ATLANTIS PRESS
Advances in Computer Science Research (ACRS), volume 54
International Conference on Computer Networks and Communication Technology (CNCT2016)

An NDN-based Query Processing Framework for Networked Databases

Zhu-hua LIAO, Zeng-de TENG*, Jian ZHANG and Yi-zhi LIU

School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China

Key Lab of Knowledge Processing & Networked Manufacturing, University of Hunan Province, China

*Corresponding author

Keywords: Named Data Networking, Relational Databases, Named-based Query, Query Routing, Answer Integration.

Abstract. Nowadays, massive databases have accumulated and networked in the large-scale and distributed networks, however, in which extensive data are still hard to be directly shared to or efficiently exploited by the users as many challenges and technical designs of query processing are arising in the networks. To address these issues, a name-based query framework is proposed by utilizing Named Data Networking (NDN). Based on the principles of NDN, our query scheme transforms SQL query statements into name-based queries firstly, and then processes these queries for routing, local querying and answers integration in distributed networks. In the system, networked query algorithms process the complex queries and distributed answers with hierarchical names. The primary implementation solutions illustrated in the paper and finally the performance analysis demonstrated that the name-based query scheme is feasible and can fulfill general network design principles.

Introduction

Nowadays, huge amounts of databases accumulated and networked in the large-scale and distributed networks, but querying data directly from large number of databases is a big challenge. So, extensive data stored on those networks, such as MANET (Mobile Ad-hoc network), P2P and DTN (Delay Tolerant Network), are still hard to be directly shared to or efficiently exploited by network applications or users. Because the databases are distributed in the dynamic networks, the query statements with complex claims and various constraints need to be routed to right databases, and then all response answers should be aggregated at intermediate nodes and are returned to corresponding clients through one or more routers. Furthermore, considering the traffic balance and wire-speed forwarding, the query statements had better refrain from spreading everywhere when they are routed and forwarded in networks. Besides these, many other difficulties have to overcome in the dynamic and distributed networks, such as query decomposition and in-network query processing [1-4].

Recently, an novel network architecture, called Named Data Networking (short for NDN/CCN)[4], is proposed for retrieving named content from dynamic and distributed networks. In which a routing mechanism (we called name-based routing mechanism) is presented for routing a users' interest by a hierarchical name, which stored in the FIB (Forwarding Information Base), rather than an IP address. Since the name-based routing mechanism can semantically look up FIBs in routers, and wisely determine right next hops in NDN, it is possible to route each complex query to right sources. If so, the

Copyright © 2017, the Authors. Published by Atlantis Press.
This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

database query will not be enslaved to the addresses of databases and sabotaged by network topology changing and the displacement of databases on networks.

In the paper, we will address the challenges of the in-network query processing and present a name-based query scheme for NDN. The main contributions are as follows:(1) We introduce a name-based query scheme for the NDN that connected with relational databases; (2) We put forward a transformation method of SQL query and a strength name-based routing algorithm for forwarding the complex query to the right databases in NDN; (3) We present various integration techniques for retrieving the response answers of different SQL queries in NDN.

Related Works

Today, existing distributed RDBMS can be classified into three main paradigms: central data repository, database federation and P2P database networks. In the central data repository paradigm, the clients can access to the central data repository by the IP address on the Internet. Another paradigm is the database federation [5] in which database sources are distributed and a centralized portal is used to receive all queries and in charge of query decomposition and distributing the sub-queries to appropriate database sources by IP addresses. In the P2P database networks, querying the distributed and dynamic structured data is the main objective. Early days' unstructured P2P systems rely on the query flooding or random walk [6], but they have poor scalability. To address the scalability issue, several methods are proposed, for example, range query [7], multi-dimensional query [8] and the routing indices (RIs) [9]. These methods rank or classified the distributed structured data and provide a list of "coarser" topics or values towards the potential database sources for different queries. So the query is generally unitary. The structured Peer-to-Peer system builds on the theory of distributed hash table (DHT) which uses flat identifiers or keys to map the data's addresses [10]. However, using flat identifiers is not inadequate for complex queries which included SQL queries. Recently, the NDN has been proposed for efficiently discovering the named data by issuing an INTEREST packet which carried a hierarchical name, and returning results by a DATA packet in a network. And many similar applications have been developed based on NDN, such as NDN-friendly file system [11], distributed file sharing application on NDN [12] and so on. For strength NDN to search networking data, Zahariadis et al. presents an approach of similar content search [13], in which the "search" is introduced as a top level namespace and uses the flooding to search similar named data in NDN. The relational routing scheme [14] is devoted to routing the semantic query and aggregating the answer for querying the semantic data in NDN, but it is not suitable for the SQL query in NDN. In this paper, we focus on providing a name-based SQL query processing for networked databases based on NDN.

In short, despite appearing various distributed query techniques and systems, almost none of them are committed to processing the SQL query for networked databases in a distributed and dynamic network. However, this capability is becoming increasingly important as more mobile ad-hoc networks, delay tolerant networks connected to lots of databases, and other structured data which addresses are untraceable.

SQL Query Scheme for NDN Databases

Data schema of Relational Database

Generally, relational databases can be classified into different application areas which can be described by DB names or mined by some reverse engineering approaches [15,16]. In a relational database, a relation (or a table) collected a type of entries (or records) with multiple fields (or columns). Usually a relation stored a type of entries can be classified into a category. If the names of databases and relations correctly represent the semantics of data stored in these databases, we can use the names to reject unrelative databases and possibly fulfill a name-based query scheme by the name-based routing mechanism in NDN. If so, the name-based query will open a new way to retrieve or manipulate data in networked databases by network routers.

Naming Data and Query

To newly created databases, relations and fields, their names can be uniformly naming by referring to a *globally data dictionary*. But to the legacy ones, their names can be extracted from respective local data dictionaries by using reverse engineering approaches. Based on these names and the requirements, user can generate different SQL query statements. For example, assumed a NDN connected a distributed database, named *ElectrionicBooks*, which stored many sorts of electronic books, and the names of all relations and fields are acquired. If a user wants to query all books' information that belong to “Computer Science” major and “Network” area and published within the past 3 years, the SQL query statement can be written as:

```
Use ElectronicBooks;
Select Name, Author, Publishhouse, Publishtime
From BookInfo, MajorInfo
Where Major =“Computer Science” and Area =“Network” and
(2016- Year(publishing time))<=3
```

In NDN, above SQL query statement should be transformed a packet which is similar to Interest packet for semantically routing and in-network query processing. From the above SQL query statement and the name of the databases, the following principal information can be extracted:

Relation Classes: ElectronicBooks / BookInfo
Parent tables: MajorInfo
Entity Classes: Computer Science / Network
Semantic Constraints: (2013- Year(publishing time))<=3

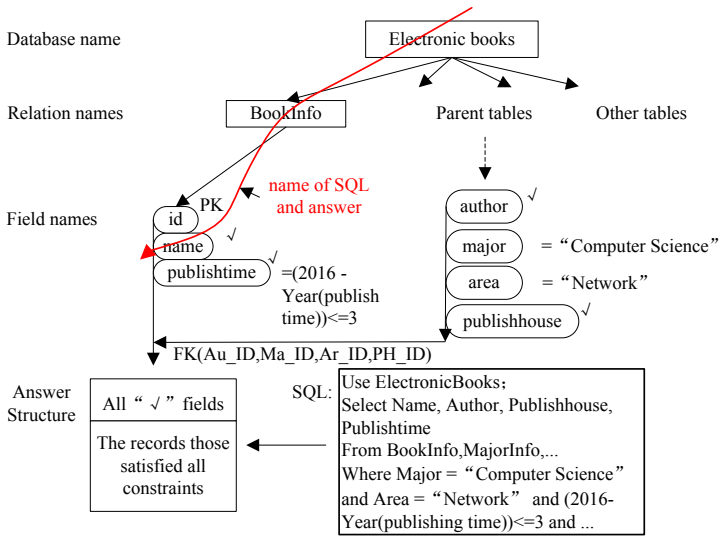

Fig.1 The hierarchical name and answer structure extraction from a SQL statement

Here, the *Relation Classes* can be used to route the query and return the answers in NDN. The *Entity Classes* is a further semantic division for a type of entities. The *Semantic Constraints* mainly specify some restrictions to select the needed entities in a relation. To efficiently forward a query in NDN, the name should simple as the naming of INTEREST. In the paper, we consider the *Entity Classes* as a semantic constraint. So we form a query packet which can be routed and processed in NDN by carrying the requisite information. Fig.1 shows how to extract the hierarchical name from a SQL query and set up the answer structure.

NDN-based SQL Query Scheme

In our scheme, we build a *Global Data Dictionary* firstly in Centralized controller for network users to create databases and tables. Then, we employ a hierarchical name and *semantic constraints* to constitute a query packet that similar to the INTEREST packet. Fig.2 presents a SQL query processing framework used for databases' query in NDN. When a client issues a SQL query, it will be translated into a NDNQL, which can be routed by queried in NDN, by a *NDNQL Translator*. Then the *Dispatcher* will pack the NDNQL and send the NDNQL packet to the nearest router. When a router received a NDNQL packet, it will look up the answers in the cache for the NDNQL. If the complete answers can be found in the cache, it will be directly returned and the NDNQL packet will be discarded. Or else, it will be forwarded to the next routers or sources by matching the items of FIBs with the *Longest Prefix Matching* (LPM) and record the arrived face for the packet. When the NDNQL packet arrived at right sources, the NDNQL will be transformed back to the original SQL query, and then it will be performed in the local databases. After the query finished, the query answers will be packed into a NDNRS packet that similar to the DATA packet and returned back via the paths that the NDNQL packet has passed through (mainly do exactly match in PITs to find the arrived faces), and will be integrated by integration algorithms (can refer the algorithm in [14]) when they arrived at the rendezvous nodes. Finally, as the aggregated results backtracked to related clients, the NDNRS packet will be unpacked as answers and will be sent to the corresponding applications or users.

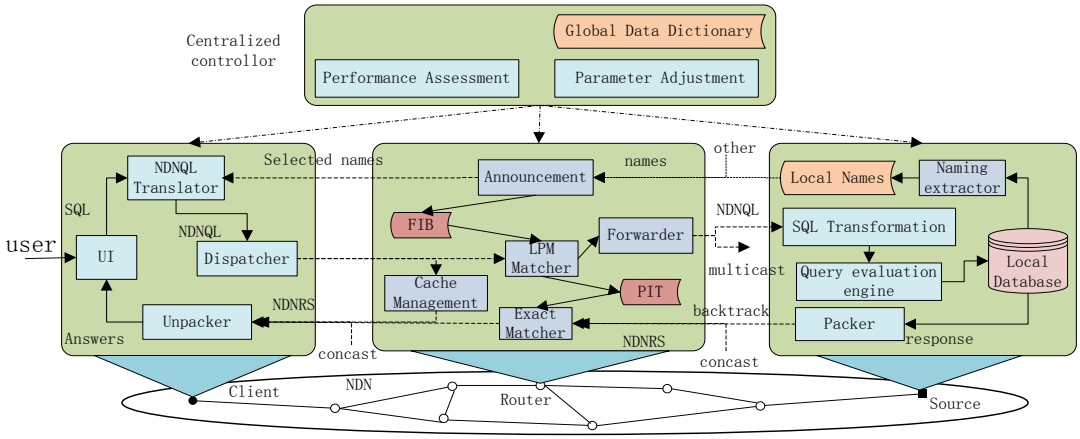

Fig.2 The processing framework of SQL query in NDN

NQNQL Description and Transformation

Usually, most RDBMSs mainly provide these widely used SQL query operations, such as *SELECTION*, *PROJECTION*, *AGGREGATION* and *JOIN*. However, we found that the query statements of these operations consist primarily of two types of information: (1) the names of databases and relations (tables) involved in the query statements; (2) the semantic constraints based on the names. In the networked databases, the SQL query operations can involve into a table, multiple copies of a table or multiple different distributed tables. Here we tend to address two operations: query operations on multiple copies or multiple tables. Next we will present the transformation of SQL query operations in detail.

NDNQL Description for SQL Query Statements

The *SELECTION*, *PROJECTION*, *AGGREGATION* operations usually involve at least a relation (or table). So we simply take the names of database and relation(s) to form a hierarchical name for routing, and then add the query statement to form a name-base query for in-network processing in NDN. So, a name-based query is consisting of two parts: *naming component* and *query statement component*. The naming component is used for query routing and answers response.

If a SQL query involved multiple tables, including child tables and parent tables, it would be best to find out the child tables, as the content of parent tables might be put in child tables. So, we take the parent tables' name as an option in the hierarchical name.

A *JOIN* query is bound to involve at least two tables. In distributed databases, these tables may be distributed, so we usually shift *JOIN* operation as s*emi-join* operation. If the tables located at different sources, the *semi-join* query should be forwarded to all these sources in a certain order. So multiple naming components need to be added to a NDNQL packet, and specifying an order for routing the *semi-join* query to the corresponding database sources.

For processing a nested query, we should consider the execution sequence of multiple sub-queries. In NDN, maybe these sub-queries ought to be performed in different sources. So, we should also find out multiple naming components and rightly use them for routing when performing different sub-queries.

Statements Transformation between SQL and NDNQL

In NDN, the format of a hierarchical name that stored in FIB is:

"Globally routable name / Organization name"

In practice, the *globally routable name* can be omitted that means the query targets the whole network, or pruning lower levels of *globally routable name* (that is keep a prefix of domain name) that means the query targets certain area of a network. So for transforming a SQL query to a NDNQL statement, we design a transformation algorithm shown in Algorithm 1.

Algorithm 1 Transform SQL to NDNQL

```
Input:  QueryStatement SQL
Output:  NDNQLStatement(s) NDNQL
    Int i=0; String[] Q, NDNQL, ES;
    DbName=GetDbName(SQL);
    If (MultipleSubQueries(SQL)==true) Then{
       For (;all sub-queries;)
       ES[i++]= all sub-queries in execution sequence order;
    }
    Else ES[i]=SQL;
    For (Int j=0; j<sizeof(ES);j++){
      TableNames=GetAllTableName(ES[j]);
      HierName=Globallyroutablename+``/"+DbName+``/\{"+TableNames+``}";
      NDNQL[j]=HierName+``;SQL:"+Q[j];
    }
    Return NDNQL;
```

When the NDNQL packet was forwarded into right sources, it should be transformed back into the original SQL query.

NDNQL Routing and Query Processing

For routing NDNQL and response answers in NDN, we need at least two kinds of packets, NDNQL packet and answer packet respectively, which are shown in the Fig.3. In a NDNQL packet, the *NDNQL Statement* is used for NDNQL forwarding and in-network querying, the *Auth Info* is used for packet and user authentication, and the *Selector* is used for given the DB owners, preferences and network scope when processing the in-network query. When a router receives the packet, it will be resolved. Then the router will use the hierarchical name of NDNQL to match the FIB for acquiring next hops. As the packet arrived at sources, the SQL statement in NDNQL will be used to query local database. Soon afterwards, the answers and the corresponding NDNQL will be packed into the answer packet (or NDNRS). Finally, the answer packet will be forwarded back to the senders by routers which picked up from the PIT by matching the NDNQL.

NDNQL packet

NDNQL Statement	Selector (Preference, Publisher, Scope, ...)	Auth Info	Nonce

Answer packet

NDNQL Statement	Signature Technique	Signed Info	Timestamp	Answers

Fig.3 NDN Packets for name-based query

NDNQL Routing on Distributed Tables

In NDN, the implementation procedures of query routing are as follows: Firstly, when a router received a NDNQL packet, the LPM matching between the hierarchical name and each entry in FIB is done. If an entry is matched, forwarding the NDNQL to the

matched routers according to the values of face field until the NDNQL is forwarded to all relevant sources. In each relevant source, the SQL statement will be extracted and performed at the local database. After the answers are ready, they and the SQL statement will be packed and backtrack to original requester(s) by following the trail of 'bread crumbs' that were left in PITs by the NDNQL packet. Fig.4 shows the in-network routing of NDNQL packet for querying distributed tables in NDN.

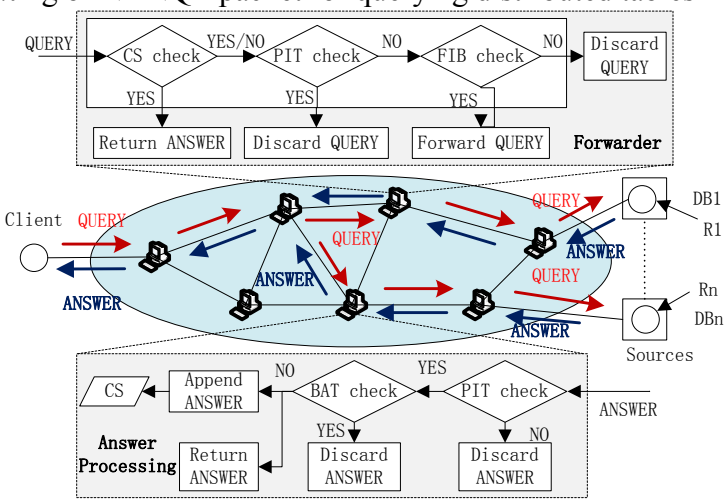

Fig. 4 In-network query routing for querying distributed tables

NDNQL Processing on Distributed Tables

As a join query involved multiple tables and these tables are stored in different sources, the NDNQL packet will be forwarded to these sources by our scheme. The semi-join is a very efficient and important method for optimizing join query in distributed databases. Fig.5(a) shows the overall processing of the semi-join query on two distributed tables in NDN. First, a NDNQL query is routed to relevant sources by our routing scheme and a hierarchical name (assume it included the name of table *R*). Then performing query in the local database with SQL statement which is packed into the NDNQL packet. When getting the answer, NDNQL needs to be reconstructed (the hierarchical name should include the name of table *S* and the SQL should be rewritten) and forwarded to another source. Finally, after the new SQL statement is performed at another source, the answer will return back to the senders by NDNQL routing algorithm.

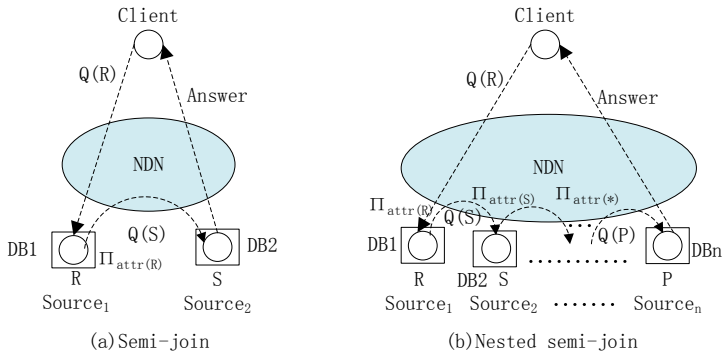

Fig. 5 In-network semi-join query on distributed tables

The nested query is a complex query which involved more tables and multiple sub-queries. If these tables are distributed, we should use nested semi-join operations to realize the nested query and transform the query after executing a sub-query. Fig.5(b)

shows the overall processing of the nested semi-join query on more than two distributed tables in NDN.

In-network Answers Integration of NDNQL

Because the in-network query processing means multiple local queries processing performed at different sources and multiple distributed answers may be received from different upstream routers or sources in NDN. For reducing the number of packets and traffic flow, the answers' integration is needed to process at rendezvous sites. Here we call the client or router a rendezvous node. But the arrival time of these answers could be not at same time, so there are three types of answers integration which depend on different situations.

(1) Synchronous Answers Integration

When multiple answers of a NDNQL arrived at a router in a short-time, they may be merged into an answer packet and remove the duplicate content.

The detail integration procedures are: First, storing the answers, and deleting the duplicate content from them for the same NDNQL. Then, integrating them and recording the arrival faces and time for these answers. To an answer packet, it may be integrated multiply at rendezvous routers.

(2) Asynchronous Answers Integration

When each answer of a NDNQL arrived at a router in a different short-time window, the duplicate content in late arrived answers need to be deleted according to the previous answers that cached in the C*ontent Store*, but the asynchronous answers are difficult to be merged into fewer packets.

(3) Recursive Answers Integration

The above mentioned that an answer packet may be integrated several times on different routers. And the answers integration is essentially data integration in multiple answer packets. In most case, the data integration process is simply putting the data together. But in some case, we can choose a computing mode to prune some data and even reduce some answer packets. For example, some *AGGREGATION* operations (e.g. MAX, AVG) of database query can make a query only need to back an answer. But to this data integration, we should be noted the computational algorithm. For example, in the recursive answers integration of AVG query, the average calculation can be repeatedly to all the local average values that responded from different neighbor nodes, but it should be considered the different weights to different local average values for the local average values that contained different number of the original data.

Analysis and Future Work

In the NDN-based query system, the performance of the distributed processing determines the availability and applicability of our scheme. Compared to the traditional database query schemes, such as database federation, P2P database system, our scheme has some competitive advantages in performance issues.

(1) Traffic balance. Though one NDNQL query may get multiple answers, most answers' sizes are comparatively small and multiple smaller packets can be merged into an answer packet. So it is easy to maintain the traffic balance between queries and answers.

(2) Wire-speed forwarding. In our scheme, the *Organization name* of naming component can simply have two layers, and many FIB entities can also be aggregated,

so the match time of names in FIB, *Content store*, PIT are shorter, so the query forwarding can be wire-speed.

(3) Dynamic adaptability: Usually, in a slightly larger network, a query needs pass through multiple hops to reach right sources. In a network, sources displacement will happen often and the network topology also will change uncertainly. But NDN can bear packets routing on the dynamic and distributed network by taking advantage of the name-based routing mechanism.

The NDN-based database query system is under active development with Java language. It can run on any TCP/IP network as an overlay system. And we are continually adding new features to improve the query performance and enhance the recall and precision ratios.

Conclusions

In the paper, we present an in-network SQL query processing framework for networked databases in NDN which can be deployed on the TCP/IP network and other ad-hoc networks. In particular, for realizing the novel NDN-based database query, the algorithm that transformed a SQL query into an in-network query form which suited routing and querying in NDN. And the in-network query processing for the SQL query in NDN are also introduced. Furthermore, as for the plethora of distributed query answers, we presented the distributed answers integration in NDN. In the future, we will be dedicated to the software development and applications of the SQL query in NDN.

Acknowledgment

This work was supported by the grant from the National Natural Science Foundation of China (Grant No.61370227) and Hunan Province Universities Innovation Platform of Open Fund Project (Grant No.14K037).

References

[1] A. Eyal, A. Gal. :Self Organizing Semantic Topologies in P2P Data Integration Systems. International Conference on Data Engineering (ICDE). 2009.

[2] Mario Gerla, Leonard Kleinrock. Vehicular networks and the future of the mobile internet. Computer Networks. Vol.55, No.2, 2011.

[3] P.Padmanabhan, L.Gruenwald, et al. A survey of data replication techniques for mobile ad hoc network databases. The VLDB Journal, Vol.17, No.5, 2008.

[4] V. Jacobson, D. K. Smetters, et al. Networking named content. the ACM CoNEXT. New York, USA, Dec. 2009.

[5] G. Olaf and S. Steffen. Federated data management and query optimization for linked open data. NDWDM, vol. 331, Springer, 2011.

[6] D. Fayel, G. Nachouki and P. Valduriez. Semantic Query Routing in SenPeer, a P2P Data Management System. NBiS , LNCS 4658. 2007.

[7] F. Banaei-Kashani and C. Shahabi. SWAM: a family of access methods for similarity-search in peer-to-peer data networks. ACM CIKM. 2004.

[8] X. Sun. SCAN: a small-world structured p2p overlay for multidimensional queries. International World Wide Web Conference, New York, USA, 2007.

[9] A. Crespo and H. Garcia-Molina. Routing indices for peer to peer systems. International Conference on Distributed Systems, 2002.

[10] M. Harren, J.M. Hellerstein, R. Huebsch, et al. Complex Queries in DHT-Based Peer-to-Peer Networks. Int'l Workshop Peer-to-Peer Systems (IPTPS). 2002.

[11] Wentao Shang, Zhe Wen, Qiuhan Ding, etc. NDNFS: An NDN-friendly File System. NDN, Technical Report NDN-0027, Revision 1: October 27, 2014.

[12] Alexander Afanasyev, Zhenkai Zhu, etc. The Story of ChronoShare, or How NDN Brought Distributed Secure File Sharing Back, IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), p. 525-530, 2015.

[13] P. Daras, T. Semertzidis, L. Makris, and M. G. Strintzis. Similarity content search in content centric networks. ACM International Conference on Multimedia, 2010.

[14] Z. Liao, G. Zhang, A. Yi, and G. Zhang. A Relation Routing Scheme for Distributed Semantic Media Query. The Scientific World Journal, 2013.

[15] Reda Alhajj. Extracting the extended entity-relationship model from a legacy relational database. Information Systems, 28 (2003) 597 – 618.

[16] D. Yeh, Y. Li, W. Chu. Extracting entity-relationship diagram from a table-based legacy database. The Journal of Systems and Software, 81, 764 – 771, 2008.