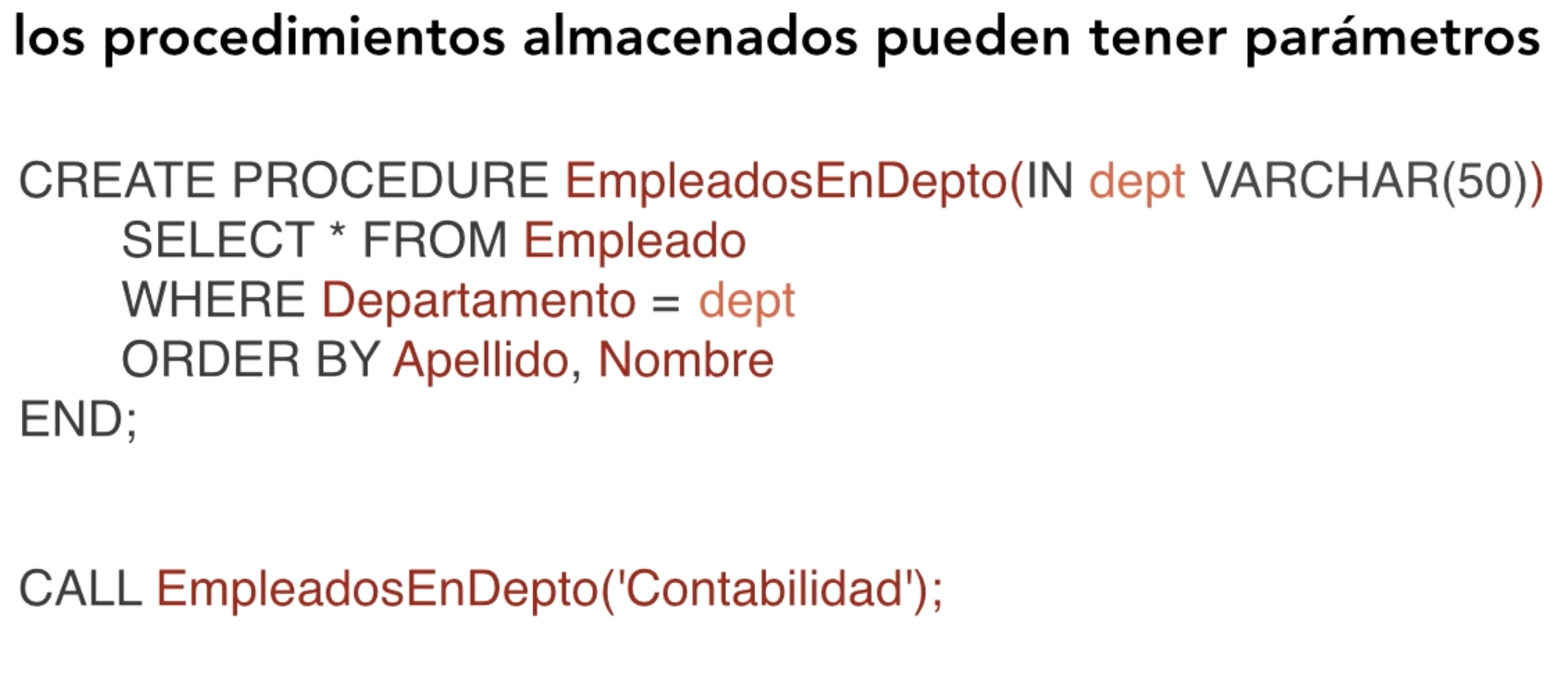
**%Green /\*\* acaban en Green \*\*/**

**‘Sm\_th’ /\*\* cualquier letra en \_ \*\*/**

## Procesos







## ORDER BY (asc / des )

**/\*\* ordenar 1 campo:lista por pasillo (asc) \*\*/**

**SELECT \* FROM groceries ORDER BY aisle;**

**/\*\* ordenar un campo (asc)y luego su interior (asc)\*\*/**

**SELECT \* FROM productos**

**ORDER BY sección, precio**

**/\*\* ordenar 1º por sección (ascendente) y luego por precio descendente\*\*/**

**SELECT \* FROM productos**

**ORDER BY sección, precio DESC**

/\*\*podemos poner más criterios de ordenación \*\*/

SELECT \* FROM personas ORDER BY apellido,nombre;

## SELECT \_\_ FROM

**/\*\* mostrar toda la tabla : \*\*/**

**SELECT \* FROM groceries;**

**/\*\* mostrar solo los campos seleccionados \*\*/**

**SELECT name,quantity FROM groceries**

## SUBCONSULTAS

**/\*\* si tenemos dos tablas y queremos hacer consulta entre elals \*\*/**

**CREATE TABLE exercise\_logs**

**(id INTEGER PRIMARY KEY AUTOINCREMENT,**

**type TEXT,**

**minutes INTEGER,**

**calories INTEGER,**

**heart\_rate INTEGER);**

**INSERT INTO exercise\_logs(type, minutes, calories, heart\_rate) VALUES ("biking", 30, 100, 110);**

**INSERT INTO exercise\_logs(type, minutes, calories, heart\_rate) VALUES ("dancing", 15, 200, 120);**

**INSERT INTO exercise\_logs(type, minutes, calories, heart\_rate) VALUES ("tree climbing", 30, 70, 90);**

**INSERT INTO exercise\_logs(type, minutes, calories, heart\_rate) VALUES ("rowing", 30, 70, 90);**

**INSERT INTO exercise\_logs(type, minutes, calories, heart\_rate) VALUES ("hiking", 60, 80, 85);**

**/\* IN \*/**

**SELECT \* FROM exercise\_logs WHERE type IN ("biking", "hiking", "tree climbing", "rowing");**

**/\*\* nueva tabla \*\*/**

**CREATE TABLE drs\_favorites**

**(id INTEGER PRIMARY KEY,**

**type TEXT,**

**reason TEXT);**

**INSERT INTO drs\_favorites(type, reason) VALUES ("biking",**

**"Improves endurance and flexibility.");**

**INSERT INTO drs\_favorites(type, reason) VALUES ("hiking",**

**"Increases cardiovascular health.");**

**/\*\* comprovamos que campos type tenemos \*\*/**

**SELECT type FROM drs\_favorites;**

**/\*\* miramos en la primera tabla esos campos que coinciden \*\*/**

**SELECT \* FROM exercise\_logs WHERE type IN ("biking", "hiking");**

**/\*\* con este método no se actualizan los campos si se modifican**

**en la tabla, por eso hacemos una consulta anidada simplemente pegando la subconsulta dentro de l consulta \*\*/**

**SELECT \* FROM exercise\_logs WHERE type IN (SELECT type**

**FROM drs\_favorites);**

## UPDATE

**/\*\* Si no ponemos una condición con WHERE se actualizarían todos los valores de la tabla \*\*/**



## WHERE

**/\*\* Filtrar pasillo >5 \*\*/**

**SELECT \* FROM groceries WHERE aisle > 5 ORDER BY aisle;**

SUM , MAX, MIN,AVG(**quantity)**

**/\*\* suma (o otras operaciones) elementos de una columna \*\*/**

**SELECT SUM(quantity) FROM groceries;**

**/\*\* cantidad máxima de un elemento (quantity en el eejmplo) \*\*/**

**SELECT MAX(quantity) FROM groceries;**

## Separar datos en tablas relacionadas

Hasta ahora, solo hemos trabajado con una tabla a la vez, y visto qué datos interesantes podemos seleccionar de esa tabla. Pero en realidad, la mayor parte del tiempo, tenemos nuestros datos distribuidos en varias tablas, y todas esas tablas están "relacionadas" unas a otras de alguna manera.

Por ejemplo, digamos que tenemos una tabla para registrar qué tan bien les va a los estudiantes en sus exámenes, e incluimos direcciones de correo electrónico en caso de que necesitemos enviar mensajes a los papás acerca de resbalones en las calificaciones:

| **nombre\_estudiante** | **correo\_estudiante** | **examen** | **calificacion** |
| --- | --- | --- | --- |
| Peter Rabbit | peter@rabbit.com | Nutrición | 95 |
| Alice Wonderland | alice@wonderland.com | Nutrición | 92 |
| Peter Rabbit | peter@rabbit.com | Química | 85 |
| Alice Wonderland | alice@wonderland.com | Química | 95 |

También podríamos tener una tabla para registrar qué libros lee cada estudiante:

| **nombre\_estudiante** | **titulo\_libro** | **autor\_libro** |
| --- | --- | --- |
| Peter Rabbit | El cuento de la señora Tiggy-Winkle | Beatrix Potter |
| Peter Rabbit | Jabberwocky | Lewis Carroll |
| Alice Wonderland | La Caza del Snark | Lewis Carroll |
| Alice Wonderland | Jabberwocky | Lewis Carroll |

También podríamos tener una tabla solo para información detallada del estudiante:

| **id** | **nombre\_estudiante** | **apellido\_estudiante** | **correo\_estudiante** | **telefono** | **cumpleaños** |
| --- | --- | --- | --- | --- | --- |
| 1 | Peter | Conejo | peter@rabbit.com | 555-6666 | 2001-05-10 |
| 2 | Alice | Wonderland | alice@wonderland.com | 555-4444 | 2001-04-02 |

¿Que piensas acerca de estas tablas? ¿Las cambiarías de alguna manera?

Hay una cosa importante que hay que darse cuenta acerca de estas tablas: describen datos relacionales, como en: describen datos que se relacionan unos a otros. Cada una de estas tablas describe datos relacionados a un estudiante en particular, y muchas de las tablas replican los mismos datos. Cuando los mismos datos están replicados en múltples tablas, puede haber consecuencias interesantes.

Por ejemplo, ¿qué pasa si cambia el correo electrónico de un estudiante? ¿Qué tablas serían necesarias cambiar?

Necesitaríamos cambiar la tabla de información del estudiante, pero como también incluimos esos datos en la tabla de calificaciones, también tendríamos que encontrar cada renglón acerca de ese estudiante, y cambiar el correo electrónico ahí también.

A menudo es preferible estar seguros de que una columna de datos en particular esté almacenada en una sola ubicación, de modo que haya menos lugares que actualizar y menos riesgo de tener diferentes datos en diferentes lugares. Si hacemos eso, necesitamos asegurarnos de tener una manera de relacionar los datos en distintas tablas, a lo cual llegaremos más adelante.

Digamos que decidimos quitar el correo electrónico de la tabla de calificaciones, porque nos dimos cuenta de que es redundante con el correo electrónico en la tabla de detalles del estudiante. Esto es lo que tendríamos:

| **nombre\_estudiante** | **examen** | **calificacion** |
| --- | --- | --- |
| Peter Rabbit | Nutrición | 95 |
| Alice Wonderland | Nutrición | 92 |
| Peter Rabbit | Química | 85 |
| Alice Wonderland | Química | 95 |

¿Cómo podríamos averiguar el correo electrónico para cada estudiante? Podríamos encontrar el renglón en la tabla de información de estudiantes, al hacer coincidir los nombres. ¿Qué pasa si 2 estudiantes tienen el mismo nombre? (¿Sabías que en Bali cada persona solo tiene 1 de 4 nombres posibles?) No podemos depender del nombre para buscar un estudiante, y en serio, nunca debemos depender en algo como el nombre para identificar algo de manera única en una tabla.

Así que lo mejor por hacer es quitar nombre\_estudiante y reemplazarlo con id\_estudiante, ya que ese es un identificador único garantizado:

| **id\_estudiante** | **examen** | **calificacion** |
| --- | --- | --- |
| 1 | Nutrición | 95 |
| 2 | Nutrición | 92 |
| 1 | Química | 85 |
| 2 | Química | 95 |

Podríamos hacer el mismo cambio en nuestra tabla de libros, al usar id\_estudiante en vez de nombre\_estudiante:

| **id\_estudiante** | **titulo\_libro** | **autor\_libro** |
| --- | --- | --- |
| 1 | El cuento de la señora Tiggy-Winkle | Beatrix Potter |
| 1 | Jabberwocky | Lewis Carroll |
| 2 | La Caza del Snark | Lewis Carroll |
| 2 | Jabberwocky | Lewis Carroll |

Te das cuenta de que tenemos el título del libro y autor repetidos dos veces para Jabberwocky? Ese es otro signo de alerta de que podríamos separar nuestra tabla en múltiples tablas relacionadas, de modo que no tengamos que actualizar múltiples lugares si algo cambia acerca de un libro.

Podríamos tener una tabla solo acerca de libros:

| **id** | **titulo\_libro** | **autor\_libro** |
| --- | --- | --- |
| 1 | El cuento de la señora Tiggy-Winkle | Beatrix Potter |
| 2 | Jabberwocky | Lewis Carroll |
| 3 | La Caza del Snark | Lewis Carroll |

Y después nuestra tabla libros\_estudiantes se convierte en:

| **id\_estudiante** | **id\_libro** |
| --- | --- |
| 1 | 1 |
| 1 | 2 |
| 2 | 3 |
| 2 | 2 |

Ya sé, esta tabla no se ve tan legible como la anterior que tenía toda la información metida en cada renglón. Pero las tablas suelen no estar diseñadas para que las lea un humano, sino para que sean lo más fáciles de mantener y menos propensas a errores. En muchos casos, puede ser mejor separar la información en múltiples tablas relacionadas, de modo que haya menos datos redundantes y menos lugares que actualizar.

Es importante entender cómo usar SQL para lidiar con datos que han sido separados en múltiples tablas relacionadas, y traer de regreso los datos de varias tablas cuando sea necesario. Hacemos eso al usar un concepto llamado "join"s (uniones) y eso es lo que te mostraré a continuación.