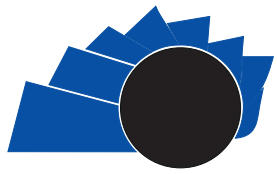
Visión Electrónica Vol. 12 No. 2 (2018) • Julio-Diciembre • p.p. 307-318 • ISSN 1909-9746 • ISSN-E 2248-4728 • Bogotá (Colombia)

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Visión Electrónica
Más que un estado sólido
https://revistas.udistrital.edu.co/index.php/visele



Visión Electrónica

VISIÓN ACTUAL

Arquitectura para la generación de consultas SQL usando lógica de conjuntos

Architecture for generating SQL queries, using logic sets

Luis Felipe Wanumen Silva[1], Darin Jairo Mosquera Palacios[2]

INFORMACIÓN DEL ARTÍCULO

Historia del artículo:
Enviado: 09/07/2018
Recibido: 01/08/2018
Aceptado: 29/08/2018

Palabras clave:
Algebra relacional
Generador de códigos
Herramientas case
Lógica de conjuntos
Sentencias SQL

Keywords:
Relational algebra
Code generator
Tools case
Logic set
SQL statements

RESUMEN

El presente artículo muestra, la arquitectura implementada en el desarrollo de una herramienta generadora de código SQL, concretamente código de consulta entre varias tablas, en donde se desean hacer operaciones que guardan relación con las operaciones de JOIN entre tablas. La herramienta explota el uso de LEFT JOIN, RIGHT JOIN, INNER JOIN Y FULL OUTHER JOIN. La arquitectura propuesta funciona para una gran cantidad de tablas, sin embargo, la herramienta por razones de eficiencia computacional en esta versión permite hasta tres tablas. La metáfora que sigue el generador de código SQL, es la metáfora impuesta por los diagramas de Venn, la cual es útil en múltiples aplicaciones computacionales y en este artículo se usó para la construcción de un generador de código en donde el usuario selecciona un diagrama de Venn concreto. Al final del artículo se muestran las modificaciones que se hace necesario implementar a la arquitectura para agregar otras funcionalidades.

ABSTRACT

This article shows the architecture implemented in the development of an SQL code generating tool (query code) between several tables, where it is desired to perform operations that are related to JOIN operations between tables. The tool exploits the use of LEFT JOIN, RIGHT JOIN, INNER JOIN and FULL OUTHER JOIN. The proposed architecture works for a large number of tables; however, the tool for reasons of computational efficiency in this version, allows up to three tables. The metaphor that follows the SQL code generator, is the metaphor imposed by Venn diagrams, which is useful in multiple computational applications and, in this article, it was used for the construction of a code generator, in which the user selects a concrete Venn diagram. At the end of the article, we show the modifications that are necessary to implement in the architecture, to add other functionalities.

[1] Ingeniero de Sistemas, Universidad Distrital Francisco José de Caldas, Colombia. Magister en Ingeniería de Sistemas y Computación, pontificia Universidad Javeriana, Colombia. Docente Universidad Distrital Francisco José de Caldas, Colombia. Correo electrónico: lwanumen@udistrital.edu.co, luchofelipe20002000@gmail.com ORCID: https://orcid.org/0000-0002-8877-5681

[2] Ingeniero de sistemas, Universidad Autónoma de Colombia, Magister en Teleinformática Universidad Distrital Francisco José de Caldas, Colombia, Docente Universidad Distrital Francisco José de Caldas, Colombia. Correo electrónico: djmosquerap@udistrital.edu.co, djmosquerap@hotmail.com. ORCID: https://orcid.org/0000-0002-4526-2683

Citar este artículo como: L. F. Wanumen y D. J. Mosquera, "Arquitectura para la generación de consultas SQL usando lógica de conjuntos". Visión Electrónica, algo más que un estado sólido, Vol. 12, No. 2, 307-318, julio-diciembre 2018. https://doi.org/10.14483/22484728.14343

1. Introducción

Los diagramas de Venn, son útiles en múltiples aplicaciones computacionales [1] e incluso podrían solucionar el problema de generar consultas SQL usando el álgebra relacional. Esto tecnológicamente sería un avance, ya que el testeo de consultas no siempre es una tarea fácil [2]. Ya otros autores han hecho incluso esfuerzos por generar sentencias SQL a partir de sistemas basados en símbolos [3]. Este artículo, describe una arquitectura que puede ser implementada por un sistema generador de consultas SQL para el caso específico de consultas que trabajan con operaciones de conjuntos. Una de las razones que motivó la arquitectura, fue la idea de desarrollar herramientas visuales que permitan generar consultas SQL, en donde se explota la hipótesis bien conocida en la cual la programación visual cuando se presenta como una interfaz entre el programador y el usuario final del sistema, genera muchas ventajas de comprensión del sistema y del código fuente [4].

La arquitectura expuesta se enfoca a solucionar los problemas que encuentran los desarrolladores de aplicaciones que manejan el lenguaje SQL quienes realizan consultas con múltiples tablas en donde la complejidad es inevitable en parte porque las combinaciones de consultas aumenta con cada tabla que se añade al sistema. Incluso cuando se trabaja con pocas tablas, por ejemplo el caso de trabajar con dos tablas, se presentan muchas opciones al realizar consultas que involucran estas tablas. La arquitectura expuesta se valida con un prototipo que genera código para dos y tres tablas inicialmente, pero que puede ser extendido a un número grande de tablas. Se presenta al final los elementos que se tendrían que mejorar para ampliar la arquitectura a fin de generar un código seguro, escalable, robusto, no vulnerable, simplificado y validado

2. Objetivo del proyecto

Se pretende generar una arquitectura e implementarla en una aplicación que permita al usuario ingresar el número de tablas que están implicadas en una consulta que puede ser resuelta con operadores JOIN. Dicha herramienta debe seguir unos lineamientos para transformar esta petición en una serie de opciones que permitan al programador de bases de datos, generar código SQL a partir de la selección de un tipo de diagrama de Venn. En la herramienta aquí mostrada, si no se tiene la posibilidad que el usuario escoja el número de tablas que están implicadas en el proceso, se recibe un archivo con sentencias DDL, el cual es procesado por la aplicación y calcula el número de tablas que están involucradas en el proceso. Esta arquitectura debe ser independiente del motor de bases de datos y del lenguaje de programación. Para validar el alcance de la arquitectura, el prototipo inicialmente debe funcionar en Access, Oracle y SQL Server.

3. Análisis del problema

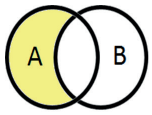
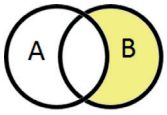
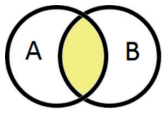
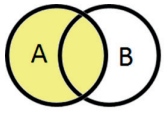
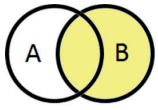
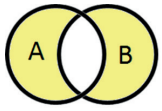
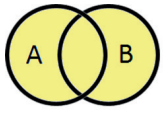
Es común el uso de diagramas de Venn para comprender el funcionamiento de consultas. Incluso, muchos docentes de asignaturas de bases de datos recurren a este tipo de diagramas para explicar las consultas entre varias tablas. La Tabla 1, muestra la relación existente entre los diagramas de Venn y los problemas a nivel de SQL que pueden ser solucionados con dichas abstracciones. (A estos problemas se les ha numerado y esta numeración se usará más tarde cuando se describa la solución propuesta).

Dicho de otra manera, cuando la aplicación muestre al usuario diagramas como los mostrados en la Tabla 1, la aplicación debe generar código SQL que funcione inicialmente en el motor de bases de datos SQL Server. Sin embargo para lograr este objetivo se debe transformar el código DDL ingresado por el usuario para establecer el número de opciones que se le va a mostrar gráficamente al usuario, dado que dependiendo el número de tablas a procesar, se tiene un número de diagramas de Venn. La Tabla 1 contiene 7 diagramas de Venn, que se pueden implementar en código SQL cuando el usuario ingrese dos tablas, sin embargo, si el usuario digita un número mayor de tablas, el número de diagramas que se le debe mostrar el usuario se incrementa notoriamente.

4. Arquitectura propuesta

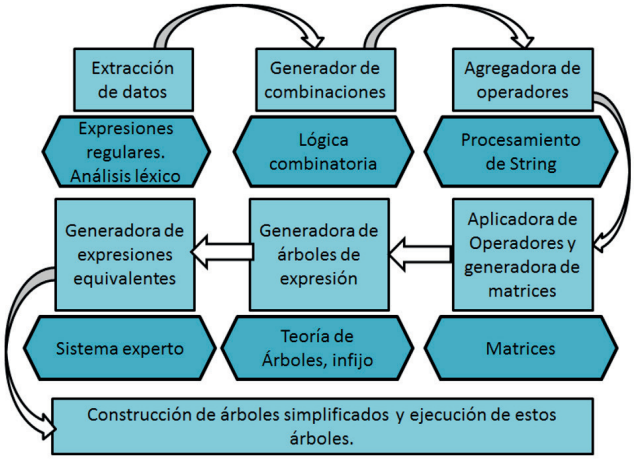
La arquitectura mostrada en la Figura 1 tiene una primera capa llamada capa de datos, la cual mediante el uso de expresiones regulares extrae los datos necesarios para la segunda capa. Esta capa permite tanto por teclado como por script conocer el número de tablas sobre la que se generará el código SQL. La capa 2 recoge esta información y usando lógica combinatoria logra identificar las posibles parejas que se pueden armar. La capa 3 incorpora los operadores a las secuencias y de esta

Tabla 1. Relación entre diagramas de venn y código sql

Notación Venn	Problema de SQL	Id
	Seleccionar los que hacen parte de A, pero que no hacen parte de B SELECT * FROM TABLAA A LEFT JOIN TABLAB B ON A.ID = B.ID WHERE B.ID IS NULL	P1
	Seleccionar los que hacen parte de B, pero que no hacen parte de A SELECT * FROM TABLAA A RIGHT JOIN TABLAB B ON A.ID = B.ID WHERE A.ID IS NULL	P2
	Seleccionar los que hacen parte de A, y de B SELECT * FROM TABLAA INNER JOIN TABLAB ON TABLAA.ID = TABLAB.ID	P3
	Seleccionar los que hacen parte de A y no hacen parte de B, más los que hacen parte de A y hacen parte de B Cualquiera de las siguientes instrucciones SELECT * FROM TABLAA LEFT JOIN TABLAB ON TABLAA.ID = TABLAB.ID SELECT * FROM TABLAA A LEFT JOIN TABLAB B ON A.ID = B.ID	P4
	Seleccionar los que hacen parte de B y no hacen parte de A, más los que hacen parte de B y hacen parte de A Cualquiera de las siguientes instrucciones: SELECT * FROM TABLAA RIGHT JOIN TABLAB ON TABLAA.ID = TABLAB.ID SELECT * FROM TABLAA A RIGHT JOIN TABLAB B ON A.ID = B.ID	P5
	Seleccionar los que hacen parte de A, pero no hacen parte de B, más los que hacen parte de B, pero no hacen parte de A SELECT * FROM TABLAA A LEFT JOIN TABLAB B ON A.ID = B.ID WHERE B.ID IS NULL UNION ALL SELECT * FROM TABLAA A RIGHT JOIN TABLAB B ON A.ID = B.ID WHERE A.ID IS NULL	P6
	Seleccione los que hacen parte de a y de b select * from tablaa inner join tablab on tablaa.id = tablab.id union all select * from tablaa a left join tablab b on a.id = b.id where b.id is null union all select * from tablaa a right join tablab b on a.id = b.id where a.id is null	P7

Fuente: elaboración propia.

Figura 1. El ejemplo de un gráfico con colores sólidos que resaltan sobre el fondo blanco



Fuente: elaboración propia.

forma haciendo procesamiento de script genera los encabezados y la primera columna para las matrices. La capa 4, genera unas matrices de posibilidad de unión, intersección, y operadores and, y OR. Estas matrices son evaluadas usando árboles de expresión, de tal forma que la capa 6 con esta información pueda calcular las expresiones equivalentes y haciendo uso del sistema Jess dicha capa entrega a la capa 7 los elementos para generar árboles definitivos que permitan ejecutar la consulta SQL.

4.1. Descripción detallada de la arquitectura propuesta

La solución entregada al problema sigue una serie de pasos que son ejecutados por una capa. En este apartado se muestran al detalle la ejecución de cada capa y en la siguiente sección se unen las piezas para mostrar la arquitectura propuesta.

4.1.1. Primera capa: Extracción de datos usando expresiones regulares

En una primera instancia el sistema recibe un script, a dicho script le extrae los nombres de las tablas y las claves primarias. Esto lo hace mediante la aplicación de expresiones regulares, en donde se detectan a partir del script DDL, los nombres de las tablas y de los campos. Al finalizar este primer proceso se tiene una lista de las tablas, campos y número de tablas con las que el sistema continuará su proceso de generación de código.

Para explicar la solución, se asume que el script contiene una tabla llamada "A" y otra llamada "B". De esta forma, la Tabla 1 es útil para mostrar el número de problemas que se presentan. Sin embargo, la forma para llegar a establecer que son siete opciones de consulta requiere la elaboración de un proceso de ingeniería basado en lógica de conjuntos y lógica proposicional.

4.1.2. Segunda capa: Generación de combinaciones necesarias

En segunda instancia los resultados del proceso anterior, se pasan a una capa generadora de combinaciones. En el caso de los datos anteriores el conjunto de las tablas está dado por:

S = {A, B}

Esta capa se encarga de hallar el número de formas diferentes en que se pueden obtener los nombres de las tablas (es decir, el número de permutaciones del conjunto). Por ejemplo, para el conjunto anteriormente mostrado, algunas formas distintas podrían ser:

Combinaciones (S1) = {A, B}, si se saca una tabla al tiempo, pero si se sacan dos tablas al tiempo, se obtiene:

Combinaciones (S2) = {AB, BA}. Sin embargo en la solución propuesta para este primer procesamiento, no se tiene en consideración el orden y por lo tanto bajo esta restricción tanto "AB" como "BA" son considerados como el mismo resultado obteniéndose (S2) = {AB}

Para comprender mejor esta parte supongamos que las tablas recibidas están dadas por el conjunto: S = {A, B, C, D}

Y dado que son 4 elementos, se generan cuatro subconjuntos de combinaciones que son:

Combinaciones (S1) = {A, B, C, D}

Combinaciones (S2) = {AB, AC, AD, BC, BD}

Combinaciones (S3) = {ABC, ABD, ACD, BCD}

Combinaciones (S4) = {ABCD}

Existen muchas alternativas para generar estos subconjuntos. La alternativa propuesta es haciendo uso del equivalente binario de un número. Por ejemplo si el conjunto de entrada es:

S = {A, B, C, D, E}

Se usa la Tabla 2 para calcular las posibles combinaciones y subconjuntos, de tal suerte que se obtienen los siguientes subconjuntos:

Combinaciones (S1) = {A, B, C, D, E}

Combinaciones (S2) = {DE, CE, CD, BE, BD, BC, AE, AD, AC, AB}

Combinaciones (S3) = {CDE, BDE, BCE, BCD, ADE, ACE, ACD, ABE, ABD, ABC}

Combinaciones (S4) = {BCDE, ACDE, ABDE, ABCE, ABCD}

Combinaciones (S5) = {ABCDE}

4.1.3. Tercera capa

La capa anterior entrega una serie de subconjuntos Esta capa es la encarada de agregar operadores de conjuntos. Estos subconjuntos son procesados para añadirles un operador de conjuntos "unión" o "intersección" y son concatenados. Por ejemplo para las salidas anteriores (de 5 combinaciones), se tienen las siguientes secuencias de operaciones:

Secuencia con Operador UNION = {A, B, C, D, E, D∪E, C∪E, C∪D, B∪E, B∪D, B∪C, A∪E, A∪D, A∪C, A∪B, C∪D∪E, B∪D∪E, B∪C∪E, B∪C∪D, A∪D∪E, A∪C∪E, A∪C∪D, A∪B∪E, A∪B∪D, A∪B∪C, B∪C∪D∪E, A∪C∪D∪E, A∪B∪D∪E, A∪B∪C∪E, A∪B∪C∪D, A∪B∪C∪D∪E}

Secuencia con Operador INTERSECCIÓN = {A, B, C, D, E, D∩E, C∩E, C∩D, B∩E, B∩D, B∩C, A∩E, A∩D, A∩C, A∩B, C∩D∩E, B∩D∩E, B∩C∩E, B∩C∩D, A∩D∩E, A∩C∩E, A∩C∩D, A∩B∩E, A∩B∩D, A∩B∩C, B∩C∩D∩E, A∩C∩D∩E, A∩B∩D∩E, A∩B∩C∩E, A∩B∩C∩D, A∩B∩C∩D∩E}. La Tabla 2 muestra cómo se usó la técnica de analizar un número binario para calcular los subconjuntos que se pueden crear a partir de este número.

4.1.4. Cuarta capa

Esta cuarta capa aplica operadores "AND", "OR" y "NOT" a las anteriores secuencias, tal como describe la Tabla 3.

Tabla 2. Uso del equivalente binario de un número para calcular subconjuntos

A	B	C	D	E	Decimal	Numero de Unos	S1	S2	S3	S4	S5
0	0	0	0	1	1		1				
0	0	0	1	0	2	1	1				
0	0	0	1	1	3	2		2			
0	0	1	0	0	4	1	1				
0	0	1	0	1	5	2		2			
0	0	1	1	0	6	2		2			
0	0	1	1	1	7	3			3		
0	1	0	0	0	8	1	1				
0	1	0	0	1	9	2		2			
0	1	0	1	0	10	2		2			
0	1	0	1	1	11	3			3		
0	1	1	0	0	12	2		2			
0	1	1	0	1	13	3			3		
0	1	1	1	0	14	3			3		
0	1	1	1	1	15	4				4	
1	0	0	0	0	16	1	1				
1	0	0	0	1	17	2		2			
1	0	0	1	0	18	2		2			
1	0	0	1	1	19	3			3		
1	0	1	0	0	20	2		2			
1	0	1	0	1	21	3			3		
1	0	1	1	0	22	3			3		
1	0	1	1	1	23	4				4	
1	1	0	0	0	24	2		2			
1	1	0	0	1	25	3			3		
1	1	0	1	0	26	3			3		
1	1	0	1	1	27	4				4	
1	1	1	0	0	28	3			3		
1	1	1	0	1	29	4				4	
1	1	1	1	0	30	4				4	
1	1	1	1	1	31	5					5

Fuente: elaboración propia.

Tabla 3. Formas de combinar las secuencias para generar encabezados de matrices.

Operador aplicado	Secuencia UNO	Secuencia DOS
AND	Secuencia con Operador UNION	Secuencia con Operador UNION
OR	Secuencia con Operador UNION	Secuencia con Operador UNION
NOT	Secuencia con Operador UNION	Secuencia con Operador UNION
NOT	Secuencia con Operador INTERSECCIÓN	Secuencia con Operador INTERSECCIÓN
NOT	Secuencia con Operador INTERSECCIÓN	Secuencia con Operador UNION
NOT	Secuencia con Operador UNION	Secuencia con Operador INTERSECCIÓN

Fuente: elaboración propia.

Por ejemplo para el conjunto S = {A, B, C} se generan seis matrices. Las tres primeras matrices usan las combinaciones de operaciones de unión entre conjuntos. La diferencia está en el hecho que la primera matriz usa el operador lógico "AND", la segunda matriz el operador lógico "OR" y la tercera matriz usa el operador lógico "NOT". Para el caso de las tablas ingresadas en el script, las tres primeras matrices se muestran en la Tabla 4:

Tabla 4. Matrices and, or y not para el problema con dos tablas

Matriz 1			
AND	A	AUB	B
A	P4	P4	P3
AUB	P3	P3	P3
B	P3	P3	P5

Matriz 2			
OR	A	AUB	B
A	P4	P4	P7
AUB	P4	P3	P3
B	P7	P3	P5

Matriz 3			
NOT	A	AUB	B
A	X	X	P1
AUB	P5	X	P1
B	P2	X	X

Fuente: elaboración propia.

Es importante observar que estas matrices corresponden a colocar en la primera fila y en la primera columna todas las posibilidades de aplicación de una operación de conjuntos. Por ejemplo si existieran tres tablas, la primera fila y la primera columna estarían dadas por: A, B, C, AUB, AUC, BUC y AUBUC. La Tabla 5 muestra las otras tres tablas generadas, usando el operador "NOT"

Tabla 5. Matrices not para el problema con dos tablas

Matriz 4			
NOT	A	AnB	B
A	x	P1	P1
AnB	X	x	X
B	P2	P2	x

Matriz 5			
NOT	A	AnB	B
A	X	P1	P1
AUB	P2	P6	P1
B	P2	P2	X

Matriz 6			
NOT	A	AUB	B
A	X	X	P1
AnB	X	X	X
B	P2	X	X

Fuente: elaboración propia.

Las X son colocadas por el analizador léxico desarrollado en el proyecto y los valores colocados en el resto de las celdas corresponden con los tipos de problemas expuestos en la Tabla 1. Posteriormente se pasa un optimizador lógico, que no es más que una capa encargada de verificar cuantas opciones hay para atacar los problemas expuestos en la tabla 1. Por ejemplo para el caso de los problemas expuestos se tiene el siguiente número de formas de atacar los problemas contando el número de veces que aparece dicha solución en las matrices, tal como se muestra en la Figura 2

Figura 2. Número de opciones para solucionar los problemas

AND	A	AUB	B
A	P4	P4	P3
AUB	P3	P3	P3
B	P3	P3	P5

NOT	A	AnB	B
A	x	P1	P1
AnB	X	x	X
B	P2	P2	x

Problema	Número Veces
p1	8
p2	7
p3	9
p4	5
p5	3
p6	1
p7	1

OR	A	AUB	B
A	P4	P4	P7
AUB	P4	P3	P3
B	P7	P3	P5

NOT	A	AnB	B
A	X	P1	P1
AUB	P2	P6	P1
B	P2	P2	X

NOT	A	AUB	B
A	X	X	P1
AUB	P5	X	P1
B	P2	X	X

NOT	A	AUB	B
A	X	X	P1
AnB	X	X	X
B	P2	X	X

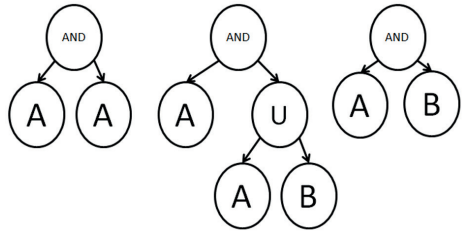
Fuente: elaboración propia.

La Figura 2 demuestra que es posible obtener una solución de múltiples formas. De hecho, si se sigue el ejercicio usando más matrices, es posible que se obtengan más formas de solucionar los problemas. Sin embargo no es necesario hacer uso de más matrices en el sentido que con estas se llega a un código SQL para cada uno de los problemas presentados en la Tabla 1.

4.1.5. Quinta capa

Posteriormente se genera un árbol de expresión basado en las anteriores matrices. La razón para el uso de árboles de expresión es que son útiles en la evaluación de expresiones ya que colocan en el nodo padre el operador y en los nodos hijos izquierdo y derecho los elementos a operar. Esto se hace en forma recursiva y permite evaluar expresiones matemáticas [5] o expresiones que usan la lógica matemática. En el caso del problema expuesto en la matriz 1 de la Figura 4 se tienen los árboles de expresión de la Figura 3.

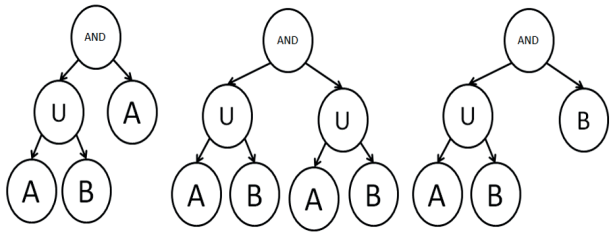
Figura 3. Árboles de expresión generador a partir de la fila 1, matriz 1 de la figura 2



Fuente: elaboración propia.

Ejemplo de estructuras del árbol generadas a partir de la fila 1 de la matriz 1 con el operador lógico AND de la Figura 2. Y en forma similar a la anterior, se generan también arboles por cada una de las filas de la Figura 2. Es así como se pueden generar los árboles de la Figura 4 a partir de la fila 2 de la Figura 2 de la tabla con el operador lógico "OR"

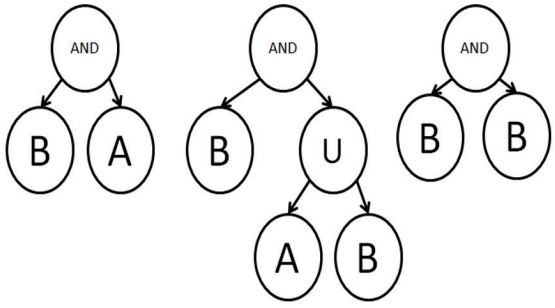
Figura 4. Árboles de expresión generador a partir de la fila 2, matriz 1 de la figura 2



Fuente: elaboración propia.

Finalmente la Figura 5, muestra los últimos tres árboles de expresión generados a partir de la fila 3, de la matriz 1 de la Figura 2.

Figura 5. Árboles de expresión generador a partir de la fila 3, matriz 1 de la figura 2



Fuente: elaboración propia.

4.6.Sexta capa

A pesar que la Figura 2 muestra el número de opciones para solucionar los problemas, este número no se puede calcular todavía con un valor exacto. Para hacerlo tendríamos que hacer una evaluación gráfica del diagrama de Venn. Sin embargo esto todavía no se ha hecho. Es más no se debe hacer, porque generaría un gasto computacional generar los conjuntos solución para cada una de las celdas de las matrices anteriores, debido a que como se aprecia en la Figura 1, son muchas las soluciones a un mismo problema. Una forma de solucionar el problema es generando un vector de expresiones equivalentes. Esto se hace usando lógica proposicional. Un api de sistemas expertos como JESS podría ayudarnos en esta parte. Con la ayuda de esta API, se obtendrían las siguientes conclusiones por ejemplo para el conjuntos S = {A, B} y al aplicar las secuencias generadas en la capa 3, obtenemos los resultados de la Tabla 6.

4.1.7. Séptima capa

La capa sexta ha entregado los datos de la Tabla 6. En la columna 1. Con estas expresiones se construye un árbol de expresiones y se comienza a ejecutar. Al finalizar se producen los resultados esperados en la Tabla 7.

5. Descripción del prototipo

El prototipo es una herramienta que funciona para generar código SQL para SQL Server, Access, y Oracle. El insumo principal para el funcionamiento de la aplicación es un archivo SQL con instrucciones de lenguaje DDL. La Tabla 6 muestra las expresiones equivalentes que se han generado para luego ser transformadas en código SQL.

Tabla 6. Resultado de analizar equivalencias

Región según diagrama Venn	Expresiones equivalentes	Problema solucionado
A^B	A^(AnB) (Matriz 4) / A^(B) (Matriz 4)/ A^(B) (Matriz 3) / (A∪B)^(B) (Matriz 3) / A^(AcB) (Matriz 5) / A^(B) (Matriz 5) / (A∪B)^(B) (Matriz 5) / A^(B) (Matriz 6)	P1
B^A	B^A (Matriz 3) / B^A (Matriz 4) B^(A∩B) (Matriz 4) / (A∪B)^(A) (Matriz 3) / (A∪B)^(A) (Matriz 5) / (B)^(A) (Matriz 5) / B^(A∩B) (Matriz 5)	P2
AB	AB (Matriz 1) / (A∪B)A (Matriz 1) / (A∪B) (A∪B) (Matriz 1) / (A∪B)B (Matriz 1) / BA (Matriz 1) / B(A∪B) (Matriz 1) / (A∪B) +(A∪B) (Matriz 2) / (A∪B) +(B) (Matriz 2) / (B) +(A∪B) (Matriz 2)	P3
A	AA (Matriz 1) / A(A∪B) (Matriz 1) / A+A (Matriz 2) / A+(A∪B) (Matriz 2) / (A∪B)+ (A∪B) (Matriz 2)	P4
B	BB / B+B	P5
(A∪B)^(A∩B)	(A∪B)^(A∩B)	P6
A+B	A+B (Matriz 2) / B+A (Matriz 2)	P7

Fuente: elaboración propia.

La Tabla 7 muestra el código fuente que ha sido usado para probar el prototipo de generación de código SQL.

Este archivo a nivel conceptual generaría las entidades "Tabla A" y "Tabla B", mostradas en la Tabla 8

Tabla 7. Código fuente insumo del prototipo

```
CREATE TABLE TABLAA
( Id int primary key,
Campo1 varchar(50)
);
CREATE TABLE TABLAB
( Id int primary key,
Campo1 varchar(50)
);
Insert into TABLAA values (1, 'Campo1')
insert into TABLAA values (2, 'Campo2')
insert into TABLAA values (3, 'Campo3')
insert into TABLAA values (4, 'Campo4')
insert into TABLAA values (5, 'Campo5')
insert into TABLAB values (1, 'String1')
insert into TABLAB values (2, 'String2')
insert into TABLAB values (3, 'String3')
insert into TABLAB values (6, 'String6')
insert into TABLAB values (7, 'String7')
```

Fuente: elaboración propia.

Tabla 8. Tablas de prueba para el prototipo

Tabla A	Tabla B
Id	Id
Campo1	Campo1
1	1
Cadena1	String1
2	2
Cadena2	String2
3	3
Cadena3	String3
4	6
Cadena4	String6
5	7
Cadena5	String7

Fuente: elaboración propia.

Este Script se guarda en el disco duro con el nombre de "scriptSQL_Server.sql.txt" y mediante la opción "abrir" se carga el archivo, tal como muestra la Figura 6.

En nuestro caso, dado que es un script que tiene instrucciones DDL y DML relacionadas con dos tablas, en la opción "Número de Conjuntos" aparece el número "2". En la Figura 7.

Aparecen unos íconos en la parte izquierda, al hacer clic sobre cada uno de ellos, se realiza la operación respectiva de consulta usando la teoría de conjuntos. De esta manera si se hace clic sobre el séptimo diagrama de Venn de la parte izquierda del software mostrado en la gráfica anterior, se obtiene el código de la Tabla 9.

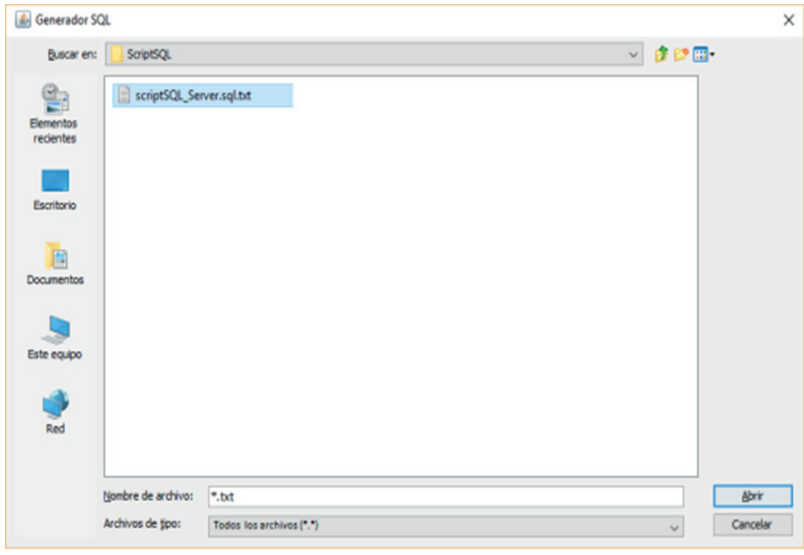
Tabla 9. Primera salida del programa

```
SELECT * FROM TABLAA A LEFT JOIN TABLAB B ON A.ID = B.ID WHERE
B.ID IS NULL
UNION ALL
SELECT * FROM TABLAA A RIGHT JOIN TABLAB B ON A.ID = B.ID
WHERE A.ID IS NULL
```

Fuente: elaboración propia.

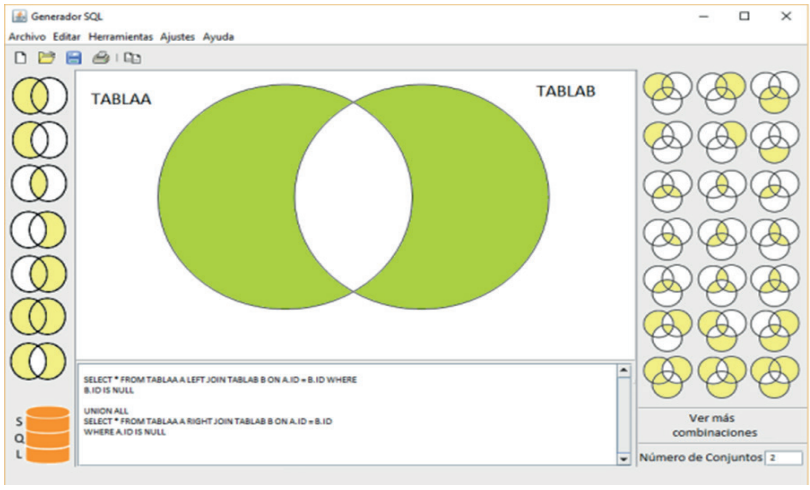
Si se hace clic sobre el sexto diagrama de Venn de la parte izquierda del sistema, se obtiene el código de la Tabla 10.

Figura 6. Opción de abrir del prototipo



Fuente: elaboración propia.

Figura 7. Área de trabajo del prototipo



Fuente: elaboración propia.

Tabla 10. Segunda salida del programa

```
SELECT * FROM TABLAA INNER JOIN TABLAB
ON TABLAA.ID = TABLAB.ID
UNION ALL
SELECT * FROM TABLAA A LEFT JOIN TABLAB B
ON A.ID = B.ID WHERE B.ID IS NULL
UNION ALL
SELECT * FROM TABLAA A RIGHT JOIN TABLAB B
ON A.ID = B.ID WHERE A.ID IS NULL
```

Fuente: elaboración propia.

En la parte de configuración se le indica a la herramienta en qué lenguaje se quiere generar el código SQL y también en qué lenguaje se carga el script de definición de datos. Es importante tener en cuenta que el script puede contener no sólo instrucciones de definición de datos, sino también de inserción de datos, caso en el cual la herramienta limpia el código de este último tipo de instrucciones para usar solamente las instrucciones de tipo DDL para establecer las operaciones de consulta usando lógica de conjuntos.

6. Resultados

El algoritmo de generación de consultas fue probado con diversos números de conectores and y or y se verificó para cada tamaño, el tiempo que tardó en generar la consulta. Este tipo de análisis fue muy importante en el sentido que permite visualizar el grado de escalabilidad que tiene el sistema actual si se usa el mismo algoritmo y se implementan consultas con mayor número de conectores. La Tabla 11 muestra los resultados obtenidos y permita visualizar que si las consultas tienen más de 100 conectores, el tiempo que requiere el sistema para visualizar y procesar los datos es extremadamente grande en el sentido que es de aproximadamente 50 segundos. Esto deja entrever que para consultas con un gran número de conectores, se hace necesario pensar en proyectos futuros tendientes a disminuir el tiempo. Una posible solución es paralelizar el algoritmo. Sin embargo 100 conectores es una cantidad muy grande y si lo normal es consultas con máximo 20 conectores, el tiempo de respuesta del sistema desarrollado es perfecto, sería aproximadamente de máximo 2 segundos en promedio.

Tabla 11. Resultados de eficiencia en el tiempo del sistema

Número de conectores	Tiempo en Milisegundos	Tiempo en Segundos
3	1600	1,6
4	3100	3,100
5	962,9032258	0,9629032258
6	1024,703088	1,024703088
7	3143,761141	3,143761141
8	20836,9509	20,8369509
9	3366,683848	3,366683848
10	1764,79313	1,76479313
20	2673,438136	2,673438136
30	3523,751157	3,523751157
50	170501,7286	170,5017286
80	10365,08824	10,36508824
100	50549,33344	50,54933344

Fuente: elaboración propia.

7. Conclusiones y trabajo futuro

Al momento de probar la ejecución de consultas, los programadores acostumbran a comprobar consultas ejecutándolas en cada uno de los casos de prueba creados por ellos mismos y comprueban que la consulta obtiene los resultados deseados en cada caso de prueba. Sin embargo el problema es cuando esos casos de prueba no son suficientes o no están diseñados bien y no permiten probar verdaderamente la ejecución de una consulta. [2].

Es posible que con las combinaciones posibles se lleguen incluso a encontrar mutaciones a las consultas, en tanto que las mutaciones en el caso de las consultas SQL son un solo cambio sintácticamente correcto que modifica el programa de una manera controlada para producir una consulta mutada que podría en algunos casos producir errores o que podría explotar alguna vulnerabilidad del motor de base de datos [2].

Si la herramienta mostrada permite generar un SQL bien formado, es posible proponer luego proyectos de generación de servicios web personalizados basados en estas consultas SQL que están bien formadas [6].

En el sistema mostrado se está generando el SQL a partir de una manipulación de cadenas SQL, pero es posible generar también el SQL a partir de un conjunto de clases fuertemente tipificadas en un esquema de bases de datos [7].

Una forma de probar si un generador de código SQL funciona bien, es construyendo un generador de datos de prueba que permita validar si el código SQL realmente funciona bien [8].

Un generador de código SQL no sólo debería preocuparse por el rendimiento de las consultas generadas, sino también por la simplicidad de las consultas que genera, en el sentido que en la medida en que estas consultas sean sencillas, las bases de datos se hacen más fáciles de usar [9].

Un generador de código SQL podría tener en cuenta también como parámetro de evaluación el tiempo de compilación de las consultas generadas, de tal forma que busque generar sentencias SQL que demanden la menor cantidad de tiempo al ser compiladas [10]

Sería bueno proponer como proyectos futuros analizadores de la eficiencia de las consultas generadas, ya que este tipo de análisis cobran mucho interés cuando estas consultas se ejecutan sobre estructuras con grandes cantidades de datos como en el caso de la Cloud [11].

Las matrices que se usan el proyecto tienen en cuenta algunas posibles combinaciones de sentencias SQL, pero si se genera una mayor cantidad de combinaciones es posible que se encuentren soluciones nuevas [12].

Se pueden proponer proyectos futuros tendientes a verificar que las consultas generadas no presentan vulnerabilidades por ataques como por ejemplo los de inyección SQL [13]. Algunos autores incluso llegan a proponer los modelos bayesianos para detectar anomalías en bases de datos SQL a fin de garantizar la seguridad de un sistema [14].

El sistema mostrado genera consultas a partir de unos script con sentencias DDL, es posible crear herramientas que hagan el trabajo inverso, es decir que a partir de las consultas generadas, obtengan las estructuras de datos [15].

Es posible proponer proyectos futuros tendientes a encontrar similitudes entre los códigos generados y predecir cuando el usuario ha realizado una consulta que se puede hacer con otro código similar [16]

Si es posible hacer una representación matemática de una consulta y con una expresión matemática es posible construir frameworks que permitan detectar contradicciones matemáticas [17] también es posible proponer proyectos futuros tendientes a encontrar contradicciones entre consultas SQL generadas. De otra parte profundizando en el tema de los árboles, es posible usar transformaciones para reconocer expresiones matemáticas [18] y como las consultas tienen una expresión que puede escribirse en forma matemática, es posible verificar si dos consultas producen el mismo resultado mediante este tipo de técnicas basada en transformación de árboles.

Una alternativa interesante que surge fruto del sistema mostrado en este artículo, es la de trasladar las sentencias SQL generadas en el sistema expuesto en expresiones de algebra relacional a fin de realizar no solamente proceso de optimización, sino de análisis semántico y de equivalencias de consultas SQL [19][20]

8. Reconocimientos

Luis Felipe Wanumen Silva, Darin Jairo Mosquera Palacios, agradecen a la Universidad Distrital francisco José de Caldas por el apoyo brindado en la elaboración de este artículo.

Referencias

[1] D. B. Wijesinghe, S. Ranathunga y G. Dias, "Computer representation of Venn and Euler diagrams," 2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer), Negombo, 2016, pp. 100-105, https://doi.org/10.1109/ICTER.2016.7829905

[2] S. Shah, S. Sudarshan, S. Kajbaje, S. Patidar, B. P. Gupta y D. Vira, "Generating test data for killing SQL mutants: A constraint- based approach," 2011 IEEE 27th International Conference on Data Engineering, Hannover, 2011, pp. 1175-1186, https://doi.org/10.1109/ICDE.2011.5767876

[3] M. S. Mahmood, M. A. Ghafoor y J. H. Siddiqui, "Symbolic execution of stored procedures in database management systems," 2016. 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, 2016, pp. 519-530, https://doi.org/10.1145/2970276.2970318

[4] S. Dahl y K. Lindqvist, "Visual programming as an interface between program and user?," [Proceedings] 1989 IEEE Workshop on Visual Languages, Rome, 1989, pp. 18-23, https://doi.org/10.1109/WVL.1989.77036

[5] W. R. Anderson y I. Imam, "A numeric parser and evaluator for algebraic inorder expressions," Southeastcon '89. Proceedings. Energy and Information Technologies in the Southeast., IEEE, Columbia, SC, 1989, pp. 1093-1098, https://doi.org/10.1109/SECON.1989.132578

[6] Q. Chi, Y. Zhu y H. Zhu, "A Research of SQL-Based Web Services Automatic Generating Strategy," 2010 IEEE Asia-Pacific Services Computing Conference, Hangzhou, 2010, pp. 707-711, https://doi.org/10.1109/APSCC.2010.33

[7] R. A. McClure y I. H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005., Saint Louis, MO, USA, 2005, pp. 88-96, https://doi.org/10.1145/1062455.1062487

[8] B. P. Gupta, D. Vira y S. Sudarshan, "X-data: Generating test data for killing SQL mutants," 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), Long Beach, CA, 2010, pp. 876-879, https://doi.org/10.1109/ICDE.2010.5447862

[9] J. Fan, G. Li y L. Zhou, "Interactive SQL query suggestion: Making databases user-friendly," 2011 IEEE 27th International Conference on Data Engineering, Hannover, 2011, pp. 351-362, https://doi.org/10.1109/ICDE.2011.5767843

[10] R. A. McClure y I. H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005., Saint Louis, MO, https://doi.org/10.1145/1062455.1062487

[11] Z. Wu, A. Song, J. Cao, L. Junzhou, L. Zhang, "Efficiently Translating Complex SQL Query to MapReduce Jobflow on Cloud," in IEEE Transactions on Cloud Computing, vol. 1, n° 99, pp.1-1, May 2017, https://doi-org.ezproxy.javeriana.edu.co/10.1109/TCC.2017.2700842

[12] S. Sen, D. Marijan, C. Ieva, A. Grime y A. Sander, "Modeling and Verifying Combinatorial Interactions to Test Data Intensive Systems: Experience at the Norwegian Customs Directorate," in *IEEE Transactions on Reliability*, vol. 66, n° 1, pp. 3-16, March 2017, https://doi.org/10.1109/TR.2016.2618121

[13] K. Kamtuo y C. Soomlek, "Machine Learning for SQL injection prevention on server-side scripting," 2016 International Computer Science and Engineering Conference (ICSEC), Chiang Mai, 2016, pp. 1-6, https://doi.org/10.1109/ICSEC.2016.7859950

[14] M. M. Drugan, "A Bayesian model for anomaly detection in SQL databases for security systems," 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, 2016, pp. 1-8, https://doi.org/10.1109/SSCI.2016.7849905

[15] C. Grant, D. Z. Wang y M. Wick, "Query-Driven Sampling for Collective Entity Resolution,"

2016 IEEE 17th International Conference on Information Reuse and Integration (IRI), Pittsburgh, PA, 2016, pp. 208-217, https://doi.org/10.1109/IRI.2016.34

[16] M. Agrawal y D. K. Sharma, "A novel method to find out the similarity between source codes," 2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON), Varanasi, 2016, pp. 339-343, https://doi.org/10.1109/UPCON.2016.7894676

[17] M. Cristia, P. Albertengo y P. R. Monetti, "Pruning Testing Trees in the Test Template Framework by Detecting Mathematical Contradictions," 2010 8th IEEE International Conference on Software Engineering and Formal Methods, Pisa, 2010, pp. 268-277, https://doi.org/10.1109/SEFM.2010.31

[18] R. Zanibbi, D. Blostein y J. R. Cordy, "Recognizing mathematical expressions using tree transformation," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, n° 11, pp. 1455-1467, Nov 2002, https://doi.org/10.1109/TPAMI.2002.1046157

[19] S. Ceri and G. Gottlob, "Translating SQL Into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries," in *IEEE Transactions on Software Engineering*, vol. SE-11, n° 4, pp. 324-345, April 1985, https://doi.org/10.1109/TSE.1985.232223

[20] S. Qian y Y. Lu, "How to achieve SQL in approximate divide and exact relation division by Relational algebra," 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE), Singapore, 2010, pp. 552-554