Journal of the Research Institute of Industrial Technology, Vol 19, 2002

그래프로 표현된 자료의 관리를 위한 데이터베이스 라이브러리 설계에 관한 연구

박 휴 찬[1], 추 인 경[2]

A Study on the Design of Database Library for the Management of Graph Represented Data

H. C. Park[1], I. K. Chu[2]

Abstract

The graph has provided a powerful methodology to solve a lot of real-world problems. There has been much research on the graph representations and algorithms. But, there are still many difficulties to apply the graph to practical domains. This paper proposes a graph library designed on the relational database. Graphs are represented in the form of relational tables and then saved in a database. Graph operations are coded in terms of the database language, SQL, and general purpose programming language, C. Users of the graph library can manage efficiently a large amount of graph data. Furthermore, graphs saved in the database can be concurrently shared among many users. The proposed graph library may be useful to represent and solve real world problems efficiently.

1. 서 론

그래프는 다양한 분야에서 복잡한 문제를 수학적으로 모델링하여 원하는 해를 구하는 아주 강력한 방법을 제공한다 [1][2]. 이러한 이유로 그래프를 일목요연하게 표현하기 위한 자료구조와 그래프 연산에 대한 효율적인 알고리즘의 개발이 진행되어 왔다 [3][4][5]. 하지만, 이렇게 개발된 그래프 자료구조와 그래프 알고리즘을 실세계의 문제에 적용하는 것은 간단한 작업이 아니다. 즉, 실세계의 문제를 그래프의 형태로 효과적으로 모델링하는 방법, 이렇게 모델링된 그래프를 체계적으로 저장하여 관리하는 방법, 그리고 이러한 그래프에 대하여 원하는 해를 구하기 위한 효율적인 프로그램의 개발 등에는 많은 노력이 요구된다.

본 논문에서는 실세계에서 발생하는 문제에 대하여 그래프를 효과적으로 적용하기 위하여 데이터베이스 기술을 응용하는 방법을 제안한다. 우선, 그래프로 모델링된 정보를 데이터베이스화하기에 적합한 표현방법인 테이블에 대하여 그 구조를 설계한다. 이렇게 설계된 테

1) 한국해양대학교 기계 정보공학부
2) 주식회사 하우리

이블에 따라 표현된 그래프를 저장, 관리하기 위한 기본적인 연산들을 정의하고 라이브러리화하기 위한 설계를 제시한다. 본 논문에서 제안하는 그래프 라이브러리는 다음과 같은 테이블들로 구성된다. 각 그래프의 기본적인 정보를 저장하는 *그래프정보 테이블*, 그래프의 한 구성요소인 vertex에 관한 정보를 저장하는 *vertex 테이블*, 또 다른 구성요소인 edge에 관한 정보를 저장하는 *edge 테이블*, 그리고 vertex와 edge에 부가된 각종 속성 정보를 저장하는 *vertex 속성 테이블*과 *edge 속성 테이블*로 구성된다. 또한, 그래프 라이브러리는 다음과 같은 기본 연산들로 구성된다. 데이타베이스 접속/해제 연산, 그래프/vertex/edge의 생성/갱신/삭제/검색 연산, 복구/완료 연산 등으로 구성된다. 이러한 그래프 라이브러리는 관계형 데이터베이스를 이용하여 구현 가능하며 데이터베이스의 여러 가지 장점 가질 수 있다. 즉, 방대한 양의 그래프를 효율적으로 저장, 관리할 수 있을 뿐만 아니라 다수의 사용자들이 저장된 그래프를 공유할 수도 있다. 또한, 라이브러리에서 제공하는 기본 함수를 이용하면 각종 응용 프로그램의 개발이 용이해질 수 있다. 본 논문에서는 기본적인 그래프 개요를 우선 살펴본 후, 그래프를 표현하기 위한 데이터베이스 테이블의 구조와 그래프의 기본적인 연산에 대한 라이브러리 설계에 대하여 논한다.

2. 그래프 개요

그래프 $G=(V, E)$는 vertex라고 불리는 요소의 유한집합인 V와 두개의 vertex 쌍으로 구성되는 edge라고 불리는 요소의 유한집합 E로 구성된다. edge가 방향을 나타내면 방향그래프(digraph/directed graph)라하고 edge가 가중치를 가지면 가중치그래프(weighted graph)라 한다. vertex u와 v를 끝점으로 가지는 edge e는 vertex u와 v에 달려있다(incident with)라고 말하고, u와 v는 서로 인접(adjacent)인 vertex라고 한다. vertex v의 차수(degree)는 v에 달린 edge의 수로서 $deg(v)$라고 표시한다.

그래프 $G=(V_G, E_G)$와 $H=(V_H, E_H)$에서 $V_H \subseteq V_G$이고 $E_H \subseteq E_G$이면 H를 G의 부그래프(subgraph)라 한다. 만약, H가 G의 부그래프이고 $V_H = V_G$이면 H는 G의 스패닝그래프(spanning subgraph)라고 한다. 그래프 G의 두 vertex를 x, y라 하면, walk는 vertex x에서 시작하여 vertex y에서 끝나는 유한한 vertex와 edge의 순열(sequence)이며 x-y walk로 표시한다. walk의 길이(length)는 walk 내의 edge의 개수이다. 이때 순열에서 edge가 중복되지 않는 것을 x-y trail, vertex가 중복되지 않는 것을 x-y path라고 한다 [6].

3. 데이터베이스 테이블 설계

그래프를 저장하기 위한 데이터베이스 테이블은 그래프에 대한 전체적인 정보를 저장하는 테이블 (*graph_info*), vertex에 관한 정보를 저장하는 두개의 테이블 (*vertex, vertex_att*), edge에 관한 정보를 저장하는 두개의 테이블 (*edge, edge_att*)로 구성된다. 각 테이블의 상호 관계는 그림 1에서 보여준다. 실선 ㉮는 그래프 ID의 참조관계를 나타내며 *graph_info* 테이블에 존재하는 그래프 ID(*g_id*)를 모든 테이블들이 참조한다. 점선 ㉯는 vertex ID의 참조관계를 나타내며 *vertex* 테이블에 존재하는 vertex ID(*v_id*)를 *vertex_att* 테이블과 *edge* 테이블이 참조한다. 절선 ㉰는 edge ID의 참조 관계를 나타내며 *edge* 테이블에 존재

하는 edge ID(e_*id*)를 *edge_att* 테이블이 참조한다.

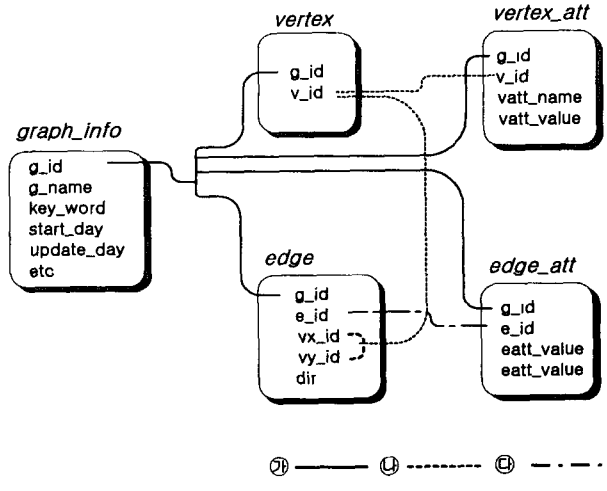

그림 1. 데이타베이스 테이블의 상호관계

3.1 그래프 정보 테이블 (graph information table)

각 그래프의 전체적인 정보를 저장하는 메타 데이터 테이블은 *graph_info*이다. 각각의 그래프를 고유하게 식별하는 그래프 ID(*g_id*)를 기본키로 하고, 그래프 이름(*g_name*), 그래프에 대한 키워드(*key_word*), 그래프 작성 시작날짜(*start_day*)와 최종 수정날짜(*update_day*), 기타정보(*ect*)로 구성된다.

[*graph_info*]

필드명	설 명
g_id	그래프 ID, (기본키)
g_name	그래프 이름
key_word	키워드
start_day	그래프 작성 시작일
update_day	그래프 최종 수정일
ect	기타정보

3.2 Vertex 정보 테이블 (vertex information table)

vertex에 관한 정보를 저장하는 테이블은 vertex의 기본정보를 저장하는 *vertex* 테이블과 vertex에 부가된 각종 속성들과 그 값을 저장하는 *vertex_att* 테이블로 구성된다. 이렇게 두개의 테이블로 분리한 이유는 한 vertex에 여러개의 속성이 존재할 수 있기 때문이다.

vertex 테이블의 구조는 vertex를 고유하게 식별하는 ID인 *v_id,* vertex가 소속된 그래프를 나타내는 *g_id*로 구성된다. 서로 다른 그래프에 동일한 vertex ID가 존재할 수 있으므로 이 vertex들을 서로 구별하기 위해 (*g_id, v_id*)가 기본키의 역할을 한다.

vertex_att 테이블의 구조는 속성의 이름을 나타내는 *vatt_name*, 속성의 값을 저장하는 *vatt_value*, 속성이 소속된 vertex와 이 vertex가 소속된 그래프를 나타내는 *v_id*와 *g_id*로 구성된다. 서로 다른 그래프와 vertex에 동일한 속성이 존재할 수 있으므로 이 속성들을 서로 구별하기 위해 (*g_id, v_id, vatt_name*)가 기본키의 역할을 한다.

[*vertex*]

필드명	설 명
g_id	vertex *v_id*가 소속된 그래프의 ID, (기본키)
v_id	vertex ID, (기본키)

[*vertex_att*]

필드명	설 명
g_id	vertex *v_id*가 소속된 그래프의 ID, (기본키)
v_id	속성 *vatt_name*가 소속된 vertex의 ID, (기본키)
vatt_name	속성의 이름, (기본키)
vatt_value	속성의 값

3.3 Edge 정보 테이블 (edge information table)

edge에 관한 정보를 저장하기 위한 테이블은 edge의 기본정보를 저장하는 *edge* 테이블과 edge에 부가된 각종 속성들과 그 값을 저장하는 *edge_att* 테이블로 구성된다. 이렇게 두 개의 테이블로 분리한 이유는 한 edge에 여러 개의 속성이 존재할 수 있기 때문이다.

edge 테이블의 구조는 edge를 고유하게 식별하는 ID인 *v_id*, edge가 소속된 그래프를 나타내는 *g_id*, edge가 연결하는 두개의 vertex *x*와 *y*를 나타내는 *vx_id*와 *vy_id*, edge의 방향성 유무를 나타내는 *dir*로 구성된다. 서로 다른 그래프에 동일한 edge ID가 존재할 수 있으므로 이 edge들을 서로 구별하기 위해 (*g_id, e_id*)가 기본키의 역할을 한다.

edge_att 테이블의 구조는 속성의 이름을 나타내는 *eatt_name*, 속성의 값을 저장하는 *eatt_value*, 속성이 소속된 edge와 이 edge가 소속된 그래프를 나타내는 *e_id*와 *g_id*로 구성된다. 서로 다른 그래프와 edge에 동일한 속성이 존재할 수 있으므로 이 속성들을 서로 구별하기 위해 (*g_id, e_id, eatt_name*)가 기본키의 역할을 한다.

[*edge*]

필드명	설 명
g_id	edge e_*id*가 소속된 그래프의 ID, (기본키)
e_id	edge ID, (기본키)
vx_id	edge가 연결하는 두 개의 vertex ID
vy_id	
dir	edge의 방향성 · *true*(*y*) : x → y, directed graph · *false*(*n*) : x — y, undirected graph

[edge_att]

필드명	설 명
g_id	edge *e_id*가 소속된 그래프의 ID, (기본키)
e_id	속성 *eatt_name*가 소속된 edge의 ID, (기본키)
eatt_name	속성의 이름, (기본키)
eatt_value	속성의 값

4. 데이타베이스 라이브러리 설계

위의 테이블들은 그래프 데이터베이스 생성시에 만들어지게 되며, 각 그래프는 다음의 라이브러리를 이용하여 이 테이블들에 저장, 관리될 수 있다. 즉, 그래프, vertex, edge, vertex 속성, edge 속성 정보에 대해서 생성, 갱신, 삭제, 검색 그리고 복구 및 완료 연산으로 구분하여 수행 가능하다. 그림 2는 그래프 라이브러리의 사용 흐름을 도식화했으며 모든 작업은 데이터베이스 연결로 시작하여 데이터베이스 연결해제로 마무리된다.

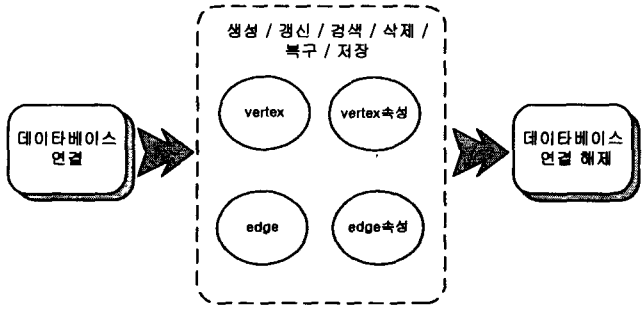

그림 2. 그래프 라이브러리 사용 흐름도

4.1 그래프 데이터베이스 생성 (graph database creation)

데이터베이스에 그래프 테이블을 생성한다. 이때 생성되는 그래프 테이블은 *graph_info* 테이블, *vertex* 테이블, *edge* 테이블, *vertex_att* 테이블, *edge_att* 테이블이다. 그래프 데이터베이스 생성은 관리자 모드에서 한번 수행된다.

- *create_graph_db ();*

4.2 그래프 데이터베이스 연결/해제 (graph database connection/disconnection)

데이터베이스에 대한 작업을 수행하기 위해 데이터베이스에 연결을 설정하고 작업 완료 후 연결을 해제하는 라이브러리이다. 기본적으로 관계형 데이터베이스로 저장되어 있는 그래프 테이블을 이용해야 하기 때문에 사용자의 계정과 암호를 이용하여 접속한다.

- *connect (id, passwd);*
  사용자 계정과 암호로 그래프 데이터베이스에 연결한다.

- *disconnect ();*
  데이터베이스 연결을 종료한다.

4.3 그래프 생성 (graph creation)

사용자는 데이타베이스와의 연결이 이루어졌으면 새로운 그래프를 작성할 것인지 기존의 그래프를 불러와 작업할 것인지를 결정하게 된다. 새로운 그래프를 생성할 경우에 *graph_info* 테이블에 새로운 그래프에 대한 그래프 ID(*g_id*)와 그래프 이름(*g_name*) 및 기타 속성에 대한 값을 입력한다. 그래프에 대한 모든 작업은 *graph_info* 테이블에 등록되어 있는 그래프에 대해서만 가능하기 때문에 새로운 그래프 ID 등록은 매우 중요한 작업이며 새로운 그래프 ID를 생성할 때 중복된 그래프 ID인지 확인할 필요가 있다.

- *create_graph (graph_id, graph_name, key, start, update, description);*
  새로운 그래프를 생성한다. 그래프 ID, 이름, 키워드, 작성시작일, 갱신일, 추가정보를 인자로 가진다.

4.4 데이터 삽입 (data insertion)

graph_info 테이블에 등록된 그래프에 대하여 vertex와 edge 그리고 각 속성 데이터를 삽입하는 라이브러리이다.

vertex 삽입은 그래프 ID와 vertex ID를 이용해서 이루어지며 *vertex* 테이블에 저장된다. 이미 존재하는 vertex ID와 중복된 vertex ID는 삽입할 수 없으므로 vertex ID를 확인할 필요가 있다. 삽입한 vertex에 대해 속성 데이터를 삽입하기 위해서는 그래프 ID, vertex ID, vertex의 속성이름 그리고 vertex의 속성값을 이용하여 이루어지며 *vertex_att* 테이블에 저장된다. 하나의 vertex에 같은 속성이름으로 한개 이상의 속성값을 가질 수 없다. 그러므로 해당 vertex에 어떤 속성들이 존재하는지 확인할 필요가 있다.

edge 삽입은 그래프 ID, edge ID, 두 개의 연결점 vertex 그리고 방향성을 이용해서 이루어지며 *edge* 테이블에 저장된다. 이미 존재하는 edge ID와 중복되는 edge ID는 삽입할 수 없으므로 edge ID를 확인할 필요가 있다. 연결점 vertex는 *vertex* 테이블에 존재하는 vertex이어야 한다. 삽입한 edge에 대해 속성 데이터를 삽입하기 위해서는 그래프 ID, edge ID, edge의 속성이름 그리고 edge의 속성값을 이용하여 이루어지며 *edge_att* 테이블에 저장된다. vertex 삽입과 마찬가지로 같은 속성이름으로 한개 이상의 속성값을 가질 수 없다. 그러므로 해당 edge에 어떤 속성들이 존재하는지 확인할 필요가 있다.

- *insert_vertex (graph_id, vertex_id);*
  vertex를 삽입한다. vertex를 삽입할 그래프 ID, 새롭게 삽입되는 vertex ID를 인자로 가진다.

- *insert_vertex_attribute (graph_id, vertex_id, vatt_name, vatt_value);*
  vertex에 속성 데이터를 삽입한다. 그래프 ID, vertex ID, 속성이름, 속성값을 인자로 가진다.

- *insert_edge (graph_id, edge_id, vertex_xid, vertex_yid, direction);*
  edge를 삽입한다. edge를 삽입할 그래프 ID, 새롭게 삽입되는 edge ID, 두개의 연결점 vertex, 방향성 여부를 인자로 가진다.

- *insert_edge_attribute (graph_id, edge_id, edge_attname, edge_attvalue);*
  edge에 속성 데이터를 삽입한다. 그래프 ID, edge ID, 속성이름, 속성값을 인자로 가진다.

4.5 데이터 갱신 (data update)

데이터베이스에 이미 존재하는 그래프 데이터를 새로운 값으로 갱신하는 라이브러리이다. 즉, 이 라이브러리는 그래프정보, vertex, vertex 속성, edge, edge 속성 테이블에 대하여 데이터를 변경하게 된다.

그래프정보 갱신은 그래프 ID를 기준 값으로 두고 새로운 그래프 ID, 그래프이름, 키워드, 작업시작일, 최종수정일, 기타정보로 한번에 갱신할 수 있다. 그래프 ID를 갱신할 경우, *vertex* 테이블, *edge* 테이블, *vertex_att* 테이블, *edge_att* 테이블에 저장되어 있는 기존의 그래프 ID는 모두 새로운 그래프 ID로 갱신된다.

vertex 갱신은 그래프 ID, vertex ID를 기준 값으로 두고 새로운 vertex ID, 속성이름, 속성값으로 갱신할 수 있다. vertex ID를 갱신할 경우 *vertex* 테이블과 *vertex_att* 테이블, *edge* 테이블(연결점 vertex)에서 기존의 vertex ID는 모두 새로운 vertex ID로 갱신된다.

edge 갱신은 그래프 ID, edge ID를 기준으로 두고 새로운 edge ID, edge 연결점 vertex, 방향성, 속성이름, 속성값으로 갱신할 수 있다. edge ID를 갱신할 경우 *edge* 테이블과 *edge_att* 테이블에 존재하는 기존의 edge ID는 모두 새로운 edge ID로 갱신된다. 존재하지 않는 데이터에 대해서 갱신을 시도하거나 새롭게 갱신된 기준 값들이 이미 그래프 데이터베이스에 존재하는 값이라면 오류 메세지를 발생한다.

1) 그래프 정보 갱신

- *update_graph_information (old_graph_id, new_graph_id, new_g_name, new_key_word, new_start, new_update, new_etc);*
  그래프의 정보를 새로운 그래프 ID, 이름, 키워드, 시작일, 수정일, 기타정보로 갱신한다.

- *update_graph_id (old_graph_id, new_graph_id);*
  그래프 ID를 새로운 값으로 갱신한다.

- *update_graph_name (graph_id, new_graph_name);*
  그래프의 이름을 갱신한다.

- *update_graph_keyword (graph_id, new_keyword);*
  그래프의 키워드를 갱신한다.

- *update_graph_startday (graph_id, new_day);*
  그래프의 작성시작일을 갱신한다.

- *update_graph_upeateday (graph_id, new_day);*
  그래프의 최종수정일을 갱신한다.

- *update_graph_comment (graph_id, new_etc);*
  그래프의 기타정보를 갱신한다.

2) vertex와 vertex 속성 갱신

- *update_vertex_information (graph_id, old_vertex_id, new_vertex_id, old_vatt_name, new_vatt_name, new_vatt_value);*
  vertex의 전체 정보를 새로운 vertex ID, 속성이름, 속성값으로 갱신한다.
- *update_vertex_id (graph_id, old_vertex_id, new_vertex_id);*
  vertex ID를 새로운 값으로 갱신한다.
- *update_vertex_attname(graph_id, vertex_id, old_vatt_name, new_vatt_name);*
  vertex의 속성이름을 갱신한다.
- *update_vertex_attvalue(graph_id, vertex_id, vatt_name, new_vatt_value);*
  vertex의 속성이름에 해당하는 속성값을 갱신한다.

3) edge와 edge 속성 갱신

- *update_edge_information (graph_id, old_edge_id, new_edge_id, new_vx_id, new_vy_id, new_direction,old_edge_attname, new_edge_attname, new_edge_attvalue);*
  edge의 전체 정보를 새로운 edge ID, 연결점, 속성이름, 속성값으로 갱신한다.
- *update_edge_id (graph_id, old_edge_id, new_edge_id);*
  edge ID를 새로운 값으로 갱신한다.
- *update_edge_connection (graph_id, edge_id, new_vx_id, new_vy_id);*
  edge의 연결점을 갱신한다.
- *update_edge_direction (graph_id, edge_id, new_dir);*
  edge의 방향성을 갱신한다.
- *update_edge_attname (graph_id, edge_id, old_edge_attname, new_edge_attname);*
  edge의 속성이름을 갱신한다.
- *update_edge_attvalue (graph_id, edge_id, edge_attname, new_edge_attvalue);*
  edge의 속성이름에 해당하는 속성값을 갱신한다.

4.6 데이터 검색 (data search)

데이터베이스에 존재하는 그래프에 관한 데이터를 검색하는 라이브러리이다. 검색 연산은 데이터베이스에 존재하는 한개 이상의 테이블에 대해 많은 죠인(join)을 통해서 이루어진다. 각 검색결과는 임시 테이블에 저장되며 사용자의 요구가 있을 때 한 행씩 읽어져 나오게 된다. 사용자는 검색된 결과에 대하여 다양한 그래프 알고리즘을 적용하여 좀 더 복잡한 작업을 수행 할 수 있다.

그래프정보 검색은 그래프 ID를 이용하여 *graph_info* 테이블에 저장되어 있는 전체 정보뿐만 아니라 각 속성에 대한 부분 정보를 검색할 수 있다. 그리고 해당 그래프에 소속되어 있는 vertex와 edge에 대한 정보도 검색할 수 있다.

vertex 검색은 그래프 ID와 vertex ID를 이용하여 *vertex* 테이블과 *vertex_att* 테이블에 저장되어 있는 해당 vertex에 대한 데이터 검색이 가능하다. vertex의 속성정보를 검색할 경

우 그래프 ID, vertex ID 그리고 vertex 속성이름을 이용하여 해당 속성값 검색이 가능하다.

edge 검색은 그래프 ID와 edge ID를 이용하여 *edge* 테이블과 *edge_att* 테이블에 저장되어 있는 해당 edge에 대한 데이터 검색이 가능하다. edge의 속성정보를 검색할 경우 그래프 ID, edge ID 그리고 edge 속성이름을 이용하여 해당 속성값 검색이 가능하다.

그밖에 vertex와 vertex들 사이, vertex와 edge 사이의 관련성을 검색하기 위한 검색 라이브러리가 있다. 즉, vertex들 사이의 인접성 여부를 검색하는 인접성 검색과 vertex에 연결된 edge의 개수를 검색하는 차수 검색이 있다.

1) fetch 라이브러리

- *graph_fetch_row (result);*
  결과 테이블의 레코드 세트에서 한 행(레코드)을 가져온다.

2) 그래프 정보 검색

- *get_all_graph ();*
  그래프 데이터베이스에 저장된 모든 그래프의 ID를 검색한다.
- *get_all_graph_name ();*
  그래프 데이터베이스에 저장된 모든 그래프의 이름을 검색한다.
- *get_graph_information (graph_id);*
  주어진 그래프 ID의 그래프 정보를 검색한다.
- *get_graph_name (graph_id);*
  주어진 그래프 ID의 그래프 이름을 검색한다.
- *get_graph_startday (graph_id);*
  주어진 그래프 ID의 작성시작일을 검색한다.
- *get_graph_updateday (graph_id);*
  주어진 그래프 ID의 최종수정일을 검색한다.
- *get_graph_comment(graph_id);*
  주어진 그래프 ID의 기타정보를 검색한다.

3) vertex와 vertex 속성 검색

- *get_all_vertices (graph_id);*
  주어진 그래프 ID에 소속된 모든 vertex ID를 검색한다.
- *get_vertex_information (graph_id, vertex_id);*
  주어진 그래프 ID의 vertex ID에 해당하는 모든 속성 정보를 검색한다.
- *get_all_vertex_attributes (graph_id, vertex_id);*
  주어진 그래프 ID의 vertex ID에 해당하는 모든 속성이름을 검색한다.
- *get_vertex_attribute (graph_id, vertex_id, vatt_name);*
  주어진 그래프 ID의 vertex ID와 속성이름에 해당하는 속성값을 검색한다.

4) edge와 edge 속성 검색

- *get_all_edges (graph_id);*
  주어진 그래프 ID에 소속된 모든 edge ID를 검색한다.

- *get_edge_information (graph_id, edge_id);*
  주어진 그래프 ID의 edge ID에 해당하는 모든 속성 정보를 검색한다.

- *get_all_edge_attributes (graph_id, edge_id);*
  주어진 그래프 ID의 edge ID에 해당하는 모든 속성이름을 검색한다.

- *get_edge_attribute (graph_id, edge_id, eatt_name);*
  주어진 그래프 ID의 edge ID와 속성이름에 해당하는 속성값을 검색한다.

- *get_edge_connection (graph_id, edge_id);*
  주어진 그래프 ID의 edge ID가 연결하는 두 vertex의 ID를 검색한다.

- *get_edge_xyconnection (graph_id, vx_id, vy_id);*
  주어진 그래프 ID의 두 vertex를 연결하는 edge ID를 검색한다.

- *get_edge_direction (graph_id, edge_id);*
  주어진 그래프 ID의 edge ID에 해당하는 방향성을 검색한다.

5) 인접성(adjacency)과 차수(degree) 검색

- *get_isolated (graph_id);*
  그래프 ID에서 독립된 vertex ID를 검색한다.

- *get_adjacent (graph_id, vertex_id);*
  그래프 ID의 vertex ID에 인접한 vertex ID를 검색한다.

- *get_degree (graph_id, vertex_id);*
  그래프 ID의 vertex ID에 대한 전체 차수를 검색한다.

- *get_indegree (graph_id, vertex_id);*
  그래프 ID의 vertex ID에 대한 입력 차수를 검색한다.

- *get_outdegree (graph_id, vertex_id);*
  그래프 ID의 vertex ID에 대한 출력 차수를 검색한다.

4.7 데이터 삭제 라이브러리 (data deletion)

데이터베이스에 존재하는 그래프 데이터를 삭제하는 라이브러리이다. 삭제 라이브러리는 그래프 ID, vertex ID, edge ID 그리고 각 속성이름을 이용하여 그래프 데이터를 삭제할 수 있다.

그래프 삭제는 그래프 ID를 이용하여 *graph_info* 테이블에 저장된 그래프 데이터를 삭제할 수 있다. 이 삭제 결과는 *vertex* 테이블, *edge* 테이블, *vertex_att* 테이블 그리고 *edge_att* 테이블에 적용되며 해당 그래프 ID에 소속되는 모든 데이터가 삭제된다.

vertex 삭제는 그래프 ID와 vertex ID를 이용하여 삭제 가능하다. vertex ID를 삭제할 경우 *vertex* 테이블과 *vertex_att* 테이블 그리고 *edge* 테이블에서 연결점 vertex로 존재하

는 해당 vertex에 대해 그 결과를 적용시킨다. vertex 속성 삭제는 그래프 ID와 vertex ID 그리고 해당 속성이름을 이용하여 *vertex_att* 테이블에 저장된 vertex 속성 데이터를 삭제할 수 있다.

edge 삭제는 그래프 ID와 edge ID를 이용하여 삭제 가능하다. edge ID를 삭제할 경우 edge 테이블과 *edge_att* 테이블에 존재하는 해당 edge에 대해 그 결과를 적용시킨다. edge 테이블의 연결점 속성과 방향성 속성은 단독으로 삭제가 불가능하며 edge ID와 함께 삭제된다. edge 속성 삭제는 그래프 ID와 edge ID, 그리고 해당 속성이름을 이용하여 *edge_att* 테이블에 저장된 edge 속성 데이터를 삭제할 수 있다.

1) 그래프 삭제
- *del_graph (graph_id);*
  데이터베이스에 저장된 그래프 중에서 주어진 그래프 ID를 삭제한다.

2) vertex와 vertex 속성 삭제
- *del_vertex (graph_id, vertex_id);*
  주어진 그래프 ID의 vertex ID를 삭제한다.
- *del_vertex_attribute (graph_id, vertex_id, vatt_name);*
  주어진 그래프 ID의 vertex ID에 대해서 해당속성을 삭제한다.

3) edge와 edge 속성 삭제
- *del_edge (graph_id, edge_id);*
  주어진 그래프 ID의 edge ID를 삭제한다.
- *del_edge_attribute (graph_id, edge_id, eatt_name);*
  주어진 그래프 ID의 edge ID에 대해서 해당 속성을 삭제한다.

4.8 복구와 완료 (recovery and commit)

복구 라이브러리는 잘못 수행한 작업을 작업수행 이전 상태로 복구하는 라이브러리이다. 복구위치(save point)를 지정하면 그 이후 작업에 대해서 잘못된 작업을 수행을 했을 경우 복구위치 이전 상태로 작업을 복구할 수 있다. 이 라이브러리는 갱신 라이브러리 및 삭제 라이브러리와 함께 사용하면 매우 유용하다.

완료 라이브러리는 작업 내용을 완료(commit)하는 라이브러리이다. 데이터베이스 연결을 해제할 때 묵시적으로 완료 작업을 수행하지만 명시적으로 완료할 수도 있다. 한번 완료된 작업에 대해서는 복구위치를 정하여도 그 이전 상태로의 복구가 불가능하다.

1) 복구
- *save_point ();*
  복구할 위치를 지정한다.
- *undo ();*
  설정된 복구위치 이전 상태로 복구한다.

2) 완료

- *commit ();*
  수행한 작업을 명시적으로 저장, 완료한다.

5. 결 론

그래프는 실세계에서 발생하는 문제들을 해결하는데 강력한 방법을 제공하므로 그래프의 체계적인 표현을 위한 데이터 구조와 효율적인 알고리즘들이 연구되어 왔다. 본 논문에서는 그래프를 데이터베이스에 저장하고 이것을 이용하는 그래프 라이브러리를 제안함으로써 효과적으로 그래프를 실세계에 응용할 수 있는 방법을 제안하였다. 즉, 그래프의 각 구성요소 및 정보를 테이블의 형태로 설계하고 이것을 데이터베이스화하여 그래프에 대한 생성, 갱신, 삭제, 검색 등의 연산을 정의하는 라이브러리를 제안하였다.

향후 연구과제는 그래프 라이브러리를 좀더 보강하고, 객체지향 데이터베이스(object-oriented database)를 이용한 그래프 라이브러리를 개발하여 장단점을 비교할 필요가 있다. 또한, 실제 응용분야에 적용하여 그 효율성을 검정하는 등의 연구가 필요하다.

참고문헌

[1] Sakti Pramanik and David Ittner, "Use of Graph-Theoretic Model for Optimal Relation Database Accesses to Perform Join," *ACM Transactions On Database System,* vol. 10, no. 1, pp. 57-74, March 1985.

[2] Alberto O. Mendelzon and Peter T. Wood, "Finding Regular Simple Paths In Graph Databases," *SIAM J. Comput.,* vol. 24, no. 6, pp. 1245-1258, December 1995.

[3] Joachuim Biskup, Uwe Räsch and Holger Stiefeling, "An Extension of SQL for Querying Graph Relations," *Computer Lang.*, vol. 5, no. 2, pp. 65-82, 1990.

[4] R. Guting, "GraphDB: Modeling and Querying Graphs in Database," *VLDB*, pp. 297-308, 1994.

[5] Lei Sheng, Z.M. Ôzsoyoğlu and G. Ôzsoyoğlu, "A Graph Query Language and Its Query Processing," *Proc. of the 15th IEEE Data Engineering*, pp. 572-581, 1999.

[6] James A. Mchug, *Algorithmic Graph Theory,* Prentice-Hall, 1998.