SQL

KHAN-apuntes

Contenido

Cláusulas

Cláusula	Descripción
FROM	Especifica la tabla de la que se quieren obtener los registros
WHERE	Especifica las condiciones o criterios de los registros seleccionados
GROUP BY	Para agrupar los registros seleccionados en función de un campo
HAVING	Especifica las condiciones o criterios que deben cumplir los grupos
ORDER BY	Ordena los registros seleccionados en función de un campo

Operadores de comparación

Operador	Significado
<	Menor que
>	Mayor que
=	Igual que
>=	Mayor o igual que
<=	Menor o igual que
<> +	Distinto que
BETWEEN	Entre. Utilizado para especificar rang de valores
LIKE	Cómo. Utilizado con caracteres comodín (? *)
In	En. Para especificar registros en un campo en concreto

S VÍDEOS

SQL. Clausulas y operadores. Video 3

Operadores lógicos

Operador	Significado
AND	Y lógico
OR	O lógico
NOT	Negación lógica

AND / OR

```
/** 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;
```

BETWEEN

/ 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;
```

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);
```

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);

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%");
```

ORDER BY

```
/** ordenar  lista por pasillo:  ORDER BY **/

SELECT * FROM groceries ORDER BY aisle;
```

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);
```

WHERE

/ Filtrar pasillo >5 **/**

SELECT * FROM groceries WHERE aisle > 5

ORDER BY aisle;

SUM , MAX(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.