KEY/VALUE (NOSQL) DATABASE

OPERATIONAL CONCEPT DOCUMENT

CSE681-PROJECT 1

-

By Ronak Bhuptani (SUID: 429019830)
Instructor: Jim Fawcett

Date: 16th September, 2015

Contents

1. Executive Summary

This document represents the Operational Concept of Key/Value (NoSQL) Database.

Over the last few years, we have seen number of technical activities with “Big Data” and real time applications. Examples include data collection of social networks such as Facebook, Amazon and Twitter, data collection and analysis from large Hardon collider, measuring weather patterns, geographic data collection of earth and many more. These applications need their database to control very large database. Traditional database system would not be able to handle this amount of load and are not perfectly suited for these applications. Unlike traditional database system, NoSQL Database is none—relational and largely distributed database system. NoSQL Database is not built on tables and it may not provide full ACID (Atomicity, Consistency, Isolation, and Durability) but still has a fault tolerant architecture. Primary users of this system will be other programming languages, software developers, instructors and teaching assistants and, database designers and administrators.

There are various methods to classify NoSQL database. Our system will use Key/Value stores method to create the system. This system will use Dictionary as its principal data model. Data is represented as a collection of a unique key and value pairs. It can be used when faster key/value access is needed in application. It also provides sharding feature so the application systems which have huge data collection and need to store their partition database into smaller parts to store them on different servers, it can use NoSQL Database System. In Physics, NoSQL database system is used to handle large number of transactions carried out by the world’s largest and most powerful particle accelerator and the data will be later analyzed

This system will be implemented in C# using the facilities of the .Net framework class libraries and Visual Studio 2015. The system will be divided into separate modules in which each module performs specific tasks. Major modules of the system are as below.

- TestExec: TestExec package is the main entry of the application. It will demonstrate that all the requirements are met.
- DBEngine and DBFactory: These packages will be used to create generic key/value in-memory database.
- Sharder: This will help to create shards of immutable database from different queries which will return a collection of keys.
- QueryEnfine and DBFactory: These packages will be used to process the simple and compound queries which will result in a collection of keys.

In implementing this system, few important issues are identified and their potential solutions are recommended. Some of the critical issues are:

- Eventual Consistency – in which database changes are made persistent eventually so queries for that data might not return updated data immediately if read from persisted XML file.
- Sharding – in which database are sharded based on different queries and are stored in XML files.
- Pluggable sharding – sharding strategies may vary from database to database and it will be difficult to support pluggable sharding in our NoSQL Database system.

The NoSQL Database System thus ensures that it can handle a huge data collection and such system can be implemented in considerable period of time and with available resources.

2. Introduction

In the last few years, Informational Companies have developed huge number of applications and many of these applications has often too large data collection that needs database system which can provide high scalability and flexibility. NoSQL Database systems are designed for such kind of applications.

2.1 Application Obligations

The main responsibility of this application to handle large data collection such that it can offer performance, high scalability, flexibility and security. The primary obligations of the system would be:

- To store the key/value pairs in dictionary data collection.
- To support add/edit/delete operations on key/value pairs.
- To support query processing on in-memory and persisted XML files.
- To shard the large data in smaller data collections.

2.2 Organizing Principles

The organizing principles are to perform the primary functionalities of NoSQL database system through Database Engine, Query Engine, Sharder and Item Factory modules while other functionalities are performed using separate isolated modules which will communicate with Test Executive module.

2.3 Key Architectural Ideas

The Key idea in NoSQL Database System architecture is to use Key/value pairs to store the data. In this application, we will be using C# language with .NET framework 4.6 and Visual Studio 2015. To store Key/value pairs, we will use Dictionary class. Dictionary uses inbuilt Hash function to map key value pairs which makes it the best choice for this application. Because of Hashing, we can perform quicker query processing and sharding which is the primary requirement of this database system.

3. Actors

NoSQL database is used to create large data collection system. The following type of users can interact with the first version of NoSQL Database system.

3.1 Code Developers

Code developers will use this system to create a new database for the application in use. On this database they can add new Key/Value pairs or delete already created values in the database table using add and delete command. They can also edit the values in database using edit command, but they cannot edit key as all the keys are unique in database system.

Code developers can extract data or data collection from database using executing simple or compound queries on the database. Code developers can also shard database in persisted XML file.

3.2 Teaching Assistants & Instructor

Teaching Assistants and Instructor will perform testing tasks on this system. They will formally witness that all the requirements of database system are fulfilled. They are also responsible for identifying faults in the system, if there are any, and to report them to the developers of the system.

3.3 Other Programming Languages

Other programming languages can connect with this NoSQL Database System using Language Interface. This language will use this system to access the previously created database or it can also use this system to create a new database system for the application it is creating. On this database system, it can use all the commands such as add, delete or edit the key/value pairs. It can also use the system to execute simple or compound queries on the database system.

3.4 Database Designers & Administrators

Database Designers and Administrators need NoSQL database to define structure of database, to create database and to monitor database system's use. As per the application requirements, they need to configure the database system. They need to identify the structure to store and represent data. They also need to authorize access to other users to use and also decides which other systems and programming languages can interact with this system.

4. Uses

NoSQL Database System was introduced to overcome the limitation of SQL database system. Today, many companies are adopting this technology for their numerous use cases. Few of the use cases are explained below.

1) Capturing and accessing big data requires a very different type of database than traditional SQL database. Developers need more flexible and unstructured or semi-structured database to solve these issues. NoSQL database system fits very well for this type of requirements. Companies such as Facebook, google, amazon and many others use NoSQL database system to overcome this issue.
2) We have used Dictionary object to save the data in-memory. When we want to find certain value by its key, it directly jumps to that element as it uses hash lookup to search data. This is O(1) complexity for one search. So for the systems when latency is important for query execution, NoSQL database system is best match.
3) This Database system supports sharding in persistent XML files. You can shard huge data collection in shards and store them on disks as an XML file. It helps the system to balance the load and distribute data on the disks. So it can be used in data warehouse applications.
4) NoSQL Database system can be used in big systems or projects to collect data and to analyze the data. It can be used to collect data and analyze it from the Large Hadron Collider, analysis of Biological Genomes, measuring weather patterns, collecting data for global climate models.
5) The NoSQL system can be used in all web applications where you need to store all the user profiles information. The profile can have numerous fields and a typical website will have huge amount of users which will generate large amount of data. To handle this amount of data, NoSQL system fits well.
6) It can also be used in Mobile applications. Mobile Developers enhance their applications periodically and NoSQL can store data in schema less format which helps developers to change the structure of database quickly.
7) NoSQL database System can be used in e-commerce applications as they require dynamic scalability, large data storage and performance.
8) It can also be used in gaming as it needs to store huge amount of data and users on the system. It has seen that users in gaming can rapidly grow from few thousands to millions in very less period of time. Such high scalability an only provided by the NoSQL Database system.

5. Modular Structure

NoSQL Database system is divided into different packages which will perform different task assigned to them. Below is package diagram of the system.

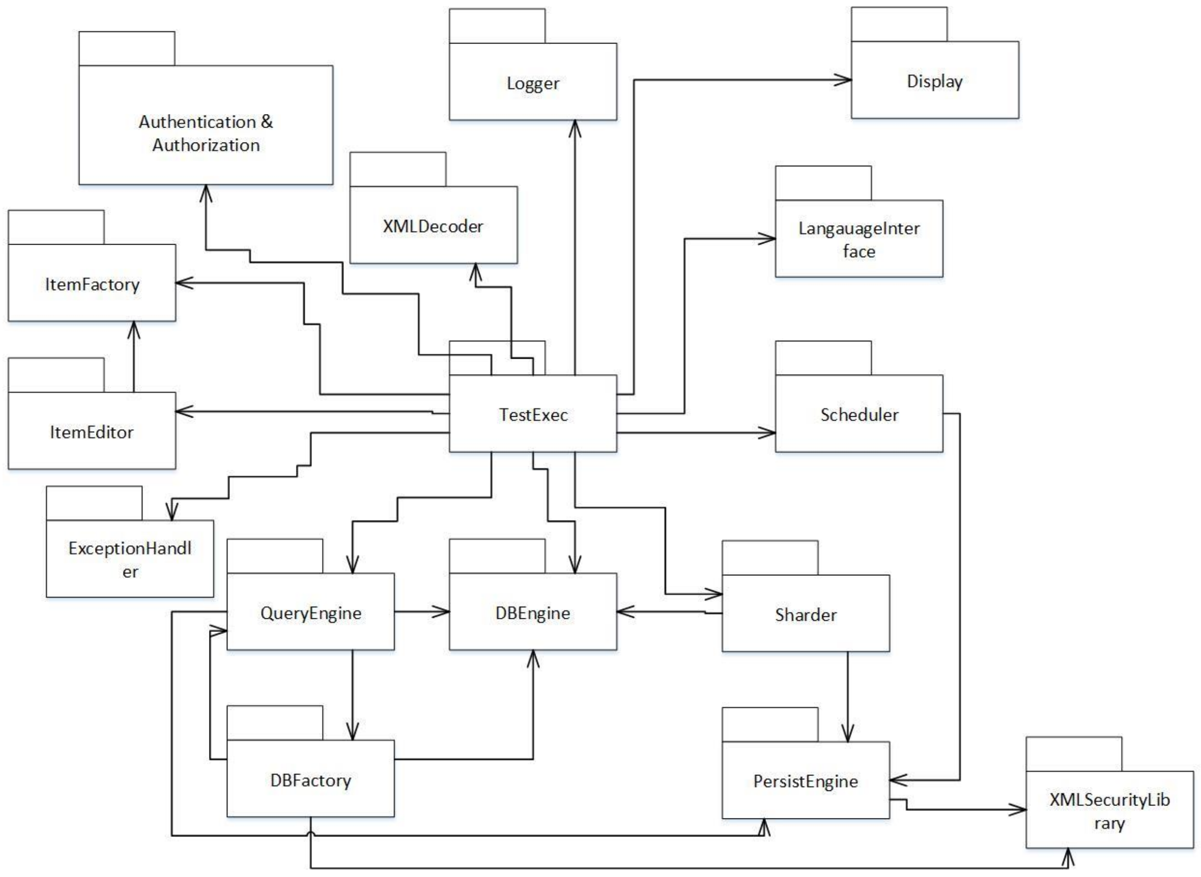


Fig. 5.1: Package diagram from NoSQL Database System

Package diagram is used here to represent the different layers of the system and how they are dependent on other packages. All the packages are explained below.

5.1 TestExec

TestExec package is the entry point of the system. Main method of this system resides in this package. This package will take input as an XML file and read it using XMLDecoder package. This package also makes calls to below packages to execute required tasks.

- Display package to show the output of different methods.
- DBFactory from which it will generate unique key and sets a value with supplied parameters.
- DBEngine, QueryEngine and DBFactory to execute queries
- ItemFactory and Itemeditor to create and update values of items.

5.2 XMLDecoder

This package is used by system to read the input XML files and pass it to TestExec package in application readable format. This information will be further processed in TestExec package.

5.3 Sharder

Every database has a limit on how many data they can store and process. If it reaches to its maximum capacity, it might slow down or become unresponsive and in extreme cases, system might crash. This is one of the critical issue and needs much attention to it. To overcome this issue, we have introduced Sharding of database. Sharding reduces the burden on individual database by spreading the rows of tables across multiple XML files on disks.

5.4 PersistEngine

PersistEngine package will store the Dictionary data to persisted XML file on hard disk. It can augment data to previously created persisted XML file when scheduler will trigger the save process after positive time intervals. It will also save data to previously created persisted XML file or create new XML files when query engine returns collection of data from execution of simple or compound queries. The proposed design of XML file is shown in Appendix 9.2

5.5 Scheduler

Scheduler will write the in-memory database to persisted XML file after positive time interval or after number of writes as described in input XML file. Scheduler will interact with TestExec to carry out its tasks. It will have one trigger function based on event and one trigger function based on time. Whenever the function is triggered, it will perform sharding on dictionary data.

5.6 Logger

Logger package handles all the functionality of logging. Logging gives you information about what your code does. It catches every action of your application with timestamp and their current and before states which will be stored in one file. This information is useful while debugging of the system and regression testing. Logging makes it much easier and smoother process.

5.7 Display

Display package will have methods to show the output of specific processes on console. For ex. If a command is run to edit value of key X to V2 from V1 then output will show as below.

Database state before the query was executed

Value of key X is changed to V1 from V2

Database state after the query is executed

Different processes will show different outputs based on the requirement. Please see Appendix 9.1.

5.8 DBFactory

DBFactory will help QueryEngine to execute compound queries. When a compound query needs to be executed, DBFactory will first divide the compound queries in N simple queries. It will fetch Dictionary data from DBEngine or PersistedEngine and execute those queries on these data. It will then temporary store all the result of simple queries and merge them as stated in query.

5.9 DBEngine

DBEngine package will have the primary Dictionary object where all the key/value pairs are stored. Its primary responsibility is to create a unique key for key/value pairs. Value will be generated from ItemFactory and will be added to dictionary object for that specific key. DBEngine will also interact with sharder to create shards on command and when scheduler event is triggered.

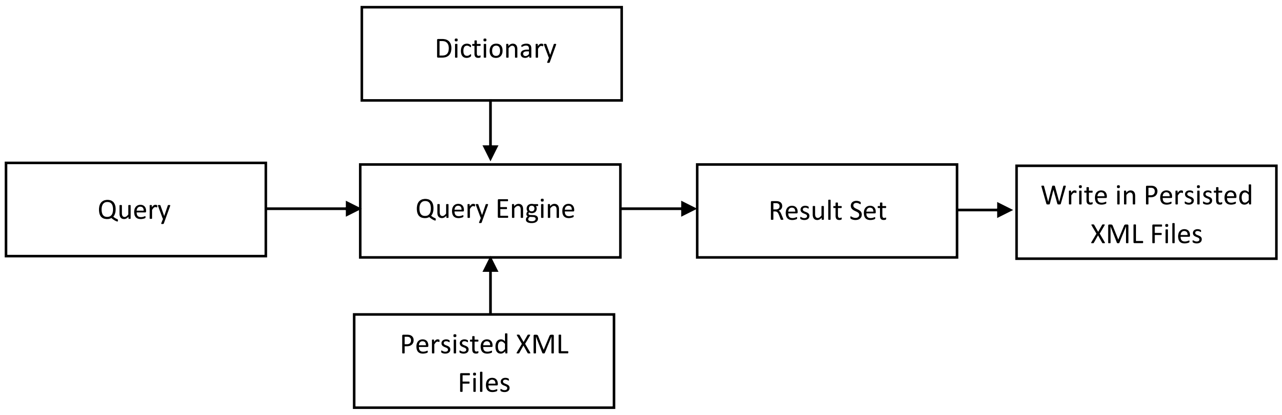
5.10 ItemEditor

This system can support editing values including addition/deletion of relationship, editing text metadata, and replacing an existing value with a new value. These tasks will be carried out by ItemEditor Package. ItemEditor is dependent on TestExec package. To edit any item in the database, first TestExec package will provide dictionary data from DBEngine and after editing data, it will be return to DBEngine through TesExec. We follow this chain of interaction to ensure the integrity of database

5.12 ItemFactory

Itemfactory will create generic type of value based on the input file. Values are consisted of metadata which includes a name string, text description of the item, time stamp, list which has a finite number of child relationship with other values and generic type of value.

5.11 QueryEngine



A typical NoSQL Database system will have a very huge data collection and users might need to fetch specific data from the database factory. Such request to fetch data will be complete by executing simple or compound queries. And these queries will be created and executed in QueryEngine Package. The functioning of QueryEngine package can be show as below.

QueryEngine will first create a simple or compound query based on the requirement and it can either execute that query on in-memory dictionary object or Persisted XML files which are on hard disks. The execution will return the result set which can be single row or multiple rows and it can be written on persisted XML file in disk. QueryEngine is one of the most important module of this system and it mainly interacts with TestExec, DBEngine, DBFactory and PersisEngine to carry out its functionality.

5.13 Authentication & Authorization

Authentication & Authorization package will handle verification of users. This package is introduced to ensure more security for the database system. This package will have one properties file, usually an XML file, and code file to read properties from that XML file which will include user credentials and their privileges. A typical XML properties file looks like as below.

```
<configuration id=”Config1” db=”db124”>
        <users>
                <user id=”user275”>
```

```
                    <username>User275</username>
                    <password>abc@123</password>
                    <read_database>1</readdatabase>
                    <write_database>1</writedatabase>
                    <modify_database>1</modify_database>
                    <shard_persist>1</shard_persist>
                    <queryexecutor>1</queryexecutor>
            </user>
            <user id="user278">
                    <username>User278</username>
                    <password>xyz@123</password>
                    <read_database>1</readdatabase>
                    <write_database>1</writedatabase>
                    <modify_database>0</modify_database>
                    <shard_persist>0</shard_persist>
                    <queryexecutor>0</queryexecutor>
            </user2>
    </users>
</configuration>
```

Properties.cs will read from this file and compare the values with credentials of input XML file and if not match, it will show an error message on console.

5.14 ExceptionHandler

Exceptions are things that occur which are not expected in the system as a part of normal process. And exceptions are tend to occur in huge system but if the system could not handle such exceptions, it can lead to system failure. To handle this kind of exception, we have introduced ExceptionHandler which will control the process flow if any exceptions are detected. Typically, whenever exceptions are detected, control is passed to special exception handling method from that point or it will return to the last known state of the system. This package will interact through all other packages through TestExec package.

5.15 XMLSecurityLibrary

This package will make persisted XML data collection more secure by using XML-Enc method which will encrypt XML data and make them secure from the attackers. To secure XML files, it will interact mainly with persistEngine. DBFactory will use this package to decrypt XML files into application readable format.

5.16 Language Interface

Function of this package is to provide interface between other programming languages and NoSQL Database System. Using this interface, other programming languages can connect this database system and can used all the features of the system such as creating a new database, adding/deleting/editing of key/value pairs, persisting database to an XML file etc. This package will only interact with TestExec package.

6. Activity Diagram

Activity diagrams are used to illustrate activities of the functionalities of business system. Below are the activity diagrams for whole system, query processing and sharding process.

6.1 High Level Activity Diagram

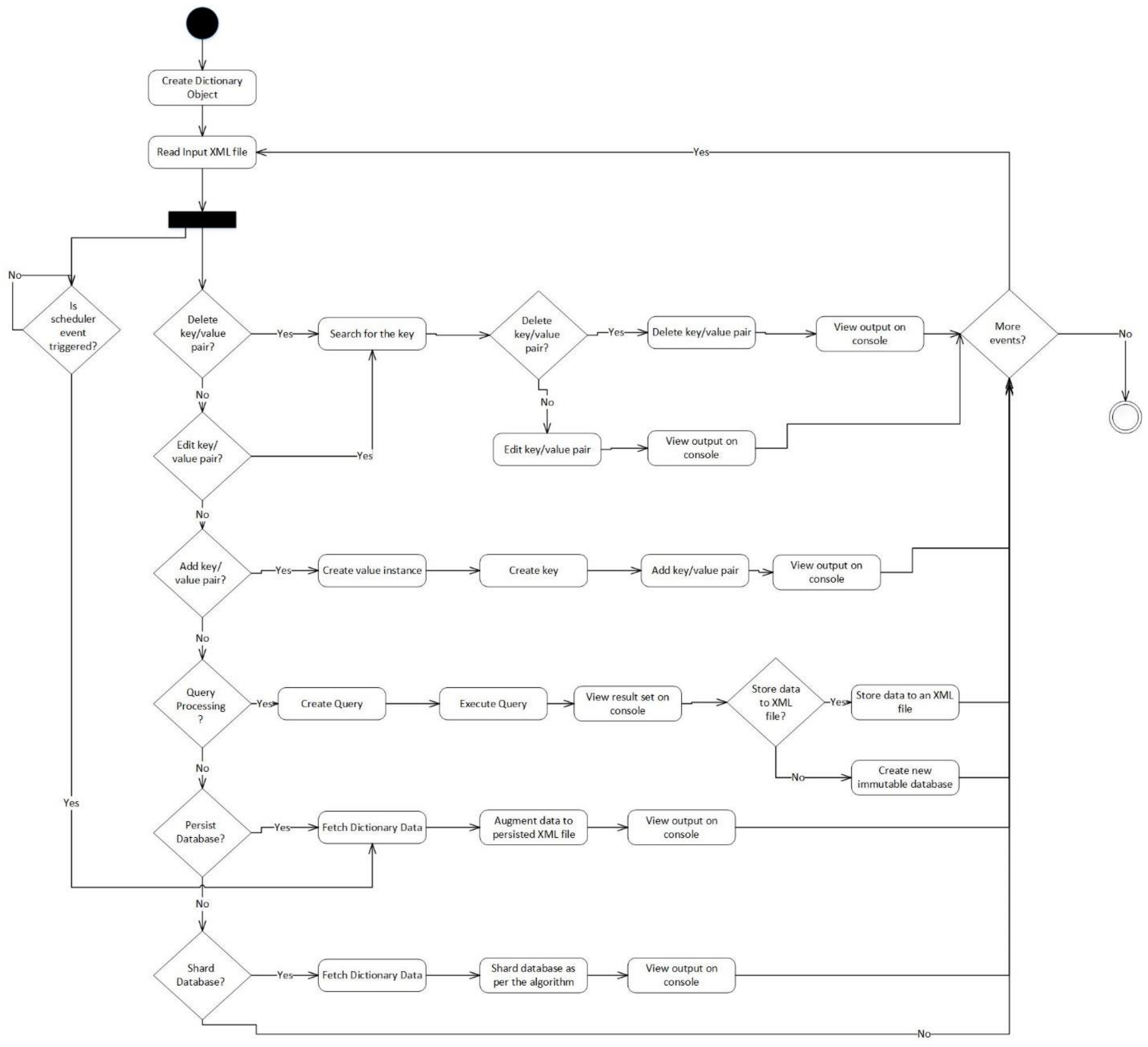


Fig.6.1: Activity diagram for whole system

Description: The above activity diagrams shows all the functionality of this database system.

1) First decision node is used to check if we need to delete a key/value pair from data collection. If the answer is yes then it will search for the key and then will delete that key/value pair and display the before/after state with the key/value pair which is deleted.
2) Second decision node is to check if we need to edit a key/value pair. The process is similar to delete process. It will first search the key and then edit key/value pair and shows the similar output as above.
3) Third decision node is used to add new key/value pair. It will first create value from Itemfactory and then create a Key from DBEngine and add the newly created key/value pair in dictionary data collection. It will then show the before/after state with the key/value pair which is added.
4) Fourth decision node is used to check if we want to process any query. Query processing is explained in more detailed in next activity diagram.
5) Fifth Node is to check if we need to persist the database in XML file. First it will fetch dictionary data and then it will augment that data to persisted XML file and will show the stats of XML files on console.
6) Sixth Node is to check if we want to shard the database. Sharding process and data retrieval from shards is explained in brief in activity diagram 6.3.
7) With all these nodes, one more decision node will be active in parallel. It will continuously check if the scheduler event is triggered or not. When the scheduler event will be triggered it will augment the data in persisted XML file.

6.2 Activity Diagram for Query Processing

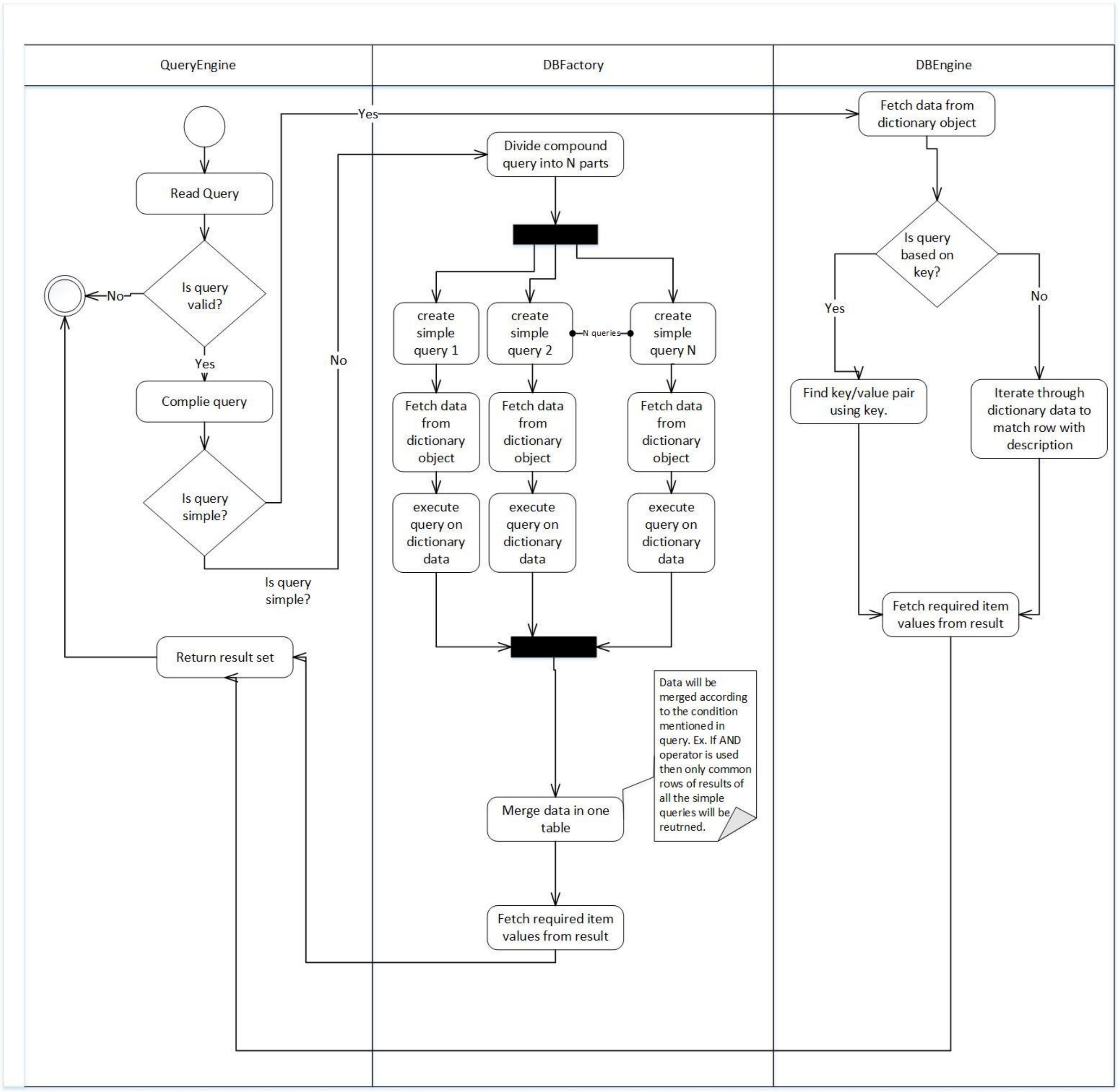

QueryEngine
DBFactory
DBEngine
Read Query
Is query valid?
No
Yes
Complie query
Is query simple?
Yes
No
Is query simple?
Return result set
Divide compound query into N parts
create simple query 1
create simple query 2
N queries
create simple query N
Fetch data from dictionary object
execute query on dictionary data
Merge data in one table
Data will be merged according to the condition mentioned in query. Ex. If AND operator is used then only common rows of results of all the simple queries will be reutrned.
Fetch required item values from result
Fetch data from dictionary object
Is query based on key?
Yes
No
Find key/value pair using key.
Iterate through dictionary data to match row with description
Fetch required item values from result

Fig.6.2: Activity diagram for Query Processing

Description: The above activity diagrams shows all the functionality of query processing.

1) QueryEngine will read the query.
2) QueryEngine will check if the query is valid or not. If the query is invalid, it will exit from that function. If the query is valid, it will proceed to next step.
3) Next step will be to compile query.
4) Then if the query is simple query then it will first collect the dictionary data from DBEngine.
5) If the query is based on key then the key/value pairs will directly be returned to queryengine as dictionary uses hashing to search data based on key.
6) If the query is based on value then we need to iterate through all the data in dictionary data.
7) Now if the query is compound, then it will be divided into N simple queries by DBFactory.
8) All the simple queries will be executed in parallel on dictionary data in DBFactory.
9) After that all the result sets will be merged based on the query parameters. For Ex., if AND operator is used then only common rows from all the result sets will be returned.
10) The returned result set from simple query or compound query will displayed on console.

6.3 Activity Diagram for Sharding Process & Data retrieval from shards

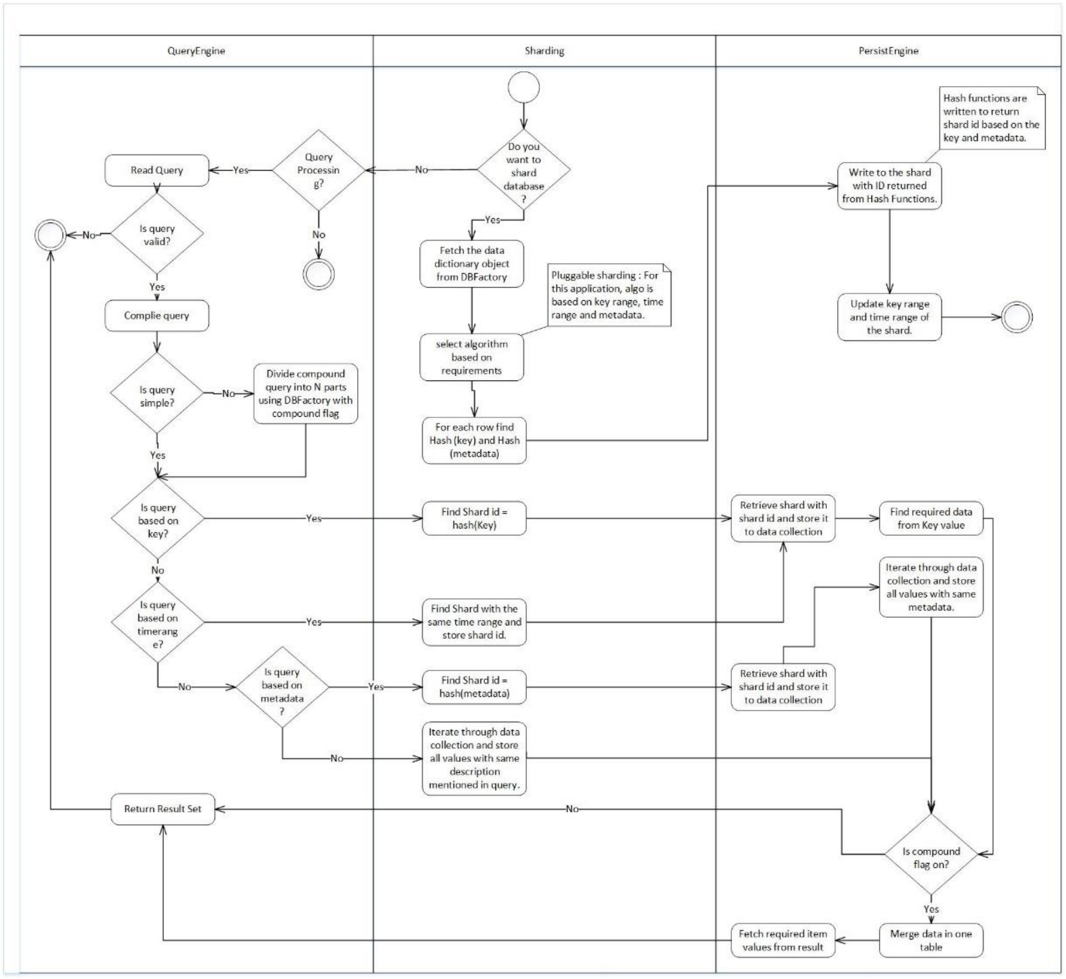

QueryEngine
Sharding
PersistEngine
Do you want to shard database ?
No
Yes
Query Processin g?
Yes
No
Read Query
Is query valid?
No
Yes
Complie query
Is query simple?
No
Divide compound query into N parts using DBFactory with compound flag
Yes
Fetch the data dictionary object from DBFactory
Pluggable sharding : For this application, algo is based on key range, time range and metadata.
select algorithm based on requirements
For each row find Hash (key) and Hash (metadata)
Hash functions are written to return shard id based on the key and metadata.
Write to the shard with ID returned from Hash Functions.
Update key range and time range of the shard.
Is query based on key?
Yes
Find Shard id = hash(Key)
Retrieve shard with shard id and store it to data collection
Find required data from Key value
No
Is query based on timerang e?
Yes
Find Shard with the same time range and store shard id.
Iterate through data collection and store all values with same metadata.
No
Is query based on metadata ?
Yes
Find Shard id = hash(metadata)
Retrieve shard with shard id and store it to data collection
No
Iterate through data collection and store all values with same description mentioned in query.
Return Result Set
No
Is compound flag on?
Yes
Merge data in one table
Fetch required item values from result

Fig.6.3: Activity diagram for Sharding Process and data retrieval from shards

Description: The above activity diagrams shows all the functionality of sharding process and data retrieval from shards.

1) First decision node is there to check if we want to shard database or retrieve database from shards.
2) If we want to shard database, then first it will fetch the data from dictionary data collection.
3) Then it will select algorithm based on the requirement of the system. For our system we have created algorithm which works to perform faster for data retrieving in case of queries based on key, key range, metadata and time range.
4) After that we will find Hash (Key) and Hash (Metadata) which will return shard id. Our Hash functions are written in a way that it will return shard id.
5) Next, we will write that data in that shard whose id was returned from hash function. If both hash function returns different shard id, then it will be written on both shards.
6) Last step before exiting, we will update the key range ad time range of the shard.
7) Now, if we have decided to execute queries on shards, it will first read query and check if the query is valid or not.
8) Next we will compile the query and check for if the query is simple or compound.
9) If the query is simple, then we will check query for 3 conditions.
    a. If query is based on metadata, then we will find associated shard using Hash (key) function.
    b. If query is based on time range, then we will find associated shard using time range of shard.
    c. If query is based on metadata then we will find associated shard using Hash (metadata) function.
    d. If query is based on none of the above criteria, then we will iterate through all the shards to retrieve the required data.
10) Then it will retrieve the required data from associated the shard returned by above conditions.
11) After that it will check if the compound flag is set. If it is not set than it is will directly return result set to QueryEngine.
12) If the query is compound then DBFactory will first divide the query in N simple query with compound flag set to 1.
13) Now, all divided compound query will work as simple query as explained above but when it reaches to decision node – “is compound flag on?” – It will return yes.
14) Now, all the result sets will be merged in one table based on operations given in query.
15) Then it will retrieve the required data and will return final result set to QueryEngine.

7. Critical Issues and Analysis

This section discusses about critical issues which are of crucial importance in relation to successful development of the system. If the required actions to address the issues are not taken, it might lead to development of defected system. Some of the critical issues and their potential solutions are discussed below.

1) **Eventual Consistency:** Scheduler will persist database contents after positive time interval or number of writes. If in-between any query is executed on XML file will not return the updated data.
   Solution: One of the possible solutions of this issue is to read the newly updated data from in-memory database rather that reading from persisted database.

2) **Sharding:** Sharding is used to shard huge data collection in small data collections. But there is a critical issue in deciding which algorithm needs to be implement so that we can access the data faster after sharding.
   Solution:
   1) To solve the issue, we will be using hash table and hash functions. The hashing will be based on key and metadata for this application. Initially we might have 10 shards but as the shard load is increased we can add more shards afterwards. Each shard will have unique shard ID. First we will find hash (key) which will return us shard id and then we will find hash (metadata) which will again return us shard id. If both are same then we will write that data in that shard else we will perform write functions on two shards. Every time we write in shard, we will also update the shard range and time range. So with every shard, we have associated key range and time range. For Ex.,
      We have key = 123 and time stamp = 10/10/2015 and Hash (key) & hash (metadata) returns shard id = 8, then we will write that data in shard 8 and we will update the key range and time range accordingly, which means if you have shard 8 with key range 125 to 200, it will be changed to 123-200 after write operation. And same for time range. This has been illustrated in activity diagram of sharding process in fig. 6.3.
      Impact on Design: By applying above algorithm, we can have key range and time range wise shards and also we can directly find shard from its key or metadata. The performance to find a data based on their key, timestamp and metadata will be faster.
   2) Another way to shard is based on time range. We can shard data which are created in specific day or month or year, based on number of shards available in system, in one XML file. So when we need to access data which are in those time range, it will be easy to find it as all the required data will be available in one shard. But the issue with this method is when we need to find data based on their key, we will need to traverse through all the shards.
   3) There is also one way to shard dictionary data. We can shard database based on the type of data it contains. For ex, one shard will have all the data with data type as string and another will have all the data with data type integer. So when the specific type of data are needed, we can find shards for them. Again the issue with this solution is same as above solution. We need to traverse through all the data for query based on key.

   Conclusion: In above three solutions, it seems that the best potential solution for our application is provided by the first solution and we will use that algorithm for sharding.

3) **Pluggable Sharding:** Sharding strategies may vary from database to database. So it will affect the performance in case of data retrieval to use same sharding algorithms for all database application.
Solution: To overcome this issue, we have introduced pluggable sharding. In pluggable sharding, we use different sharding algorithms for different database applications. If the most of the queries are based on key then we can use the algorithm mentioned above but if the main concentration of queries are on value then we need to tweak the algorithm such that it perform faster when queries are executed based on values.

4) **Query Handling:** Query processing is another critical issue in the system. The application can execute simple or compound queries. The issue here is how can we perform queries on in-memory database and persisted XML files?
Solution: We use dictionary data collection of C# framework in this system. So the in-memory database can be accessed directly through the key value if queries are based on key. And it will be faster as dictionary uses hashing to map key/value pairs. So if the query is simple and search is based on Key the dictionary will use hash function to search for the key/value pair and if key is based on data then we will need to iterate through the data dictionary to find the matching pattern. Now if the query is compound then first we will divide query into N simple queries and all N queries will be executed on dictionary data in parallel. The result set from all the queries will then be merged into single result table based on the conditions declared in query. For ex., if query need to find data with metadata X or description Y, then it will first divide queries into two simple queries and then execute both queries in parallel. And all the values with metadata X and all the values with description Y will be merged into one result table. This has been illustrated in Query processing activity diagram 6.2
For query processing in persisted XML file, the query can be executed based on the key values, metadata values, key range and time range. To execute query based on key, first we need to find Hash (key) which will return a shard id. After that we need to retrieve data from that shard to in memory dictionary data. And with key value we can directly search for the key/value pair as dictionary will use hashing to search. So it will be faster as well. Queries based on other criteria can also be executed as illustrated in activity diagram 6.3 for sharding.

5) **Heterogeneous Data Support:** This application has heterogeneous data support which means it can store different type of values in database. The issue here is how we can store different types of data using one data type?
Solution: We can use "object" type from .NET framework, which is unified type system of C#. The benefit to use this type is you can assign any type of variable to object type. We can create one interface which will handle all the type handling functionalities for this object. The value object will look like as below:

```
Public class value<Key,Data>{
        private string Name;
        private DateTime timestamp;
        private string description;
        private List<key> children;
        private object payload;
}
```

6) **Recovery from failures:** For the large systems, the system is supposed to verify and validate the inputs as well as it needs to handle the exceptions if any process results in undefined state which can eventually crash the whole system.
Solution: For handling recovery scenarios, the application has one package – ExceptionHandler – which will handle all the exceptions that are resulted due to undefined reasons such as wrong type of input, segmentation faults, invalid syntax of queries and many more.

7) **Programming Language Binding:** One of the critical issues is how other programming languages can connect with this database system successfully to access the in-memory or persisted data collection.
Solution: The solution is to create a LanguageInterface which will work as communicator between two components, i.e. NoSQL Database and other Programming Language. This interface will provide access to programming languages to use the features of this database system.

8) **Key/Value Size:** This is most critical issue in this system. The issue is no .NET object can be over 2GB in size. So the big objects such as video files which are larger than 2 GB cannot be stored in this database.
Solution: To overcome such issues, we can fragment this type of data in objects which will be less than or equal to 2 GB. To use the data again, we will restore the values by amending the fragmented data together.

9) **Data Security Issue:** Another critical issue which needs our attention is database security. Mainly the database will be stored in main memory as a Dictionary object and as an XML file in disks. The system needs a way to ensure that the data is secured and is not accessible directly from XML file. NoSQL database may become even more susceptible to exploits once attackers are able to identify hidden security or software weaknesses.
Solution: To overcome this critical issues, we have introduced one package – XMLSecurityLibrary – which can encrypt and decrypt data of XML file using XML-Enc a specification, governed by a W3C recommendation.

10) **Authentication & Authorization:** Another critical issue related to security is authentication and authorization. A system will be vulnerable to data leaking if it does not verify that the person or other software is specific user of the system. A system also needs to confirm that a specific person has privileges to perform any particular action.
Solution: Solution to this issue is handled by the Authentication & Authorization package of the system. It contains properties file of the system which will declare all the permitted users and their privileges. When XML input file is executed by the system, it will verify the user and its privileges before executing any queries. If the user is not specified in properties file, it will throw a failure message.

11) **Demonstrating Requirements:** How can we test all the requirements using one input XML file on console display?
Solution: TestExec package will handle this issue by careful automation of series of tests and supported by processing the output on console using Display package.
Impact on Design: It will be effective to provide a method for each test that displays the requirement number and displays database state before and after each change.

8. Conclusion

In conclusion, NoSQL Database System can be used in system when huge data collection is needed. It provides users scalability, flexibility, safety and performance in database system. This OCD explains which actors and how they can use the system. We have discussed different use cases where NoSQL System fits best. The system is divided into cohesive packages which can interact with each other to perform all the task defined in requirements. It also provides reusability.

We also covered all the activities being carried out by the system using 3 diagrams in section 6. It describes how the events take place in the system and makes it easy to understand the system. We also discussed few critical issue which can result in serious design flaw if not provided the proper solutions mentioned in section 7. It can be ensured from this OCD that the system can be developed in considerable period of time and with available resources.

9. Appendix

9.1 Output Formats

I have displayed outputs of few functionalities in this section. I have create one sample Dictionary data collection as below to display the functioning:

Key	Value				
	Metadata	Time stamp	Description	Children	Value
124	First Name	10/10/2015 : 11:12:57	It describes about person's first name.	NULL	Ronak
354	Last Name	10/10/2015 : 11:14:25	It describes about person's last name.	NULL	Bhuptani
268	Address	10/10/2015 : 11:15:29	It stores info about person's address.	NULL	Syracuse
645	Phone Number	10/10/2015 : 11:20:29	It stores person's phone number.	NULL	1234567890

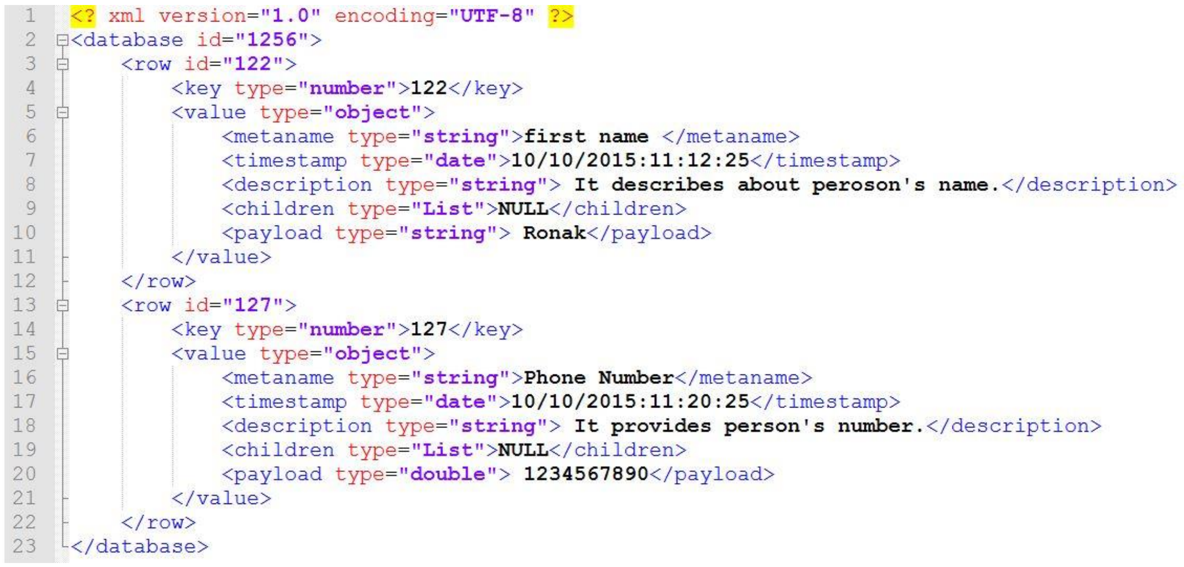
1) After adding key/value pair: For ex, I am adding a key value pair. Output will be show as below.
   Output:
   >Database db_name had 4 rows.
   >New row with "key = 283, Metadata = College, Time Stamp = 10/10/2015 : 11:23:45, Description = It says where the person is studying, Children = NULL, value = Syracuse University" is added.
   >Database db_name has 5 rows.
2) After editing/deleting key/value pair: These functions will have similar outputs except the second row. It will have information about this specific function.
3) After query processing: Let's say one query is executed and it has 7 rows of data. Table display after execution of query can be disable from Input XL file.
   Output:
   >Query "Query description" has been executed successfully and it has returned 7 rows.
   >"7 rows of data will be shown here"
4) Sharding: For ex, we have sharded 3 rows form dictionary data in shard id = "1265".
   Output:
   > 3 rows have been sharded into shard 1265.

5) Scheduling: if the scheduler event is triggered when server is running, the output will be as follow.
   Output:
   >the scheduler has started…
   >the scheduler has sharded 1000 rows in shard 1232.
   >the scheduler has stopped.

This are proposed output formats, meaning it may change during implementation.

9.2 Sample Output XML File

Dictionary data can be sharded or augmented in persisted XML file. The proposed format of an XML file is as below.



```xml
<? xml version="1.0" encoding="UTF-8" ?>
<database id="1256">
    <row id="122">
        <key type="number">122</key>
        <value type="object">
            <metaname type="string">first name </metaname>
            <timestamp type="date">10/10/2015:11:12:25</timestamp>
            <description type="string"> It describes about peroson's name.</description>
            <children type="List">NULL</children>
            <payload type="string"> Ronak</payload>
        </value>
    </row>
    <row id="127">
        <key type="number">127</key>
        <value type="object">
            <metaname type="string">Phone Number</metaname>
            <timestamp type="date">10/10/2015:11:20:25</timestamp>
            <description type="string"> It provides person's number.</description>
            <children type="List">NULL</children>
            <payload type="double"> 1234567890</payload>
        </value>
    </row>
</database>
```

10. References

1) https://en.wikipedia.org/wiki/NoSQL
2) https://en.wikipedia.org/wiki/XML_Encryption
3) http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html
4) http://ecs.syr.edu/faculty/fawcett/handouts/webpages/BlogNoSql.htm
5) https://en.wikipedia.org/wiki/Shard_(database_architecture)