

工學碩士 學位論文

**Development Of a Graph Library
Using the Relational Database**

指導教授 朴 然 讚

2001年 2月

韓國海洋大學校 大學院

工學科 秋 仁 京

本 論 文 李 聖 大 工 學 碩 士 學 位 論 文 認 准

委 員 長 工 學 博 士 辛 沃 根 印

委 員 工 學 博 士 孫 周 永 印

委 員 工 學 博 士 朴 侏 讚 印

2001年 1月

韓 國 海 洋 大 學 校 大 學 院

工 學 科 秋 仁 京

Development Of a Graph Library Using the Relational Database

In Kyung Chu

Department of Computer Engineering, Korea Maritime University,
Pusan, Korea

Abstract

The graph provides a powerful methodology to solve a lot of real-world problems. There has been much research on graph representations and algorithms. But, there are still many difficulties to apply graphs to practical domains.

This paper proposes a graph library developed 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 database languages such as the SQL. Users of the graph library can efficiently manage a large amount of graph data. Furthermore, graphs saved in the database can be concurrently shared among many users. The proposed graph library will be useful to represent and solve real world problems efficiently.

1	1
1.1	1
1.2	3
1.3	5
2	6
2.1	6
2.2	8
3	13
3.1	13
3.1.1	14
3.1.2	18
3.2	20
3.2.1	21
3.2.2	24
4	27
4.1	28
4.2	/	28
4.3	28
4.4	29
4.5	30
4.6	31
4.7	33
4.8	34

5	35
5.1	35
5.2	40
6	42
	43
Appendix	45

가 .

(graph library)

. ,

(relational database)

가 가

가

가

가 [11].

(library)

가

가 .

가

.

vertex

, edge

attributes

vertex

, edge

(graph library)

(connect/disconnect

library), (graph creation library),
 (insertion library), (update library),
 (searching library), (deletion library)
 (recovery and commission library)

1.2

(object oriented database)[1-4]
 (relational database)[5-7]
 가 .
 Tony [1] (protocol verification)
 finite-state machine
 (transaction) global-state
 (logical error)
 (relational algebra) .

Joachim [2]

SQL

(genealogies), (graph relation)
 , (concept hierarchies),
 query SQL

가

Lei [5] (OQL:Object-oriented Query Language) 가 (Object-oriented Graph Data Model) GOQL(Graph Object-oriented Query Language) .

O-Algebra GOQL (paths) (sequence) 가 .

Guting [6]

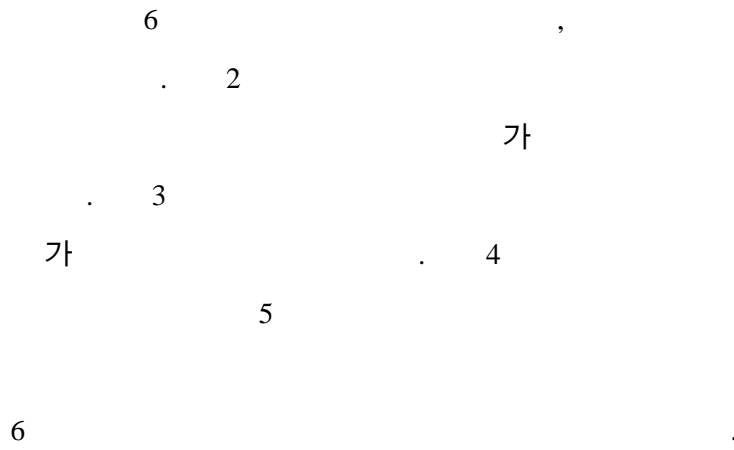
가

referencing objects object identity, attributes . [5] [6]

가 .

(DBMS : DataBase Management System) 가 (Relational Database)

1.3



2

가

2.1

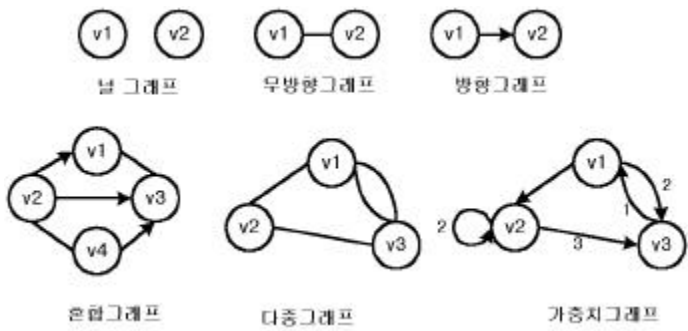
(graph

library)

[8- 19]. 2.1

Graph $G = (V, E)$ vertex
 $V(G)$ vertex edge $E(G)$
vertex edge
(undirected graph) edge가
가 (directed graph)가
edge vertex 가 , (v_i, v_j)
 (v_j, v_i) edge edge가
vertex vertex 가
edge vertex $\langle v_i, v_j \rangle$
 v_i edge (tail) v_j edge (head)

$\langle v_i, v_j \rangle$ $\langle v_j, v_i \rangle$ edge가 .
 (digraph) .
 (mixed graph) edge edge가 .
 edge edge .
 (multigraph) .
 edge 가 (weight)
 edge 가 (weighted graph)
 (network) . vertex
 vertex vertex (isolated vertex)
 vertex (null
 graph) vertex edge가
 (complete graph) .



2.1

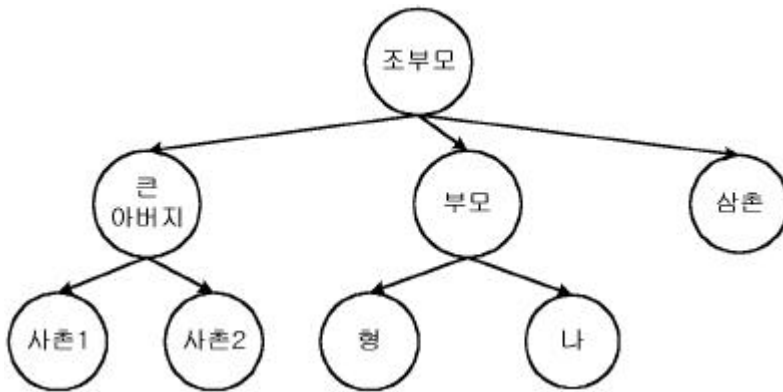
Fig. 2.1 The Examples Of Graph

edge $e \in E$ 가 vertex (v_i, v_j) vertex v_i v_j edge e
 (incident) edge
 vertex (adjacent) .
 (indegree) vertex
 edge (outdegree) vertex
 edge . vertex
 (total degree) .
 vertex edge .

2.2

[2]. vertex edge 가
 가 node 가 edge 가 가
 . 가
 .
 가 .
 1) 가
 가 vertex 가 edge
 . 2.2 가
 .
 - : .
 - : vertex

- : none.
- : ANCESTOR , DESCENDANT ,
RELATIONSHIP 가 binary .



2.2 가
Fig. 2.2 The Family Relationship

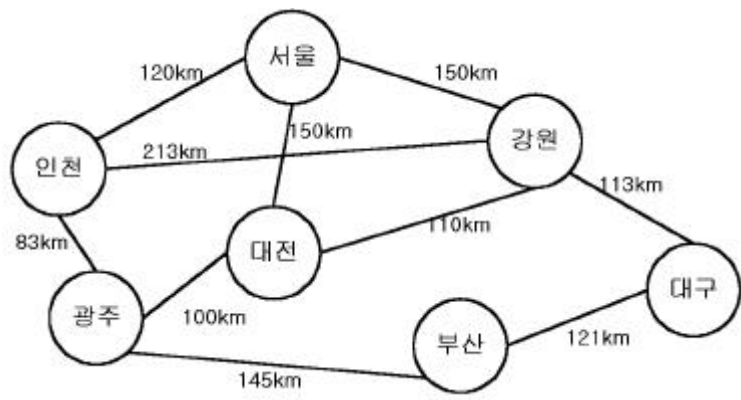
2)

가 2.3

- : 200

- : vertex

- :
 - : CITY, INHABITANIS, DISTANCE



2.3

Fig. 2.3 The Length Of Loads

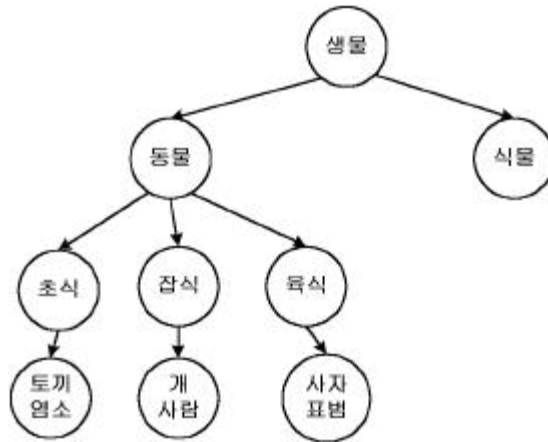
3)

2.4

- :
 - :

- : none.

- :

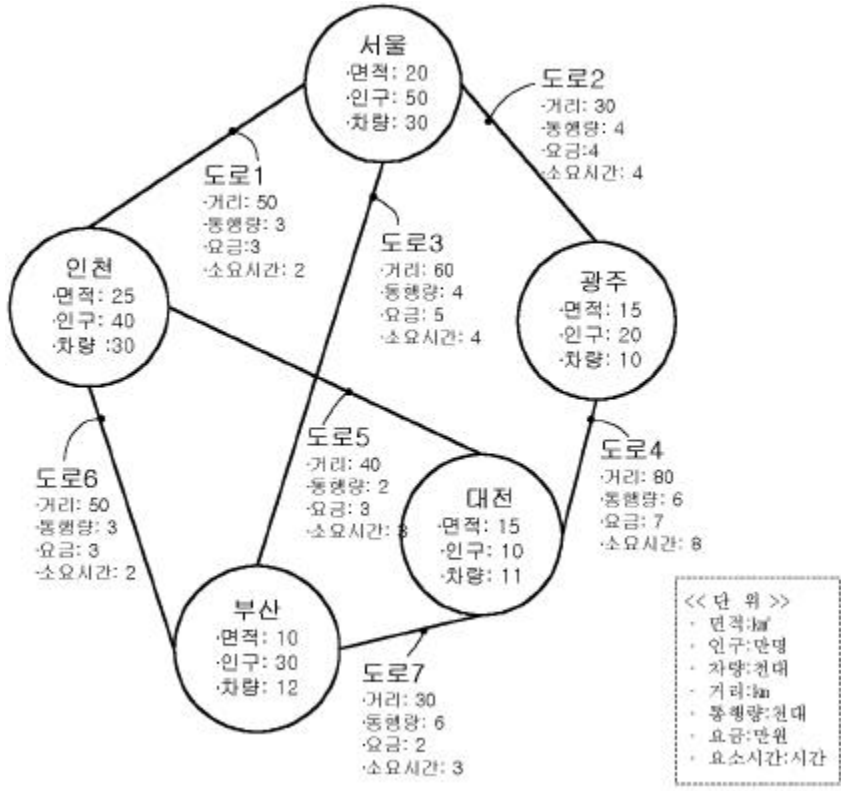


2.4

Fig. 2.4 The Hierarchies Of Animal Concept On Ecology

2.5

edge . vertex ,
가 가 vertex edge .



2.5

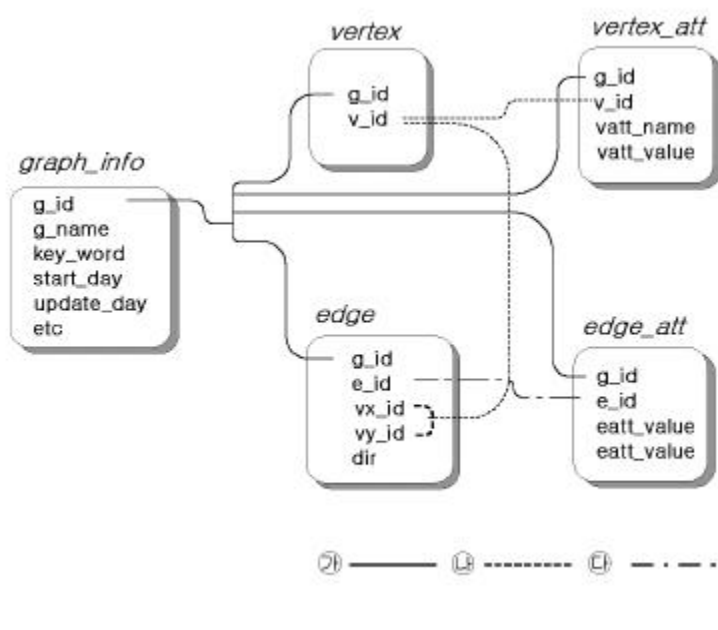
Fig. 2.5 The Relationship Of Cities And Roads

3

vertex
, edge
vertex edge

3.1

vertex edge
vertex edge
가 가
가
ID가 ID *graph_info*
ID . 3.1
ID *graph_info*
ID
vertex ID *vertex*
vertex ID *vertex_att* *edge*
edge ID *edge* edge
ID *edge_att*



3.1

Fig. 3.1 The Relationship Of Integrated Graph Tables

3.1.1

5 .
graph_info , *vertex* *vertex*
 , *edge* *edge* , *vertex* *edge* 가
vertex_att *edge_att* .

3.1

3.1

Table 3.2 The Structure Of Integrated Graph Tables

· *graph_info*

Attribute	Data Type	
<i>g_id</i>	VARCHAR	ID, PK(Primary key)
<i>g_name</i>	VARCHAR	
<i>key_word</i>	VARCHAR	
<i>start_day</i>	VARCHAR	
<i>update_day</i>	VARCHAR	
<i>etc</i>	VARCHAR	

· *vertex*

Attribute	
<i>g_id</i>	vertex <i>v_id</i> 가 ID, PK(Primary key)
<i>v_id</i>	vertex id , PK(Primary key)

· *edge*

Attribute	
<i>g_id</i>	edge <i>e_id</i> 가 ID, PK(Primary key)
<i>e_id</i>	edge ID, PK(Primary key)
<i>vx_id</i>	<i>e_id</i> 가 vertex <i>v1_id</i> <i>v2_id</i>
<i>vy_id</i>	
<i>dir</i>	· <i>true</i> / <i>y</i> : x → y , digraph · <i>false</i> / <i>n</i> : x — y, graph

· *vertex_att*

<i>A ttribute</i>	
<i>g_id</i>	v_id가 ID, PK(Primary key)
<i>v_id</i>	vertex ID, PK(Primary key)
<i>vatt_name</i>	vertex
<i>vatt_value</i>	vertex

· *edge_att*

<i>A ttribute</i>	
<i>g_id</i>	e_id가 ID, PK(Primary key)
<i>e_id</i>	edge ID, PK(Primary key)
<i>eatt_name</i>	edge
<i>eatt_value</i>	edge

1) *graph_info*

graph_info

가
 .
 ID(*g_id*),
 (*g_name*), (*key_word*),
 (*start_day*), (*update_day*)
graph_info 가
 가 (etc) .

2) *vertex*

vertex vertex
ID(*g_id*) vertex ID(*v_id*) 가 .
ID 가 vertex가
.

3) *edge*

edge edge
ID(*g_id*) edge ID(*e_id*) 가 .
가 edge가
. *vx_id* *vy_id* *e_id*가 vertex
vertex vertex .
(*dir*) .

4) *vertex_att*

vertex_att vertex 가
. ID(*g_id*) vertex ID(*v_id*),
(*vatt_name*) 가 . vertex
(*vatt_value*) 가
.

5) *edge_att*

edge_att edge 가
ID(*g_id*) edge ID(*e_id*),
(*eatt_name*) 가 . edge
(*eatt_value*) 가

3.1.2

2.5

vertex

가

edge

3.2 2.5

3.2

Table 3.2 The Application Of Integrated Graph Tables

· *graph_info*

<i>g_id</i>	<i>g_name</i>	<i>key_word</i>	<i>start_day</i>	<i>update_day</i>	<i>etc</i>
G1			05-DEC-00	15-DEC-00	
G2			20-OCT-99	30-OCT-99	
⋮	⋮	⋮	⋮	⋮	⋮

· *vertex*

<i>g_id</i>	<i>v_id</i>
G1	
G1	
⋮	⋮
G2	
⋮	⋮

· *edge*

<i>g_id</i>	<i>e_id</i>	<i>vx_id</i>	<i>vy_id</i>	<i>dir</i>
G1	1			n
G1	2			n
⋮	⋮	⋮	⋮	⋮
G2	1	air2	air4	y
⋮	⋮	⋮	⋮	⋮

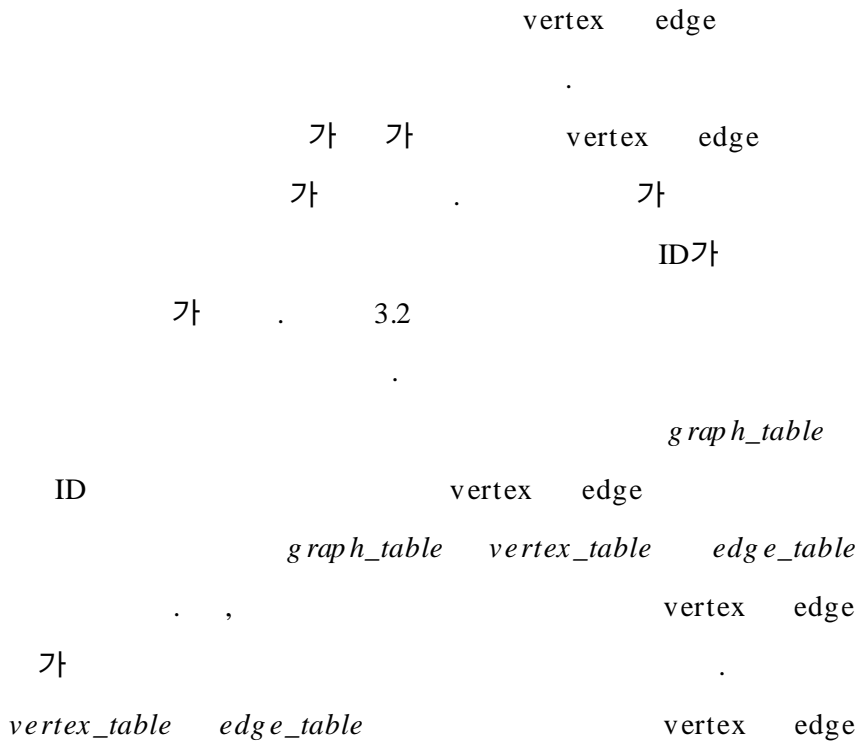
· *vertex_att*

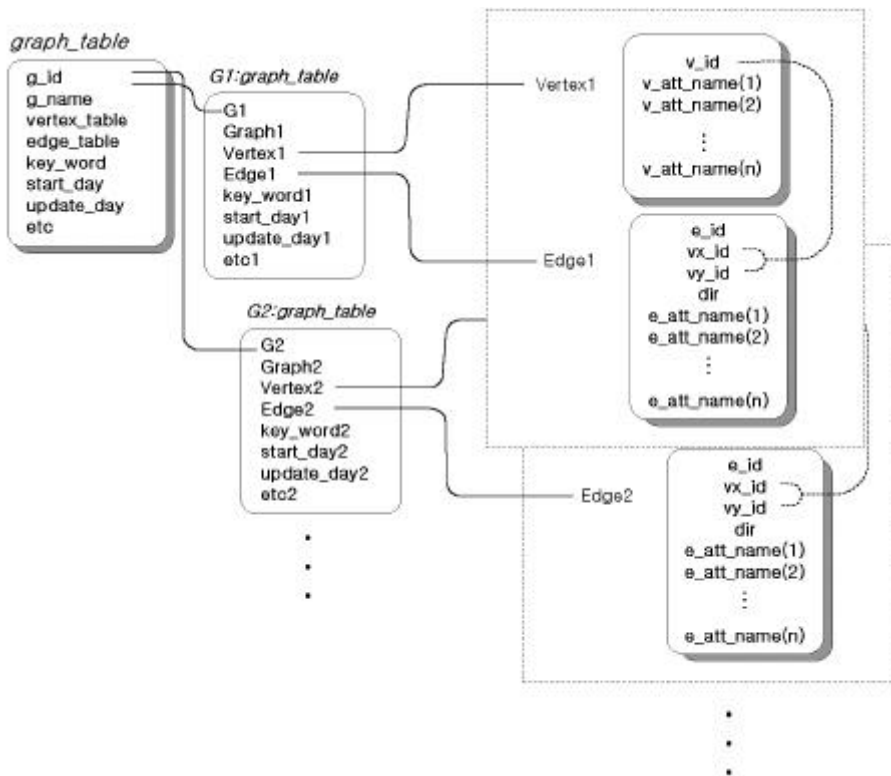
<i>g_id</i>	<i>v_id</i>	<i>vatt_name</i>	<i>vatt_value</i>
G1			20
G1			50
G1			30
G1			25
⋮	⋮	⋮	⋮

· *edge_att*

<i>g_id</i>	<i>e_id</i>	<i>eatt_name</i>	<i>eatt_value</i>
G1	1		50
G1	1		3
G1	1		3
G1	1		2
⋮	⋮	⋮	⋮

3.2





3.2

Fig. 3.2 The Relationship Of Separated Graph Tables

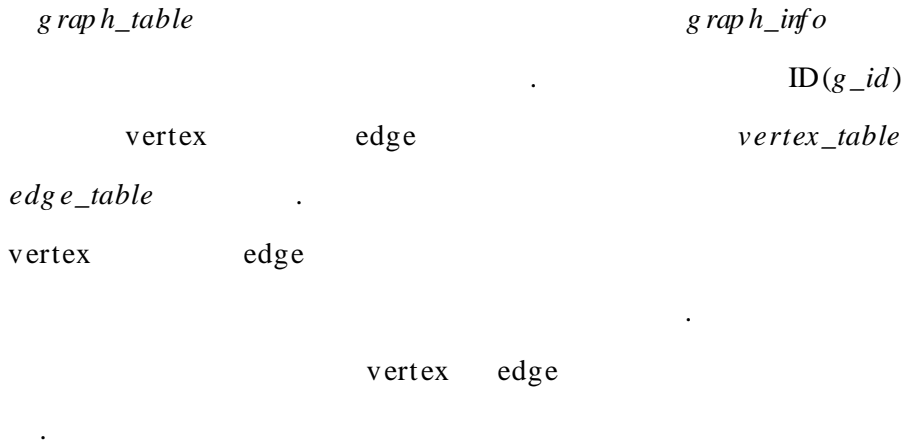
3.2.1

3

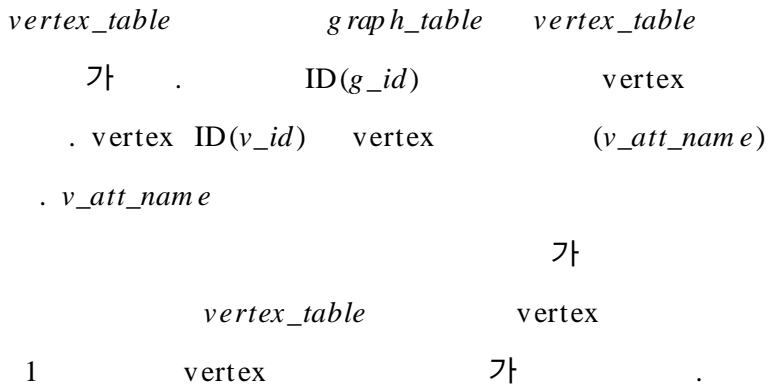
graph_table, vertex edge
vertex_table *edge_table*

3.2

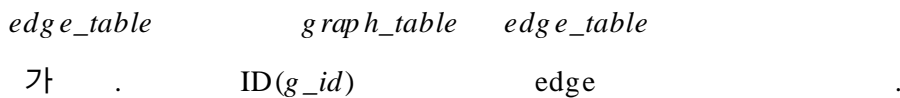
1) *graph_table*



2) *vertex_table*



3) *edge_table*



edge ID(*e_id*) *e_id*가 vertex *vx_id* *vy_id*
dir 가 . *edge_table*
vertex_table 가
가
edge_table edge
1 edge 가 .

3.2

Table 3.2 The Structure Of Separated Graph Tables

· **graph_table**

<i>Attribute</i>	Data Type	
<i>g_id</i>	VARCHAR	ID, PK(Primary key)
<i>g_name</i>	VARCHAR	
<i>vertex_table</i>	VARCHAR	vertex
<i>edge_table</i>	VARCHAR	edge
<i>key_word</i>	VARCHAR	
<i>start_day</i>	VARCHAR	
<i>update_day</i>	VARCHAR	
<i>etc</i>	VATCHAR	

· *vertex_table*

<i>Attribute</i>	
<i>v_id</i>	vertex ID, PK(Primary key)
<i>v_att_name(1)</i>	vertex (1)
<i>v_att_name(2)</i>	vertex (2)
⋮	⋮
<i>v_att_name(n)</i>	vertex (n)

· *edge_table*

<i>Attribute</i>	
<i>e_id</i>	edge ID, PK(Primary key)
<i>vx_id</i>	e_id가 vertex
<i>vy_id</i>	vx_id vy_id
<i>dir</i>	· <i>true</i> /y : x → y · <i>false</i> /n : x ← y
<i>e_att_name(1)</i>	edge (1)
⋮	⋮
<i>e_att_name(n)</i>	edge (n)

3.2.2

2.5

3.3

3.3

Table 3.3 The Application Of Separated Graph Tables

· *graph_table*

g_id	g_name	vertex_table	edge_table	key_word	start_day	update_day	etc
G1					05 -DEC- 00	15 -DEC- 00	
G2					20 -OCT- 99	30 -OCT- 99	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

v_id			
	20	50	30
	25	40	30
	15	20	10
	15	10	9
	10	30	12

e_id	vx_id	vy_id	dir				
1			n	50	3	3	2
2			n	30	4	4	4
3			n	60	4	5	4
4			n	80	6	7	8
5			n	40	2	3	3
6			n	50	3	3	2
7			n	30	6	2	3

가

가

가

가

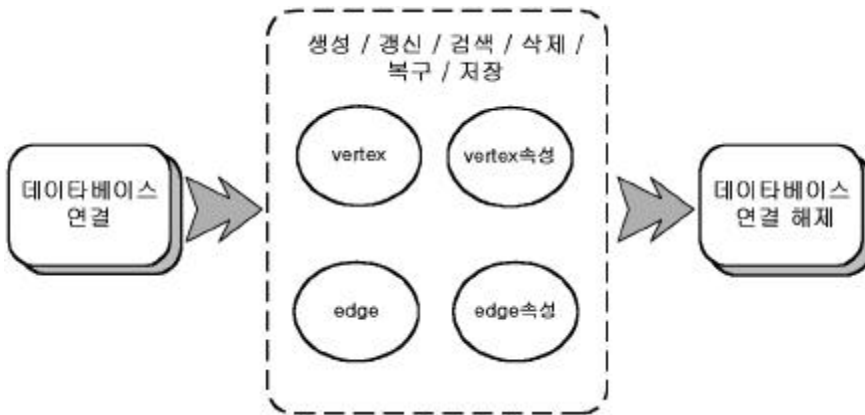
4

3.1

(graph library)

2

4.1



4.1

Fig. 4.1 The Flowchart Of Graph Library

, vertex, edge vertex edge

, , ,
가 .

4.1 (graph database creation)

3.1 *graph_info*, *vertex*
, *edge*, *vertex_att*, *edge_att*

4.2 / (graph database connection / disconnection library)

DBMS
. DBMS

4.3 (graph creation library)

DBMS

graph_info

$graph_info$ ID(g_id) (g_name)
 .
 가
 ID
 ID ID 가 .

4.4 (insertion library)

$graph_info$ vertex edge
 .
 vertex ID vertex ID
 $vertex$. vertex ID
 vertex ID vertex ID 가 .
 vertex 가 ID,
 vertex ID, vertex vertex
 $vertex_att$. vertex
 가 .
 vertex 가 .
 edge ID, edge ID, vertex
 $edge$.
 edge ID edge ID edge ID
 가 . vertex $vertex$
 vertex . edge 가
 ID, edge ID, edge edge
 $edge_att$. vertex

가
 edge
 가

4.5 (update library)

, vertex vertex , edge edge

ID(*old_graph_id*)
 ID(*new_graph_id*), (*new_graph_name*),
 (*new_keyword*), (*new_start*), (*new_update*)
 (*new_etc*)

ID
 ID *graph_info* ID가

ID(*new_graph_id*)

vertex , *edge* , *vertex_att* , *edge_att*

ID (*old_graph_id*) (*new_graph_id*)

graph_info

vertex ID, vertex ID(*old_vertex_id*)

vertex (*old_vatt_name*)

(join)

가

ID *graph_info*

vertex

edge

vertex

ID vertex ID

vertex

vertex_att

vertex

가

ID, vertex

ID

vertex

가

edge

ID edge ID

edge

edge_att

edge

가

ID, edge ID

edge

가

vertex vertex, vertex edge

가 . vertex

(adjacency searching library)

vertex

edge

(degree searching library)가 .

4.7 (deletion library)

. ID, vertex ID, edge ID .

ID *graph_info*

. *vertex* ,

edge , *vertex_att* *edge_att*

ID 가 .

vertex ID vertex ID 가 .

vertex ID *vertex* *vertex_att*

edge vertex vertex

. vertex ID vertex ID

(*vatt_name*) *vertex_att*

vertex .

edge ID edge ID 가 .

edge ID *edge* *edge_att*

edge . *edge*

가 가 edge ID

. edge ID edge ID,

(*eatt_name*) *edge_att* edge

.

(recovery library)

4.8

(recovery and commit library)

(save point)

(commit)

save point

가 가

5

3

4

5.1

2.5

가

'G1'

1)

```
connect('myfall', 'myfall');
```

2)

```
create_graph('G1', ' ', ' ', ' ', '05-DEC-00',  
            '15-DEC-00', '  
            ');
```

· *graph_info*

<i>g_id</i>	<i>g_name</i>	<i>key_word</i>	<i>start_day</i>	<i>update_day</i>	<i>etc</i>
G1		' '	05-DEC-00	15-DEC-00	

$G1$. vertex
 edge . 가

3) *insert_vertex*('G1', ' ');
insert_vertex('G1', ' ');
insert_vertex('G1', ' ');
insert_vertex('G1', ' ');
insert_vertex('G1', ' ');

· *vertex*

<i>g_id</i>	<i>v_id</i>
G1	
G1	
G1	
G1	
G1	

4) *insert_edge*('G1', ' 1', ' ', ' ', 'n');
insert_edge('G1', ' 2', ' ', ' ', 'n');
insert_edge('G1', ' 3', ' ', ' ', 'n');
insert_edge('G1', ' 4', ' ', ' ', 'n');
insert_edge('G1', ' 5', ' ', ' ', 'n');
insert_edge('G1', ' 6', ' ', ' ', 'n');
insert_edge('G1', ' 7', ' ', ' ', 'n');

5) *insert_vertex_attribute*('G1', ' ', ' ', 20');
insert_vertex_attribute('G1', ' ', ' ', 50');
insert_vertex_attribute('G1', ' ', ' ', 30');
insert_vertex_attribute('G1', ' ', ' ', 25');
insert_vertex_attribute('G1', ' ', ' ', 40');
insert_vertex_attribute('G1', ' ', ' ', 30');

```

6) insert_edge_attribute('G1', '1', '1', '50');
   insert_edge_attribute('G1', '1', '2', '3');
   insert_edge_attribute('G1', '1', '3', '3');
   insert_edge_attribute('G1', '1', '4', '2');
   insert_edge_attribute('G1', '2', '1', '30');
   insert_edge_attribute('G1', '2', '2', '4');
   insert_edge_attribute('G1', '2', '3', '4');
   insert_edge_attribute('G1', '2', '4', '4');

```

· *edge*

<i>g_id</i>	<i>e_id</i>	<i>vx_id</i>	<i>vy_id</i>	<i>dir</i>
G1	1			n
G1	2			n
G1	3			n
G1	4			n
G1	5			n
G1	6			n
G1	7			n

· *vertex_att*

<i>g_id</i>	<i>v_id</i>	<i>vatt_name</i>	<i>vatt_value</i>
G1			20
G1			50
G1			30
G1			25
G1			40
G1			30

· *edge_att*

<i>g_id</i>	<i>e_id</i>	<i>eatt_name</i>	<i>eatt_value</i>
G1	1		50
G1	1		3
G1	1		3
G1	1		2
G1	2		30
G1	2		4
G1	2		4
G1	2		4

1) 6) 가

'G1'

가

7) 가 55 가

`update_vertex_attribute('G1', ' ', ' ', '55');`

<i>g_id</i>	<i>v_id</i>	<i>vatt_name</i>	<i>vatt_value</i>
⋮	⋮	⋮	⋮
G1			55
⋮	⋮	⋮	⋮

8) 2 .

`get_edge_connection('G1', ' 2');`

`==>` ,

9) .

`del_vertex('G1', ' ');`

· vertex

<i>g_id</i>	<i>v_id</i>
G1	
G1	
G1	
G1	

· edge

<i>g_id</i>	<i>e_id</i>	<i>vx_id</i>	<i>vy_id</i>	<i>dir</i>
G1	4			n
G1	5			n
G1	6			n
G1	7			n

· vertex_att

<i>g_id</i>	<i>v_id</i>	<i>vatt_name</i>	<i>vatt_value</i>
G1			25
G1			40
G1			30
G1			15
G1			20
G1			10

· edge_att

<i>g_id</i>	<i>e_id</i>	<i>eatt_name</i>	<i>eatt_value</i>
G1	4		80
G1	4		6
G1	4		7
G1	4		8
G1	5		40
G1	5		2
G1	5		3
G1	5		3


```

while(key !=ESC) /* edge */
{
    scanf("%s %s %s %s",graph_id,edge_id,eatt_name,eatt_value);
    insert_vertex_attribute(graph_id,edge_id,eatt_name,eatt_value);
}

/*      ID g1      vertex ID      */
result=get_all_vertex("g1");
while(row=graph_fetch_row(result)) {
    printf(" vertex list : %s", row);
}
}

```

```

< >
vertex list :

```

6

가

가

- [1] Tony T, Lee and Ming-Yee Lai, "A Relational Algebraic Approach to Protocol Verification," IEEE Transactions on Software Engineering, Vol.14, No.2, pp.184-193, February 1998.
- [2] Joachuim Biskup, Uwe Räsch and Holger Stiefeling, "An Extension of SQL for Querying Graph Relations," Computer Language, Vol.15, No.2, pp.65-82, 1990.
- [3] 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.
- [4] Alberto O. Mendelzon and Peter T. Wood, "Finding Regular Simple Paths In Graph Databases," SIAM J. Compute, Vol.24, No.6, pp.1245-1258, December 1995.
- [5] Lei Sheng, Z.M. Özsoyoğlu and G. Özsoyoğlu, "A Graph Query Language and Its Query Processing," the 15th IEEE Data Engineering, pp.572-581, 1999.
- [6] Guting,R., "GraphDB: Modeling and Querying Graphs in Database," VLDB, pp.297-308, 1994.
- [7] Michael V. Mannino and Leonard D. Sharpiro, "Extensions to Query Languages for Graph Traversal Problems," IEEE Transactions on Knowledge and Data Engineering, Vol.2, No.3, pp.353-363, September 1990.

- [8] Robin J.Wilson, John J.Watkins, *Graphs an Introductory Application*, John Wiley & Sons.Inc, 1990.
- [9] James A. Mchugh, *Algorithmic Graph Theory*, Prentic-Hall Inc. 1998.
- [10] Ronald Gould, *Graph Theory*, The Benjamin/Cummings Publishing Company.Inc, 1988.
- [11] Abraham Silverschatz, Henry F.Korth and S. Sodarshan, *Database System Concepts*, The McGraw-Hill Companies Inc, 1997.
- [12] Jayavel S, Kristin T, Gang H, Chun Z, David D and Jeffrey N, "Relational Database for Querying XML Documents: Limitations and Opportunities," the 25th VLDB Conference, pp.302-314, 1999.

Appendix

- 1** (graph database creation)
- 2** /
(graph database connection / disconnection library)
- 3** (graph creation library)
- 4** (insertion library)
- 5** (update library)
- 6** (searching library)
- 7** (deletion library)
- 8** (recovery and commit library)

1. (graph database creation)

- *create_graph_db()*;
graph_info, vertex, edge
, vertex_att, edge_att.

2. / (graph database connection / disconnection library)

- *connect(char *id, char *passwd)*;
(id) (passwd)

- *disconnect()*;
connect

3 (graph creation library)

- *create_graph(char *graph_id, char *graph_name, char *key,*
*char*start, char *update, char *description)*;

. ID, , ,
, , 가 가 .

4 (insertion library)

- *insert_vertex(char *graph_id, char *vertex_id);*
vertex . vertex (graph_id)
vertex(vertex_id) 가 .

- *insert_vertex_attribute(char *graph_id, char *vertex_id, char *vatt_name, char *vatt_value);*
vertex . (graph_id) vertex
(vertex_id), 가 .

- *insert_edge(char *graph_id, char *edge_id, char *vertex_xid, char *vertex_yid, char *direction);*
edge . edge (graph_id)
edge(edge_id) (vertex_xid, vertex_yid)
(direction) 가 .

- *insert_edge_attribute(char *graph_id, char *edge_id, char *edge_attname, char *edge_attvalue);*
edge . (graph_id) edge
(edge_id), 가 .

5 (update library)

1)

- *update_graph_information(char *old_graph_id,*

```

char *new_graph_id, char *new_g_name,
char *new_key_word, char *new_start,
char *new_update, char *new_etc);
(old_graph_id) ID(new_
graph_id), , , , .

- update_graph_id(char *old_graph_id, char *new_graph_id);
ID(old_graph_id) (new_graph_id)
.

- update_graph_name(char *graph_id, char *new_graph_name);
.

- update_graph_keyword(char *graph_id, char *new_keyword);
.

- update_graph_startday(char *graph_id, char *new_day);
.

- update_graph_up_eateday(char *graph_id, char *new_day);
.

- update_graph_comment(char *graph_id, char *new_etc);
.

```

2) vertex vertex

- *update_vertex_information(char *graph_id, char *old_vertex_id, char *new_vertex_id, char *old_vatt_name, char *new_vatt_name, char *new_vatt_value);*

vertex(*old_vertex_id*) vertex ID(*new_vertex_id*),

- *update_vertex_id(char *graph_id, char *old_vertex_id, char *new_vertex_id);*

vertex ID(*old_vertex_id*) (*new_vertex_id*)

- *update_vertex_atname(char *graph_id, char *vertex_id, char *old_vatt_name, char *new_vatt_name);*

vertex

- *update_vertex_attvalue(char *graph_id, char *vertex_id, char *vatt_name, char *new_vatt_value);*

vertex (*vatt_name*) vertex

3) edge edge

- *update_edge_information(char *graph_id, char *old_edge_id, char *new_edge_id, char *new_vx_id, char *new_vy_id, char *new_direction,*

```

        char *old_edge_attname,
        char *new_edge_attname,
        char *new_edge_attvalue );
edge(old_edge_id)                                edgID(new_edge_id),
, , .

- update_edge_id(char *graph_id, char *old_edge_id, char
    *new_edge_id);
edge ID(old_edge_id)                            (new_edge_id)
.

- update_edge_connection(char *graph_id, char *edge_id,
    char *new_vx_id, char *new_vy_id);
edge .

- update_edge_direction(char *graph_id, char *edge_id, char
    *new_dir);
edge .

- update_edge_attname(char *graph_id, char *edge_id, char
    *old_edge_attname, char *new_edge_attname);
edge .

- update_edge_attvalue(char *graph_id, char *edge_id,
    char *edge_attname, char *new_edge_attvalue);
edge (eatt_name) edge
.

```

6 (searching library)

1) fetch

- *graph_fetch_row(char *result);*
() 가

2)

- *get_all_graph();*
ID .

- *get_all_graph_name();*
.

- *get_graph_information(char *graph_id);*
ID(*graph_id*)
(, ,)

- *get_graph_name(char *graph_id);*
ID(*graph_id*) .

- *get_graph_startday(char *graph_id);*
ID(*graph_id*)
.

- *get_graph_updateday(char *graph_id);*
ID(*graph_id*)

- *get_graph_comment(char *graph_id);*
ID(*graph_id*)

3) vertex vertex

- *get_all_vertices(char *graph_id);*
ID(*graph_id*) vertex ID

- *get_vertex_information(char *graph_id, char *vertex_id);*
ID(*graph_id*) vertex ID(*vertex_id*)
(vertex , vertex)

- *get_all_vertex_attributes(char *graph_id, char *vertex_id);*
ID(*graph_id*) vertex ID(*vertex_id*)

- *get_vertex_attribute(char *graph_id, char *vertex_id, char *vatt_name);*
ID(*graph_id*) vertex ID(*vertex_id*)

4) edge edge

- *get_all_edges(char graph_id);*
ID(graph_id) edge ID

- *get_edge_information(char *graph_id, char *edge_id);*
ID(graph_id) edge ID(edge_id)
(edge , , edge , edge)

- *get_all_edge_attributes(char *graph_id, char *edge_id);*
ID(graph_id) edge ID(edge_id)

- *get_edge_attribute(char *graph_id, char *edge_id, char *eatt_name);*
ID(graph_id) edge ID(edge_id)

- *get_edge_connection(char *graph_id, char *edge_id);*
ID(graph_id) edge ID(edge_id)가
vertex ID

- *get_edge_xyconnection(char *graph_id, char *vx_id, char*

```

        *vy_id);
        ID(graph_id)    vertex vx    vy
    edge ID(edge_id)    .

```

```

- get_edge_direction(char *graph_id, char *edge_id);
        ID(graph_id)    edge ID(edge_id)
    .

```

5) (adjacent) (degree)

```

- get_isolated(char *graph_id);
        ID(graph_id)    vertex ID
    .

```

```

- get_adjacent(char *graph_id, char *vertex_id);
        ID(graph_id)    vertex ID(vertex_id)
    vertex ID
    .

```

```

- get_degree(char *graph_id, char *vertex_id);
        ID(graph_id)    vertex ID(vertex_id)
    .

```

```

- get_indegree(char *graph_id, char *vertex_id);
        ID(graph_id)    vertex ID(vertex_id)
    .

```

- *get_outdegree(char *graph_id, char *vertex_id);*
ID(*graph_id*) vertex ID(*vertex_id*)

7 (deletion library)

1)

- *del_graph(char *graph_id);*

ID .

2) vertex vertex

- *del_vertex(char *graph_id, char *vertex_id);*
ID(*graph_id*) vertex ID(*vertex_id*)

- *del_vertex_attribute(char *graph_id, char *vertex_id, char *vatt_name);*
ID(*graph_id*) vertex ID(*vertex_id*)

3) edge edge

