FUNCIONES MULTICAPA

```
SELECT AVG(salary), MAX(salary), MIN(salary), SUM(salary)  FROM employees e ;
```

```
SELECT MIN(hire_date), MAX(hire_date) FROM employees e;
SELECT COUNT(*) FROM employees e WHERE department_id = 10;
SELECT COUNT(salary)  FROM employees e WHERE department_id = 9;
```

GROUP BY

```
SELECT department_id, ROUND(AVG(salary),2) FROM employees e GROUP BY department_id ;
SELECT AVG (salary) FROM employees e GROUP BY department_id ;
SELECT department_id, job_id, SUM(salary) FROM employees e2 GROUP BY department_id, job_id ORDER
BY department_id ;
SELECT department_id, COUNT(last_name) FROM employees e;
SELECT department_id, job_id, COUNT(last_name) FROM employees e GROUP BY department_id ;
SELECT department_id, AVG(salary) FROM employees;
```

GROUP BY HAVING

```
SELECT department_id, MAX(salary) FROM employees e GROUP BY department_id  HAVING
max(salary>10000);
SELECT job_id, SUM(salary) "NÓMINA DE SUELDOS" FROM employees e WHERE job_id NOT LIKE 9 GROUP BY
job_id HAVING SUM(salary) >13000  ORDER BY SUM(salary);
```

EJERCICIOS PÁGINA 9

Uniendo tablas SQL

INNER JOIN devuelve solo las filas que coinciden con la condición de combinación y elimina todas las demás filas que no coinciden con la condición de combinación.

Un **NATURAL JOIN** es similar a un JOIN ... USING, la diferencia es que automáticamente prueba la igualdad entre los valores de cada columna que existe en ambas tablas.

La diferencia entre INNER JOIN y NATURAL JOIN:

En INNER JOIN, debe especificar una condición de combinación que la combinación interna utiliza para unir las dos tablas. Mientras que en la combinación natural, no escribe una condición de combinación. Simplemente escriba los nombres de las dos tablas sin ninguna condición. Luego, la combinación natural probará automáticamente la igualdad entre los valores para cada columna que exista en ambas tablas. La unión natural infiere la condición de unión automáticamente.

En NATURAL JOIN, todas las columnas de ambas tablas con el mismo nombre se compararán entre sí. Por ejemplo, si tenemos dos tablas con dos nombres de columna en común (las dos columnas existen con el mismo nombre en las dos tablas), entonces la combinación natural unirá las dos tablas comparando los valores de ambas columnas y no solo de una. columna.

SQLite JOIN ... USING

El INNER JOIN se puede escribir usando la cláusula "USING" para evitar la redundancia, por lo que en lugar de escribir "ON Students.DepartmentId = Departments.DepartmentId", puede simplemente escribir "USING (DepartmentID)".

Puede usar "JOIN .. USING" siempre que las columnas que comparará en la condición de combinación tengan el mismo nombre. En tales casos, no es necesario repetirlos usando la condición on y simplemente indique los nombres de las columnas y SQLite lo detectará.

La diferencia entre INNER JOIN y JOIN .. USANDO:

Con "JOIN ... USING" no escribe una condición de unión, solo escribe la columna de unión que es común entre las dos tablas unidas, en lugar de escribir table1 "INNER JOIN table2 ON table1.cola = table2.cola" escribimos es como "table1 JOIN table2 USING (cola)".

NATURAL JOIN – BASADA EN RELACIONES NATURALES

```
SELECT d.department_id, d.department_name, d.location_id, l.city  FROM departments d NATURAL JOIN
locations l ;
```

JOIN – USING – BASADA EN RELACIONES NATURALES CON UN ÚNICO ARGUMENTO

```
SELECT e.first_name, d.department_name  FROM employees e JOIN departments d USING(department_id)
```

```
SELECT e.first_name, d.department_name  FROM employees e INNER JOIN departments d
USING(department_id)
```

INNER JOIN / JOIN – ON – SIMILAR A JOIN – USING, PERO CON ARGUMENTOS REDUNDANTES

```
SELECT e.employee_id, e.last_name, e.department_id, d.department_id, d.location_id  FROM
employees e JOIN departments d ON (e.department_id = d.department_id);
```

```
SELECT e.employee_id, l.city FROM employees e JOIN departments d ON d.department_id =
e.department_id JOIN locations l ON d.location_id = l.location_id
```

SELF JOIN – join a entre una misma tabla

```
SELECT e.last_name, e2.last_name FROM employees e  join employees e2 ON (e.manager_id =
e2.manager_id);
```

LEFT OUTER JOIN – DEVUELVE SOLO FILAS TABLA IZQUIERDA QUE COINCIDAN CON LA TABLA DE LA DERECHA

El estándar SQL define tres tipos de UNIONES EXTERIORES: IZQUIERDA, DERECHA y COMPLETA, pero SQLite solo admite la UNIÓN EXTERIOR IZQUIERDA natural.

```
SELECT e.last_name, e.department_id, d.department_name  FROM employees e LEFT OUTER JOIN
departments d ON(e.department_id = d.department_id);
```

CROSS JOIN – PRODUCTO CARTESIANO T.TABLA.A*T.TABLA.B

A CROSS JOIN da el producto cartesiano para las columnas seleccionadas de las dos tablas unidas, haciendo coincidir todos los valores de la primera tabla con todos los valores de la segunda tabla.

En otras palabras, produce un conjunto de resultados que es el número de filas en la primera tabla multiplicado por el número de filas en la segunda tabla si no se usa la cláusula WHERE junto con CROSS JOIN.

```
SELECT e.last_name, d.department_name  FROM employees e  CROSS JOIN departments d ;
```

EJERCICIOS PÁGINA 25

SUBCONSULTAS

```
SELECT first_name, last_name, salary FROM employees WHERE salary  > (SELECT salary FROM employees
WHERE last_name = 'Hunold');
```

SUBCONSULTAS DE FILA ÚNICA

```
SELECT last_name, job_id, salary FROM employees WHERE job_id = (SELECT job_id FROM employees
WHERE first_name = 'Luis') AND salary > (SELECT salary FROM employees WHERE first_name =
'Karen');
```

```
SELECT last_name, job_id, salary FROM  employees WHERE salary = (SELECT MIN(salary) FROM
employees);
```

```
SELECT  department_id, MIN(salary) FROM employees GROUP BY department_id HAVING MIN(salary) >
(SELECT MIN(salary) FROM employees WHERE department_id = 10);
```

SUBCONSULTAS MULTI-FILA

```
SELECT employee_id, first_name, last_name  FROM employees e  WHERE department_id IN (1, 3, 8, 10,
11) ORDER BY first_name, last_name ;
```

```
SELECT e.employee_id, e.first_name, e.last_name FROM employees e  WHERE department_id  IN (SELECT
d.department_id FROM departments d WHERE d.location_id = 1700) ORDER BY first_name, last_name ;
```

```
SELECT e.employee_id, e.first_name, e.last_name FROM employees e  WHERE department_id NOT IN
(SELECT d.department_id FROM departments d WHERE d.location_id = 1700) ORDER BY first_name,
last_name ;

SELECT employee_id, first_name, last_name, salary  FROM employees WHERE salary >= (SELECT
MIN(salary) FROM employees GROUP BY department_id) ORDER BY first_name, last_name ;

SELECT employee_id, first_name, last_name, salary FROM employees WHERE salary IN (SELECT
MAX(salary) FROM employees GROUP BY department_id) ORDER BY first_name, last_name ;
```

SUBCONSULTAS COMO TABLA

```
SELECT ROUND(AVG(average_salary),0) FROM (SELECT AVG(salary) average_salary FROM employees GROUP
BY department_id) department_salary;
```

SUBCONSULTAS COMO ATRIBUTO

```
SELECT employee_id, first_name, last_name, salary, (SELECT ROUND(AVG(salary),0) FROM employees)
average_salary, salary - (SELECT ROUND(AVG(salary),0) FROM employees) difference FROM employees
ORDER BY first_name, last_name ;
```

SUBCONSULTA CON COMPARACIÓN MULTICOLUMNA

```
SELECT employee_id, manager_id, department_id FROM employees WHERE (manager_id,department_id) IN
(SELECT manager_id, department_id FROM employees WHERE first_name = 'John') AND first_name <>
'john';
```

SUBCONSULTAS CORRELACIONADAS

```
SELECT employee_id, first_name, last_name  FROM employees WHERE department_id  IN (SELECT
department_id FROM departments d WHERE  d.location_id = 1700) ORDER BY first_name, last_name ;

SELECT department_name FROM departments d WHERE NOT EXISTS (SELECT 1 FROM employees e WHERE
e.salary > 10000 AND e.department_id = d.department_id) ORDER BY department_name ;
```

EJERCICIOS PÁGINA 43

OPERACIONES DE CONJUNTOS

UNION – MISMOS VALORES/CANTIDAD/ORDEN

```
SELECT MIN(e.salary) min_salary, e.job_id  FROM employees e UNION SELECT j.min_salary, j.job_id
FROM jobs j ORDER BY min_salary;
```

UNION ALL – VALORES DUPLICADOS

```
SELECT MIN(e.salary) min_salary, MAX(e.salary) max_salary, e.job_id  FROM employees e UNION ALL
SELECT j.min_salary, j.max_salary, j.job_id  FROM jobs j ORDER BY min_salary;
```

INTERSECT – DEVUELVE FILAS QUE COINCIDAN EN AMBOS SELECT

```
SELECT MIN(e.salary) min_salary, e.job_id  FROM employees e INTERSECT SELECT j.min_salary,
j.job_id  FROM jobs j;
```

EJERCICIOS PÁGINA 52

MANIPULANDO DATOS DML

INSERTANDO NUEVAS FILAS

CREAR PRIMERO LA TABLA:

```
CREATE TABLE products (   ID INT PRIMARY KEY,product_name VARCHAR (255) NOT NULL, product_summary
VARCHAR(255), product_description VARCHAR(4000) NOT NULL, price NUMERIC (11, 2) NOT NULL,
discount NUMERIC (11, 2));
```

INSERTAR REGISTROS EN LA TABLA

```
INSERT INTO products (
	ID,
	product_name,
	product_summary,
	product_description,
	price,
	discount
)
VALUES
	(
		1,
		'McLaren 675LT',
		'Inspired by the McLaren F1 GTR Longtail',
		'Performance is like strikin and the seven-speed dual-clutch gearbox is twice as fast
now.',
		349500,
		1000
	),
	(
		2,
		'Rolls-Royce Wraith Coupe',
		NULL,
		'Inspired by the words of Sir Henry Royce, this Rolls-Royce Wraith Coupe is an
imperceptible force',
		304000,
		NULL
	),
	(
		3,
		'2016 Lamborghini Aventador Convertible',
		NULL,
		'Based on V12, this superveloce has been developed as the Lamborghini with the
sportiest DNA',
		271000,
		500
	);
```

UPDATE

```
UPDATE employees SET department_id = 50 WHERE employee_id  = 113;

SELECT employee_id, department_id FROM employees e ;
```

ACTUALIZANDO FILAS

```
UPDATE employees  SET job_id = (SELECT job_id FROM employees WHERE employee_id = 205), salary =
(SELECT salary FROM employees WHERE employee_id = 205) WHERE employee_id  = 113;


SELECT employee_id, job_id FROM employees e
```

ELIMINANDO FILAS

Explicar desde las láminas, para no borrar registros.

EJERCICIOS PÁGINA 63

TRANSACCIONES TCL

Explicar desde las laminas