Tema 4

Queries en el FROM

Se trata de un tipo de consulta que se realiza sobre otra consulta. En una consulta normal, escribiríamos lo siguiente:

SELECT * FROM nombre_tabla;

En este tipo de consultas, lo que hacemos es reemplazar la tabla por otra consulta (también llamada “subconsulta”), de forma que el motor considere el resultado de esa consulta como una tabla:

SELECT campo1, campo2
FROM (SELECT * FROM tabla);

Por ejemplo, si queremos conocer la cantidad de estudiantes nacidos en la década del 90 que realizan más de un curso, podemos hacer una *query* anidada.

Empezamos por conocer los estudiantes nacidos en la década del 90:

select *
from estudiante
where fecha_nacimiento like "199%";

Resultado:

Tabla 41. Resultado

	legajo	nombre	apellido	fecha_nacimiento	carrera
▶	36485	Romina	Nieva	1999-11-26	Mecánica
	41258	Ramiro	Ríos	1994-12-06	Sistemas
	43651	Cristian	Gómez	1995-03-19	Mecánica
	47521	María	Velazquez	1998-01-02	Sistemas
	47961	Alexis	Reinoso	1994-12-17	Sistemas
	48952	Gabriel	Morales	1996-10-03	Sistemas

Fuente: elaboración propia.

Luego, obtenemos los estudiantes que realizan cursos:

```
select *
from estudiante e inner join inscripcion i on e.legajo =
i.ESTUDIANTE_legajo
where fecha_nacimiento like "199%"
;
```

Resultado:

Tabla 42. Resultado

legajo	nombre	apellido	fecha_nacimiento	carrera	numero	CURSO_codigo	ESTUDIANTE_legajo	fecha_hora
36485	Romina	Nieva	1999-11-26	Mecánica	6	103	36485	2012-04-28 18:45:00
41258	Ramiro	Ríos	1994-12-06	Sistemas	1	101	41258	2012-05-02 18:45:00
41258	Ramiro	Ríos	1994-12-06	Sistemas	2	102	41258	2012-04-02 18:45:00
43651	Cristian	Gómez	1995-03-19	Mecánica	7	103	43651	2012-04-28 18:45:00
43651	Cristian	Gómez	1995-03-19	Mecánica	11	101	43651	2012-04-21 18:45:00
47521	María	Velazquez	1998-01-02	Sistemas	13	102	47521	2012-04-03 18:45:00
47961	Alexis	Reinoso	1994-12-17	Sistemas	3	102	47961	2012-01-02 20:01:00
47961	Alexis	Reinoso	1994-12-17	Sistemas	4	103	47961	2012-04-28 18:45:00
47961	Alexis	Reinoso	1994-12-17	Sistemas	8	101	47961	2012-04-28 18:45:00

Fuente: elaboración propia.

Después, obtenemos los estudiantes nacidos en la década del 90 que realizan más de un curso:

```
select e.legajo, count(*)
from estudiante e inner join inscripcion i on e.legajo =
i.ESTUDIANTE_legajo
where fecha_nacimiento like "199%"
group by e.legajo having count(*) > 1
;
```

Resultado:

Tabla 43. Resultado

legajo	count(*)
41258	2
43651	2
47961	3

Fuente: elaboración propia.

Y, por último, tomando el resultado de esa consulta como la tabla “ej_query_anidada”, contamos la cantidad de registros que nos

devuelve. Para realizar eso, debemos poner la consulta anterior en el FROM y hacer el conteo:

```
SELECT count(*)
FROM (select e.legajo, count(*)
from estudiante e inner join inscripcion i on e.legajo =
i.ESTUDIANTE_legajo
where fecha_nacimiento like "199%"
group by e.legajo having count(*) > 1) as ej_query_anidada;
```

Resultado:

Tabla 44. Resultado

	count(*)
▶	3

Fuente: elaboración propia.

Podemos querer mostrar la información del profesor que dicta el curso con más alumnos. En ese caso, primero hacemos una consulta para averiguar la cantidad de estudiantes en cada curso:

```
select CURSO_codigo, count(*) cant_estud
from alkemy.inscripcion i
group by CURSO_codigo;
```

Resultado:

Tabla 45. Resultado

	CURSO_codigo	cant_estud
▶	101	4
	102	5
	103	3

Fuente: elaboración propia.

Luego, utilizamos esa consulta en el FROM de otra consulta, de la siguiente manera:

```
select p.*, c.codigo, max(curso_max.cant_estud) as "cantidad de
estudiantes"
from
```

```
(select CURSO_codigo, count(*) cant_estud
From inscripcion i
group by CURSO_codigo) curso_max
inner join curso c on curso_max.CURSO_codigo = c.codigo
inner join profesor p on c.PROFESOR_id = p.id;
```

De esta forma, “joineamos” el curso con más estudiantes (ya que nos traemos el máximo **max(curso_max.cant_estud)**); y, “joineando” con la tabla de profesores, obtenemos los datos del profesor, el código del curso con más estudiantes y la cantidad de estudiantes:

Tabla 46. Estudiantes

	id	nombre	apellido	fecha_nacimiento	salario	codigo	cantidad de estudiantes
▶	1	Juan	Pérez	1990-06-06	55000	101	5

Fuente: elaboración propia.

Queries en el WHERE

Otra manera de anidar las consultas es colocarlas en la sección del WHERE.

```
SELECT *
FROM tabla1
WHERE campo operador (SELECT campo FROM tabla2 WHERE
condiciones);
```

Para estos casos, utilizamos el valor (o el conjunto de valores) que devuelve la consulta interna (o subconsulta) para compararlo con el valor de un campo en la consulta externa.

Veamos un ejemplo:

Para obtener toda la información de los estudiantes que realizan los cursos cuyos códigos son 101, 104 o 105, podemos ejecutar lo siguiente:

```
SELECT *
FROM estudiante e
WHERE e.legajo IN (SELECT i.ESTUDIANTE_legajo
                   FROM inscripcion i
                   WHERE i.CURSO_codigo IN (101,104,105)
);
```

El código resaltado en negrita representa la consulta interna que será la que se anidará.

Esta consulta interna da como resultado lo siguiente:

Tabla 47. Estudiante_legajo

	ESTUDIANTE_legajo
▶	41258
	39685
	47961
	43651

Fuente: elaboración propia.

Y el resultado final de la consulta completa es este:

Tabla 48. Resultado final

	legajo	nombre	apellido	fecha_nacimiento	carrera
▶	39685	Brenda	Medrano	2000-09-25	Sistemas
	41258	Ramiro	Ríos	1994-12-06	Sistemas
	43651	Cristian	Gómez	1995-03-19	Mecánica
	47961	Alexis	Reinoso	1994-12-17	Sistemas

Fuente: elaboración propia.

Este tipo de consultas no solo se realiza utilizando el operador IN: también pueden emplearse otros operadores como “=”, “>”, “<”, entre otros. Sin embargo, para esos casos, debemos asegurarnos de que la subconsulta devuelva solamente un registro; de lo contrario, el motor no sabrá qué registro tomar para realizar la operación correspondiente.

Otro operador posible es el EXISTS, el cual nos permite verificar la existencia de un registro en el resultado de una subconsulta. Este operador excede el contenido del curso, pero puedes ver ejemplos de su uso en la siguiente publicación:

Fuente: **W 3 Schools** (s. f. e). SQL EXISTS Operator. Recuperado de https://www.w3schools.com/sql/sql_exists.asp

Las *queries* anidadas en el WHERE puede emplearse, también, para identificar registros duplicados. Por ejemplo, si necesitamos validar que no haya estudiantes inscriptos más de una vez a un curso, podemos ejecutar la siguiente consulta para identificar registros duplicados en la tabla INSCRIPCION:

```
Select *
From inscripcion as t1
where (select count(*)
            from inscripcion as t2
            where t1.CURSO_codigo = t2.CURSO_codigo
      and t1.ESTUDIANTE_legajo = t2.ESTUDIANTE_legajo) > 1;
```

Como se puede observar en el ejemplo, consultamos dos veces la tabla de inscripciones (para evitar ambigüedades, ponemos alias a cada tabla “t1” y “t2”): una para traer todos los registros (t1) y otra para contar la cantidad de inscripciones cuyo código y legajo en t2 coinciden con el código y legajo en t1. Gracias al “>1”, obtendremos solamente aquellos registros que devuelvan un número mayor que 1 en ese conteo.

Documentación

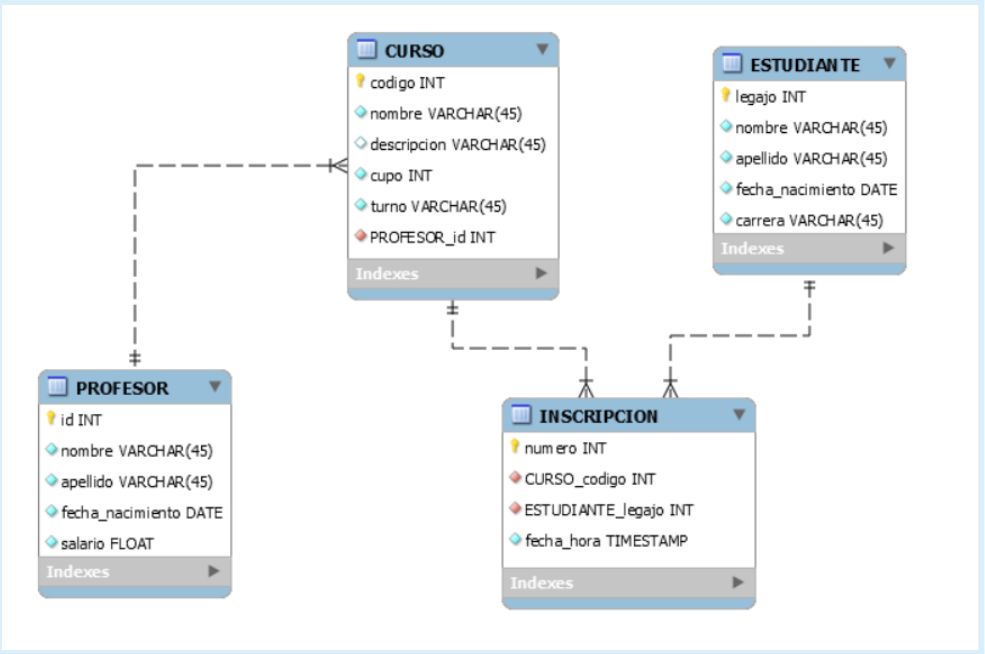
Videos

Fuente: Píldoras Informáticas [pildorasinformaticas]. (04 de octubre de 2015). Curso SQL. Subconsultas I. Vídeo 10 [YouTube]. Recuperado de https://www.youtube.com/watch?v=rGPb5E1UAJA

Ejercitación 4

El objetivo de este ejercicio es continuar con la resolución de consultas aplicando funciones de agregación.

Figura 6: Modelo relacional



Siguiendo con el modelo relacional visto a lo largo de esta cápsula, lleva a cabo las siguientes dos actividades:

1) Escribe una consulta que devuelva la cantidad de profesores que dictan más de un curso en el turno Noche.

   where turno = "Noche"
   group by id having count(*) > 1) turno_noche;

2) Escribe una consulta para obtener la información de todos los estudiantes que no realizan el curso con código 105.

Formato de entrega: la actividad se entrega a través de una URL correspondiente al repositorio sobre el que se haya trabajado.

Referencias

[Imagen sin título sobre modelo relacional], (s. f.). Recuperado de https://www.incanatoit.com/2015/10/tabla-referencia-cruzada-pivot-sql-server-2014.html

Cardenas Valenzuela, V. H. [Victor Hugo Cardenas Valenzuela] (s. f.). *Explicación de los Índices en el SQL Server de Microsoft* [YouTube]. Recuperado de https://www.youtube.com/watch?v=XWX1YvS5Kec

deividcoptero Programación [deividcoptero Programación], (27 de enero de 2014). *Tutoriales SQL Server #15 Funciones de Agregado* [YouTube]. Recuperado de https://www.youtube.com/watch?v=iAcv1jxEuGs

Microsoft (2019). *Índices agrupados y no agrupados descritos*. Recuperado de https://docs.microsoft.com/es-es/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15

Microsoft (2019a). *Guía de diseño y de arquitectura de índices de SQL Server*. Recuperado de https://docs.microsoft.com/es-es/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver15

Píldoras Informáticas [pildorasinformaticas]. (04 de octubre de 2015). *Curso SQL. Subconsultas I. Vídeo 10* [YouTube]. Recuperado de https://www.youtube.com/watch?v=rGPb5E1UAJA

Richardson, B. (2018). *¿Cuál es la diferencia entre Índices Agrupados y No Agrupados en SQL Server?* Recuperado de https://www.sqlshack.com/es/cual-es-la-diferencia-entre-indices-agrupados-y-no-agrupados-en-sql-server/

SEO - Blogger y Más [SEO - Blogger y Más], (13 de noviembre de 2013). *Crear consultas con MAS de una tabla en SQL SERVER* [YouTube]. Recuperado de https://www.youtube.com/watch?v=SVP4nFnD7_w

Tecnología Binaria [Tecnología Binaria;]. (03 de agosto de 2016). *Cláusula GROUP BY | Curso SQL Server - #58* [YouTube]. Recuperado de https://www.youtube.com/watch?v=renHFr2jeqk

W 3 Schools (s. f. a). *SQL COUNT(), AVG() and SUM() Functions*. Recuperado de https://www.w3schools.com/sql/sql_count_avg_sum.asp

W 3 Schools (s. f. b). *SQL GROUP BY Statement*. Recuperado de https://www.w3schools.com/sql/sql_groupby.asp

W 3 Schools (s. f. c). *SQL HAVING Clause*. Recuperado de https://www.w3schools.com/sql/sql_having.asp

W 3 Schools (s. f. d). *SQL Tutorial*. Recuperado de https://www.w3schools.com/sql/default.asp

W 3 Schools (s. f. e). *SQL EXISTS Operator*. Recuperado de https://www.w3schools.com/sql/sql_exists.asp

W 3 Schools (s. f.). *SQL MIN() and MAX() Functions*. Recuperado de https://www.w3schools.com/sql/sql_min_max.asp


alkemy