CSE 681-Software Modeling and Analysis

Project #1

KEY / VALUE DATABASE

OPERATIONAL CONCEPT DOCUMENT (OCD)

VERSION: 1.0

Sai Krishna Bandaru
SUID: 464418513
Date: 09/16/2015
Instructor-Dr. Jim Fawcett

Table of Contents

Figures

1. Executive Summary

A Key/Value store, or Key/Value database, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash. Dictionaries contain a collection of objects, or records, which in turn have many different fields within them, each containing data. These records are stored and retrieved using a key that uniquely identifies the record, and is used to quickly find the data within the database.

The key motivation to implement Key/Value database is to store and analyse the large amount of data, extreme examples like, data collection and analyses from the Large Hadron Collider, the Sloan Sky Survey, analyses of Biological Genomes, measuring weather patterns, collecting data for global climate models, and analyzing client interactions in social networks. Conventional SQL databases are not well suited for these kinds of applications. While they have worked very well for many business applications and record keeping, they get overwhelmed by massive streams of data. Developers are turning to "noSQL" databases like MongoDB, couchDB, Redis, and Big Table to handle massive data collection and analyses.

The Key/Value database is used to store information as key-value pair, with each key-value pair also has some metadata associated with it. This database also supports the following features:

- Support addition and deletion of key/value pairs.
- Support editing of values including relationship and also metadata in with each element.
- Database can be restored or augmented from an existing XML file as well as write its contents out to an XML file.
- Can save database after some time interval or particular number of writes.
- Supports querying of stored data based on their key, relationships and the metadata.
- Support creation of new immutable database based on the results of the query.

The primary users of the tool will be instructor, teaching assistant and developer. Instructor, teaching assistant can use this tool to test and check if the given requirements are met, and also see if the critical issues are addressed. Developer can use this document to maintain and extend the Key/Value database module to implement different no-SQL applications, which require analysis on huge streams of real-time data.

We have thought of some critical issues and listed below that are associated with this entire system that need to be addressed are identified and solutions are discussed in detail in further sections.

- Retrieving of in information from the database.
- Implementing compound queries.
- Scheduling the save process, with positive time interval or on number of writes.
- Sharding of large database into small files.
- Augmenting the sharded files to retrieve data or to update database.

This Key/Value Database is also built in such a way that this serves as database component of Remote Key/Value Database where multiple users can access concurrently by requesting multiple read and writes at the same point of time.

2. Introduction

Key/Value database provides a mechanism for storage of data that is modelled in means other than the tabular relations used in relational databases, which helps us to retrieve data more efficiently for some applications. This is one example of “NoSQL”, whose approach is that simpler "horizontal" scaling to clusters of machines, which is a problem for relational databases, and finer control over availability, making some operations faster in NoSQL and others faster in relational databases. Though, NoSQL stores compromise consistency in favour of availability, partition tolerance, and speed. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages, lack of standardized interfaces, and huge previous investments in existing relational databases.

2.1. Obligation

The Key/Value noSQL model has goals that often prove to be difficult to implement with SQL databases. A noSQL database is designed to support one or more of the following:

- Handle very large collections of data, high throughput with data from streams.
- Support tree models for its data, with children relationship that helps to perform queries and retrieve information in constant time.
- Support heterogeneous collections of data, where value of the key-value pair can be any generic type.
- Support retrieval of data through compound queries.
- Persist data in files, where need to save the database state into a file and retrieve it when needed.
- When there is a huge data to handle, shred the database in to multiple files and also augment store data across files to perform data retrieval.

2.2. Organizing Principles

- The key/value data store uses C# data dictionary to store key/value relationships which internally uses hash map data structure.
- The main principle is to develop independent packages for specific functionality which will communicate only through Executive package.
- Shall use factory methods to construct a Value with supplied parameters and construct immutable database from query results.

2.3. Key architectural designs

The Key/Value Database is an example of NoSql database which has the advantage of retrieving data in constant time. For querying of data in constant time the data dictionary supported by C# language is very much helpful. To maintain a persistent Key/Value database we implement scheduler which triggers the save on in-memory database based on time and number of updates performed over in memory database which helps to have consistency along with persistent database. There is one more key design we can implement in when storing database value in auxiliary store like XML files in this example, which is sharding of

data store that we use for persistency. We divide entire information into small chunks of data which increases performance and availability of the database.

3. Use Cases

There are different types of users that interact with Key/Value Database package. The typical users can be Instructor, Teaching Assistant, Developer and other applications that are built over Key/Value database.

3.1. Instructor or Teaching Assistant

Instructor or Teaching Assistant will use the application to check the functionality, so that the all the requirements are met that are given as the part of project statement. The task is to use the package and test if all the critical issues are addressed without fail. More importantly see if the current Key/Value database is extended for future projects.

3.2. Developer

Essentially student developer is responsible for demonstrating that each of the requirements in this project statement. When this package is extended, developer may perform required maintenance activities to customize it to the application. The student developer may also reuse the package as a database component for implementing Project 4 and 5.

3.3. Other Applications

The applications which want to process huge volumes of data and query for results at constant time can use this package as their database, so that their purpose is solved. Examples are MongoDB, couchDB, Redis.

3.3.1. Project 4

In Project 4 which is Remote Key/Value Database where multiple users access Key/Value Database that is developed now should be access concurrently, so we need to design our database in such a way the concurrent access is supported. This has to support concurrent read and writes for multiple users that login remotely.

3.3.2. Event Logging

Storing information about event logging of a large user base application has a huge data stream flowing into database, in such cases it is really hard to store and retrieve information. Here Key/Value Database is built to handle such huge inflow of streams of data, and also the retrieval of information is in constant time. So Key/Value database can be used to implement user logging database.

4. Application Activities

A Key/Value Database is an example of NoSql, where data is stored as key-value pair. As like any database one has to retrieve information from the stored data. This Key/Value Database enable us to edit value of a key-value pair and also the metadata associated with it. We can retrieve information not only by providing key but also using child relationships. The database can be made persistent by implementing scheduler which can save information into XML storage files. Most of the applications of key-value database have to handle a large amount of data, so instead of writing into a single database file it can be sharded into multiple files.

4.1. Construct database: Parse the input file

User inputs file name that needs to be read in order to construct the start-up database so that one can demonstrate that requirements are met. Read the XML file and construct key-value database, where key is auto generated to corresponding value. Each key value pair has metadata associated with it. Metadata contains name, description of the value, date and time the value was written to the database and finite number of child relationships. Basically the database is stored in a hash map that can be accessed with its auto generated key.

4.2. Enable editing Values

As each instance can be represented as a key-value pair, where key is unique and cannot be changed. The value part can be altered up on the user request. Hence one can alter the metadata components such as adding and deleting relationships, editing text metadata and replacing an existing value instance with a new instance.

4.3. Query and Compound Queries

Typically in any database we retrieve data from it by posting a query to database, similarly in Key/Value database also we can get data which is the value object based on key. Key-Value database also returns a set of keys satisfying the query condition whose value objects are resided in the database. The resultant set of keys is also a structure resembling a different immutable database with the result keys. When it comes to a compound query the database splits the query into individual series of queries, the first query (inner most query) gets a set of keys on which we implement the next query until the execution of compound query is done. Also we need to take about the data which is not residing in in-memory database. In that case we retrieve information from persistent database. The whole process is depicted in the following activity flow diagram below.

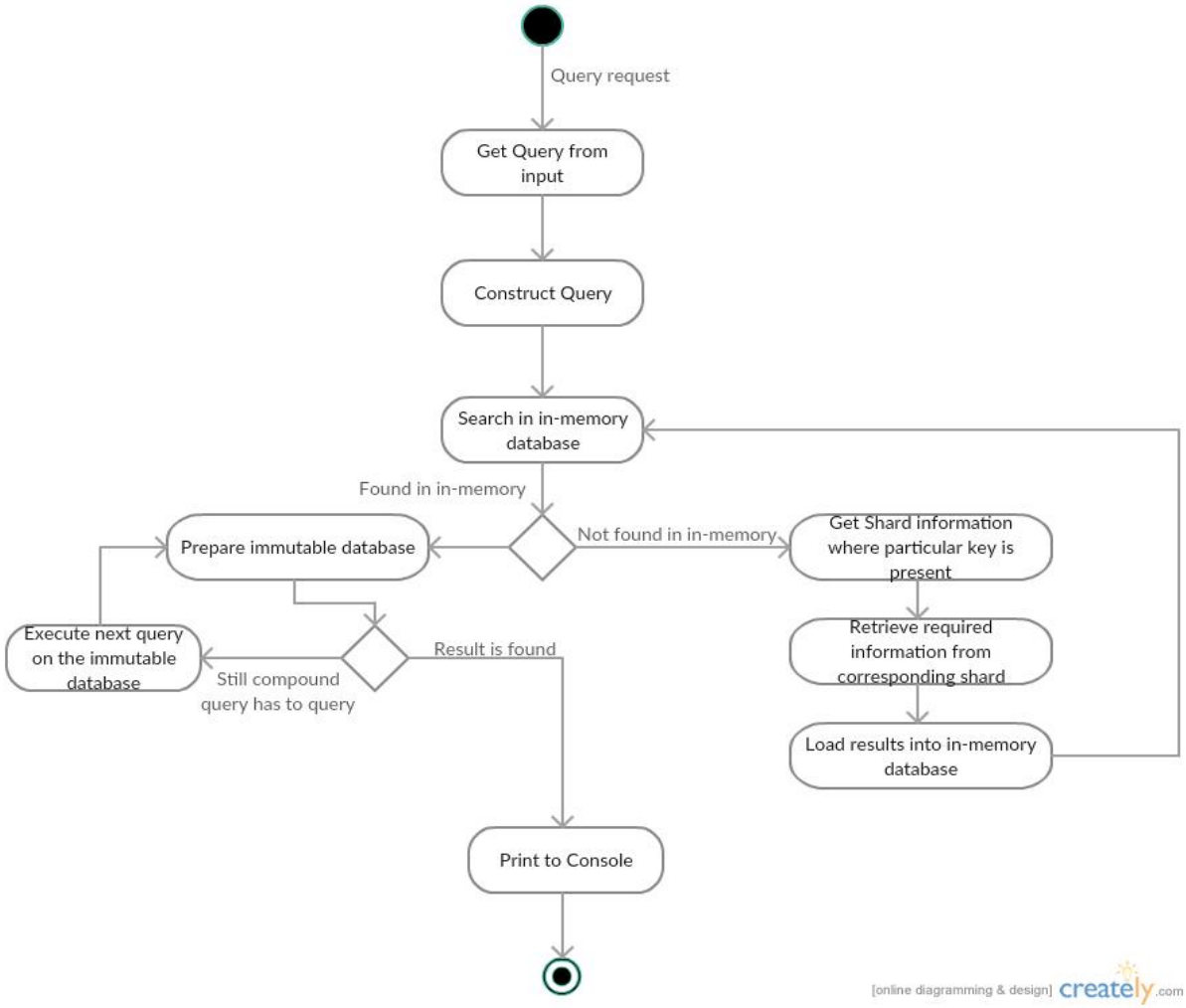

Figure 1-Query Request Activity Flow

4.4. Scheduling for persistent database

As each and every database, Key/Value database also stores the database into external resource other than in main memory structure. This is like a persistent backup from which we can generate the whole database for the next time the database to bring up. But unlike Relational databases which updates the external backup after each change, this Key-Value database writes the information to files after some positive interval of time or particularly after making certain number of modifications to the in memory structure. So here in this Key-Value database we store the data into an XML file and can retrieve the database from the file as in when needed. One more challenge is that if the data is huge then the main memory cannot accommodate space for the whole database to stay in memory, so in such cases we shard the database in to blocks of databases based on a parameter like time and store the information in different XML files.

4.5. Sharding of persistent database

Most of the applications of Key/Value database have to do with handling of huge volumes of data coming and residing in the database. It is practically impossible to have such huge

database in main memory, thus database system breaks the whole database into shards based on some parameter which can be time. So, when ever database moves into next time frame the scheduler starts writing the new time frame database into new XML file which is a new shard. While in the present Key/Value database we have fixed number of shards, when a key needs to be stored in persistent shards we find the hash value of the key in key-value pair and find mod value of hash value which gives the shard number into which we need to write key-value instance. Similarly, we find in which shard instance of key-value pair is residing the same hash value- mod with number of shards logic. The following activity flow diagram helps to understand the sharding logic better.

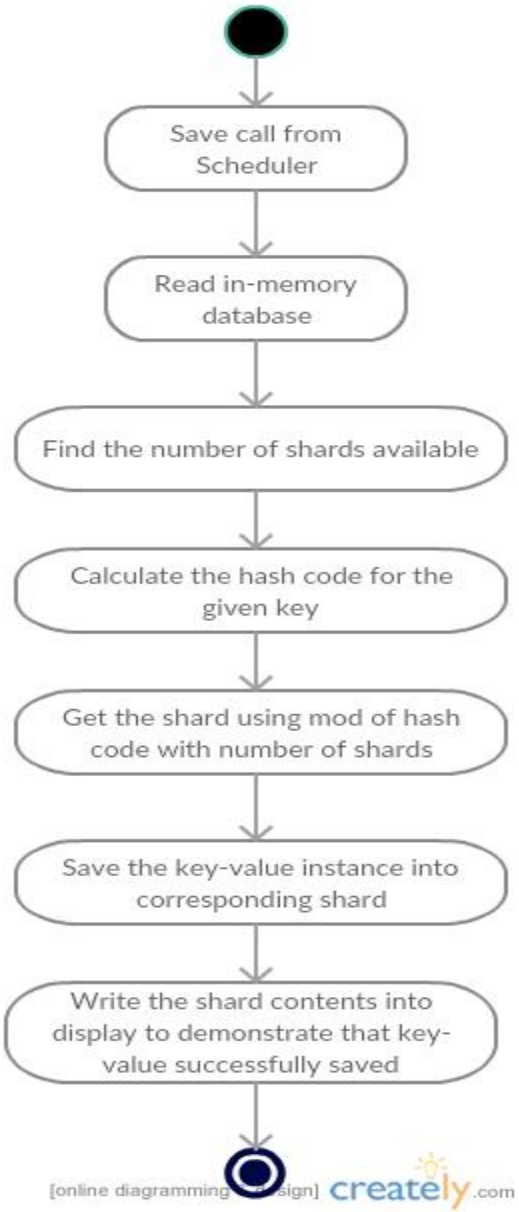

Figure 2- Sharding Implementation Activity Flow

4.6. Displaying results

The Display module provides support for displaying the Key-Value database results onto the standard output. These are typically output from a query, and are basically designed to show that all requirements given are met successfully.

4.7. Activity Diagram

The activity diagram below represents the sequential order of activities performed as part of the execution steps of this tool. The diagram also depicts the overall flow of control. The activity diagram gives reader an abstract idea of what activities the database will perform for implementing Key-Value database.

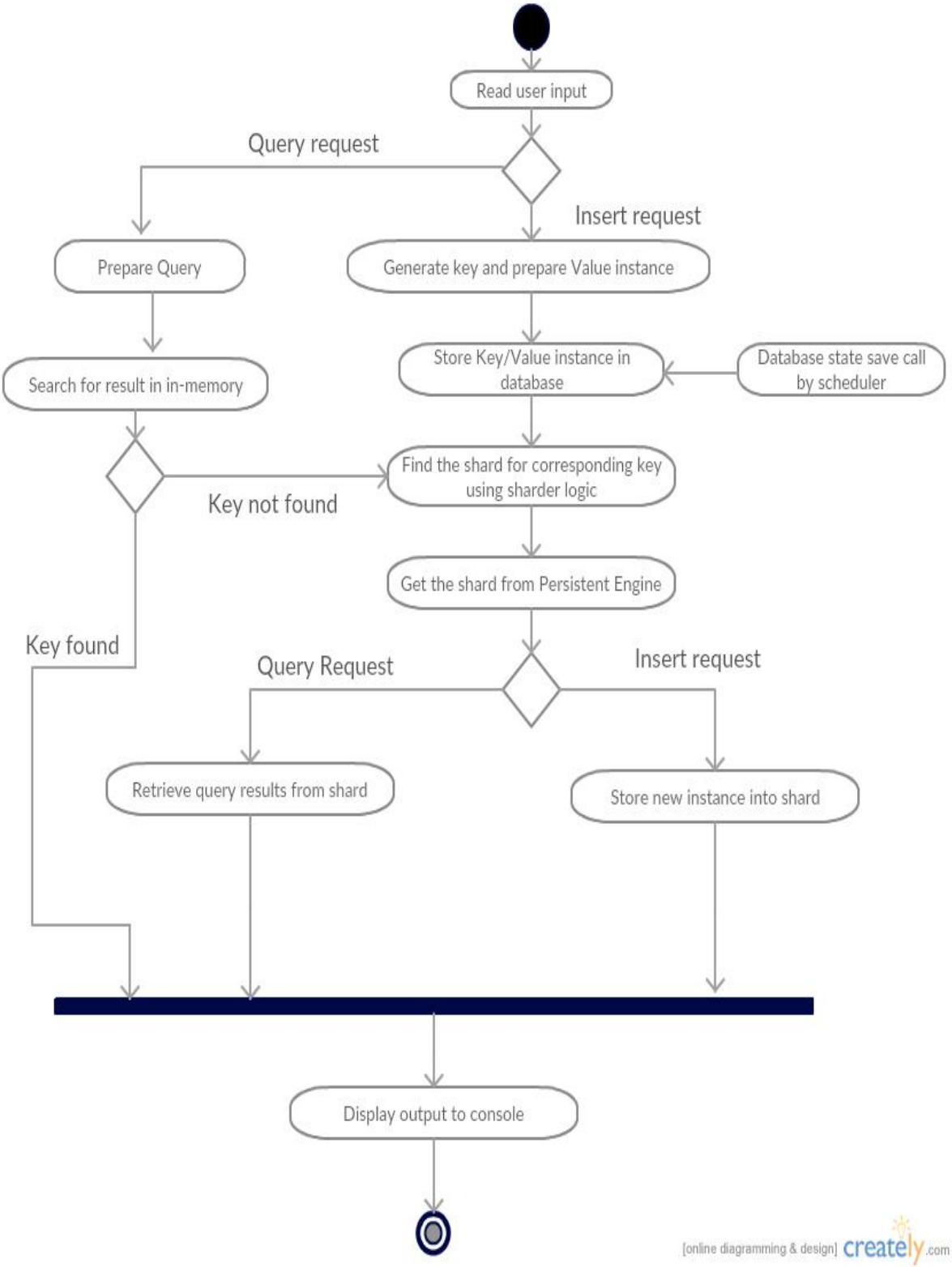

Figure 3- Key/Value Database Activity Flow

5. Packages or Modules

Based on the functionality, all the activities or tasks are split into different packages which have independent roles to play. The following are the packages that belong to Key-Value database.

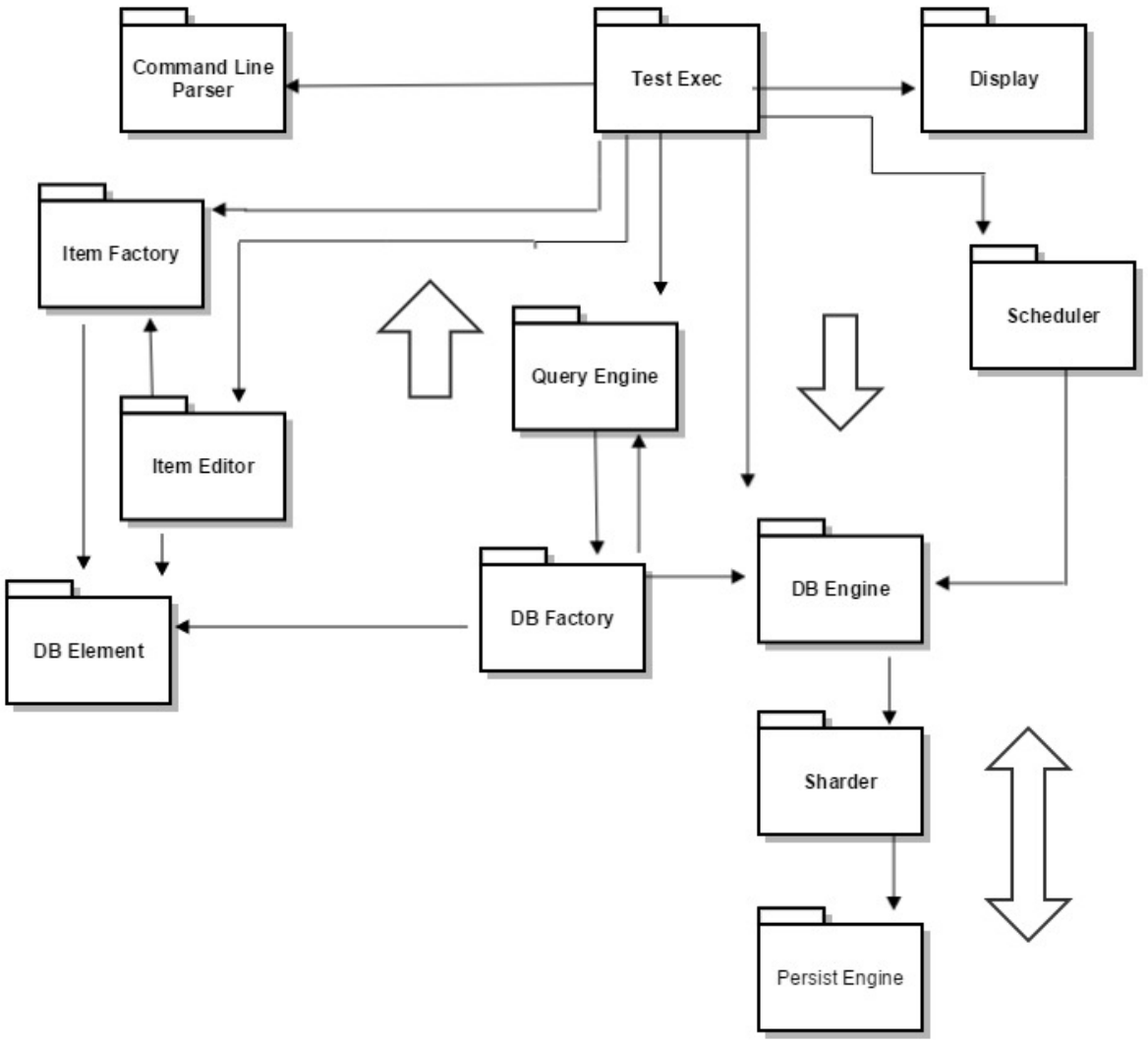

Figure 4-Key/Value Database Package Diagram

5.1. Test Executive

This is the package that acts as an interface between user and different modules of application that carries output information from one package which is fed as input for other package and also displays result on the shared output device with the help of display package. This package is designed to demonstrate that all the requirements are met. This package also parses the input XML file and is used to create the start up database.

5.2. Command line Parser

Command line parser is the package which takes command line XML file name and parses the file, which contains generic value. The Value is fed to Item Factory for generation of Key-Value Object dictionary object.

5.3. Item Factory

Initially a database is constructed from XML file that will be provided in command line. XML parser requests for the Value object corresponding to generic value type, Item factory constructs Value object with all the required metadata items and generates corresponding key and returns it to DB Engine.

5.4. DB Element

DB element package has the knowledge about the key-value instance structure and this helps to generate value instance and default key-value pair. This gets calls from Item Factory and DB Factory to create Key-Value instances.

5.5. Item Editor

This package's task is to update different components of key-value instance, like editing of values including the addition and/or deletion of relationships, editing text metadata, and replacing an existing value's instance with a new instance. But it inhibits any operation that involves in change of key.

5.6. Query Engine

The Query Engine is responsible to retrieve required information from the database. Here, Test Executive asks for particular information in form of a query, if it is a simple query the package works with DB Engine and gets all the keys which match the query. Then the Query Engine creates an immutable database with the resultant query keys and DB Factory. While if it is a compound query, Query Engine breaks the compound query into set of simple queries, then we generate an immutable database at the end of each simple query, on which the next simple query works on. The last query output database is the resultant of the compound query.

5.7. DB Factory

DB Factory is a package that acts as an interface that takes the resultant query keys as input and creates an immutable database for Query Engine. This immutable database contains key-value pairs where the value part is referenced to already existing in memory database.

5.8. DB Engine

DB Engine handles key functionality of Key Value database, this manages in memory database. When one wants to insert new key-value record into database, the instance is created at Item Factory and is sent to DB Engine. This adds key-value instance to database based on its key, this also helps retrieve information from database by accepting different queries which in turn returns set of keys that satisfy the condition. DB Engine also takes

persistent database into consideration while retrieving information for different queries. The persistent database is in turn saved from in-memory database of DB Engine upon request from scheduler.

5.9. Scheduler

The Scheduler package task is to trigger a save on in-memory database (DB Engine), so as to have a persistent copy of the whole database. This scheduler may trigger save call on two different conditions, when is elapses the time period set to schedule save call and when it meets maximum number of writes between two save calls. And also, when the current in-memory copy of database is replaced it save the state of database by writing it to persistent database. The save call is sent to DB Engine to backup a persistent copy for the database. The time interval between two save calls and maximum number of writes that can happen over in-memory database can be defined by user through Test Executive package.

5.10. Sharder

Sharder takes care of persistent database by dividing and maintaining multiple shards which are typically XML files. When the Scheduler calls for save over in-memory database, Sharder decides which XML file to write into the instances of in-memory database. This Sharder has logic to decide on, shard by generating hash value for particular key and dividing the hash value by number of shards where the reminder gives shard to write into, and while implementing data retrieval this sharder finds the XML file which has the required key in the similar way.

5.11. Persist Engine

Persist Engine is responsible for storing database in to XML file. This works with Sharder to save Key/Value database into different XML files. When sharder requests to store in particular XML file it operates like, finds the corresponding file and stores in it. When the sharder requests to load XML file, the file is given back to Sharder to load the database.

6. Critical Issues

6.1. How to implementing Sharding?

Sharding is something completely application specific, there are some Sharder packages which Shard on time, but here in this Key/Value Database we have fixed number of shards to which we write our database. Selecting a shard is based on the key; generate an auxiliary hash value that is divided by number of shards for which reminder gives XML file into which Key/Value instance has to reside (mod value of hash value to number of XML files). Now persistent database has been created, while we perform a query over database and if required key is doesn't exists in in-memory chunk of database. Then Sharder gets a request to look up for that particular key, same as how we performed save we get hash value of key whose mod value give the shard (XML file) details.

6.2. Construction of immutable database

The result of a query is an immutable database, which also a Key/Value database structure but the challenge for construction such database is, we may have child relationships in metadata that doesn't exists in our immutable database which results in a dangling reference. The solution for this problem is to have list of all keys in corresponding resultant immutable database, so each time we try to traverse through child relationships we ignore all the keys that doesn't exists in the key list of resultant database.

6.3. Issue with deleting of Key-Value instance

As the database supports deletion of key-value instances, once we delete an instance the child relationships which holds corresponding key that is deleted will point to an instance that doesn't exists in the database. So in order to eliminate this invalid child references, there has to be a check for reliability of database so as to remove all the child references that contain corresponding key. Hence we update all the child relationships after deleting a key-value instance.

6.4. Retrieving information for Compound Query

A compound query is like a nested query, in which we generate an immutable Key/Value database after executing each query. The problem with compound query is, if it is a series of queries where do we start with. So in case of compound query system has to do an additional work of preparing query, which means we parse the user's input and store nested query in a container where inner most query is stored at last and we start from inner most query gives an immutable database on which the query that contains inner most query works on. In this way compound queries can be implemented.

6.5. To have Data Reliable and Fault Tolerant

For every database availability is a key issue, which means that database has to be up and processing user requests all the time even when is has some issues with some part of the data. For this we need to implement multiple copies of shard (XML files), here each shard

has a parity check if the shard doesn't pass the parity check which means that corresponding shard some corrupt information so we load reserve copy of it. In order to implement complete fault tolerant system we have multiple copies of each shard which has backup data of the same shard take at different time in past. So we can track back for a consistent shard and retrieve back consistent information.

6.6. Maintaining eventual consistency

In Key/Value Database as we have persistent database which stores whole database into some shards of XML files, each time we request information we get an instance that resembles a key-value pair from persistent database when we change the value object of that particular pair there is some delay for replicating the change in to persistent store. This delay depends on the time scheduler triggers a save call or the number of times u perform a write operation in in-memory database. When another user requests for same instance before writing the changes into persistent database there is some data inconsistency. So in order to eliminate this problem of data inconsistency, we can set scheduler for very less save call time or maintaining less number of writes for each save. There can be one more solution is to provide commit feature for the user, which overrides scheduler save call; in this way we can assure important data is saved promptly.

6.7. Authentication

As Key/Value Database has the ability to edit values and also add or delete key-value pair instances, there are risks of misusing access of database which might be updating of a value instance data without proper privileges and addition or deletion of a key-value instance which make the database an un-reliable one. The solution for this problem is to have an authentication system which tracks all the users that are logged in by identifying each user by their credentials and tracking of their activities by storing the request of changes they make.

6.8. When input is inconsistent

The Key/Vale database interacts with user in different ways, and there are chances of getting inconsistent data from the user, be it the input XML file to construct startup database or any request like editing value or querying for values. So the Key/Value database has to process each input from user and write out appropriate message to user, in case of any inconsistent information is found. Example, when the input XML file contains any inconsistent data the Key/Value database has to print the details where it found inconsistency so that user can correct it.

6.9. Getting User familiar with the Key/Value database

As Key/Value database is a new type of database which doesn't use general SQL queries for its operations, user has to be familiar with all the operations that he can perform over database. So one of the solutions is to write into console helper information which briefly tells the user of operations he can work with.

7. References

[1] Key/Value database Requirements document, provided by Dr. Fawcett
http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project2-F2015.htm

[2] Key-Value Database helper code, provided by Dr. Fawcett
http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project2HelpF15

[3] NoSql details from Blog, provided by Dr. Fawcett
http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Parser/

8. Appendix

The objects of this DBElement class are the values in our key/value database. The payload field contains the actual value; the child relationships are stored in the List data structure.

```
public class DBElement<Key, Data>
{
   public string name { get; set; }
   public string descr { get; set; }
   public DateTime timeStamp { get; set; }
   public List<Key> children { get; set; }
   public Data payload { get; set; }
}
```

DB Engine class contains the .net dictionary object, functions are exposed which performs the specific operations over the data dictionary. Few of the notable functions are implemented in the below code snippet.

```
public class DBEngine<Key, Value>
  {
    // - Data Dictionary which holds the key value pairs
    private Dictionary<Key, Value> dbStore;

    public DBEngine()
    {
      dbStore = new Dictionary<Key, Value>();
    }

   // - DB Engine class which contains the .net dictionary object
    public bool insert(Key key, Value val)
    {
      if (dbStore.Keys.Contains(key))
        return false;
      dbStore[key] = val;
      return true;
    }
   // - DB Engine class which gets the value from the .net dictionary object
    public bool getValue(Key key, out Value val)
    {
      if(dbStore.Keys.Contains(key))
      {
        val = dbStore[key];
        return true;
      }
      val = default(Value);
      return false;
    }
    // - DB Engine class which contains the .net dictionary object
       public IEnumerable<Key> Keys()
       {
             return dbStore.Keys;
       }


       // - DB Engine class which deletes the key from the .net dictionary object
       public bool deleteKey(Key key)
       {
              if (dbStore.Keys.Contains(key))
                 {
```

```
                dbStore.Remove(key);
            }
        return false;
    }
}

// - DB Engine class which updates the value from the .net dictionary
object
public bool updateValue(Key key, Value val)
{
            if (dbStore.Keys.Contains(key))
            {
                dbStore[key] = val;
                return true;
            }
            return false;
}
```

Example XML Shard file

<?xml version="1.0" encoding="UTF-8"?>
<sharddatabase>
<data-pair>
<key>KEY_464418513</key>
<value>
<metadata> <name> Sai Krishna BAndaru</name>
<description> Student</description>
<timestamp> 15-SEPTEMBER-2016 23:50:06</timestamp>
<children> <child> KEY_999356566 </child>
<child> KEY_5465865948 </child>
</children>
</metadata>
<payload>Hello</payload>
</value>
</data-pair>
</sharddatabase>