Kỷ yếu Hội nghị Quốc gia lần thứ VIII về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR); Hà Nội, ngày 9-10/7/2015
DOI: 10.15625/vap.2015.000217

VIẾT LẠI TRUY VẤN ĐỂ SỬ DỤNG KHUNG NHÌN THỰC CÓ HÀM THỐNG KÊ TRONG POSTGRESQL

Nguyễn Trần Quốc Vinh
Trường Đại học Sư phạm, Đại học Đà Nẵng
ntquocvinh@ued.vn

TÓM TẮT - *Khung nhìn thực là kết quả thực thi truy vấn được lưu lại trong cơ sở dữ liệu. Hệ quản trị cơ sở dữ liệu có thể sử dụng khung nhìn thực với số lượng bản ghi nhỏ chứa kết quả có sẵn để trả lời các truy vấn một cách nhanh chóng, thay vì đọc dữ liệu từ các bảng gốc và xử lý phức tạp trên lượng lớn dữ liệu. Công nghệ khung nhìn thực đã được triển khai trong các hệ quản trị cơ sở dữ liệu thương mại (Oracle, DB2, SQL Server). Từ phiên bản 9.3 và hiện nay là phiên bản 9.4, PostgreSQL hỗ trợ lệnh tạo khung nhìn thực và cập nhật toàn phần bất đồng bộ khung nhìn thực. Tuy nhiên, PostgreSQL chưa hỗ trợ khai thác khung nhìn thực một cách tự động. Tác giả nghiên cứu xây dựng, tích hợp và đánh giá mô-đun viết lại truy vấn để khai thác khung nhìn thực trên cơ sở truy vấn nối trong có hàm thống kê một cách thông minh trong PostgreSQL. Kết quả thử nghiệm cho thấy hiệu quả khi viết lại truy vấn để sử dụng khung nhìn thực - tăng tốc độ thực thi của truy vấn lên nhiều lần, đặc biệt là các truy vấn phức tạp sử dụng lượng dữ liệu lớn.*

Từ khóa *- Khung nhìn thực; hàm thống kê; PostgreSQL; xử lý truy vấn; viết lại truy vấn; can thiệp mã nguồn.*

I. ĐẶT VẤN ĐỀ

Quy mô hoạt động quản lý ngày càng được mở rộng nhanh chóng, kéo theo lượng dữ liệu phải xử lý và độ phức tạp trong truy vấn ngày càng cao. Nhiều cơ sở dữ liệu (CSDL) với dung lượng hàng terabytes, yêu cầu xử lý thông tin ngày càng phức tạp nhưng đòi hỏi phải nhanh chóng, chính xác, thậm chí phải đáp ứng tức thời các yêu cầu trong thời gian thực. Việc thực thi một truy vấn phức tạp trên lượng dữ liệu lớn từ CSDL thường yêu cầu chi phí lớn tài nguyên để thực hiện, kể cả thời gian. Điều đó làm ảnh hưởng đến việc ra quyết định, cũng như hiệu quả hoạt động của một tổ chức. Vấn đề này đặt ra bài toán, làm thế nào để tăng tốc độ thực thi truy vấn. Trong phạm vi bài viết này, tác giả đề cập đến phương pháp ứng dụng công nghệ khung nhìn thực (KNT, materialized view) để tăng tốc độ thực thi truy vấn.

Khung nhìn (ảo) đại diện cho một truy vấn và được sử dụng giống như một bảng. Khi truy cập vào khung nhìn, truy vấn đứng phía sau sẽ được thực thi. Ý tưởng ứng dụng KNT – kết quả thực thi của các truy vấn được giữ lại trong CSDL, xuất hiện từ những năm 80 của thế kỷ trước, nhưng KNT chỉ được triển khai thực tế từ năm 2000 trong ba hệ quản trị (HQT) CSDL thương mại Oracle, MS SQL Server, IBM DB2. Trong Oracle, KNT được gọi là "materialized views" và được phân làm ba loại - read only, updateable và writeable [1]. Trong IBM DB2, KNT được gọi là bảng thực hoá truy vấn (materialized query tables, MQT) và có hai loại - MQT được duy trì bởi hệ thống và MQT được duy trì bởi người dùng. Microsoft SQL Server có công nghệ tương tự gọi là khung nhìn chỉ mục hoá (indexed views). KNT được tạo ra với ý tưởng ban đầu là một công cụ hỗ trợ cho các kho dữ liệu và các hệ thống hỗ trợ ra quyết định. Tuy nhiên, nó có thể được ứng dụng cho bất kỳ CSDL nào [2]. Ứng dụng KNT là công nghệ mới đang đặt ra nhiều vấn đề cần nghiên cứu. Trong đó, nhiều công trình nghiên cứu đã công bố liên quan đến thuật toán cập nhật KNT [2-4]. Công trình [5] thực hiện sinh tự động mã nguồn trong ngôn ngữ C của các bẫy sự kiện (trigger) cho các sự kiện thêm, cập nhật, xoá dữ liệu trên tất cả các bảng gốc tham gia vào truy vấn tạo KNT. Các bẫy sự kiện đó triển khai các thuật toán thực hiện cập nhật gia tăng, đồng bộ KNT.

Công trình [6] thực hiện xây dựng mô-đun viết lại truy vấn hỗ trợ KNT và việc sử dụng KNT trong HQT CSDL PostgreSQL. Tuy nhiên, nghiên cứu [6] còn nhiều hạn chế. Chẳng hạn, i) mô tả cách thức viết lại truy vấn còn chung chung cho một vài trường hợp truy vấn đơn giản; ii) cách thức so sánh truy vấn và xử lý truy vấn để tìm kiếm khả năng sử dụng KNT thô sơ. Một số công trình đã công bố có nói về cách thức so sánh truy vấn và lấy KNT để trả lời truy vấn cho một số dạng biểu thức ở mức độ truy vấn SQL dưới dạng văn bản [4; 7-9]. Một số công trình khác [4; 7; 10] cũng nghiên cứu về cách thức sử dụng KNT để trả lời truy vấn, nhưng cũng chỉ dưới dạng ý tưởng chung cho một số dạng đơn giản và cũng chủ yếu nghiên cứu thuật toán cập nhật gia tăng.

PostgreSQL là HQT CSDL mã nguồn mở hàng đầu, được sử dụng rộng rãi trên thế giới và được khuyến cáo bởi Bộ Thông tin và Truyền thông Việt Nam (Thông tư 41/2009/TT-BTTTT). Từ phiên bản 9.3 và hiện tại là phiên bản 9.4, PostgreSQL hỗ trợ các lệnh tạo KNT (CREATE MATERIALIZED VIEW) và cập nhật bất đồng bộ theo cách thực thi lại truy vấn và thay thế toàn bộ nội dung đang có trong bảng KNT (REFRESH MATERIALIZED VIEW). Khiếm khuyết lớn nhất đó là chưa có tính năng viết lại truy vấn để cho phép khai thác KNT [11] một cách thông minh. Ở đây, ta đề cập đến tính năng phát hiện khả năng biến đổi truy vấn tương đương để sử dụng toàn bộ hoặc một phần KNT để trả lời truy vấn. Truy vấn sẽ lấy kết quả từ KNT thay vì lấy dữ liệu từ các bảng gốc và xử lý. Điều này giúp tăng tốc độ thực thi truy vấn phức tạp trên lượng dữ liệu lớn, giúp nâng cao hiệu suất hoạt động của cả hệ thống, nâng cao hiệu quả thực thi của các truy vấn phức tạp trên cơ sở dữ liệu lớn trong PostgreSQL.

Trong bài viết này, tác giả nghiên cứu xây dựng và tích hợp mô-đun viết lại truy hỗ trợ KNT vào mã nguồn của PostgreSQL nhằm khai thác KNT trên PostgreSQL một cách hiệu quả. Truy vấn quan tâm bao gồm phép nối trong, phép gộp nhóm và các hàm thống kê (aggregate functions: SUM, COUNT, AVG, MIN, MAX); không bao gồm truy vấn lồng, phép nối ngoài và truy vấn đệ quy. Nghiên cứu này xét đến cả khả năng kết quả truy vấn người dùng có thể được tính hoàn toàn từ KNT và khả năng kết quả thực thi truy vấn người dùng chứa kết quả thực thi truy vấn tạo KNT. Khi đó, để trả lời truy vấn người dùng, HQT CSDL phải nối bảng KNT với các bảng khác. Ngoài ra, nghiên cứu này còn khắc phục các nhược điểm của công trình [6] và triển khai trong thực tiễn, đặc biệt trong cách thức xử lý và so sánh truy vấn để tìm kiếm khả năng sử dụng KNT.

II. VIẾT LẠI TRUY VẤN

KNT là kết quả truy vấn được giữ lại trong CSDL dưới dạng bảng. Nếu truy vấn người dùng nhập vào được viết lại hướng qua truy vấn tạo KNT, thì có thể lấy kết quả từ KNT, thay vì lấy dữ liệu từ các bảng gốc và xử lý. Có vô số mẫu truy vấn khác nhau. Tuy nhiên, bài viết này chỉ xem xét một số mẫu truy vấn và sử dụng KNT để trả lời các truy vấn đó. Để đơn giản, ta ký hiệu truy vấn tạo KNT là Q^M và truy vấn do người dùng gửi đến HQT CSDL là Q^U. Truy vấn viết lại để sử dụng KNT được ký hiệu là Q^R. Tất nhiên, Q^R phải tương đương Q^U.

Truy vấn Q^x bao gồm các hàm thống kê có thể được biểu diễn như sau: $Q^x = (C^x, A^x, LC^x, LA^x, F^x, J^x, W^x, G^x)$. Trong đó:

- $S^x = C^x \cup A^x = \{S_1^x, S_2^x, .., S_m^x\}$ - tập các cột/biểu thức được lựa chọn trong mệnh đề SELECT. $C^x = \{C_1^x, C_2^x, .., C_p^x\}$ là tập hợp các cột của các bảng trong mệnh đề SELECT. $A^x = \{A_1^x, A_2^x, .., A_q^x\}$ là tập hợp các hàm thống kê với biểu thức (E) trên các cột từ bảng gốc như SUM(E), COUNT(E), MIN(E) và MAX(E). Để phục vụ quá trình cập nhập gia tăng đồng bộ KNT cũng như tăng khả năng sử dụng KNT sau này, AVG(E) tự động được chuyển thành SUM(E) và COUNT(E). E không chứa các hàm thống kê. $LA^x = \{LA_1^x, LA_2^x, .., LA_q^x\}$ là tập hợp các bí danh (alias) của các biểu thức tương ứng trong A^x; LA_i^x là bí danh của A_i^x. $LC^x = \{LC_1^x, LC_2^x, .., LC_q^x\}$ là tập hợp các bí danh của các biểu thức tương ứng trong C^x; LC_i^x là bí danh của C_i^x. Mặc định C_i^x có dạng $T_j^x.LC_i^x$ hoặc $LC_i^x - LC_i^x$ trùng với tên cột trong C_i^x. Tập hợp các cột trong bảng KNT Tmv chính là $LC^x \cup LA^x$.

- F^x- mệnh đề FROM. Mệnh đề FROM là sự kết hợp của tập các bảng gốc $T^x = \{T_1^x, T_2^x, .., T_n^x\}$ được sử dụng trong truy vấn và J^x - tập hợp các điều kiện của các phép nối giữa các bảng trong T^x.

- W^x - mệnh đề WHERE, điều kiện chọn lựa bản ghi để xử lý. Trong trường hợp truy vấn bao gồm phép nối tường minh, J^x không rỗng. Ngược lại, J^x rỗng và W^x bao gồm cả J^x.

- $G^x = \{G_1^x, G_2^x, .., G_k^x\}$ - tập các cột/biểu thức gộp nhóm mệnh đề GROUP BY. Mặc định đã có sự biến đổi truy vấn tạo KNT trong quá trình tạo KNT để kết quả bao gồm cả các biểu thức trong mệnh đề GROUP BY; nghĩa là, S^x tự thân đã bao gồm G^x.

Với truy vấn không bao gồm hàm thống kê, S^x không chứa hàm thống kê, A^x, L^x và G^x – rỗng. Tương ứng, ta có truy vấn tạo KNT $Q^M = (C^M, A^M, LC^M, LA^M, F^M, J^M, W^M, G^M)$, truy vấn của của người dùng $Q^U = (C^U, A^U, LC^U, LA^U, F^U, J^U, W^U, G^U)$ và truy vấn viết lại $Q^R = (C^R, A^R, LC^R, LA^R, T^R, J^R, W^R, G^R)$. Bảng KNT Tmv bao gồm các cột $C^M \cup L^M$. Nghiên cứu quan tâm đến các dạng truy vấn và viết lại truy vấn theo mức độ phức tạp từ thấp đến cao.

A. Q^U có thể được tính hoàn toàn từ Q^M

Ở đây ta xét trường hợp $S_i^U = f(S^M)$, $T^U = T^M, J^U = J^M, W^U = W^M$ và $G^U \subseteq G^M$. $f(S^M)$ là biểu thức đại số trên các phần tử của S^M. Nghĩa là, S_i^U có thể trùng với một phần tử nào đó của S^M, cũng có thể được tính thông qua các toán tử cộng, trừ, nhân, chia đại số trên các S_j^M.

1. Trường hợp: $S^U \subseteq S^M$ ($C^U \subseteq C^M$, $A^U \subseteq A^M$) và $G^U \subseteq G^M$.

Trước tiên, xét trường hợp $S^U \subseteq S^M$ ($C^U \subseteq C^M$, $A^U = A^M = \{\text{SUM(E)}, \text{COUNT(E)}, \text{MIN(E)}, \text{MAX(E)}\}$) và $LA^M = \{\text{sum}, \text{count}, \text{min}, \text{max}\}$. Với E là biểu thức đại số trên các cột mà ít nhất một trong số đó không thuộc G^M.

Gọi r^M là kết quả phép nối tất cả các bảng trong T^M với điều kiện nối J^M và áp dụng điều kiện W^M. Tương ứng, có r^U, T^U, J^U và W^U. Vì $T^U = T^M, J^U = J^M, W^U = W^M$ nên $r^U = r^M$. Vì $G^U \subseteq G^M$, mỗi bản ghi $r_l^M(r_l^U)$ thuộc nhóm thứ $y - G^{M_y}$ trên r^U (r^M) thì cũng thuộc nhóm thứ $z - G^{U_z}$ tương ứng ($G^{U_z} \subseteq G^{M_y}$), các biểu thức trong G^U trùng với các biểu thức trong G^M có giá trị bằng nhau theo từng cặp. Các bản ghi thuộc nhóm theo G^{U_z} tạo thành h nhóm theo G^{M_y}.

Xét SUM(E) có mặt trong cả A^U và A^M, tương ứng là A_i^U và A_j^M. Trong Tmv có cột sum chứa kết quả SUM(E) theo G^M – bộ giá trị (G^M, sum). Mỗi nhóm theo G^{U_z} có h giá trị SUM(E) theo G^{M_y}. Vậy, tổng của h giá trị SUM(E) theo G^{M_y} chính là SUM(E) theo G^{U_z}. Nói cách khác, (G^U, SUM(sum)) trên Tmv chính là (G^U, SUM(E)) trên r^U (r^M).

Xét COUNT(E) có mặt trong cả A^U và A^M, tương ứng là A_i^U và A_j^M. Trong Tmv có cột count chứa kết quả COUNT(E) theo G^M – bộ giá trị (G^M, count). Mỗi nhóm theo G^{U_z} có h giá trị COUNT(E) theo G^{M_y}. Vậy, tổng của h

giá trị COUNT(E) theo G^{M_y} chính là COUNT(E) theo G^{U_z}. Nói cách khác, (G^U, SUM(count)) trên Tmv chính là (G^U, COUNT(E)) trên r^U (r^M).

Xét MIN(E), MAX(E) có mặt trong cả A^U và A^M, tương ứng là A_i^U và A_j^M. Trong Tmv có cột min chứa kết quả MIN(E) theo G^M – bộ giá trị (G^M, min). Mỗi nhóm theo G^{U_z} có h giá trị MIN(E) theo G^{M_y}. Vậy, giá trị nhỏ nhất trong số h giá trị MIN(E) theo G^{M_y} chính là MIN(E) theo G^{U_z}. Nói cách khác, (G^U, MIN(min)) trên Tmv chính là (G^U, MIN(E)) trên r^U (r^M). Tương tự, (G^U, MAX(max)) trên Tmv chính là (G^U, MAX(E)) trên r^U (r^M).

Vậy, với $S^U \subseteq S^M$ ($C^U \subseteq C^M$, $A^U = A^M = \{\text{SUM(E)}, \text{COUNT(E)}, \text{MIN(E)}, \text{MAX(E)}\}$) và $G^U \subseteq G^M$, $Q^R = (LC^U, \{\text{SUM(sum)}, \text{SUM(count)}, \text{MIN(min)}, \text{MAX(max)}\}, \phi, LA^U, \{\text{Tmv}\}, \phi, \phi, G^U)$.

$S^U \subset S^M$ ($C^U \subset C^M$ và/hoặc $A^U \subset A^M$) là trường hợp riêng của $S^U \subseteq S^M$. Truy vấn viết lại sẽ là: $Q^{R2} = (C^{R2} \subseteq C^R, A^{R2} \subseteq A^R, L^U, \{\text{Tmv}\}, \phi, \phi, G^U)$.

Nếu $S^U = S^M$ ($C^U = C^M$, $A^U = A^M$) và $G^U = G^M$, Q^U tương đương với Q^M. Truy vấn viết lại trong trường hợp này sẽ là: $Q^R = (LC^U, L^M, \phi, LA^U, \{\text{Tmv}\}, \phi, \phi, \phi)$.

2. Xét trường hợp $A^U = A^M = \{\text{SUM(E)}, \text{COUNT(E)}, \text{MIN(E)}, \text{MAX(E)}\}$

Với $L^M = \{\text{sum}, \text{count}, \text{min}, \text{max}\}$, E là biểu thức đại số trên các cột thuộc G^M. Tất cả các cột tham gia vào E đều có trong Tmv. Vì $G^U \subseteq G^M$, mỗi bản ghi $r_l^M(r_l^U)$ thuộc nhóm thứ $y - G^{M_y}$ trên r^U (r^M) thì cũng thuộc nhóm thứ $z - G^{U_z}$ tương ứng, các biểu thức trong G^U trùng với các biểu thức trong G^M có giá trị bằng nhau theo từng cặp. Các bản ghi thuộc nhóm theo G^{U_z} tạo thành h nhóm theo G^{M_y}.

Xét A_i^U = "SUM(E)". Mỗi bản ghi thứ y trong Tmv đại diện cho nhóm thứ $y - G^{M_y}$ bao gồm count bản ghi thuộc r^U (r^M). Nếu xét trên r^U (r^M), bộ giá trị tương ứng các cột trong G^{M_y} sẽ được lặp lại count lần. Mỗi nhóm theo G^{U_z} có h giá trị tích E*count theo G_y^M. Tổng của h giá trị tích E*count theo G^{M_y} chính là SUM(E) theo G^{U_z}. (G^U, SUM(E*count)) trên Tmv chính là (G^U, SUM(E)) trên r^U (r^M).

Với A_i^U = "COUNT(E)", (G^U, SUM(count)) trên Tmv chính là (G^U, COUNT(E)) trên r^U (r^M).

Với A_i^U = "MIN(E)", (G^U, MIN(E)) trên Tmv chính là (G^U, MIN(E)) trên r^U (r^M).

Với A_i^U = "MAX(E)", (G^U, MAX(E)) trên Tmv chính là (G^U, MAX(E)) trên r^U (r^M).

Vậy, $Q^R = (LC^U, \{\text{SUM(E} * \text{count)}, \text{SUM(count)}, \text{MIN(E)}, \text{MAX(E)}\}, \phi, LA^U, \{\text{Tmv}\}, \phi, \phi, G^U)$.

3. Trường hợp $S_i^U = f(S^M)$ và $G^n \subseteq G^k$

Cần chú ý rằng, trong quá trình biến đổi Q^M để tạo KNT, biểu thức đại số trên hàm thống kê $f(S^M)$ đã được phân tách thành các thành phần và được lưu trữ riêng lẻ theo các biểu thức trong mệnh đề S^M. Kết hợp hai trường hợp a) và b) cho thấy $f(S^M)$ có thể được tính một cách dễ dàng.

B. Kết quả thực thi Q^U chứa kết quả thực thi Q^M

Nghiên cứu giới hạn các truy vấn thoả mãn điều kiện $C^M \subset C^U$, $A^U \subseteq A^M$, $T^M \subseteq T^U$, $J^M \subseteq J^U$, $W^M = W^U$, $G^M \subset G^U$, các cột thuộc các bảng trong T^M tham gia vào phép nối giữa $T^U \backslash T^M$ và T^M có mặt trong C^M và không tạo thành khoá trong các bảng đó. Ý tưởng chung là hướng tới các truy vấn, mà ở đó phần liên quan gộp nhóm có thể được tách thành một truy vấn lồng tham gia vào truy vấn toàn cục.

Xem xét ví dụ KNT mv2 trong bảng 1 với Q^M đưa ra danh sách khách hàng tổng số tiền và tổng số hàng đã mua: SELECT sales.cust_id, sum(quantity_sold*unit_cost) as tongtien, sum(sales.quantity_sold) as tongban FROM sales, costs WHERE sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY sales.cust_id. Bảng KNT là mv2(cust_id, tongtien, tongban).

Truy vấn Q^U đưa ra danh sách khách hàng tổng số tiền và tổng số hàng đã mua, có thể hiện thông tin khách hàng như họ tên, quốc gia: SELECT countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name, SUM(quantity_sold*unit_price) AS tongtien, sum(sales.quantity_sold) as tongban FROM countries, customers, sales, costs WHERE countries.country_id = customers.country_id AND customers.cust_id = sales.cust_id AND sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name.

Truy vấn này có thể được viết lại dưới dạng sử dụng truy vấn lồng: SELECT countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name, tongtien, tongban FROM countries, customers, (*SELECT sales.cust_id, SUM(quantity_sold*unit_price) AS tongtien, sum(sales.quantity_sold) as tongban FROM sales, costs WHERE sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY sales.cust_id*) AS Tmv WHERE countries.country_id = customers.country_id AND customers.cust_id = Tmv.cust_id. Có thể tạo KNT Tmv cho truy vấn lồng, lúc đó, có thể viết lại Q^U thành Q^R: SELECT countries.country_id, country_name,

customers.cust_id, cust_first_name, cust_last_name, tongtien, tongban FROM countries, customers, Tmv WHERE countries.country_id = customers.country_id AND customers.cust_id = Tmv.cust_id.

Thông thường, HQT CSDL thực thi truy vấn Q^U trên theo các bước theo thứ tự: i) Nối các bảng countries, customers, sales và costs với nhau; ii) thực hiện gộp nhóm theo countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name; iii) thực hiện các biểu thức trong mệnh đề SELECT cho mỗi nhóm. Tuy nhiên, với truy vấn Q^U này thì thứ tự các bước thực thi như sau vẫn cho kết quả đúng: i) Nối các bảng sales, costs; ii) thực hiện gộp nhóm theo cust_id; iii) tính cust_id và SUM(quantity_sold*unit_price), sum(sales.quantity_sold) cho mỗi nhóm – thu được bảng kết quả tạm Tmv; iv) nối Tmv với countries và customers; v) thực hiện mệnh đề SELECT countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name, total. Tmv ở đây chính là bảng KNT Tmv.

Xét trường hợp thứ nhất:

- $C^M \subset C^U, A^U \subseteq A^M, T^M = T^U, J^M = J^U, W^M = W^U, G^M \subset G^U$;

- Ít nhất một khoá của mỗi bảng trong T^1 có mặt trong C^M. T^1 – tập hợp tất cả các bảng có cột có mặt trong $C^U \backslash C^M$. J^1 – tập hợp các điều kiện nối giữa Tmv và T^1 trên các khoá của các bảng trong T^1. Điều này đảm bảo mỗi bản ghi trong Tmv tương ứng với một bản ghi duy nhất trong các bảng trong T^1 và ngược lại.

Khi đó, $Q^R = (C^U \backslash C^M \cup LC^M, \phi, LA^R, \phi, \{\text{Tmv}\} \cup T^1, J^1, \phi, \phi)$. Với $LA^R = \{LA_i^M | A_i^M \in A^U \cap A^M\}$.

Trường hợp thứ hai:

- $C^M \subset C^U$; $A^U \subseteq A^M$; $T^M \subset T^U$; $W^M = W^U$; $G^M \subset G^U$; $J^M \subset J^U$ (cho tất cả các bảng thuộc T^M);

- J^2 – tập hợp các điều kiện nối giữa $T^U \backslash T^M$ và T^M: các cột thuộc các bảng trong T^M tham gia vào phép nối giữa $T^U \backslash T^M$ và T^M có mặt trong C^M và không tạo thành khoá trong các bảng đó. Điều kiện này đảm bảo: thứ nhất, đảm bảo nối được Tmv và $T^U \backslash T^M$; thứ hai, mỗi bản ghi trong kết quả phép nối các bảng trong T^M tương ứng với duy nhất một bản ghi (nếu có) với các bảng trong $T^U \backslash T^M$. Khi sự tương ứng giữa các bản ghi trong kết quả phép nối các bảng trong T^M và các bản ghi thuộc các bảng trong $T^U \backslash T^M$ là nhiều – một, nhóm theo các cột trong T^M là đủ, nhóm theo các cột trong $T^U \backslash T^M$ sau đó không làm thay đổi các nhóm bản ghi.

Khi đó, $Q^R = (C^U \backslash C^M \cup LC^M, \phi, LA^R, \phi, \{\text{Tmv}\} \cup (T^U \backslash T^M), J^2, \phi, \phi)$, với $LA^R = \{LA_i^M | A_i^M \in A^U \cap A^M\}$.

Từ hai trường hợp này có thể suy ra Q^R cho trường hợp chung, đó là $C^M \subset C^U$, $A^U \subseteq A^M$, $T^M \subseteq T^U$, $J^M \subseteq J^U$, $W^M = W^U$, $G^M \subset G^U$, các cột thuộc các bảng trong T^M tham gia vào phép nối giữa $T^U \backslash T^M$ và T^M có mặt trong C^M và không tạo thành khoá trong các bảng đó: $Q^R = (C^U \backslash C^M \cup LC^M, \phi, LA^R, \phi, \{\text{Tmv}\} \cup T^1 \cup (T^U \backslash T^M), J^1 \cup J^2, \phi, \phi)$.

III.XÂY DỰNG VÀ TÍCH HỢP MÔ ĐUN VIẾT LẠI TRUY VẤN TRONG POSTGRESQL

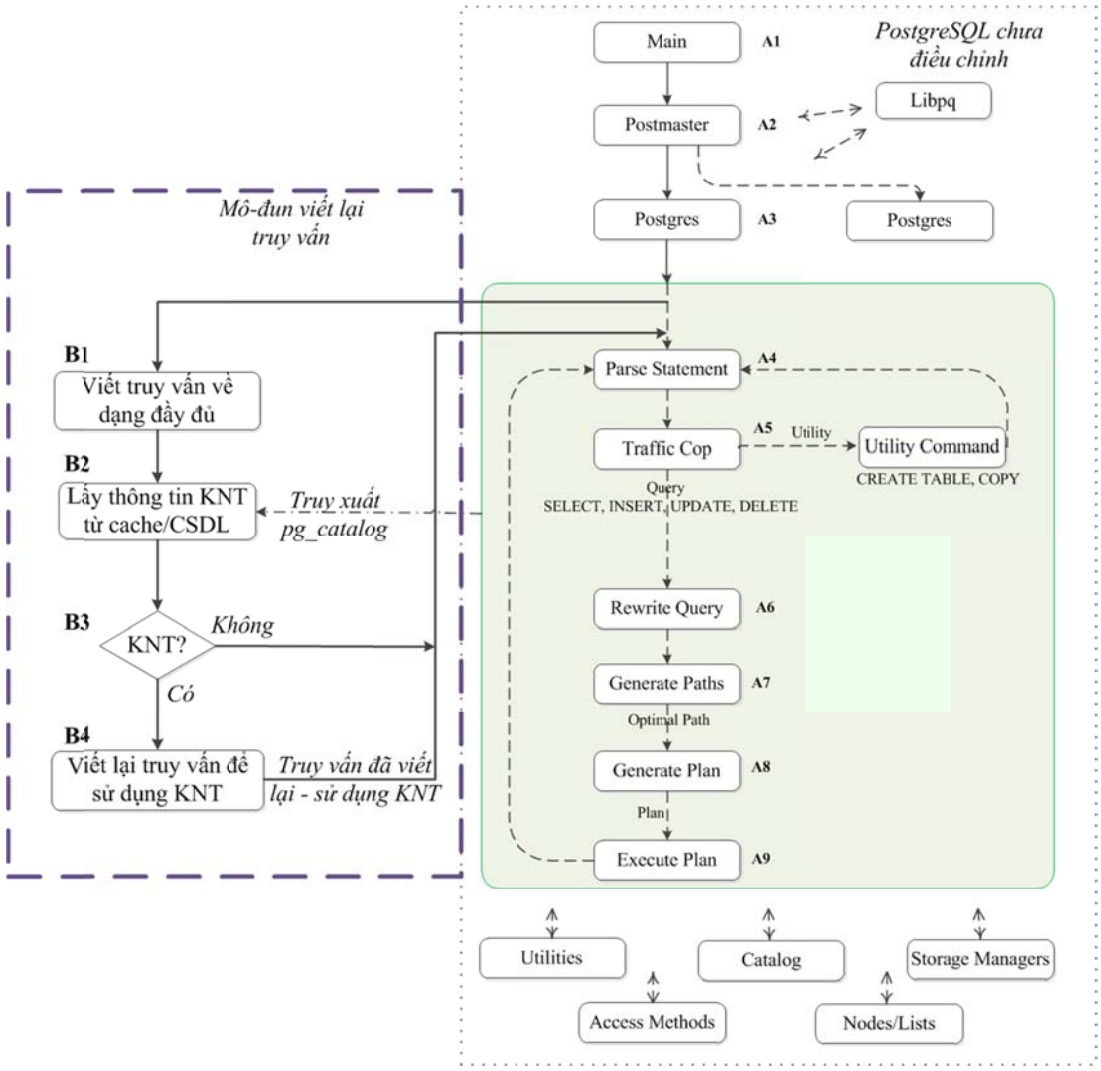
A. Nghiên cứu mã nguồn của PostgreSQL

Để xây dựng được mô-đun, tác giả tìm hiểu về quá trình xử lý truy vấn bên trong mã nguồn của PostgreSQL (xem khối "PostgreSQL chưa điều chỉnh" trên hình 1). Các hàm Postgres nhận các truy vấn từ máy khách, gọi các hàm xử lý và trả về kết quả cho máy khách. Posgres tương tác với máy khách thông qua các hàm libpq. Ban đầu, chuỗi truy vấn sẽ đi vào ở vị trí Postgres (A3). Sau đó, chuỗi được phân tích cú pháp (Parse Statement, A4) bằng công cụ lex và yacc, trả về một cây truy vấn (Query Tree). Cây truy vấn được truyền đến các hàm viết lại truy vấn (Rewrite Query, A6) để viết lại truy vấn theo các luật của hệ thống (không có luật viết lại truy vấn để hỗ trợ KNT do PostgreSQL chưa có KNT). Tiếp theo, hệ thống sẽ liệt kê các kế hoạch thực hiện (Generate Paths, A7) có thể. Sau đó, nó lựa chọn kế hoạch tối ưu (Optimal Path) và tạo cây kế hoạch (Generate Plan, A8) gửi đến các hàm thực thi để thực hiện (Execute Plan, A9) [12].

B. Lưu trữ thông tin KNT

Một công cụ tạo KNT được xây dựng. Nó tiếp nhận truy vấn, thực hiện phân tích truy vấn, tạo bảng KNT và đưa thông tin về KNT vừa được tạo vào các bảng thông tin KNT, phục vụ cho việc xác định khả năng sử dụng KNT để trả lời truy vấn sau này. Các bảng này được tạo ra trong sơ đồ hệ thống pg_catalog. Nghiên cứu đề xuất tổ chức lưu trữ thông tin truy vấn tạo bảng KNT trên các bảng pg_catalog.pg_mv(**mvid**, query, mvtable, s, w, j), pg_catalog.pg_mv_select(**selcalal**, *mvid*, selcaolexp), pg_catalog.pg_mv_from(***mvid***, ***table***), pg_catalog.pg_mv_groupby(***mvid***, ***tabcol***).

Bảng pg_catalog.pg_mv lưu thông tin tổng quát của truy vấn tạo bảng KNT, gồm tên KNT (mvid), chuỗi truy vấn tạo KNT (query), danh sách bảng tham gia trong truy vấn (mvtable), mệnh đề select (s), mệnh đề where (w), mệnh đề join (j). Bảng pg_catalog.pg_mv_select lưu thông tin mệnh đề select, gồm tên KNT (mvid), bí danh cột (selcolal), biểu thức (selcolexp). Bảng pg_catalog.pg_mv_from lưu thông tin bảng tham gia vào truy vấn, gồm tên KNT (mvid), tên bảng (table). Bảng pg_catalog.pg_mv_groupby lưu thông tin các cột thuộc mệnh đề group by, gồm tên KNT (mvid) và tên cột nhóm theo (tabcol).



Hình 1. Quá trình xử lý truy vấn của PostgreSQL và mô-đun viết lại truy vấn

C. Xây dựng mô-đun

Mô-đun viết lại truy vấn hỗ trợ KNT sẽ viết lại Q^U về dạng đầy đủ, so sánh Q^U và Q^M để lựa chọn KNT có thể và viết Q^R tương đương với Q^U. Mô-đun được viết bằng ngôn ngữ C. Khi PostgreSQL khởi động, nó sẽ đọc toàn bộ thông tin về KNT từ các bảng đã đề cập trong **Error! Reference source not found.**.**Error! Reference source not found.** từ lược đồ pg_catalog và lưu vào cache của hệ thống. Rõ ràng, truy xuất dữ liệu từ cache sẽ nhanh hơn rất nhiều so với từ các bảng. Cache sẽ được bổ sung khi có thay đổi điều chỉnh hoặc tạo mới KNT. Việc tìm thông tin trong các bảng được thực hiện theo quy trình như sau. Lấy id của bảng theo tên bảng từ hàm RelnameGetrelid, hệ thống sẽ quét bảng pg_class và trả về id của bảng ứng với tên bảng. Gọi hàm heap_open để mở bảng theo id của bảng ở chế độ NoLock. Tiếp theo, gọi hàm heap_beginscan khởi tạo bộ duyệt để quét các bảng và hàm heap_getnext để duyệt qua các bản ghi của bảng. Khi duyệt hết các bản ghi, dừng bộ duyệt bằng hàm heap_endscan và đóng quan hệ với hàm heap_close.

Trên hình 1 - quá trình xử lý truy vấn của PostgreSQL và những bổ sung viết lại truy vấn hỗ trợ KNT, trước khi truy vấn người dùng nhập vào được xem xét có thể viết lại qua KNT hay không, thì mô-đun sẽ lấy thông tin KNT (B2) từ các bảng ở **Error! Reference source not found.**.**Error! Reference source not found.**, lưu vào các mảng. Mỗi phần tử của mảng có kiểu cấu trúc ứng với thông tin của một KNT. Điều này giúp duyệt qua các KNT để lựa chọn KNT có thể.

Mô-đun viết lại truy vấn sẽ viết lại truy vấn dạng đầy đủ (B1) thông qua điều chỉnh giai đoạn phân tích cú pháp (A4). Truy vấn đầy đủ là truy vấn trong đó các bảng được viết dưới dạng schema.table và các cột dưới dạng table.column. Để lựa chọn KNT có thể (B3), tác giả tiến hành so sánh từng thành phần của truy vấn người dùng nhập vào (đã viết lại đầy đủ (B1)) với các thành phần của truy vấn tạo KNT được lưu trong kiểu cấu trúc (B2). Trước tiên, điều kiện $W^U = W^M$ được kiểm tra. Trường hợp Q^U có thể được tính hoàn toàn từ Q^M (II.A) được xét trước, nếu không đạt thì xét trường hợp Q^U chứa Q^M (II.B). So sánh lần lượt các thành phần theo thứ tự S^U và S^M, T^U và T^M, J^U và J^M, G^U và G^M.

Việc so sánh các biểu thức trong S^U với S^M, đặc biệt là W^U với W^M và J^U với J^M được tiến hành như đề xuất của các công trình [4; 7; 9]. Biểu thức đại số được chuyển tất cả về một vế, thực hiện khai triển, sắp xếp các thành phần theo chiều giảm dần theo nhãn của các phần tử; sau đó được so sánh như so sánh các chuỗi ký tự. Biểu thức luận lý được biến đổi về dạng chuẩn tắc tuyển, sắp xếp theo nhãn của các phần tử dựa theo thứ tự ưu tiên và tính giao hoán của các phép toán, sau đó được so sánh với nhau như so sánh các chuỗi ký tự. Có thể cách giải quyết này chưa phải là tối ưu; có thể phải phát triển phương pháp hiệu quả để so sánh các biểu thức đại số và các biểu thức luận lý dưới dạng các cây. Tuy nhiên, giải pháp này cũng giúp nhận biết được các biểu thức trùng nhau nhưng được viết dưới các dạng khác nhau.

Khi lựa chọn được KNT có thể rồi, phải sinh Q^R sử dụng KNT tương đương với Q^U. Các dạng truy vấn viết lại được trình bày trong mục **Error! Reference source not found.**. Thay vì lấy dữ liệu từ các bảng gốc, thì dữ liệu sẽ được lấy từ KNT. Điều đó giúp tiết kiệm được chi phí nối các bảng, chi phí gộp nhóm theo các cột và tính toán các hàm thống kê. Chuỗi truy vấn được viết lại (B5) sẽ được truyền vào hàm thực thi truy vấn trong postgres (A3) để thực thi truy vấn viết lại (B5) theo cách thực hiện truy vấn thông thường.

D. Tích hợp vào mã nguồn

Trên hình 1, từ quá trình xử lý truy vấn bên trong mã nguồn của PostgreSQL, tác giả điều chỉnh mã nguồn để xây dựng và tích hợp mô-đun viết lại. Mô-đun viết lại truy vấn hỗ trợ KNT phải được chèn vào vị trí trước khi PostgreSQL viết lại truy vấn theo các luật của hệ thống tức là sau vị trí (A5), trước vị trí (A6). Nếu có KNT có thể (B3) thì truy vấn sẽ được viết lại tới KNT đó (B4) và gọi lại hàm thực thi với tham số đầu vào là truy vấn tới KNT (B5). Ngược lại, nếu không có KNT có thể thì truy vấn người dùng nhập vào sẽ tiếp tục được xử lý viết lại theo các luật (A6) mà PostgreSQL đưa ra.

Mô-đun viết lại sẽ có tham số là chuỗi truy vấn người dùng nhập vào và cây truy vấn (Query Tree) - kết quả của giai đoạn phân tích cú pháp (A4). Kết quả trả về của mô-đun là chuỗi truy vấn được viết lại tới KNT nếu có (B5) hoặc là NULL nếu không. Sau đó chuỗi truy vấn qua KNT (B5) được thực thi như một truy vấn thông thường bằng cách truyền chuỗi truy vấn (B5) đến hàm thực thi truy vấn trong PostgreSQL (A3).

IV.THỬ NGHIỆM VÀ ĐÁNH GIÁ

Sau khi tích hợp mô-đun viết lại truy vấn hỗ trợ KNT vào mã nguồn của PostgreSQL, tác giả tiến hành biên dịch, cài đặt và cấu hình máy chủ theo quy trình [11]. Tiếp theo, tác giả tạo các KNT, tiến hành chạy các truy vấn thử nghiệm, đo lường thời gian thực thi để đánh giá tính khả thi của mô-đun tích hợp. Môi trường chạy thử nghiệm là hệ điều hành Windows 10 64 bit, Intel core i5 1.7x4 GHz, RAM 4G DDR3, HDD SATA3 500 GB 7200 vòng/phút. Trên CSDL bán hàng mẫu gồm các bảng countries - 23 bản ghi, customers – 55.500, sales – 91.8845, costs – 82.112 với các chỉ mục được tạo trên các khoá chính và khoá ngoại, tác giả tạo một số bảng KNT lưu trong CSDL với các truy vấn ở bảng 1, thực thi 10 lần các truy vấn ở bảng 2 trong HQT CSDL PostgreSQL chưa tích hợp mô-đun viết lại và trong trường hợp đã tích hợp mô-đun viết lại, tác giả đã thu được thời gian thực thi trung bình được làm tròn đến ms như trên bảng 3.

Kết quả đo lường ở bảng 3 cho thấy, với các truy vấn phức tạp sử dụng lượng dữ liệu lớn thì KNT hỗ trợ thực thi rất hiệu quả, thời gian chạy truy vấn nhỏ hơn nhiều lần so với trường hợp không hỗ trợ KNT, nâng cao hiệu suất hoạt động của cả hệ thống; đặc biệt hiệu quả khi số lượng nhóm các bản ghi trong quá trình xử lý là lớn, tỉ lệ tổng số lượng bản ghi trên số lượng nhóm lớn. Nhìn chung, mô-đun viết lại truy vấn là hiệu quả khi Q^U có thể sử dụng KNT. Ngược lại, nếu không thể sử dụng KNT, thì mô-đun không hiệu quả vì tốn chi phí xử lý quét các KNT để lựa chọn KNT có thể viết lại truy vấn. Tuy nhiên, chi phí này là nhỏ và có thể chấp nhận được. Tùy thuộc vào môi trường chạy truy vấn, độ lớn của cơ sở dữ liệu, độ phức tạp của truy vấn mà chênh lệch thời gian thực thi truy vấn trên PostgreSQL đã tích hợp mô-đun viết lại và PostgreSQL chưa tích hợp mô-đun viết lại là cao hay thấp. Đối với các cơ sở dữ liệu với số lượng bản ghi nhỏ, truy vấn đơn giản thì mô-đun viết lại truy vấn hỗ trợ KNT có thể không hiệu quả.

Trường hợp kết quả thực thi Q^U được tính bằng cách sử dụng KNT và các bảng khác (trường hợp 4 trong các bảng 2 và bảng 3) cho hiệu quả thấp hơn trường hợp kết quả thực thi Q^U có thể được tính hoàn toàn chỉ dùng KNT (trường hợp 1-3 trong các bảng 2 và bảng 3) vì lỉ lệ độ phức tạp giữa Q^U và Q^M thấp hơn nhưng xử lý viết lại thì phức tạp hơn, tuy nhiên, hiệu quả vẫn rất cao (giảm 519 lần thời gian thực thi). Các truy vấn phức tạp bao gồm hàm thống kê thường sử dụng nhiều bảng khác nhau, nhưng các cột tham gia vào các hàm thống kê trong các truy vấn đó thường từ các bảng chứa dữ liệu phục vụ mô tả chi tiết, ít khi nào từ cả bảng chứa dữ liệu mô tả phân loại. Vì thế, việc triển khai trường hợp kết quả thực thi Q^U được tính bằng cách sử dụng KNT và các bảng khác cũng là rất hữu ích.

Bảng 1. Truy vấn tạo KNT

KNT	Q^M	Mục đích
mv1	SELECT countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name, SUM(quantity_sold*unit_price) AS total FROM countries, customers, sales, costs WHERE countries.country_id = customers.country_id AND customers.cust_id = sales.cust_id AND sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name	Đưa ra danh sách khách hàng với thông tin quốc gia và tổng số tiền đã mua hàng

mv2	SELECT sales.cust_id, sum(quantity_sold*unit_cost) as tongtien, sum(sales.quantity_sold) as tongban FROM sales, costs WHERE sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY sales.cust_id	Đưa ra danh sách khách hàng tổng số tiền và tổng số hàng đã mua

Bảng 2. Các truy vấn chạy thử nghiệm và KNT có thể viết lại

#	Q^U	Dùng KNT	Mẫu	Q^R
1	SELECT countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name, SUM(quantity_sold*unit_price) AS total FROM countries, customers, sales, costs WHERE countries.country_id = customers.country_id AND customers.cust_id = sales.cust_id AND sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY countries.country_id, country_name, customers.cust_id	mv1	II.A	SELECT country_id, country_name, cust_id, cust_first_name, cust_last_name, total FROM mv1
2	SELECT countries.country_id, country_name, SUM(quantity_sold*unit_price) AS total FROM countries, customers, sales, costs WHERE countries.country_id = customers.country_id AND customers.cust_id = sales.cust_id AND sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY countries.country_id, country_name	mv1	II.A	SELECT country_id, country_name, SUM(total) AS total FROM mv1 GROUP BY country_id, country_name
3	SELECT sales.cust_id, sum(quantity_sold*unit_cost) /sum(quantity_sold) as tb FROM sales, costs WHERE sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY sales.cust_id	mv2	II.A	SELECT cust_id, tongTien/ tongBan as tb FROM mv2
4	SELECT countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name, SUM(quantity_sold*unit_price) AS tongtien, sum(sales.quantity_sold) as tongban FROM countries, customers, sales, costs WHERE countries.country_id = customers.country_id AND customers.cust_id = sales.cust_id AND sales.prod_id = costs.prod_id AND sales.time_id = costs.time_id GROUP BY countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name	mv2	II.B	SELECT countries.country_id, country_name, customers.cust_id, cust_first_name, cust_last_name, tongtien, tongban FROM countries, customers, mv2 WHERE countries.country_id = customers.country_id AND customers.cust_id = mv2.cust_id
5	SELECT cust_city_id, cust_city, count(cust_id) as sokh FROM customers GROUP BY cust_city_id, cust_city			
6	SELECT customers.cust_id, count(prod_id) as goods FROM customers inner join sales on customers.cust_id = sales.cust_id GROUP BY customers.cust_id			

Bảng 3. Đánh giá hiệu quả tích hợp mô-đun viết lại truy vấn

Q^U	Thời gian, chưa tích hợp mô-đun (T1, ms)	Đã tích hợp mô-đun		T1/T2	Hiệu quả
		Dùng KNT	Thời gian (T2, ms)		
1	211594	Có	93	2275	Có
2	21589	Có	62	348	Có
3	34143	Có	71	480	Có
4	273220	Có	519	526	Có
5	219	Không	267	0.822	Không
6	34289	Không	34358	0.997	Không

Nghiên cứu chưa quan tâm đến các truy vấn bao gồm truy vấn lồng, phép nối ngoài và truy vấn đệ quy, dù đó cũng là lĩnh vực rất quan trọng nhưng ít phổ biến hơn. Trên thực tế, với hầu hết các trường hợp truy vấn bao gồm truy vấn lồng, người dùng có thể chủ động viết lại dưới dạng truy vấn bao gồm phép nối trong, không bao gồm truy vấn lồng và ứng dụng khả năng viết lại của mô-đun. Việc sử dụng nhiều KNT để trả lời một truy vấn cũng cần được nghiên cứu. Chẳng hạn, cho bài toán tính tổng số lượng mỗi mặt hàng đã nhập, đã xuất, còn lại ở mỗi kho. Thông thường,

người ta phải tổ chức hai khung nhìn để tính tổng xuất và tổng nhập, sau đó viết truy vấn bao gồm phép nối ngoài trái sử dụng hai khung nhìn. Rõ ràng, có thể xây dựng hai KNT thay vì hai khung nhìn và xây dựng cơ chế đủ thông minh và hiệu quả để nhận biết khả năng sử dụng nhiều KNT để trả lời một truy vấn.

V. KẾT LUẬN

Nghiên cứu xây dựng được quy luật viết lại và triển khai thực tế trong HQT CSDL PostgreSQL các truy vấn dạng phổ biến nhất – truy vấn bao gồm phép nối trong với điều kiện chọn lựa bản ghi ở Q^U tương đương với Q^M theo hai trường hợp: i) kết quả thực thi Q^U có thể được tính hoàn toàn chỉ dùng KNT; ii) kết quả thực thi Q^U được tính bằng cách sử dụng KNT và các bảng khác.

Tác giả đã nghiên cứu quy trình xử lý truy vấn bên trong mã nguồn của PostgreSQL, xây dựng mô-đun viết lại truy vấn theo tiêu chuẩn mã nguồn của PostgreSQL, can thiệp và tích hợp được mô-đun vào mã nguồn của PostgreSQL, thử nghiệm và đánh giá tính khả thi của mô-đun. Kết quả thử nghiệm mô-đun cho thấy hiệu quả khi viết lại truy vấn để sử dụng KNT - tăng tốc độ thực thi của truy vấn lên nhiều lần, đặc biệt là các truy vấn phức tạp sử dụng lượng dữ liệu lớn. Với các trường hợp truy vấn không sử dụng KNT, thời gian thực thi lớn hơn so với khi không có mô-đun viết lại vì chi phí tìm kiếm KNT. Tuy nhiên, chênh lệch là không đáng kể.

VI.TÀI LIỆU THAM KHẢO

[1] "Materialized Views - Oracle to SQL Server Migration", <http://www.sqlines.com/oracle/statements/create_materialized_view> (Truy cập: 20/02/2013).

[2] Nguyễn T. Q. V., "Ứng dụng khung nhìn thực để nâng cao tốc độ thực thi truy vấn", Tạp chí Khoa học và công nghệ, Đại học Đà Nẵng, 1(30), 2009, tr. 59-65.

[3] Zhou J., Larson P.-A., Goldstein J., "Partially materialized views", Technical Report MSR-TR-2005-77, Microsoft Research, 2005.

[4] Zaharioudakis M., Cochrane R., Lapis G., Pirahesh H., Urata M., "Answering complex SQL queries using automatic summary tables", In Proc. of SIGMOD Conference, 2000, pp. 105-116.

[5] Nguyễn T. Q. V., Trần T.N., "Nghiên cứu xây dựng mô-đun sinh tự động mã nguồn trigger trong ngôn ngữ C thực hiện cập nhật gia tăng, đồng bộ các khung nhìn thực trong PostgreSQL", HTKH Quốc gia về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR), VII-2014, tr. 440-448.

[6] Nguyễn V.Q., Nguyễn T.Q.V., "Nghiên cứu xây dựng và tích hợp mô-đun viết lại truy vấn hỗ trợ khung nhìn thực trong PostgreSQL", Tạp chí Khoa học và Công nghệ - Đại học Đà Nẵng, 8(69), 2013, tr. 169-175.

[7] Srivastava D., Dar S., Jagadish H.V., Levy A.Y., "Answering Queries with Aggregation Using Views", Proceedings of the 22th International Conference on Very Large Data Bases, 1996, Morgan Kaufmann Publishers Inc.: 318-329.

[8] Kungurtsev A.B., Nguyen T.Q.V., "The analysis of feasibility of applying the materialized views in information systems", Odes'kyi Politechnichnyi Universytet. Pratsi, 2(20), 2003, pp. 102-106.

[9] Kungurtsev A.B., Nguyen T.Q.V., Blashko A.A., "Сравнение запросов в реляционных базах данных для построения материализованных представлений - Comparison of queries in a relational database to build materialized views", Praci UNDIRT, Ukraine, 3(39), 2004, pp. 35-38.

[10] Kungurtsev A.B., Nguyen T.Q.V., "Data extraction from materialized views in information systems", Odes'kyi Politechnichnyi Universytet. Pratsi, 1(23), 2005, pp. 82-87.

[11] PostgreSQL, "PostgreSQL 9.4 Documentation", 2015.

[12] Group T.P.G.D., "PostgreSQL Backend Flowchart", <http://www.postgresql.org/developer/backend/> (Truy cập: 20/5/2014).

QUERIES REWRITING FOR USING MATERIALIZED VIEWS WITH AGGREGATE FUNCTIONS IN POSTGRESQL

Nguyen Tran Quoc Vinh

ABSTRACT - *Materialized views are retained executed query results and help answer queries quickly instead of taking data from the original tables. The materialized views technology is implemented in 3 commercial database management systems (Oracle, DB2, SQL Server). Since v.9.3 and now, v.9.4, PostgreSQL supports commands for creating materialized views and updating asynchronously with full refresh. It does not support automatic using of materialized views. In this article, the author explores the process of query processing in PostgreSQL, aims to build, integrate the module that queries with aggregate functions are rewriten to use the materialized views in PostgreSQL by the clever way.*

Keywords *- materialized views; aggregate functions; PostgreSQL; query processing; query rewriting; open source intervention.*