CHAPTER-4

Designing Graph Database Models from Existing Relational Databases
CHAPTER-4

DESIGNING GRAPH DATABASE MODELS FROM EXISTING RELATIONAL DATABASES **

4.1 INTRODUCTION

Graphs and hyper graphs are extensively used to describe various concepts in database technology- ranging from data modeling to data processing. There are many graph based approaches applied in RDBMS [24][38][40]. Recently the Graph Database is a hot topic of research in database technology [5][42][43][45]. Graph database is becoming popular because it addresses a set of complex data related application domains such as WWW, social networking, genomics [40][41][42][43][46][48]. Hyper graphs and graphs are very much used in describing graph database structures, the complex interactions among the data [38][47]. The interactions among the data items forms a complex graph and the prime objective of graph database is to find a certain pattern(sub graph) in the complex graph using graph theory and graph algorithms. It is difficult to manage the complex interactions in relational databases and studies show that graph database is better than any RDBMS [40][44].

In this chapter, few concepts of graph oriented design methodologies used in DBMS are discussed. A novel method to construct graph database models from existing relational and object oriented database is proposed. In this approach, the dependency graphs for the entities in the system are transformed into star graphs. This star graph model is transformed into a hyper graph model for the relational database, which, in turn, can be used to develop the domain relationship model that can be converted in to a

** This chapter has been published as a research paper in the journal “International Journal of Computer Applications”(IJCA), (0975 – 8887) Volume 74 ,No.1, July, 2013, pp. 25-31.
graph database model. For discussion, the SUPPLIER-PART database [39] is considered.

4.2 GRAPH BASED APPROACH IN DATABASE DESIGN

Figure 4.1 shows the dependency graph for the supplier-part database. Considering the dependencies available among the attributes of the entities, the candidate keys are identified. For example, there are two candidate keys (S# and SNAME) for the entity SUPPLIER and one candidate key (P#) for the entity PART.

![Dependency Graph for SUPPLIER-PART database](figure4.1)

Choosing S# as the primary key for SUPPLIER and P# for PART, the star graphs for these entities as shown in fig. 4.2 are obtained. This analysis will determine the attributes from the entities SUPPLIER and PART which will be included in the relationship set (intersecting portions marked as X in fig. 4.1).

![Star Graphs for SUPPLIER and PART](figure4.2)

*Figure 4.1: Dependency Graph for SUPPLIER-PART database*

*Figure 4.2: Star Graphs for SUPPLIER and PART*
Finally, a dependency graph for the relationship SHIPMENT as shown in fig. 4.3 (a) is obtained. From this dependency graph, the star graph for SHIPMENT can be drawn as shown in fig. 4.3 (b). Finally, the following reference graph for the database as shown in fig. 4.4 is obtained.

![Diagram](image)

**Figure 4.3: Dependency Graph and the Star Graph for SHIPMENT**

![Diagram](image)

**Figure 4.4: Reference Graph for SUPPLIER PART Database**

![Diagram](image)

**Figure 4.5: Hyper Graph Model for SUPPLIER-PART Database**
4.3 GRAPH DATA MODEL USING TUPLE ORIENTED DEPENDENCY

The graph structure of the database can be deduced from the mathematical model [24] of the database. If $x_i$ be the attributes in the key set, $k_S$ of the set $S$ representing a database table, then the central node of the star graph will contain $x_i$. The mathematical model for the SUPPLIER-PART database is as follows.

\[
\text{DRS NAME} = \text{SUPPLIER-PART}
\]

\[
\text{SET SUPPLIER} = \{\{S\#\},\{\text{SNAME}\},\{\text{STATUS}\}, \{\text{CITY}\}\}
\]

\[
\text{SET PART} = \{\{P\#\},\{\text{PNAME}\},\{\text{COLOR}\}, \{\text{WEIGHT}\}, \{\text{CITY}\}\}
\]

\[
\text{SET SP} = \{\{S\#\},\{P\#\},\{\text{DATE}\}, \{\text{QTY}\}\}
\]

To find the relationships among these sets (tables) take the intersection of the sets as follows.

\[
\text{SUPPLIER} \cap \text{PART} = \emptyset \quad \text{(Eq. 4.1)}
\]

\[
\text{SHIPMENT} \cap \text{SUPPLIER} = \{\{S\#\}\} \quad \text{(Eq. 4.2a)}
\]

\[
\{S\#\} \in k_S \quad \text{(Eq. 4.2b)}
\]

\[
\text{SHIPMENT} \cap \text{PART} = \{\{P\#\}\} \quad \text{(Eq. 4.3a)}
\]

\[
\{P\#\} \in k_P \quad \text{(Eq. 4.3b)}
\]

[Note that, Equation 4.2(a) and Equation 4.3(a) can be obtained from the reference graph shown in fig. 4.4]

From this it is found that there is a relationship (m: n) between SUPPLIER and PART through SHIPMENT. The central node of the star graph for SHIPMENT will contain $S\#$, $P\#$, $\text{DATE}$ (see fig. 4.3 (b)). Therefore, a graph (hyper graph) representation of the database (see fig. 4.5) can be obtained from the mathematical model of the database. To convert the database to a graph (hyper graph) model, the mathematical model of the database is required.
This mathematical model can also be obtained from the database tables in the system by using a reverse engineering approach [24] or from the schema diagram. Consider the following relational data model instance of the SUPPLIER-PART database.

<table>
<thead>
<tr>
<th>Table 4.1: SUPPLIER</th>
<th>Table 4.2: PART</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S#</strong></td>
<td><strong>SNAME</strong></td>
</tr>
<tr>
<td>S1</td>
<td>RAM</td>
</tr>
<tr>
<td>S2</td>
<td>HARI</td>
</tr>
<tr>
<td>S3</td>
<td>BOB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>P#</strong></th>
<th><strong>PNAME</strong></th>
<th><strong>COLOR</strong></th>
<th><strong>WEIGHT</strong></th>
<th><strong>CITY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>NUT</td>
<td>RED</td>
<td>12</td>
<td>KOLKOTTA</td>
</tr>
<tr>
<td>P2</td>
<td>BOLT</td>
<td>GREEN</td>
<td>15</td>
<td>CHENNAI</td>
</tr>
<tr>
<td>P3</td>
<td>SCREW</td>
<td>BLUE</td>
<td>13</td>
<td>MUMBAI</td>
</tr>
<tr>
<td>P4</td>
<td>BOLT</td>
<td>RED</td>
<td>15</td>
<td>CHENNAI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4.3: SHIPMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S#</strong></td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>S3</td>
</tr>
</tbody>
</table>

In fig. 4.7, nodes are created for each attribute values in the database instance. All nodes in this model are simple nodes. The hyper edges can also store information. The suppliers and the parts are connected by shipments, so the hyper edges can be labeled as shown in fig. 4.8. In this model, S1, S2, S3 are nodes representing star graphs for
SUPPLIERS. P1, P2, P3, P4 that represent the nodes for PARTs. Moreover, the tuple oriented dependency is considered here. The attribute level dependencies are not considered.

Figure 4.7: Hyper Graph Model for SUPPLIER-PART Database Instance using Tuple Oriented Dependencies.
Domain dependency is the dependency among the domain values. In this representation, number of nodes is substantially reduced, hence the complexity is reduced. A very small number of values from a domain are used several times in the database tables. The dependencies in the star graph and the relationships among the star graphs are also described by using the domain dependencies. So, a domain relationship diagram can be drawn to describe a relational database. The graph data model of fig. 4.7 is represented using domain oriented dependencies as shown in fig. 4.9. The domain relationship model for the SUPPLIER-PART database is represented in fig. 4.10. The relationships among the domains of the attributes are considered. The prime focus is now the domains in the system, not the entities. Entities share domains. For example, as shown in fig. 4.10, entities SUPPLIER and PART share the same domain (CITY) values. Therefore, graph database model can be created from existing relational database by considering the relationships among domains in the system. The
dependencies among the domains in the star graph can be 1: n or n: 1. For example, one CITY can be related to may S# or any number of P#. So, in the database instance for the SUPPLIER-PART database, only one node for one CITY value “KOLKOTA” will be created and there may be many edges to connect it to many nodes for different S#-values and many different P# -values (See fig. 4.11). For an m: n relationship, a hyper edge is created connecting the m-side key node, the n-side key node and the other key nodes to any dependent node.

Figure 4.9: Hyper Graph Model for SUPPLIER-PART database Instance using Domain Oriented Dependencies.
Figure 4.10: Domain Relationship Model for SUPPLIER-PART Database

Note that, for the composite keys in the relational database, hyper edges are created to connect the nodes for the attributes (domains) in the composite keys to any node for dependent attributes (domains). Therefore, the domain relationship model can be represented as a hyper graph (see fig. 4.12). If a graph model for a database instance of SUPPLIER-PART database is created, then this model will follow the structure as shown in fig. 4.12.

Figure 4.11: Connection of KOLKOTTA to three different S# values and P# values

Figure 4.12: Hyper graph model for Domain Relationship Diagram for SUPPLIER-PART Database
By considering the domain dependencies and creating only one node for a particular
domain value appearing several times in a table (or in many tables), the data retrieval
speed is enhanced and reduces the extra processing of node data as well. For example,
consider the graph data model in fig. 4.8 and fig. 4.9. In the data model in fig. 4.8 a
query to find all suppliers and parts in KOLKOTTA needs seven node accesses and lots
of processing of node information. But in the data model in fig. 4.9, this query needs to
locate the first supplier (links in the look-up index for S#) node which has a relationship
to the node “KOLKOTTA” and once the node “KOLKOTTA” is located, all S#s and
P#s can be located in constant time. This is because it is assumed that a node in the
dominating factor in the search is the time required to find the first node having the
search-key or the node in the look-up index which has a relationship to the node having
the search-key. In this case, the node for the search key “KOLKOTTA” is located
through the first node in the look-up for S#. The number of node accessed is 4 in the
model with domain oriented dependency, whereas, the number of node accessed is 7 in
the model with tuple oriented dependency [see Table 4.4].

In the next section, a method to create a graph model for the database instance of a
relational database is discussed.

4.5 CREATING GRAPH MODEL FOR A RELATIONAL DATABASE INSTANCE

In this approach, the tables in the database are read once for a particular period
(session) say for a day and a graph model is created for these data. This graph is stored
in the server’s memory. All subsequent transactions are performed on this graph model.
Adding data to the tables requires creating nodes for the domain values and creating
edges to connect these nodes accordingly. If a node for the domain value is already in the graph, it is required to create only the appropriate edge in the graph to represent the relationship.

4.5.1 Creating the Domain Relationship Diagram

From the mathematical model, the cardinality of the relationships cannot be found directly. But the cardinality information is hidden in the model.

Let A and B are two set representing two tables in the database. If \( A \cap B = b_i \) such that \( b_i \in B \) and \( b_i \) does not belong to \( K_A \), where \( K_A \) is the key set of A, then the cardinality of the relationship between A and B is 1:1 or 1:n. Therefore the relationship between \( a_j \) and \( b_i \) is assumed to be 1: n. Now, let us consider a set C (table) with \( k_A \) and \( k_B \), where \( k_A \) and \( k_B \) are the keys of A and B. If \( k_A \) and \( k_B \) does not belong to \( K_C \) then there are two relationships, one between A and C and the other between B and C which are 1: n, i.e. C is related with both A and B. If \( k_A \in K_C \) and \( k_B \in K_C \), then there is one m:n relationship between A and B. So, for independent set or the set having a 1: n relationship with other set, a 1: n relationship from each describing attribute to the key attribute in the set is created. For a derived set create an m: n relationship between the foreign keys of the sets. For all attributes in the key set of the derived set, create m: n relationship to the relationship created between the foreign keys. For other attributes in the describing set of the derived set, create n: 1 relationship from the relationship created between the foreign keys.

4.5.2 Algorithm for Creating Domain Relationship model

Step 1: Develop an E-R model by using reverse engineering approach. [24]

Identify the unique domains for the attributes.
Step 2: Develop the Domain Relationship Model

a) For the entities create relationships (1: n) from the key attribute to all describing attributes. For entities having composite key, an n-ary relationship is obtained.

b) For n:1 relationship in the E-R model, create an 1:n relationship from the key of n-side entity to the key of the 1-side entity. For m:n relationship in the E-R model, create m:n n-ary relationship from the keys of the participating entities in the m:n relationship and other distinguishing attributes in the m:n relationship to the describing attributes of the m:n relationship.

![Diagram](image)

*Figure 4.13: (a) Graph models for Supplier S1 (b) A shipment of Part P3 by Supplier S3*

4.5.3 Algorithm for Creating Graph Database Model from the Table data

Step 1: Read the Table data and find the distinct domain values in the database.

Step 2: Create nodes for these distinct domain values.
Step 3: Create edges for the relationship following the Domain Relationship Diagram.

For each row in a Table create edges from the domain value for the key in the Table to other non-key domain values and label the edge with the attribute names as shown in fig. 4.13(a). If the Table has composite key then create a hyper edge to connect the key domain values to other non-key domain values as shown in fig. 4.13(b).

[Fig. 4.18 shows the complete graph data model created by following the Algorithm 4.5.3.]

4.6 QUERYING DATA FROM THE GRAPH MODEL

The graph data model for the SUPPLIER-PART database basically consists of four graph patterns G1, G2, G3 and G as shown in fig. 4.15. Querying from the database is now equivalent to mining sub graphs from the graph model.

![Figure 4.14: Four Sub Graphs in SUPPLIER-PART Graph Database](image)

4.6.1 Get all supplier details.

This query is equivalent to matching the graph pattern (G1) of fig. 4.15 from the graph data model.

![Graph Pattern G1](image)
4.6.2 Get all parts details.

This query is equivalent to matching the graph pattern (G2) of fig. 4.16 from the graph data model.

![Figure 4.16: Sub Graph G2 of the graph in fig. 4.14](image)

4.6.3 Get all shipments details.

This query is equivalent to matching the graph pattern (G3), shown in fig. 4.17, from the graph data model.

![Figure 4.17: Sub Graph G3 of the Graph in fig. 4.9](image)

<table>
<thead>
<tr>
<th>APPROACH</th>
<th>SUPPLIER (A)</th>
<th>PART (B)</th>
<th>SHIPMENT (C)</th>
<th>TOTAL (A+B+C)</th>
<th>SUPPLIER-PART DATABASE (D)</th>
<th>TO FIND ALL S# AND P# RELATED TO &quot;KOLKOTA&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuple Oriented</td>
<td>12</td>
<td>20</td>
<td>18</td>
<td>50</td>
<td>44</td>
<td>6</td>
</tr>
<tr>
<td>Domain Oriented</td>
<td>10</td>
<td>16</td>
<td>11</td>
<td>37</td>
<td>30</td>
<td>7</td>
</tr>
</tbody>
</table>
4.7 COMPARISON OF THE APPROACHES

A comparison between Tuple Oriented Dependency model and Domain Oriented Dependency model is shown in Table 4.4. The last column(8) of the table shows the number of nodes accessed to find all suppliers in KOLKOTTA and all parts produced in KOLKOTTA in the respective models. The number of nodes required to model the data in SUPPLIER in tuple oriented model is 12 and 10 in the domain oriented model. If the complete database is modeled using tuple oriented approach the number of nodes is 44 whereas the number of nodes is 30 in domain oriented approach for the complete database (see column 6 in Table 4.4).
4.8 CONCLUSION

In this chapter, an approach to design graph database model from existing relational databases is discussed. This approach can be used to design new graph databases models. This approach can also be applied to derive graph database models from existing object-oriented databases. This approach can also be automated. It is implemented by using Neo4j graph database. (See Appendix-I)