Determinando la existencia de relaciones en bases de datos relacionales mediante el algoritmo DFS

RESUMEN: Las bases de datos relacionales son la estrategia más utilizada para almacenar y recuperar datos de una manera organizada, soportando el lenguaje SQL como su interfaz de consulta. Sin embargo, el poder de SQL se ve limitado para expresar consultas analíticas complejas para extraer información de bases de datos cuando se desconoce su esquema entidad-relación. Para solventar esta problemática, en este artículo se propone un enfoque que permite determinar y validar la existencia de relaciones entre tablas de bases de datos relacionales mediante la utilización del algoritmo DFS como mecanismo de búsqueda no informada. Se implementó realizando un grafo equivalente de la base de datos a través de listas de adyacencia en Python, el cual se suministra al algoritmo DFS para encontrar la ruta más corta; es decir, las relaciones entre las tablas de la base de datos, permitiendo la construcción de los productos cartesianos necesarios para conformar una consulta SQL compleja y dinámica que permite la extracción de los datos aun desconociendo el esquema entidad-relación. El enfoque se integró a un sistema de web de apoyo a la toma de decisiones demostrando la viabilidad del enfoque. La aplicación de este enfoque permite desarrollar sistemas de información eficientes.

PALABRAS CLAVE: bases de datos relacionales, búsqueda en profundidad, DFS, consultas SQL dinámicas, grafos.

Colaboración
Héctor Adán Morales Lugo; Patricia Elizabeth Figueroa Milán; Nicandro Farias Mendoza; Jesús Alberto Verduzco Ramírez; Ramona Evelia Chávez Valdez, Tecnológico Nacional de México / Instituto Tecnológico de Colima

ABSTRACT: Relational databases are the most used strategy to store and retrieve data in an organized way, supporting the SQL language as its query interface. However, the power of SQL is limited to express complex analytical queries to extract information from databases when its entity-relationship schema is unknown. In order to solve this problem, this article proposes an approach that allows determining and validating the existence of relationships between relational database tables by using the DFS algorithm as an uninformed search mechanism. It was implemented by making an equivalent graph of the database through adjacency lists in Python, which is supplied to the DFS algorithm to find the shortest path; that is, the relationships between the tables in the database, allowing the construction of the Cartesian products necessary to form a complex and dynamic SQL query that allows the extraction of the data even though the entity-relationship schema is unknown. The approach integrated a web-based decision support system demonstrating the viability of the approach. The application of this approach allows the development of efficient information systems.

KEYWORDS: relational databases, depth-first search, DFS, dynamic SQL queries, graphs.

INTRODUCCIÓN

Las bases de datos (BD) son la principal solución para satisfacer la necesidad de almacenar y recuperar datos de una manera organizada. Permiten representar la estructura del negocio de una empresa, optimizando y automatizando los procesos cotidianos de comercialización; por lo cual, los datos mundiales se duplican cada dos años con una gran cantidad de transacciones que requieren soluciones de almacenamiento. Dentro de estas soluciones, las más utilizadas son las bases de datos relacionales (BDR) y las bases de datos no relacionales, conocidas comúnmente como NoSQL (Not only SQL) [1].

Actualmente, las bases de datos que han tomado mayor popularidad son las bases de datos NoSQL, ya que soportan la gestión de datos estructurados, no estructurados y semiestructurados en formas no tabulares. Sin embargo, a pesar de su popularidad, éstas no reemplazarán completamente a las BDR principalmente por que las BDR se distinguen por su consistencia y seguridad [2], así como por la gran cantidad de herramientas que soportan para inteligencia de negocios, análisis, toma de decisiones e informes [3]. Además, en [4] se demostró que las BDR proporcionan un mejor rendimiento general en comparación por ejemplo con una base de datos orientadas a grafos, un tipo de las bases de datos NoSQL.

Las BDR almacenan los datos en forma de filas y columnas dentro de una tabla, manteniendo las relaciones entre éstas mediante una llave externa [5]. Utilizan únicamente el Lenguaje Estructurado de Consultas (SQL, por sus siglas en inglés) para consultar y manipular los datos almacenados en éstas. No obstante, la versión estándar de SQL no puede expresar efectivamente consultas avanzadas y analíticas. Por ejemplo, un escenario en donde el poder de SQL es limitado es cuando se requiere extraer información de una BD externa al sistema, de la cual se desconoce su esquema. En este caso, la extracción o acceso a la información se convierte en una tarea complicada.

Una de las estrategias para solventar esta problemática es la implementación de algoritmos de búsqueda no informa ya que, no necesitan conocer información sobre su dominio [6]. Entre estos algoritmos, el algoritmo de Búsqueda en Profundidad (DFS, por sus siglas en inglés) es de utilidad para determinar la conectividad informática, recorre cada ruta de cada nodo dentro de un conjunto de nodos unidos por aristas (grafo o árbol) hasta más no poder. Si esto se traslada a una BD, el algoritmo DFS permitiría determinar las relaciones entre las tablas de cualquier BD relacional, a partir de la creación de un árbol o grafo equivalente al esquema de la BD y con esto, construir las consultas SQL de forma dinámica sin tener que utilizar varios lenguajes de datos.

Para clarificar esta problemática, considere un ejemplo real en donde se tiene una base de datos externa al sistema, conformada por información de pedidos, detalle de pedidos, productos, clientes y vendedores, así como las asociaciones entre éstas. A través del sistema, se desea conocer el nombre del producto que compró determinado cliente, el sistema debe poder determinar si la consulta que el usuario desea es válida; por lo cual, al ser una base de datos externa, de la cual se desconoce su esquema es necesario descubrir si existe una relación entre la tabla clientes y la tabla productos para poder extraer la información solicitada. Para efectos de esta clarificación, supon-

ga que la base de datos externa tiene el esquema de tablas y relaciones ilustrado en la Figura 1.

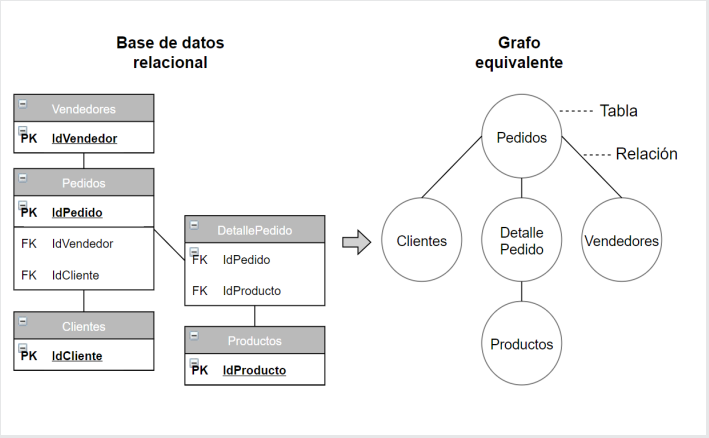

Figura 1. Mapeo de base de datos relacional a un grafo

Como se puede observar, las tablas Clientes y Productos no están relacionadas entre si de manera directa, siendo necesario encontrar el camino que permita determinar las relaciones entre ambas tablas. Como solución para esto, en este artículo se propone realizar un mapeo de la base de datos relacional a un grafo (ver Figura 1), en donde los nodos representan a las tablas y los enlaces entre los nodos las relaciones a través de la llave foránea de cada tabla. Obteniendo el grafo, se ejecuta el algoritmo de búsqueda DFS para determinar la existencia de relaciones entre las tablas involucradas y con esto construir posteriormente la consulta SQL mediante productos cartesianos que permita realizar la extracción de la información solicitada.

Hasta donde se tiene conocimiento existen pocos trabajos relacionados con el objetivo de esta investigación. La mayoría de ellos describen la transformación completa del esquema de BDR a un grafo, implicando un tiempo considerable de procesamiento y complejidad alta de implementación. Por ejemplo, en [7] se describe un método para transformar completamente la BDR con sus atributos y relaciones, a bases de datos orientada a grafos compatible con Neo4j. Este enfoque requiere del conocimiento total de la BDR y de los algoritmos para realizar la transformación de la BDR a una base de datos orientada a grafo, lo que implica un proceso no automatizado y complejo.

Por otro lado, otras soluciones describen sistemas de búsqueda por palabras clave para ejecutar consultas complejas sin necesidad de utilizar consultas SQL, como es el caso de [8], en el cual se describe un sistema de búsqueda de objetos relacionales llamado Ross, que utiliza un enfoque novedoso para encontrar las mejores rutas de unión entre relaciones y predecir relaciones entre tablas no relacionadas. En [9] se describe un sistema similar que realiza búsquedas de

palabras clave en una BDR mediante la creación de un conjunto de relaciones candidatas, las cuales se evalúa y finalmente son ejecutadas en el manejador de base de datos.

Otra estrategia es la implementación de consultas recursivas utilizando extensiones de SQL para implementar Expresiones Comunes de Tabla (CTE, por sus siglas en inglés). Sin embargo, además de requerir conocer el esquema de la base de datos, este estándar solo soporta consultas estratificadas, es decir, el cómputo recursivo en un estrato debe ser completado antes de los agregados y la negación a los resultados producidos [10].

Como se puede observar, las soluciones anteriores proponen la conversión de la BDR a una base de datos NoSQL o la utilización de algoritmos complejos o mecanismos de búsqueda mediante palabras clave lo cual implica conocimiento del esquema entidad relación de la base de datos, complejidad y mayor tiempo de desarrollo. Por el contrario, en este artículo se describe la aplicabilidad del algoritmo DFS como mecanismo de búsqueda para determinar las relaciones de una BDR y posteriormente utilizarlas para construir consultas complejas, ya que está optimizado para determinar si el camino o ruta existe o no. Es importante mencionar que, como caso de uso, este algoritmo se implementó en un sistema web de apoyo a la toma de decisiones para la comercialización de plantas ornamentales, permitiendo crear herramientas dinámicas para la realización de consultas asíncronas a la base de datos, dando soporte a la toma de decisiones mediante tablas, gráficas y reportes.

MATERIAL Y MÉTODOS

La implementación del mecanismo y del algoritmo para determinar y validar las relaciones existentes entre las tablas de una BDR, se llevó a cabo mediante el empleo del lenguaje de programación Python utilizando listas de adyacencia o diccionarios y el framework Django; además, se utilizó REST para la construcción de una interfaz de programación de aplicaciones web, permitiendo la ejecución de consultas asíncronas a la base de datos y la reducción de tiempo de procesamiento y análisis de cantidades masivas de información a través de Django REST framework.

La metodología para la determinación de las relaciones entre las tablas de una base de datos se describe en el diagrama de flujo de la Figura 2 y 3. Como se podrá observar, el algoritmo DFS se ejecutará siempre y cuando exista más de una tabla en la consulta solicitada por el usuario, determinando la existencia de caminos entre las tablas y validando la ejecución de la consulta final. Si alguno de los caminos no es válido, termina el proceso y la consulta no se ejecuta.

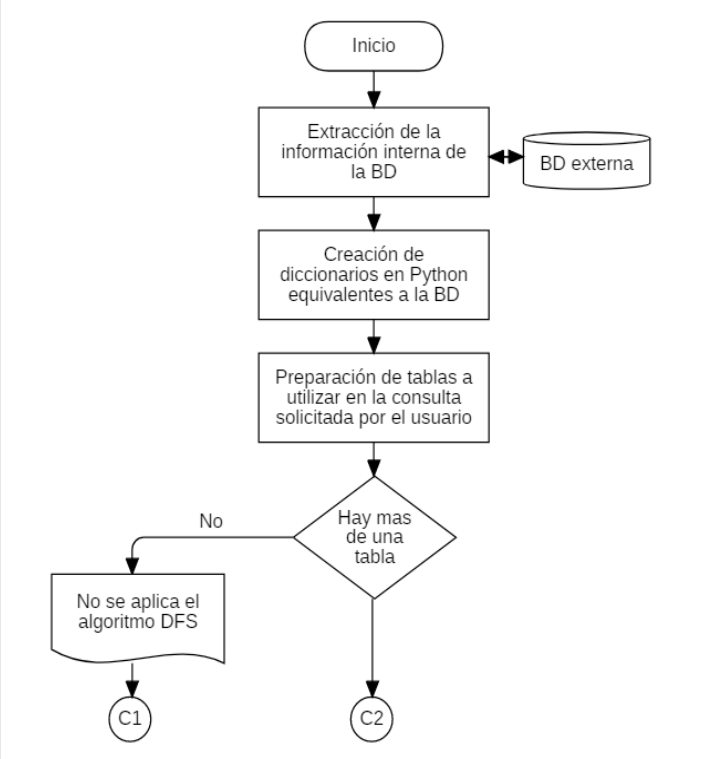

Figura 2. Diagrama de flujo para la búsqueda y validación de relaciones en BDRs con DFS.

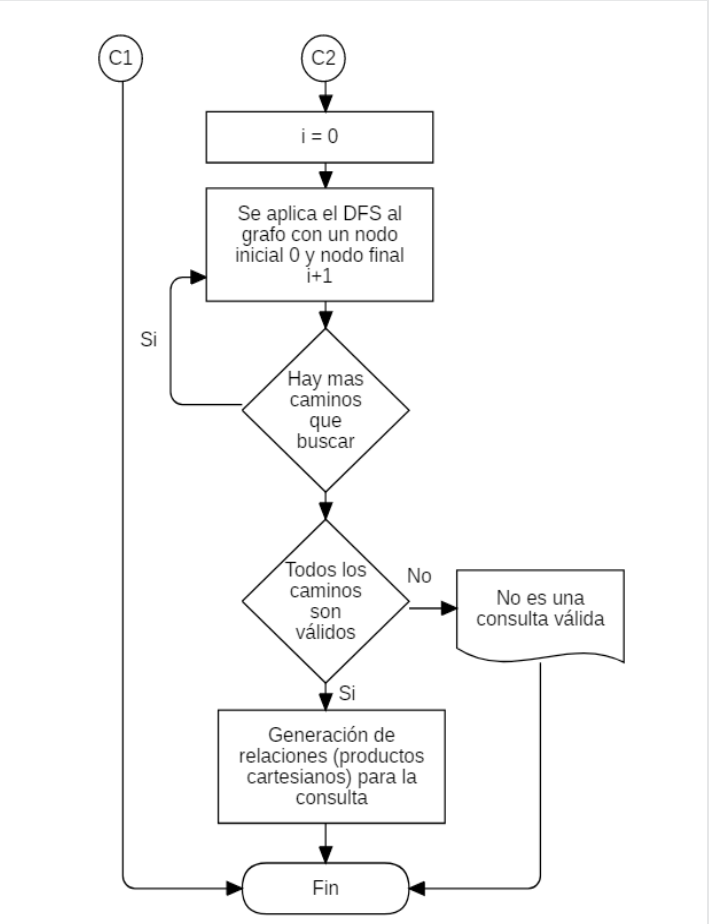

Figura 3. Continuación del diagrama de flujo para la búsqueda y validación de relaciones en BDRs con DFS.

A continuación, se describe paso a paso la metodología llevada a cabo para la determinación de relaciones e ilustrada en la Figura 2 y 3.

Extracción de información de la BD

Independientemente del lenguaje que se utilice es necesario, después de establecer conexión con la base de datos, realizar una consulta SQL a la BD para extraer la información interna (tablas y relaciones) como se ilustra en la Figura 4, debido a que se desconoce a priori la estructura de ésta.

```
"SELECT TABLE_NAME, COLUMN_NAME,
CONSTRAINT_NAME, REFERENCED_TABLE_NAME,
REFERENCED_COLUMN_NAME FROM
INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE
TABLE_SCHEMA='BASEDEDATOS'"
```

Figura 4. Consulta SQL para extracción de tablas y relaciones de una base de datos.

Como se observa en la Figura 4, se crea una consulta SQL que será de utilidad para determinar el esquema ER de la BD, donde se selecciona información como: nombres de tablas y columnas, llaves, referencias a tablas y columnas asociadas.

Creación de diccionarios equivalentes a la BDR

Como ya se mencionó en la sección anterior, es necesario conocer la información interna de la BD, y a partir de ésta, crear una representación de la BDR mediante un grafo, el cual es conjunto de nodos unidos por aristas que permiten representar relaciones entre sí. El grafo es requerido por el algoritmo DFS para buscar la ruta (relación) entre dos nodos (tablas) del grafo representando la BDR.

La representación del grafo se puede lograr a través de diversas estrategias; sin embargo, una de las estrategias más utilizadas son las listas de adyacencia. En una lista de adyacencia se mantiene una lista maestra de todos los vértices (nodos) y cada vértice en el grafo mantiene una lista de los otros vértices a los que está conectado, permitiendo encontrar fácilmente todos los enlaces (aristas) que están directamente conectados a un vértice particular, es decir las relaciones existentes entre éstos [11]. Por lo tanto, se implementó una lista de adyacencia utilizando un diccionario de Python, en donde la llave representa el nodo y el valor es un arreglo que hace referencia a los otros nodos a los que está conectado. Para comprender este proceso considere una BDR conformada por 5 tablas: pedidos, clientes, detallePedido, vendedores y productos (véase Figura 5).

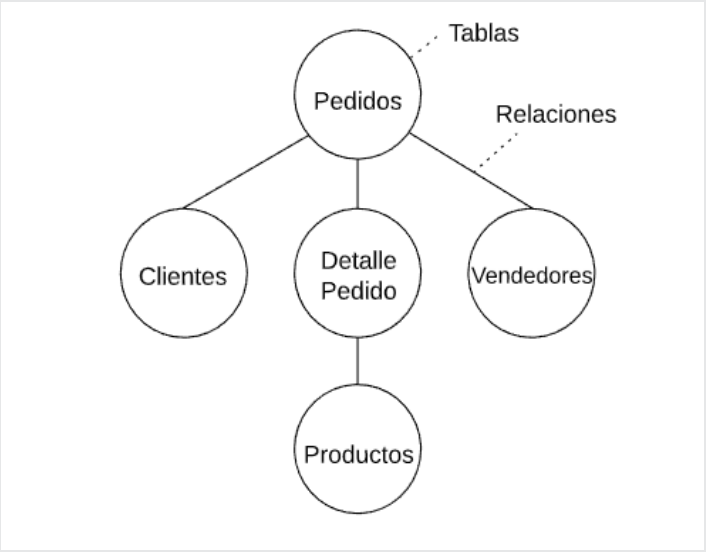

Figura 5. Representación de una base de datos en grafo.

El grafo de la BDR de la Figura 5, quedaría representado mediante una lista de adyacencia en Python utilizando diccionarios como se ilustra en la Figura 6.

```
grafo = {
    "Clientes":["Pedidos"],
    "Pedidos":["Clientes","DetallePedido",
               "Vendedores"],
    "DetallePedido":["Pedidos","Productos"],
    "Productos":["DetallePedido"],
    "Vendedores":["Pedidos"]
```

Figura 6. Representación de grafo en diccionario Python.

El diccionario presentado en la Figura 6, representa los nombres de las tablas y sus relaciones. Al final, se traslada este diccionario a un diccionario representando sus claves y valores con etiquetas numéricas en vez de nombres de tablas, como se muestra en la Figura 7.

```
grafo_num = {
    "1":["2"],
    "2":["1","3","5"],
    "3":["2","4"],
    "4":["3"],
    "5":["2"]
}
```

Figura 7. Representación de grafo con etiquetas numéricas en diccionario Python.

Como se observa en la Figura 7, se le asigna un número a cada tabla de la BD ya que, esto facilita la búsqueda con el algoritmo DFS. Sin embargo, es importante mencionar que el algoritmo DFS funciona correctamente tanto con el diccionario de la Figura 5 como con el diccionario de la Figura 6.

Una vez obtenido el grafo numérico, se seleccionan las tablas de la consulta construida por el usuario desde la vista del sistema web a priori y se aplica el algoritmo DFS siempre y cuando exista más de una tabla, ya que el algoritmo requiere de dos nodos como mínimo para buscar y regresar un camino.

Aplicación del algoritmo DFS

Para verificar si existe algún camino por el cual se pueda relacionar la información, el algoritmo DFS toma como valores de entrada los siguientes parámetros: 1) el grafo con etiquetas numéricas equivalente a la base de datos; 2) el nodo inicial (por el cual empezará la búsqueda) y el nodo final o de destino. Esto se logra en Python implementando una función que recibe 3 parámetros importantes, un grafo, un punto inicial y uno final (véase Figura 8). El algoritmo siempre toma dos tablas del array de tablas que utiliza la consulta y les asigna un punto inicial y un punto final. No importa el orden y no tienen un peso, solamente importa obtener si existe relación entre esas dos tablas.

```
def encontrar_Camino(grafo, inicial, final, camino=[]):
    camino = camino + [inicial]
    if inicial == final:
        return camino
    if inicial not in grafo:
        return None
    for nodo in str(grafo[inicial]):
        if nodo not in camino:
            nuevo_camino = encontrar_Camino(grafo,nodo,final,camino)
            if nuevo_camino: return nuevo_camino
    return None
```

Figura 8. Algoritmo DFS para encontrar el camino entre dos puntos. Fuente [12].

Como se observa en la Figura 8, se utiliza el algoritmo DFS para verificar la existencia de camino entre un punto inicial y un punto final, regresando un camino (si es que existe), el cual consiste en una serie de números del recorrido del nodo inicial hasta el nodo final.

Generación de productos cartesianos para la consulta

Una vez obtenidos los caminos válidos, el algoritmo DFS selecciona la ruta más corta con lo que se determinarán las relaciones entre las tablas involucradas de la consulta final, considerando el esquema de la BD obtenido con la consulta mostrada en la Figura 4. Posteriormente, se crean los productos cartesianos correspondientes a cada tabla. En SQL un producto cartesiano o unión cruzada es la unión de información de dos tablas donde las columnas de una tabla se unen con otra [13]. Para esto, un ciclo en Python construye los productos cartesianos requeridos, regresando un array de relaciones válidas para ejecutar la consulta que permitirá extraer la información solicitada.

Siguiendo esta metodología, cualquier sistema de información puede utilizar cualquier BDR, aun cuando se desconozca a priori la estructura de ésta, para determinar las relaciones entre sus tablas y con esto poder realizar consultas dinámicas.

RESULTADOS

Los resultados obtenidos fueron evaluados mediante la implementación del enfoque y del algoritmo planteado anteriormente en un sistema web de apoyo a la toma de decisiones para la comercialización de plantas ornamentales, siguiendo la metodología mostrada anteriormente. Para su comprensión, considere el ejemplo en donde el usuario requiere conocer el nombre del cliente y el nombre del producto que compró. Para esto, el usuario construye una consulta en el sistema web arrastrando los parámetros nombre de cliente y nombre del producto a una tabla en la vista del sistema web. Por consiguiente, la Figura 9 ilustra el grafo equivalente de la BDR.

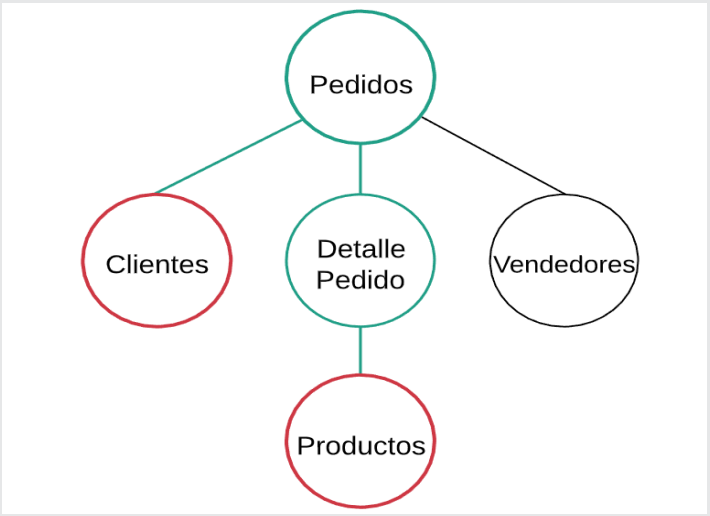

Figura 9. Tablas seleccionadas en la consulta.

Como se puede observar en la Figura 9, las tablas en rojo son las tablas requeridas para obtener información y las tablas en verde son las tablas que se encuentran en el camino entre éstas. Debido a que la tabla Clientes y la tabla Productos no están relacionadas entre sí, se debe verificar si existe algún camino por el cual se pueda relacionar la información. Para esto se utiliza el algoritmo DFS, el cual recibe el diccionario de la Figura 7 y las tablas necesarias para la consulta.

El algoritmo DFS realiza un ciclo para encontrar los caminos y al final validar las relaciones; por lo cual, en

cada iteración recibe la tabla 0 (Clientes) como punto inicial y la tabla i+1 (Productos) como punto final. Al finalizar, el algoritmo regresa una representación numérica de la ruta para llegar desde la tabla Clientes hasta la tabla Productos, siendo esta ruta: [1, 2, 3, 4]; es decir, [Clientes, Pedidos, DetallePedido, Productos]. Con esto se valida la existencia de un camino entre la tabla Clientes y Productos y se construyen las relaciones de la consulta final mediante la construcción de los productos cartesianos como se muestra en la Figura 10.

```
relaciones = [
    "DetallePedido.IDproducto=Productos.IDproducto",
    "Pedidos.IDpedido=DetallePedido.IDpedido",
    "Clientes.IDcliente=Pedidos.IDcliente"
]
```

Figura 10. Array de relaciones válidas.

Lo anterior se traduce de la siguiente manera:
1. La tabla 1 se relaciona con la 2:
Clientes.IDcliente = Pedidos.IDcliente
2. La tabla 2 se relaciona con la 3:
Pedidos.IDpedido = DetallePedido.IDpedido
3. La tabla 3 se relaciona con la 4: DetallePedido.IDproducto = Productos.IDproducto

Posteriormente, los tres productos cartesianos resultantes se integran a la consulta final del sistema web de apoyo a la toma de decisiones y se añaden a la cláusula WHERE para terminar la construcción de la consulta dinámica como se muestra en la Figura 11.

```
SELECT nombreCliente, nombreProducto
FROM Clientes, Pedidos, DetallePedido, Productos
WHERE Clientes.IDcliente = Pedidos.IDcliente AND
Pedidos.IDpedido = DetallePedido.IDpedido AND
DetallePedido.IDproducto = Productos.IDproducto
```

Figura 11. Uniones cruzadas agregadas a la consulta SLQ.

Finalmente, la consulta descrita al inicio de esta sección puede ser ejecutada debido a que el algoritmo DFS encontró las relaciones necesarias entre las tablas para que esta sea ejecutada correctamente en la BD y extraiga la información que el usuario consultó. El resultado de la ejecución de la consulta se puede observar en la Figura 12.

Figura 12. Tabla del sistema web BI, que muestra el resultado de la consulta.

Como se observa en la Figura 12, el sistema web de apoyo a la toma de decisiones para la comercialización de plantas ornamentales, para el cual se diseñó el algoritmo, despliega la consulta en una tabla con el nombre del cliente y el nombre del producto que compró, utilizando solamente los campos de la BD nombreCliente y nombreProducto. Lo anterior se logra gracias a que el algoritmo DFS permite encontrar la ruta necesaria para extraer la información de las tablas requeridas y así constituir las uniones cruzadas y validarlas; mientras que el sistema se encarga de construir el SELECT y el FROM y así ejecutar la consulta dinámica para visualizar la información necesaria.

Evaluación cualitativa

Además, como parte de los resultados, se realizó una evaluación cualitativa del método propuesto en este artículo, comparados con otros métodos tradicionales o de mayor complejidad como se observaron en los trabajos relacionados, que cumplan con el objetivo de determinar la existencia de relaciones entre tablas de una BD y construir consultas dinámicas. (véase Tabla 1).

Aportaciones

Considerando además, que las bases de datos relacionales son las bases de datos principalmente utilizadas para el desarrollo de sistemas de información, proporcionando interoperabilidad también con los sistemas ya existentes, se considera que la aplicación de este algoritmo permite reducir tiempo y complejidad en el proceso de extracción y procesamiento de la información, ya que podría ayudar a desarrollar sistemas BI para cualquier empresa que busque gestionar su información y requieran un sistema capaz de utilizar cualquier base de datos relacional y ejecutar consultas dinámicas complejas que pueda

operarse sin necesidad de conocimientos avanzados en el área, totalmente transparente para los usuarios, construyendo cualquier tipo de consultas de n número de tablas sin afectar la ejecución de la consulta o extraer información innecesaria que pueda afectar el análisis de ésta.

Tabla 1. Tabla comparativa de métodos para la determinación de relaciones en BDRs.

Método	Ventajas	Desventajas
Consultas SQL normales [14]	Permiten crear consultas SQL rápidas y específicas, utilizando tablas y relaciones conocidas.	Se requiere conocer el esquema entidad-relación para construir varias consultas SQL para extraer la información solicitada.
Expresiones de tabla comunes (CTEs) recursivas [15]	Permiten crear consultas recursivas evaluando los resultados previos, por lo que pueden utilizarse para búsqueda de caminos en un grafo.	Se requiere conocer el esquema ER y es necesario generar una vista con la información de las relaciones entre las tablas para determinar los caminos.
Procesamiento de lenguaje natural, búsqueda de objetos [8]	Permite encontrar las mejores rutas de unión entre relaciones y predecir relaciones entre tablas no relacionadas.	- Complejidad alta de implementación. - Se requiere conocer el esquema ER.
DFS y métodos utilizados en esta investigación	- DFS Permite determinar relaciones en una BDR sin conocer sus tablas y relaciones. - Permite trabajar con múltiples BDRs. - Permite la creación de consultas SQL sin conocimiento sobre el lenguaje.	- Se requiere la utilización de BDRs para generar el grafo. - Las relaciones deben de estar definidas.

CONCLUSIONES

La gran mayoría de consultas a una base de datos son simples, ya que estas se realizan en aplicaciones que se basan en cuatro operaciones básicas como: crear, obtener, actualizar y eliminar; sin embargo, existen escenarios que requieren consultas analíticas y complejas, en done el poder de SQL es limitado. Por lo tanto, el enfoque propuesto mediante la utilización del algoritmo DFS permite determinar las relaciones existentes entre tablas de una base de datos aun cuando éstas no estén relacionadas directamnte, siendo este enfoque de gran utilidad para desarrollar sistemas de información eficientes ya que, posibilita la utilización de cualquier base de datos aún si se desconoce su esquema entidad-relación. El algoritmo se aplicó a un sistema web de apoyo a la toma de decisiones, el cual cuenta con una BDR histórica que contiene información de las ventas de plantas ornamentales. Con la aplicación del algoritmo DFS, se obtuvieron las relaciones de la BD, permitiendo construir y ejecutar consultas dinámicas a la BD histórica, dando un resultado positivo para los operadores del sistema ya que les permitía crear consultas complejas sin necesidad de conocimientos avanzados en SQL. La implementación de este enfoque impacta en la habilidad del sistema para responder eficientemente a las consultas de los usuarios. Se explicó el enfoque utilizando un caso de uso muy sencillo; sin embargo, con los datos reales del sistema web al cual se integró se observa la eficiencia y la mejora en el proceso de construcción de consultas complejas.

Como trabajo futuro se espera comparar el funcionamiento del algoritmo utilizando la sentencia JOIN de SQL para construir las relaciones y mejorar el funcionamiento del algoritmo prediciendo relaciones en bases de datos que no contengan relaciones bien estructuradas ya que una de las limitantes es que la BD debe de ser relacional y además contener relaciones bien estructuradas.

AGRADECIMIENTOS

Al Tecnológico Nacional de México campus Instituto Tecnológico de Colima y al Consejo Nacional de Ciencia y Tecnología (CONACYT).

BIBLIOGRAFÍA

[1] Raut, D. A. B. (2017). NOSQL Database and Its Comparison with RDBMS. International Journal of Computational Intelligence Research, 13(7), 1645-1651.

[2] Kunda, D., & Phiri, H. (2017). A Comparative Study of NoSQL and Relational Database. Zambia ICT Journal, 1(1), 1-4. https://doi.org/10.33260/zictjournal.v1i1.8

[3] Aftab, Z., Iqbal, W., Almustafa, K. M., Bukhari, F., & Abdullah, M. (2020). Automatic NoSQL to Relational Database Transformation with Dynamic Schema Mapping. Scientific Programming, 2020, 1-13. https://doi.org/10.1155/2020/8813350

[4] Pacaci, A., Zhou, A., Lin, J., & Özsu, M. T. (2017). Do We Need Specialized Graph Databases?: Benchmarking Real-Time Social Networking Applications. Proceedings of the Fifth International Workshop on Graph Data-Management Experiences & Systems-GRADES'17, 1-7. https://doi.org/10.1145/3078447.3078459

[5] Aftab, Z., Iqbal, W., Almustafa, K. M., Bukhari, F., & Abdullah, M. (2020). Automatic NoSQL to Relational Database Transformation with Dynamic Schema Mapping. Scientific Programming, 2020(1), 1–13. https://doi.org/10.1155/2020/8813350

[6] Chandel, A., & Sood, M. (2014). Searching and Optimization Techniques in Artificial Intelligence: A Comparative Study & Complexity Analysis. In-

ternational Journal of Advanced Research in Computer Engineering & Technology 3(3), 6.

[7] Bordoloi, S., & Kalita, B. (2013). Designing Graph Database Models from Existing Relational Databases. International Journal of Computer Applications, 74(1), 25-31. https://doi.org/10.5120/12850-9303

[8] Yin, X., Han, J., & Yang, J. (2005). Searching for Related Objects in Relational Databases. Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM. 12-2005, 227–236.

[9] Hristidis, V., & Papakonstantinou, Y. (2002). DISCOVER: Keyword Search in Relational Databases. Proceedings of the 28th VLDB Conference, 670–681. https://doi.org/10.1016/B978-155860869-6/50065-2

[10] Wang, J., Xiao, G., Gu, J., Wu, J., & Zaniolo, C. (2020). RASQL: A Powerful Language and its System for Big Data Applications. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2673–2676. https://doi.org/10.1145/3318464.3384677

[11] Miller, B. W., & Ranum, D. L. (2013). Problem Solving with Algorithms and Data Structures using Python (2a ed.). Franklin Beedle & Associates.

[12] Python Software Foundation. (2019). Python Patterns-Implementing Graphs. Obtenida el 29 de Abril de 2020, de la página electrónica: https://www.python.org/doc/essays/graphs/

[13] w3resource. (2020). SQL Cross Join. Obtenida el 29 de Abril de 2020, de la página electrónica: https://www.w3resource.com/sql/joins/cross-join.php

[14] Dobson, R. (2017). List Dependencies for SQL Server Foreign Keys [Artículo]. Mssqltips. Obtenida el 31 de Julio de 2020, de la página electrónica https://www.mssqltips.com/sqlservertip/4753/list-dependencies-for-sql-server-foreign-keys/

[15] Kołodziejski, M. (2019). Get to Know the Power of SQL Recursive Queries [Artículo]. Learn SQL. Obtenida el 31 de Julio de 2020, de la página electrónica https://learnsql.com/blog/get-to-know-the-power-of-sql-recursive-queries/