An Algorithm for Transformation of Data from MySQL to NoSQL (MongoDB)

Rupali Arora
Department of Computer Science & Engineering
Thapar University, Patiala-147004(India)

Rinkle Rani Aggarwal
Department of Computer Science & Engineering
Thapar University, Patiala-147004(India)

***Abstract*— Relational databases have been the dominant model since 1980's, for storing, retrieving and managing data in computer industry. But relational databases do not fit into the current scenario and losing its importance due to fixed schema requirement and inability to scale well. With the development of internet and cloud services like Google and Amazon data is growing fast. Today's Web and mobile applications are designed to support large number of concurrent users by spreading load across multiple servers. The relational databases are facing many problems to contend with these trends. Recently, NoSQL solutions have emerged as a solution to these problems. A growing number of industries and users are migrating to NoSQL solutions. So, there arises need to transform the data from relational databases to NoSQL databases. In this paper an algorithm has been proposed for transformation of data from MySQL to MongoDB. This algorithm has been implemented using Net Beans Java IDE with the help of Pentaho integration tool.**

Keywords— MySQL, NoSQL, MongoDB, Pentaho, Key-value, Column-oriented, document-based, graph database.

I. INTRODUCTION

Traditional databases have been used since a long time, for data storage in web and business applications that required few user writes. With the advent of Web 2.0 applications and social-networking applications such as Facebook, Twitter and LinkedIn that required million of user reads and writes, storage of information, support and maintenance, have become the biggest challenge. Facebook and Twitter accumulate terabytes of data everyday for millions of its users [1]. Google have to store large amount of data in the form of web pages and links. In 2008, it already passed the mark of more than 1 trillion unique URLs [2]. These kinds of services are likely to grow massively in next few years [3]. Traditional relational databases have many problems to confront these challenges. Relational databases have schema evolution problems. The powerful characteristics of relational databases that are requirement of fixed data model and referential integrity are not according to needs of changing business requirements. Scaling and semi-structured data storage is the requirements of today's businesses. NoSQL technologies were developed as a solution for storing "Big Data" and to deal with large scaling needs. NoSQL ("Not Only SQL") are non-relational and horizontal scalable databases. These databases are flexible to store unstructured data and support high-availability. NoSQL databases do not use Joins to relate data and SQL to retrieve data.

II. CLASSIFICATIONS OF NOSQL SOLUTIONS

NoSQL solutions are identified into four Classes: Key-Value data stores, Column-oriented data stores, Document data stores and Graph databases [4]. These data stores work differently and used for different applications. The detailed description of the classes and different tools available of these classes is given in this section.

A. Key-value Data Store

Like relational databases, Key-value data stores are consists of rows and columns. But key-value data stores have only two columns: key and value. These storage systems are basically associative arrays of key-value pairs. Each key is unique and used to retrieve the value associated with it. These values can be anything that is objects, list or hashes. Because of this, there is no need of fixed data model and key-value data stores are schema less. The query speed of these data stores is higher than the relational databases and is used for fast look-up. These data stores are designed to handle massive data loads and used in schema evolving applications, e.g. they can be used for storing market data from stock exchanges.

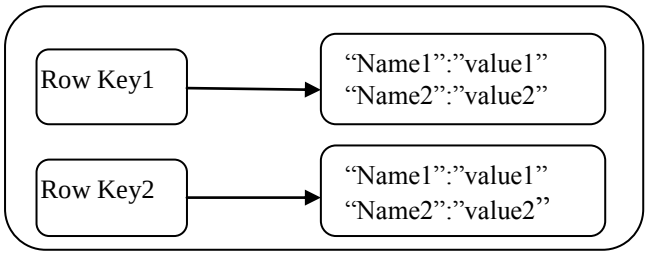

Figure 1: Representation of Key-value Data Store

Figure 1 shows the representation of the key value data store. Data is stored in the form of keys and values, where keys are simple objects and values can be lists, sets or hashes again. Unique row key is used to distinguish each row. The popular key value data store are Riak [5], voldemort [6], Redis [7], Tokyo Cabinet [8].

1) Riak:

Riak is a fully distributed, scalable key-value data store developed by Basho [5]. It was open-sourced under Apache2 license in August, 2009. Riak is written in Erlang. It provides Multi-Version Concurrency Control (MVCC)

for updates. In Riak, vector clocks are used as version control mechanism, which are assigned when values are updated. Riak has both the features of key value and document databases. Like document database, in Riak objects are stored and fetched in JSON (JavaScript Object Notation) format and can have multiple fields and can be grouped into buckets (collections in document databases). But Riak does not have query mechanism of the document database. Only primary key lookup can be done. It also supports links which are relationships between objects. Links simplifies traversal requests and it also supports MapReduce in both Erlang and JavaScript.

2) Project Voldemort:

Voldemort is an open-source Key-value data store. It is developed by LinkedIn and currently in use at LinkedIn [6]. It is written in Java and is available under the Apache 2.0 license. Thrift API is used to access the database; there are native bindings to high-level languages as well that employ serialization via Google protocol buffers [9]. Voldemort provides Multi-version concurrency control (MVCC) for updates. In this data store replicas are updated asynchronously, so it is eventually consistent and does not guarantee consistent data. In voldemort data is automatically replicated over multiple servers and partitioned automatically so that each server contains only subset of data. Voldemort supports Berkeley DB and Random Access File storage, means it can store data in RAM but plugging in a storage engine is also permitted. In addition to simple scalar values voldemort also supports lists and records.

3) Redis:

Redis is an open source, BSD licensed key value data store [7]. Redis is written in ANSI C. It stores the data in the form of keys and values, where values associated with keys can be lists, ordered lists and sets. Redis load the whole document into main memory. So, all operations are run in memory but periodically save the data on disk asynchronously. Due to memory operations its performance is high. Redis supports master-slave replication to scale reads and Sharding to distribute writes. This data store also offers shell for simple interaction.

4) Tokyo Cabinet:

Tokyo Cabinet is an open source key value data store [8]. It was developed by Mikio Hirabayashi in Japan mainly for Japan's largest SNS site mixi.jp, it has been a very mature project yet [10]. Tokyo Cabinet is back-end server dealing with all of the data structures like B-Trees and Hash Tables. Tokyo Tyrant is networking interface that provides remote access to the data stored in Tokyo Cabinet. They support ACID transactions and binary array data types. They also support master/slave or dual master replication and have no support for automatic sharding.

B. Column-family Data Store

Column-family data stores are similar to key-value data stores but here the data is stored by columns. Columns of similar data are stored in a same file on disk known as column family. A column family contains row key similar to row key in key value store. A row key contains super-column or column. Super column is a column that contains other columns but not super-columns. A column is a tuple of name and value. Figure 2 shows the representation of column-family data stores. Compared to relational databases keyspace corresponds to database, column family to table and row key to primary key, column name/key to column name and column value to column value. The leading technologies in this area are Google's Big Table [11], HyperTable [12], Cassandra [13] and HBase [14].

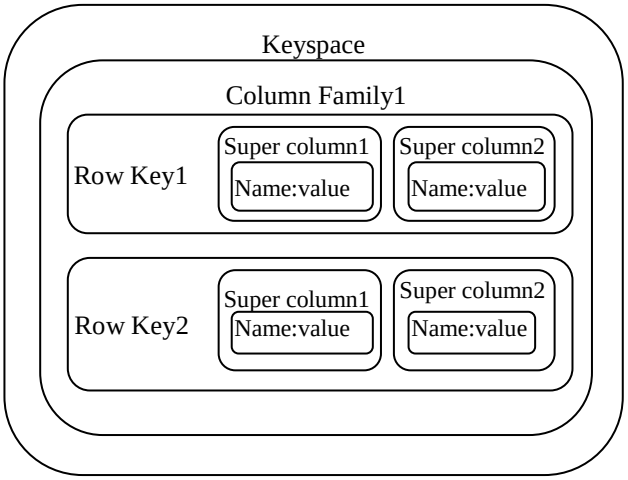

Figure 2: Representation of Column-family Data Store

1) Google's Big Table:

Big Table as described by Google, is "a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers" [11]. Big Table was designed to achieve distributed storage, high availability and high performance. Big Table provides structured storage of data like traditional relational databases but adopts a different model from relational databases. The data model of Big Table can be inferred as multidimensional sorted map. The data structure of Big Table is collection of key-value pairs. Big Table consists of column families. Column families consist of rows. Rows consist of column with key-value pairs. Timestamp are used to version the value of each column key.Google Big Table uses Google File System, Chubby Lock Services and SST table technologies. Many projects at Google store data in BigTable, including web indexing, Google Earth and Google Finance [11].

2) HyperTable:

HyperTable is an open source column-oriented data store developed by Zvents Inc. in January 2009 [12]. HyperTable was designed to solve the problems of scalability. HyperTable delivers maximum efficiency and optimum performance. It is based on the design of Google (Google's Big Table) to meet the scalability requirement.

HyperTable can be deployed on the top of the Hadoop HDFS or cloud store KFS file systems. It supports SQL like query language known as HQL. It is written in C++. HyperTable has tales consists of rows having unique rowID. Each row consists of column families that can have number of column qualifiers. Each of these column qualifiers defines a unique column value. Replication and partition of tables over servers is done using key ranges.

3) Cassandra:

Facebook is one of the most popular social networking websites. It is used by millions of people to connect with friends and relatives. Facebook applications required a data store that can manage billions of writes per day. To meet these requirements, Facebook engineer Prashant Malik and Dynamo's original author Avinash Lakshman have designed a distributed, scalable, column-oriented datastore that combines Google Big Table's structured data model with Amazon Dynamo's eventually consistent, decentralized storage model.

Cassandra is an open source data store with flexible schema [13]. Cassandra is written in JAVA and Thrift API is used to access Cassandra. Other APIs such as REST APIs can also be used. Cassandra can also have super-columns with in column families that have related columns. It provides another level of grouping. Cassandra is distributed data store which is composed of lots of database nodes. Data is replicated on these nodes based on eventual-consistency. To achieve scalability only nodes need to be added. All nodes in a cluster are same. Cassandra uses an order hash index which gives benefit of both hash table and B-Tree index. Cassandra supports rich data model and powerful query language.

4) *HBase:*

HBase is an open-source, distributed column-oriented data store [35]. It is based on HDFS (Hadoop Distributed File System) [14]. It is written in JAVA. It supports ACID transactions. Thrift interface is used to access the HBase and it also supports REST calls. The HBase data model is based on BigTable data model [11]. Each HBase table has a name. Rows have RowID, which is user defined array. Tables are organized into regions, which are stored separately. Each region has a contigious RowID defined by (first row, last row). Columns are organized into column family. HBase can handle both structured and semi-structured data naturally. It has dynamic and flexible data model and does not constrain the data types and kind of the data to be stored. For one column, it can store integer in one row and string in other row.

C. Document Databases

A document database is used to store, retrieve and manage semi-structured data. In this database, data is stored in the form of documents. The format of document can be XML, YAML, and JSON. The documents are stored into collection. Compared to relational database, document corresponds to record (tuple) and collection corresponds to table. But unlike relational databases, documents do not need to have static schema that is documents could have completely different fields. Any number of fields can be added to the documents without wasting space by adding same empty fields to the other documents in a collection. So, document databases are easy to use and dynamic data can be easily created. The document can be complex and the entire object model can be stored in one document and can be read and written at once. So, there is no need to create a series of insert statements and complex procedures. The documents are independent. It improves the performance and decreases side effects of concurrency. The popular document databases are CouchDB [15], MongoDB[16], ThruDB[17], Terrastore [18]. The representation of document database is shown in Figure 3.

1) CouchDB:

CouchDB (Cluster of Unreliable Commodity Hardware Data Base) is schema-less, fault-tolerant, distributed document database [15]. It is maintained by Apache Software Foundation and written in Erlang. CouchDB is schema free, it means in this data-store data is stored in arbitrary JSON format. To access the CouchDB data store RESTful HTTP/JSON API is used.

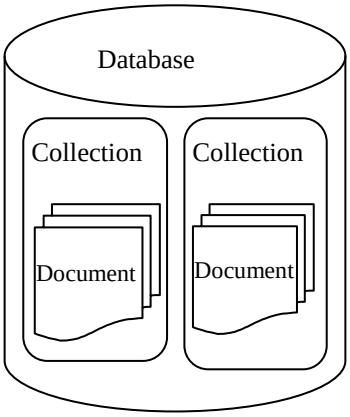

Figure 3: Representation of Document Database

To manage concurrent access to the documents, CouchDB uses Multi-Version Concurrency Control (MVCC). It means documents are versioned using SequenceID. If someone fetches the document for updates, CouchDB will notify the application. Then application combines the update or overwrites the update with its update. So the file on disk is always in consistent state. To query the database "views" are used. Views are of two types: Temporary views and permanent views. Temporary views are loaded into memory and permanent views are stored on disk [13]. Views are basically map/reduce functions, implemented in JavaScript, that process the data stored and return the results. CouchDB keeps these views updated by running new inserts through map/reduce functions. CouchDB supports asynchronous replication for scaling. It has interfaces with many programming languages like JAVA, C, PHP, LISP, Python that convert Native API calls into RESTful calls.

2) *MongoDB*:

MongoDB is a GPL open-source document database developed by 10gen in 2008. It is written in c++ and its query language is javascript. The word mongo in its name comes from word humongous [16]. MongoDB does not support Joins or ACID transactions. MongoDB was designed to handle growing data storage needs. MongoDB provides interactive JavaScipt shell for database management. It supports a rich, ad-hoc query language of its own. In addition, there exist bindings for many programming languages like Java, C, C++, Erlang, Python, Ruby and more. The data type of many languages is built-up of key-value pairs and some languages support JSON. It is easy to create documents for MongoDB using these languages. MongoDB keeps all of the most recently used data in RAM. When indexes are created for queries, all data sets fits in RAM and queries are run from memory. There is no query cache in MongoDB. All queries are run directly from the indexes or data files. Data is physically written to the disk with in 100 milliseconds.

3) Terrastore:

Terrastore is open-source, distributed database [17]. It is based on Terracotta that is an industry proven and clustering technology. It uses an Apache 2.0 license. It is written in JAVA and can be accessed through HTTP protocol. It provides advanced scalability and elasticity features without sacrificing consistency. It provides per-document consistency feature that is latest value is obtained. It is schema-less, it means data is stored in JSON format. It is easy to work with Terrastore, as fully working cluster can be easily installed by using only few commands and there is no need to edit any XML file.

4) *ThruDB*:

Thrudb is a set of simple services built on top of Facebook's Thrift framework that provides indexing and document storage services for building and scaling websites [18]. Its main purpose is to provide flexible, fast and easy-to-use services to the developers. It has multiple storage back ends like BerkeleyDB, Disk, MySQL, and S3. Thrudb provides web scale data management by having various services.

Thrudoc is a simple key value storage system designed to work on a number of underlying storage backends, such as files on disk or S3 records [18]. Thrucene provides lucene based indexing for searching data from document. Throxy provides services of partitioning and load balancing. Thruqueue provides persistent message queue services.

D. Graph Databases

The websites like Facebook, Twitter and LinkedIn are the consequences of web 3.0. These websites accumulate terabits of data every day and the data accumulated by these websites is more connected and semi-structured. The use of relational databases leads to problems because of data modeling (fixed-schema requirement) and horizontal scalability constraints. Technologies like graph database were developed to store this kind of data.

Graph database uses graph structure consisting of nodes, edges and properties to store and represent the data. Graph database is index-free adjacency storage system. This means that every element contains direct pointer to every adjacent element and no index look-ups are necessary. These are suited for the applications in which there are more interconnection between the data like social networks and web-site link structure, as the graph is natural way to represent relationship between user/nodes. Most common graph databases are Neo4j [19], OrientDB[20], AllegroGraph RDFstore [21] and HyperGraphDB [22][23]. Figure 4 represent schema of graph databases, here nodes represent entities like people, item. Properties are the attributes of these nodes and edges are the lines that connect nodes to each other. The edges represent relationships between nodes and can also have properties.

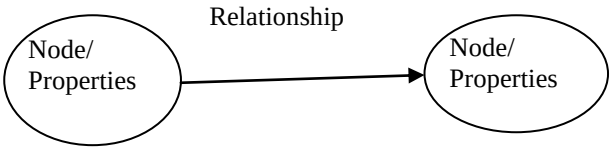

Figure 4: Representation of Graph in Graph Database

1) Neo4j:

Neo4j is one of the leading open source graph database [19]. It is supported by Neo Technology. It stores the data in the form of graphs rather than tables. It is implemented in Java, but it has binding for many languages like Python, Ruby, Jython, Scala. It supports ACID transactions and master-slave replication. Neo4j is exceptionally scalable and mainly used in embedded applications. Cypher and Gremlin query languages can be used for graph traversals.

2) OrientDB:

OrientDB is popular open-source document-graph database having features of both document and graph database [20]. Like document databases, it stores the data in the form of documents. Like graph databases, relationships are managed as direct connections between records. It is purely written in java. It supports ACID transactions and recovers pending documents on crash. It supports extension of SQL language that handle relationships without using Joins. It is extremely light. It uses MVRB-Tree indexing algorithm, which is derived from the Red-Black Tree and B^+ Tree and has fast-insertion and ultra fast lookup. It can work in schema-less mode, schema-full and a mix of both.

3) *AllegroGraph RDFstore:*

AllegroGraph RDFstore is a persistent RDF graph database that purportedly scales to "billions of triples while maintaining superior performance" [21]. It is mainly used for developing semantic web applications. It can store data and meta-data as RDF triples, which is standard format for linked data. AllegroGraph supports SPARQL, RDFS++ and prolog reasoning from various client

applications. It is developed by Franz Inc. It supports ACID transactions (Commit, Rollback, Check pointing).

4) *HyperGraphDB:*

HyperGraphDB is general purpose, open-source data storage mechanism based on a powerful knowledge management formalism known as directed hypergraphs [23]. This model allows a natural representation of higher-order relations, and is particularly useful for modeling data of areas like knowledge representation, artificial intelligence and bio-informatics [24]. It is persistent data model designed for AI, Knowledge representation and Semantic Web. It can also be used as an embedded object-oriented database for java projects of all sizes.

III. ALGORITHM DESIGN AND IMPLEMENTATION

In this section an algorithm for transformation of data from relational database to document database has been proposed. MySQL is selected as relational database and MongoDB is selected as document database.

A. MySQL

MySQL [25] is most popular open source relational database management system. It is owned by Oracle. It is used in a variety of projects and is stable. MySQL is fast, robust, easy to use, multi-user and multi-threaded SQL database server. MySQL support ACID transactions and foreign keys. Because of its simple installation and setup procedure, it is used by small companies as well as large production environments such as Twitter. In Twitter, it is used in the way of a key-value store [2]. MySQL is written in C and C++. It has drivers available in most programming languages that allow programs to access the API. An interactive shell is provided for SQL queries. The programs that use the provided native drivers can also use SQL to interact with the database.

The relational schema of database used as example for implementation and design is shown in Figure 5. The database is based on Post-Comment system. A user can post any number of posts at any time on the site. User can also give comments and sub-comments under any post. Tags are used to classify the posts. The database consists of four tables: User, Post, Comment and Tag. User table attributes are uid, username, realname, email, homeurl, pswd. Uid is primary key. The user can login into the system by using username and password. So, user must have valid account. Homeurl is the url of its home page. Post table has pid, uid, title, body, primaryurl, and time attributes, where pid is the primary key of the post table. Uid is foreign key that refers to the primary key of the user table. Any user can post an article without having valid uid. For an outsider, uid is null. Attributes of comment table are cid, parented, postid, userid, title, comment, time, score, descriptor, where cid is the primary key of table. Nested comments are also allowed. Nested comments are comment under a comment. Postid is the id of the post under which user has commented. Parentid is the id of the comment under which sub-comment is written. Attributes of tag table are tagid, name, pid. Tagid is primary key. Pid is foreign key which refers to the pid of the post table. Tags are used to categorize the posts.

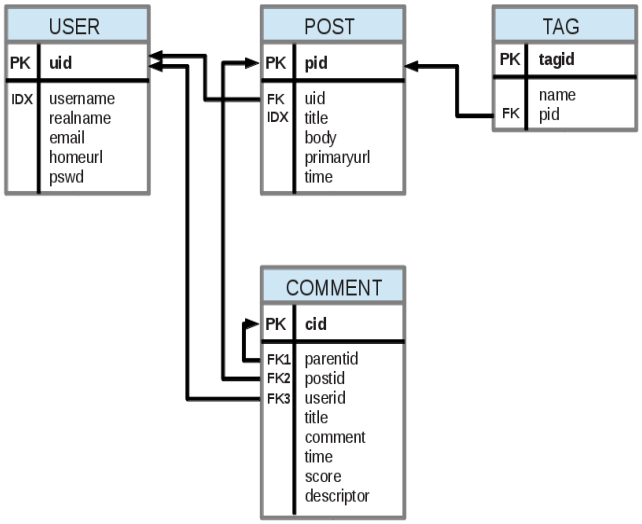

Figure 5: Schema of Sample Relational Database

B. Pentaho Integration Tool

Pentaho is one of the business analytic tools. Pentaho brings together IT and business users to easily access, explore and analyze all data that impacts business results. Pentaho and 10gen also offers MongoDB-based big data analytics solution to the market. This solution combines MongoDB with Pentaho's visual interfaces for high-performance data input, output and manipulation, as well as data discovery, visualization and predictive analytics. This makes it easy and productive for IT staff, developers, data scientists and business analysts to operationalize, integrate and analyze both big data and traditional data sources [26].

C. Proposed Algorithm

Based on the data model and features of the MongoDB, an algorithm has been proposed for data transformation from relational database to MongoDB document-oriented database.

Algorithm: DataTransformation

Input: Source database as S
Output: Target database as T

```
1.  String tbl[]:= get_table_names(S)
2.  String embdtbl[]:= get_embed_table_names(tbl[])
3.  String embedded:= get_embedded_table(embdtbl[])
4.  for all i € {1. . tbl.len} do
5.          integer toBeEmbedded:=0
6.          for all j € {1. . embdtbl.len} do
7.                  if (tbl[i]==embdtbl[j]) then
8.                          toBeEmbedded:=1
9.          end for
10.         if (! toBeEmbedded)
11.                 String
    colnames[]:=get_column_names(tbl[i])
```

```
12.                 for all k € {1. . colnames.len} do
13.                     String colvalues[]:=
    get_column_values(colnames[k])
14.                 end for
15.             generate_tbl[i]_text_file (colnames[] ,
    colvalues[])
16.             String PK[]:= get_primary_keys(tbl[i])
17.             String ETN[]:=
    get_exported_keys_table_names(tbl[i])
18.             Boolean ETNcheck:=
    check_for_null(ETN[])
19.                 if(!ETNcheck) then
20.                     String EPK[] :=
    get_exported_tables_primary_key (ETN[])
21.                     Generate_tbl[i]_keys_text_file
    (PK[], EPK[])
22.                 end if
23.                 else
24.                     Generate_tbl[i]_keys_text_file
    (pk[])
25.         end if
26.   end for
27.   String clmJoin[]:= get_col_for_join(embdtbl[])
28.         for all m € {1. . embdtbl.len} do
29.             if (embdtbl[m] == embedded) then
30.                 String embeddedtbl:= embdtbl[m]
31.             else
32.                 String embeddingtbl:= embdtbl[m]
33.         end for
34.   String
    colnames[]:=get_column_names(embeddedtbl
    LEFT JOIN embeddingtbl)
35.         for all k € {1. . colnames.len} do
36.             String colvalues[]:=
    get_column_values(colnames[k])
37.         end for
38.   generate_embedding_text_file
    (colnames[],colvalues[])
39.   String PK[]:= get_primary_keys(embeddedtbl])
40.   String ETN[]:=
    get_exported_keys_table_names(embeddedtbl])
41.   boolean ETNcheck:= check_for_null(ETN[])
42.         if(!ETNcheck) then
43.             String EPK[] :=
    get_exported_tables_primary_keys (ETN[])
44.             generate_embeddedtbl_keys_text_file
    (PK[], EPK[])
45.         end if
46.         else
47.             generate_embeddedtbl_keys_text_file
    (pk[])
48.   Intergrate_with_pentaho ( )
49.   Load_data_into_MongoDB( )
```

D. Implementation

The proposed algorithm has been implemented by using NetBeans Java IDE. For this purpose, MySQL is chosen as relational database. Data is transformed from relational database (MySQL) to document database (MongoDB). The steps involved in the transformation are as follows.

Step 1: In first step, connection with the relational database (i.e. MySQL) is established. Once the successful connection has been made to the MySQL server, then user chooses the database from the existing relational databases, which needs to be created in MongoDB. That is the database, whose data is to be transformed into MongoDB is selected.

Step 2: From the table list of the chosen database, the tables whose corresponding embedded documents are to be created in MongoDB, are chosen. In MongoDB, related data of two or more tables can be stored in a single document. This is known as embedding and documents are known as embedded documents.

Step 3: Select the columns of the embedded table on which join has to be applied.

Step 4: Using above information, text files are generated in the format accepted by Pentaho Data Integration tool as shown in Figure 6.

Step 5: Pentaho is used to load the data into MongoDB. Pentaho takes the text files as input and generates the corresponding collection in MongoDB.

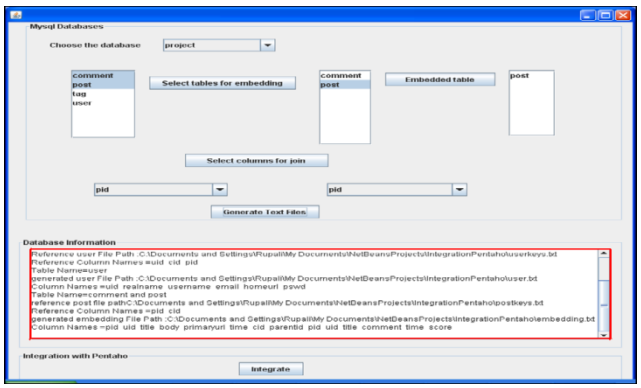

Figure 6: Text File Generation

IV. CONCLUSION AND FUTURE SCOPE

In this report, classes of NoSQL solutions and popular tools of these classes have been described. As many organizations are using NoSQL databases, there is need to transform the data from relational database to NoSQL database. In this report, a data transformation approach has been proposed for the transformation of relational database into MongoDB document database and the proposed technique is implemented using NetBeans Java IDE.

The proposed technique transforms the data from relational database to document based NoSQL class. This work can be extended for other classes of NoSQL databases also. The presented work has semi-automatic post algorithm steps. The work can be extended in this aspect to make transformation fully automatic.

REFERENCES

[1] M. Vardanyan (2011, May) Picking the Right NoSQL Database Tool. [Online]. Available: http://blog.monitis.com/index.php/2011/05/22/picking-the-right-nosql-database-tool/?attest=true&opt=vers

[2] N. Ruflin , H. Burkhart and S. Rizzotti, "Social-Data Storage Systems," in *Databases and Social Networks*, Athens, Greece, 2011, pp. 7-12.

[3] S. Higginbotham. (2010, September) Sensor Networks Top Social Networks for Big Data. [Online]. Avaialble: http://gigaom.com/2010/09/13/sensor-networks-top-social-networks-for-big-data-2/

[4] S. Edlich, A. Friedland, J. Hampe and B. Brauer, *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*, 2nd ed.: Hanser Fachbuchverlag, 2010.

[5] The Riak website. [Online]. Available: http://basho.com/riak/

[6] Project voldemort: A distributed database. [Online]. Available: http://www.project-voldemort.com/voldemort/

[7] J. Zawodny, "Redis: Lightweight key/value store that goes the extra mile," *Linux Magazine*, August 2009.

[8] Tokyo Cabinet: a modern implementation of DBM. [Online]. Available: http://fallabs.com/tokyocabinet/

[9] Protocol Buffers. Google's Data Interchange Format. [Online]. Available: http://code.google.com/p/protobuf.

[10] J. han, E. Haihong, D. Guan, and D. Jian, "Survey on NoSQL database," in *Pervasive Computing and applications (ICPCA) ,2011 6th Int. Conf.*, 2011, pp. 363-366.

[11] F. Chang et al., "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th symposium on Operating systems design and implementation*, Berkeley, CA, USA, 2006, pp. 205–218.

[12] The HyperTable website. [Online]. Available: http://hypertable.org/

[13] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35-40, April 2010.

[14] L. George, *HBase: The Definitive Guide*.: O'Reilly Media, 2011, pp. 45-89.

[15] (2010) The CouchDB website. [Online]. Availble: http://couchdb.apache.org/

[16] The MongoDB website. [Online]. Availble: http://www.mongodb.org/

[17] Terrastore. [Online]. Available: http://code.google.com/p/terrastore/

[18] T. J. Luciani. Thrudb: Document Oriented Database Services. [Online]. Available: http://thrudb.googlecode.com

[19] The Neo4j website. [Online]. Available: http://www.neo4j.org/

[20] The OrientDB website. [Online]. Available: http://www.orientdb.org/

[21] The AllegroGraph RDFstore website. [Online]. Available: http://www.franz.com/agraph/allegrograph/

[22] B. Iordanov, "HyperGraphDB: A Generalized Graph Database," in *WAIM'10 Proceedings of the 2010 int. conf. on Web-age information management* , 2010, pp. 25-36.

[23] The HyperGraphDB website. [Online]. Availble: http://www.hypergraphdb.org/index

[24] R. Angles, "A Comparison of Current Graph Database Models," in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference*, Arlington,VA, 2012, pp. 171-177.

[25] M. Widenius and D. Axmark, "MySQL Introduction," *Linux Journal*, vol. 1999, no. 67, November 1999.

[26] J. bleuel. (2012, May) Kettle and NoSQL: MongoDB. [Online]. Available: http://kettle.bleuel.com/2012/05/23/kettle-and-nosql-mongodb/