International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 9 (2015) pp. 23207-23221

http://www.ripublication.com

A Tool To Convert E-R Diagram To Property Graph Database

Dimple Chehal[1], Hanu Bhardwaj[2]
[1] *Manav Rachna College of Engineering/Department of Computer Science & Engineering, Faridabad, India*
Email: dimplechehal@gmail.com
[2] *Manav Rachna College of Engineering/Department of Computer Science & Engineering, Faridabad, India*
Email: hanu.mrce@mrei.ac.in

Abstract

In order to move to graph databases, which are very capable to handle big data, with E-R diagram as the base model, an organization need not first design a relational database and then migrate to a graph database, instead it should be able to directly map the E-R diagram to an equivalent graph database design. This is achieved by designing and implementing a tool that automates the same. The tool makes the transition from one representational scheme to another easier by taking entities, relationships, cardinality of the relationships involved in the E-R, attributes of the entities and the relationships as input and generating the design of the corresponding graph database. The tool is advantageous to use in case where an organization wants to build its graph database from the scratch using E-R diagram as the base model.

Index Terms: Cardinality, Database, Entity, Graph, Relationship

Introduction

In today's times, Graphs are being widely used to represent highly connected networks. They form ideal structures to represent data as using them any scenario can be modeled easily, for instance, social networks. Property graph is one such variant of Graph model that helps us to easily understand the dense and interrelated objects of any network. This notion of graph database is flexible since no further constraint is imposed on graphs and their topology [1]. In it, objects are referred to as nodes and are connected to each other to form relationships via directed edges. Nodes and relationships both possess properties. Traversing and querying a graph database is relatively easier than other data base management system such as relational database

due to their incapability to handle big data with ease because they store data in the form of table's rows and columns. Relationships between objects (entities) are maintained using foreign key constraints but handling them becomes inefficient when the data sets start growing. RDBMS is implementation of models that describes the data or information of the domain of interest. Entity-Relationship (E-R) Model is one such model that describes data in the form of entities, relations and their attributes.

A. Motivation

A Tool designed and developed to convert a model to another one, makes the transition from one representational scheme to another easier. Although a methodology and a tool for converting relational databases to Graph Databases exist, there does not exist any tool to automate the conversion from E-R diagram to Property Graph database. This eliminates the need to map E-R diagram to Relational database followed by migration to graph database. Ref. [1] Database updates in E-R to graph database approach are efficient since the simple insertion or deletion of an object requires less database accesses than migration from relational to graph database. Ref. [2] Also, Query performance and responsiveness is maintained in milliseconds due to index free adjacency property of graph databases. Moreover, due to schema-free nature of a graph database, a graph database solution can also evolve as the business evolves.

B. Related work

Ref. [2] specifies the following characteristics of property graph:

- It contains nodes and relationships
- Nodes contain properties (key-value pairs)
- Relationships are named and directed, and always have a start and end node
- Relationships can also contain properties

A graph database management system has Create, Read, Update, and Delete (CRUD) methods to expose a graph data model. Graph databases are built to be used with transactional (OLTP) systems. They either use native graph storage or serialize the graph data into some general purpose data store. The queries to a graph database are localized to only a portion of the graph leading to constant performance even when the dataset grows. On the other hand, Relational model becomes burdened with large join tables, sparsely populated rows, and lots of null-checking logic. Further, graphs are flexible as new relationships, nodes or sub graphs can be added to an existing structure without disturbing existing functionality.

Ref. [3] Specifies why graph databases are better than relational databases as data is natively stored and queries are expressed in terms of graph traversal operations. An earlier approach to map relational database to graph database required tuples to be mapped to nodes and foreign keys to edges, but this approach did not take into account the query load and can make graph traversals expensive. Their technique converts a relational database r into a graph database g while minimizing the number of accesses needed to answer queries over the target. Query language for graph data models are classified into two main categories:

1. SPARQL and Cypher: queries are in the form of graphs and their evaluation relies on graph matching between the query and the database. Their usage is limited by the fact that graph matching is very expensive for large databases.
2. XPath and XQuery: languages that rely on expressions denoting paths of the database. They are more suitable since they traverse the graph for getting the desired result.

Ref. [1] also specifies why graph databases are better than relational databases. Relational databases are not suitable due to their inability to manage data that involves graph structure since they are not able to capture their inherent graph structure. Moreover complex joins are involved in highly connected data which are not easy to manage in cases where the dataset is huge. Lastly, GDBMs do not rely on a schema. This makes them flexible as they can evolve as the organization's data evolves. This is due to the fact that in Graph databases each node carries the information about its neighbors and no global index of reachability exists between nodes. This in turn makes traversal of an edge from a node independent on the data size. Hence, local analysis can be performed very efficiently on GDBMS which makes these systems suitable in scenarios where the size of data increases rapidly.

Nodes representing objects of the same class (e.g., different users) can differ in the number and data type of a specific property. This makes the data model general and able to capture various graph-based data models. Three main characteristics of Graph Databases have been used in the methodology to move to design of graph database that is based on a model.

- GDBMS stores data by means of a multigraph, usually called *property graph*, where both nodes and edges are labeled with data in the form of key-value pairs called properties. Here, edges represent relationships.
- Index free adjacency: the existence of an edge between two nodes of graph can be tested by visiting those nodes and does not require the existence of an external, global, index.
- The third feature common to GDBMSs is the fact that data is queried using path traversal operations.

Aim of design of graph database is to minimize data access operations needed in graph traversals at query time which can be done by

1. by adding edges between nodes (dense strategy)
2. by merging different nodes (compact strategy)

Dense strategy means adding as many edges as possible between nodes representing conceptual entities. This will reduce the length of paths between nodes and also the number of data access operations needed at run time. *Compact* strategy relies on aggregating in the same node data that are related but are stored in different nodes which may lead to possible data inconsistencies. The aim when trying to convert ER to graph database design is to reduce the number of access operations needed to retrieve related data of the application.

About The Tool

The tool takes entities, relationships, cardinality of relationships and attributes of the entities and the relationships as input. It then assigns weights and directions to the relationships' edges based on their cardinalities. Ref. [1] Further, it calculates the sum of the weights of the incoming edges and the outgoing edges of each entity. This is followed by grouping the entities whose weights satisfy a criterion thereby generating the design of a graph database. Entities that are disconnected form a group by themselves. Nodes involved in many-many relationships are not grouped. Weight calculations are required as apart from the entities involved in one-one relationships that can be partitioned/ grouped as one node, there also exist other entities that can be grouped with other node. The later is an entity involved in many-to-one relationship and is an entity having higher cardinality which apart from being related to the entity having lower cardinality may have one-to-many relationship with some other entity with it being the node with lower cardinality.

There are 7 main interfaces the user comes across during the conversion process of E-R diagram to graph database design. The tool has 4 input UIs for taking entity input, relationship input, viewing all relationships entered by the user and cardinality input. The tool has two intermediate UIs: first one, Oriented E-R consists of information regarding the relationship's type, relationship's direction and the weight assigned to each relationship and the second one, Partitioned Oriented E-R consists of information about aggregation of entities to form nodes. Finally, there are two output UIs, one consists of graph template in the form of tables and second one consists the same in the form of graph. Both can be saved in excel and image format respectively.

C. Steps

Let, ei represent an entity, where i is the number of entities and relationship be represented by R. Ref. [4] Min-max notation consists of a pair of integers that specify the minimum and maximum participation of entity e in relationship R in the form of ei min, ei max, where 0<=min<=max and max>=1. These indicate that e must participate in at least min and at most max relationship instances in R at any point in time. w^+ and w^- are integers that represent sum of the weights of the incoming and outcoming edges for an entity e. Ref. [1] The steps followed to convert ER to a graph database design based on methodology are:

1. For every Entity **e** in E-R diagram, input its name and no. of attributes of **e** and attribute names of **e**.
2. For every Entity **e1** related to Entity **e2** via relationship **R**, input **R's** name, **e1**, **e2**, **no. of attributes of R** and **attribute names of R.**
3. For every Entity **e1** related to Entity **e2** via relationship **R,** input cardinality for **e1** and **e2** in min-max notation.
4. If (((e1min == "0") && (e1max == "1") && (e2min == "0") && (e2max == "1")) || ((e1min == "0") && (e1max == "1") && (e2min == "1") && (e2max == "1")) || ((e1min == "1") && (e1max == "1") && (e2min == "0") && (e2max == "1")) || ((e1min == "1") && (e1max == "1") && (e2min == "1")

&& (e2max == "1"))), then it's a **One-One**, **bidirectional** relationship with **ZERO weight**

5. Else If (((e1min == "0") && (e1max == "N") && (e2min == "0") && (e2max == "N")) || ((e1min == "0") && (e1max == "N") && (e2min == "1") && (e2max == "N")) || ((e1min == "1") && (e1max == "N") && (e2min == "0") && (e2max == "N")) || ((e1min == "1") && (e1max == "N") && (e2min == "1") && (e2max == "N"))), then it's a **Many-Many**, **bidirectional** relationship with **weight=2.**
6. Else
   - If (((e1min == "0") && (e1max == "1") && (e2min == "0") && (e2max == "N")) || ((e1min == "0") && (e1max == "1") && (e2min == "1") && (e2max == "N")) || ((e1min == "1") && (e1max == "1") && (e2min == "0") && (e2max == "N")) || ((e1min == "1") && (e1max == "1") && (e2min == "1") && (e2max == "N"))), then it's a **Many-One**, **direction=arrow head pointing towards Entity with higher multiplicity**(arrowpointingtowardsEntity2) with **weight=1.**
   - Else it's a **One-Many**, **direction=arrow head pointing towards Entity with higher multiplicity** (arrowpointingtowardsEntity1) with **weight=1.**
7. For every Entity **e1** related to Entity **e2** via relationship **R**, display, **e1**, **R, e2**, **e1min, e1max, e2min, e2max, Relationship_type, Relationship_direction and Relationship_weight**
8. For every Entity **e1** related to Entity **e2** via relationship **R,** get **R's direction** and **weight.**
   - If **direction=bidirectional,** then update **e1**'s $\mathbf{w}^+$ and $\mathbf{w}^-$
   - Else if **direction= arrowpointingtowardsEntity2,** then update **e1's** $\mathbf{w}^+$
   - Else if **direction= arrowpointingtowardsEntity1,** then update **e1's** $\mathbf{w}^-$
9. For every Entity **e2** related to Entity **e1** via relationship **R,** get **R's direction** and **weight.**
   - If **direction=bidirectional,** then update **e2**'s $\mathbf{w}^+$ and $\mathbf{w}^-$
   - Else if **direction= arrowpointingtowardsEntity2,** then update **e2's** $\mathbf{w}^-$
   - Else if **direction= arrowpointingtowardsEntity1,** then update **e2's** $\mathbf{w}^+$
10. For every Entity **e,** get its updated $\mathbf{w}^+$ and $\mathbf{w}^-$
    - If (($\mathbf{w}^-$ < 1 && $\mathbf{w}^+$ <= 1) || ($\mathbf{w}^-$ <= 1 && $\mathbf{w}^+$ < 1))then get Entity **e2** related to Entity **e** and aggregate **e** with **e2**
    - Else **do not aggregate** e
11. For every Entity **e** that is not to be aggregated, get its attributes; **e** becomes a **NODE** and **attributes** of **e** become properties of that **Node.**
12. For every Entity **e1** that is to be aggregated with Entity **e2, e1-e2** becomes a **NODE. Attributes** of **e1** and **e2** become **properties** of **Node.** Also, attributes of relationship **R** become properties of the aggregated **NODE.**
13. Replace occurrence of entity e1, entity e2 with the newly aggregated node e1-e2 to generate edges from the Relationships.

D. Tool Snapshots

The example taken to convert ER into graph database design consists of the following entities and their attributes are inputted using the AddEntity GUI (Fig. 1) as follows:

Table 1: Example: Entities And Their Attributes

Entity Name	**Attribute Name**
Comment	cid,msg
User	uname,uid
ExternalLink	eid,url
Category	ctid, description
Blog	bname,bid

Figure 1: Adding Entity 'Comment' And Its Attributes

Initially, unique ER ID is inputted by user so as to retrieve the graph template (graph as well as excel format) at any later point in time. The relationships between entities and attributes of relationships inputted as shown in Fig. 2 are as follows:

Table 2: Relationship list

Entity name1	**Relationship name**	**Entity name2**
Comment	contain	External Link
Comment	post	User
Comment	tag	User
Comment	publish	Blog
Blog	follower	User
Blog	admin	User
Blog	about	Category

Table 3: Relationship Name And Their Attributes

Relationship name	**Attribute name**
contain	ab1contain
post	date
tag	tagid
publish	Publish date
follower	days
admin	level
about	ababout

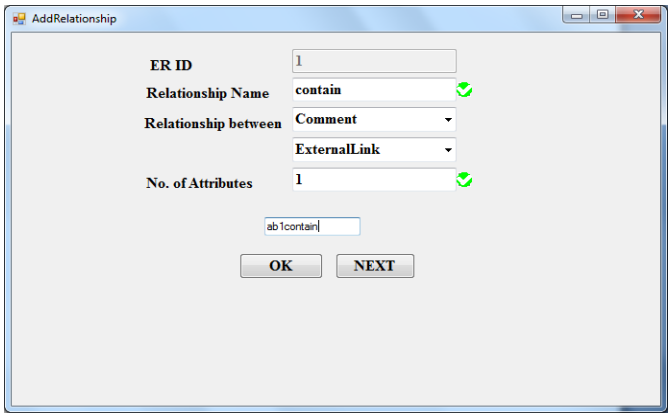

Figure 2: Adding Relationship ‘Contain’ Between ‘Comment’ And ‘Externallink’

Fig. 3 shows all the relationships inputted by the user so far. The next step is to input the cardinalities of all the relationships in min-max notation as shown in Fig. 4.

Table 4: Cardinality List In Min Max Notation

Entity name1	Relationship name	Entity name2	e1 min	e1 max	e2 min	e2 max
Comment	contain	External Link	1	1	1	1
Comment	post	User	1	1	0	N
Comment	tag	User	0	N	0	N
Comment	publish	Blog	1	1	0	N
Blog	follower	User	0	N	0	N
Blog	admin	User	1	1	0	N
Blog	about	Category	1	1	0	N

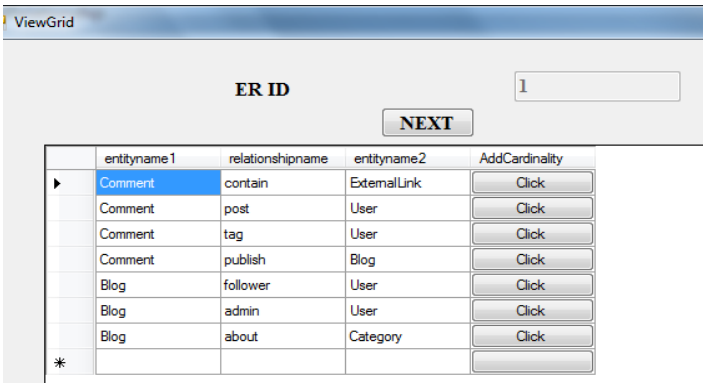

Figure 3: List of All The Relationships Entered

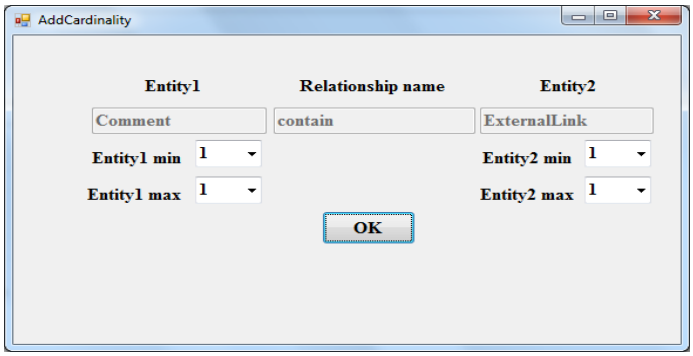

Figure 4: Adding Cardinality For 'Contain' Relationship

Fig. 5 shows the intermediate UI (Oriented ER) displays the inputted relationship's type, direction and weight assigned to it. Fig. 6 is the intermediate UI (Partitioned OER) which displays the information regarding aggregation of entities. Fig. 7, the Graph Template UI displays the design of the graph database i.e. nodes and edges in a tabular form.

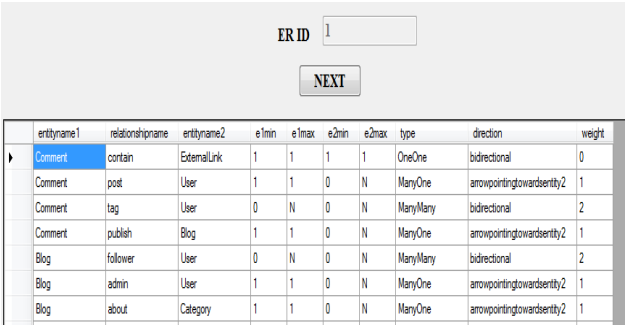

Figure 5: Oriented ER displaying the relationship type, direction and weight

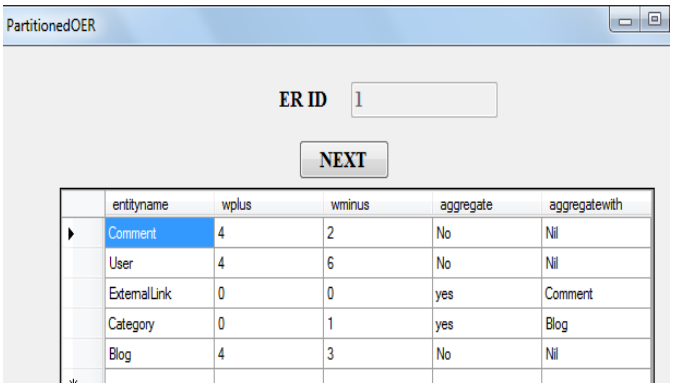

Figure 1: Partitioned OER UI displaying information regarding aggregation of nodes

Fig. 7 is the Graph Template UI which displays the design of the graph database i.e. nodes and edges in a tabular form. The aggregated node properties consist of the individual entities' attributes and the edge properties consist of the relationship's attributes. In case there is a relationship that binds two entities which are aggregated upon conversion, its attributes become the property of the aggregated node.

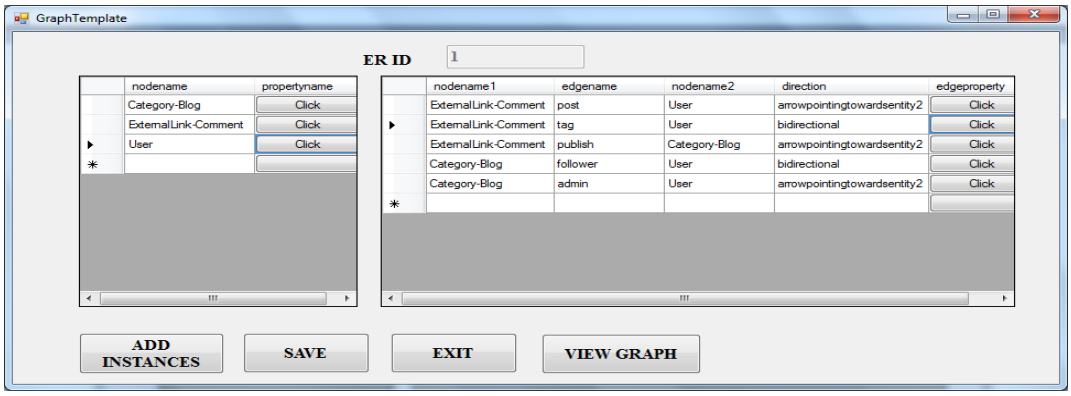

Figure 7: Graph Template UI Displaying The Graph Database Design

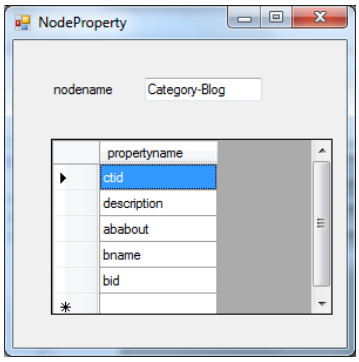

Figure 8: Properties of aggregated node 'category-blog'

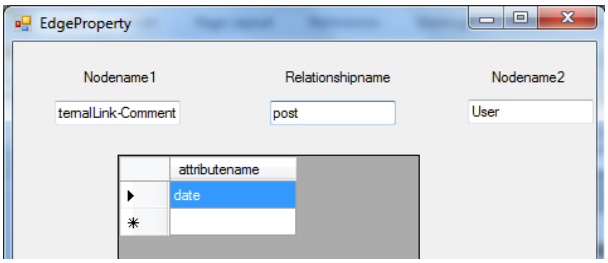

Figure 9: Properties of Edge 'Post'

Fig. 8 displays the properties of the aggregated node Category-Blog. This node has properties of the individual Category and Blog node and of the relationship 'about' which related these two entities in the ER diagram. Fig. 9 displays the property of the edge 'post'. Upon clicking on the save option as shown in Fig. 7, nodes' and edges' information are saved in excel format onto the user's system at a specified path for any future use. Also, View Graph option helps to visualize the graph template in graphical format as shown in Fig. 10, which consists of the basic information as to what will be the nodes and edges after the conversion process is over. The user may exit right here or he may go on to add instances of nodes and edges using the Add Instances option as shown in Fig.7. Using this, he needs to specify the number of nodes' instances and the value of their properties. This is followed by choosing edges between these nodes' instances and filling up edges' properties. This helps him to visualize a graph that has nodes and edges with their attribute-value pair as shown in Fig. 11

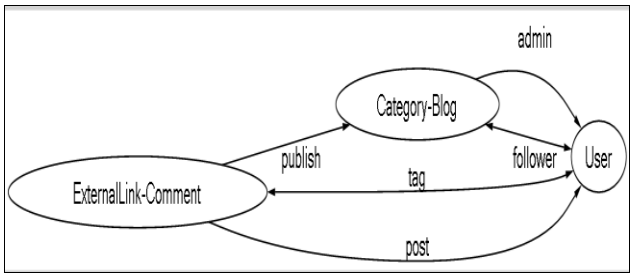

Figure 10: Example Graph Database Design

Fig. 11 shows that Category-Blog, ExternalLink-Comment and User node were inputted with 3,1 and 2 instances respectively. The properties of these instances were inputted after clicking on the corresponding Action button.

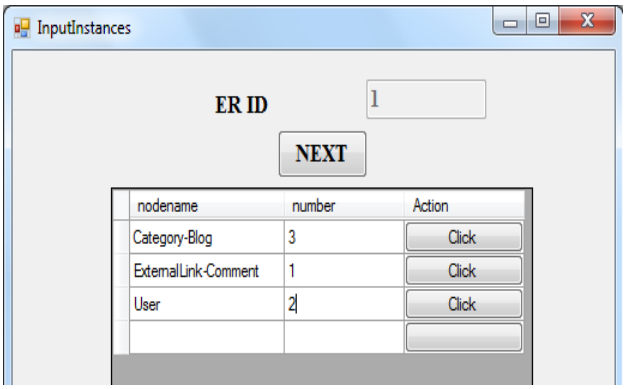

Figure 11: Input Number of Node Instances

Fig. 12 displays the properties uname and uid for node User1 and User2 both. Upon clicking on the Value button their values can be inputted by the user as shown in Fig. 13. Table V shows all the values inputted for all the instances of all the nodes.

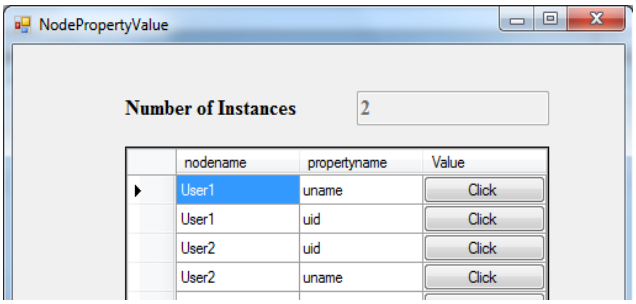

Figure 12: Two Instances Of Node 'User' : User1 And User2

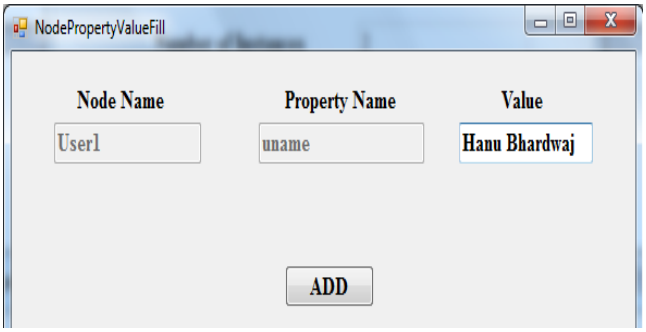

Figure 13: Inputting Value For Node User1's Property Uname

Table 5: Value of Property of All Node Instances

Nodename	Property	Value
Category-Blog1	ctid	ct01
Category-Blog2	ctid	ct02

Category-Blog3	ctid	ct02
Category-Blog1	description	A
Category-Blog2	description	B
Category-Blog3	description	B
Category-Blog1	ababout	ab1
Category-Blog2	ababout	ab2
Category-Blog3	ababout	ab3
Category-Blog1	bname	Inf.Systems
Category-Blog2	bname	Database
Category-Blog3	bname	Computer Science
Category-Blog1	bid	b01
Category-Blog2	bid	b02
Category-Blog3	bid	b03
ExternalLink-Comment1	ab1contain	somevalue
ExternalLink-Comment1	eid	el1
ExternalLink-Comment1	url	htttp://link.com
ExternalLink-Comment1	cid	c01
ExternalLink-Comment1	msg	Good News!
User1	uname	Hanu Bhardwaj
User2	uname	Dimple Chehal
User1	uid	u01
User2	uid	u02

After this, the user needs to input the value of edge properties. In order to accomplish this, first he needs to select the edgename followed by the node instance's name as shown in Fig. 14 for the edge 'post'. Table VI shows all the values inputted for all the instances of all the edges.

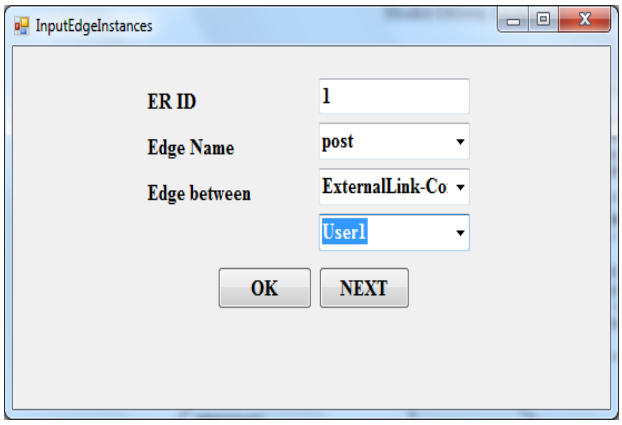

Figure 14: Edge 'Post' Between Node 'Externallink-Comment1' And 'User1'

Table 6: Value of Property of All Edge Instances

Node name1	Edge name	Node name2	Property name	Value
ExternalLink-Comment1	post	User1	date	25/3/2015
ExternalLink-Comment1	tag	User2	tag_id	tg01
ExternalLink-Comment1	publish	Category-Blog2	publishdate	1/4/2015
Category-Blog1	follower	User1	days	10
Category-Blog1	follower	User2	days	20
Category-Blog2	follower	User1	days	15
Category-Blog3	follower	User1	days	5
Category-Blog1	admin	User2	level	low
Category-Blog2	admin	User1	level	medium
Category-Blog3	admin	User2	level	High

Only After all the values of properties of nodes as well as edges have been inputted by the user, he should be able to visualize the complete graph as shown in Fig. 15. This can be saved onto the user's system as an image file.

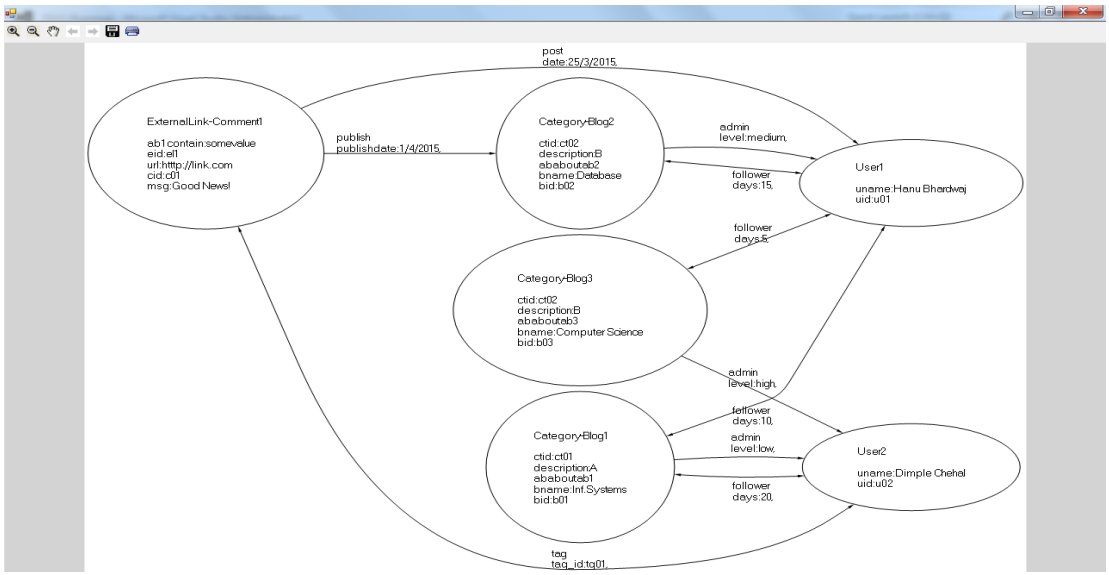

Figure 15: Complete Graph With Attribute-Value Pair

Experience

While the tool is developed as an attempt to implement the methodology provided in [1], some modifications are done so as to achieve its objective. Firstly, [1] specifies the node aggregation condition for an entity e1 as: If (($\mathbf{w}^-$ < =1 && $\mathbf{w}^+$ <= 1), then **e1** related to Entity **e2** should be aggregated with **e2.**But, it doesn't mention what action should be taken in cases where an entity e's $\mathbf{w}^-$<1 and $\mathbf{w}^+$>1 (source node), $\mathbf{w}^-$ >1 and

$\mathbf{w}^+ <1$ (sink node), $\mathbf{w}^- =1$ and $\mathbf{w}^+ =1$. The idea gauged from the example provided is that terminal nodes (nodes participating in a single one-many relationship) be aggregated so as to decrease the query time. This implies that aggregation condition for an entity e1 should be: If (($\mathbf{w}^- < 1$ && $\mathbf{w}^+ <= 1$) || ($\mathbf{w}^- <= 1$ && $\mathbf{w}^+ < 1$)) then entity **e2** related to Entity **e1** should be aggregated with e1. Secondly, the example provided in [1] doesn't specify about the conversion of the attributes of the relationship that relates two entities which are to be aggregated. We have converted them into properties of the aggregated node itself.

Table 7: Conversion Table

Notation used in [1]	Notation used to implement
Directions in Oriented ER are in the form of edges 1. Bidirectional: arrow heads on both side of edge 2. ManyOne: one arrow edge pointing to the entity with higher multiplicity	Directions in Oriented ER are in the form of string: 1. Bidirectional: bidirectional 2. ManyOne: arrowheadpoitningtowardsentity2

Conclusion

The tool will be helpful in cases where an organization wants to build graph database from the scratch using E-R diagram as the base model. This in turn will eliminate the need to map E-R diagram to Relational database followed by migration to graph database. Ref. [1] the aggregation of nodes helps to reduce the query response time. Also, due to the automation performed by the tool, manual efforts required to convert E-R diagram to graph database will be reduced. However, this tool doesn't handle ternary relationships. The approach to handle ternary relationships can be to convert it into binary relationships and then perform the processing. Moreover, the tool handles only one ER at a time with the help of an ER ID provided by the end-user initially. It provides him with the option to save the graph template in excel as well as graphical format. Also, at any later point in time, he can retrieve the template information by keying in the ER ID. The extension to this tool can be a mechanism by which Neo4j can directly take this tool's output as its input and proceed further.

References

[1] Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone, "Model-Driven Design of Graph Databases", in 32nd International Conference on Conceptual Modeling, ER 2014, Atlanta, GA, USA, pp. 172–185, 27-29 October 2014.

[2] Ian Robinson, Jim Webber and Emil Eifrem, Graph databases,United States of America: O'Reilly Media, Inc., 2013.

[3] Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone, "Converting Relational to Graph Databases", Proceedings of the First International Workshop on Graph Data Management Experience and Systems (GRADES 2013), New York, NY, USA, June 23, 2013.

[4] Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Systems, India:Dorling Kindersley(india) Pvt. Ltd., 2008.