CHAPTER 1

INTRODUCTION

We has seen the most technological environment in the last fifty years, with the advancement in every field have made the human life easier and more comfortable. Airlines industry has renowned into one of the most revolutionary and interesting industries of today’s world. Airline industry has completed the dream of humans to flying high in the sky. Today, millions of people fly every day. This has strengthened not only the economies of places but also connected people and cultures. The betterment of technology has led to big success in the area of flight ticket booking over the years.

1.1 Objective

The aim of our project is to study the graph database as well as to design and develop a software using graph database which would automate the major airline operation like providing the facilities for the reservation of the online air tickets or any other operations through an effective and yet simple GUI for a normal passenger intending to travel through airways.

1.2 Needs

The need of this system was realized since the beginning stages of the airline industry when information such as routes, aircrafts, schedules and fares about flights was published by airlines in large books. Travel agents had the difficult task of looking into separate books for reservations that involve multiple airlines. It was a dream to get a real time picture of available seats because airlines share information at day end only. Prior to this, manual systems required centralized reservation centres, groups of people in a room with the physical cards that represented inventory, in this case, seats on airplanes (Desmond, 2000). In this regard, an improved airline reservation system is implemented to assist Overland Airways with a variety of airline administration tasks and service passenger needs from the time of initial reservation through completion of the flight. An improved airline reservation system will be developed with facilities that will enable passengers of Overland Airways book available flights, select seats for their flights, print boarding passes, access aircraft maintenance report, check-in online for their flight with the hope that it will help the airline by streamlining their process of reservation without human interaction so as to enable them perform well in the highly competitive market place.

1.3 Basic Concept

In our project we are using a graph database because of its various advantages. In computing, a graph database is a database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data. A key concept of the system is the graph (or edge or relationship), which directly relates data items in the store. The relationships allow data in the store to be linked together directly, and in many cases retrieved with one operation. Graph databases, by design, allow simple and fast retrieval of complex hierarchical structures that are difficult to model in relational systems. Graph databases are similar to 1970s network model databases in that both represent general graphs, but network-model databases operate at a lower level of abstraction and lack easy traversal over a chain of edges. Compared with relational databases, graph databases are often faster for associative data sets and map more directly to the structure of object-oriented applications. They can scale more naturally to large data sets as they do not typically need costly join operations (here costly means when executed on databases with non-optimal designs at the logical and physical levels). As they depend less on a rigid schema, they are marketed as more suitable to manage ad hoc and changing data with evolving schemas.

1.4 Applications

1. Fraud detection:

Real-time analysis of data relationships is essential to uncovering fraud rings and other sophisticated scams before fraudsters and criminals cause lasting damage.

2. Network and IT operations:

Graph databases are inherently more suitable than RDBMS for making sense of complex interdependencies central to managing networks and IT infrastructure.

3. Master data management:

   Organize and manage your master data with the flexible and schema-free graph database model in order to get real-time insights and a 360° view of your customers.

4. Social networking:

   Easily leverage social connections or infer relationships based on activity when you use a graph database to power your social network application.

5. Real time recommendation engine:

   Graph-powered recommendation engines help companies personalize products, content and services by leveraging a multitude of connections in real time.

6. Identity and access management:

   Quickly and effectively track users, assets, relationships and authorizations when you use a graph database for identity and access management

CHAPTER 2

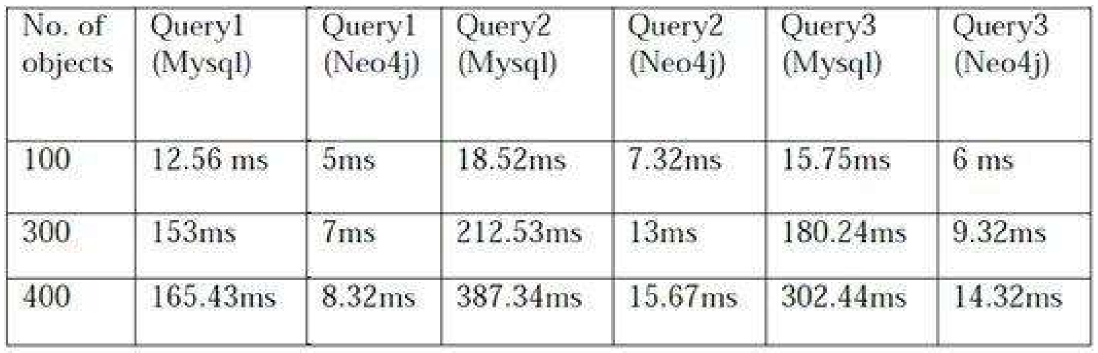
LITERATURE SURVEY

In this paper (Relational database and graph database: a comparative analysis, authors Surajit Medhi, Hemanta K. Baruah) we found that,

In recent years, the importance of storing and analyzing data in the form of graph has been increasing. Graph database is now used in social networks, recommendation systems, biological network, web graph etc. and these graphs are highly interconnected. In relational database, data are stored in tabular form. For less connected or static data, relational database is perfect, but for highly connected data such as the network of hyperlinks in the World Wide Web, it is complex for representing in relational database. A similar kind of problem arises when we want to model the social networks like Facebook, Twitter etc. It has been accepted that the web data can also be represented and visualized using the graph database.

The relational database and graph database are useful to represent data. However, if we compare with reference to storing and retrieving highly connected data, graph databases appear to give better results. This means that we can more easily design and retrieve the results of queries if we use graph databases instead of using relational databases. When we want to add a new relationship to graph database, we do not need to restructure the database again. As we can see from our experiment, we get better results for executing the predefined queries in graph databases than in relational database with respect to time. The retrieval times for quires give us a conclusion that graph database is suitable to use in commercial purposes such as developing social network, stock market, and recommendation engine and network management.

Execution Time for Different Queries for Different Objects



No. of objects	Query1 (Mysql)	Query1 (Neo4j)	Query2 (Mysql)	Query2 (Neo4j)	Query3 (Mysql)	Query3 (Neo4j)
100	12.56 ms	5ms	18.52ms	7.32ms	15.75ms	6 ms
300	153ms	7ms	212.53ms	13ms	180.24ms	9.32ms
400	165.43ms	8.32ms	387.34ms	15.67ms	302.44ms	14.32ms

In this paper (A Comparison of a Graph Database and a Relational Database, authors Chad Vicknair, Michael Macias, Zhendong Zhao) we found that,

Relational databases have been around for many decades and are the database technology of choice for most traditional data-intensive storage and retrieval applications. Retrievals are usually accomplished using SQL, a declarative query language. Relational database systems are generally efficient unless the data contains many relationships requiring joins of large tables. Recently there has been much interest in data stores that do not use SQL exclusively, the socalled NoSQL movement. Examples are Google's BigTable and Facebook's Cassandra. This paper reports on a comparison of one such NoSQL graph database called Neo4j with a common relational database system, MySQL, for use as the underlying technology in the development of a software system to record and query data provenance information. This paper is a comparison of the relative usefulness of the relational database MySQL and the graph database Neo4j to store graph data. A directed acyclic graph (DAG) is a common data structure to store data provenance information relationships. The goal of this study was to determine whether a traditional relational database system like MySQL, or a graph database, such as Neo4j, would be more effective as the underlying technology for the development of a data provenance system. A graph is one of the fundamental data abstractions in computer science.

Both systems performed acceptably on the objective benchmark tests. In general, the graph database did better at the structural type queries than the relational database. In full-text character searches, the graph databases performed significantly better than the relational database. The fact that the indexing mechanism used in the graph database was based on strings made the numeric queries less efficient. While a query on non-integer numeric data, such as doubles, was not included in the benchmark tests, the result would have likely been even worse for the graph database. The documentation on the Lucene site suggests padding numerics with spaces or zeroes, as appropriate. While this works, it is too restrictive for the purpose of storing user-supplied parameters and values in the payload. In scientific data, it is not reasonable to set a particular precision level since some parameters might require two decimal places and others may require ten or more. Speed issues related to index searching in Neo4j for numbers are related to the Lucene. This problem is known, and at the time of this writing, numerical indexing is being developed for Lucene. The other factor that must play a key role in choosing a database system for a data provenance system is security. The lack of support in Neo4j is an issue. Further investigation into Neo4j might yield workarounds to the search issues documented here. There are add-on components that allow Neo4j to be accessed as a Resource

Description Framework (RDF) store, and potentially queried with SPARQL, an RDF query language, which has a W3C recommendation. That functionality is not well documented and was not tested in this study. Overall, for the data provenance project, it seems premature to use the graph database for a production environment where many queries will be on parameters stored in a semi structured way even in the face of Neo4j's much better string searches. In addition, the need for securing user data is imperative. And lack of support in Neo4j is a significant drawback.

In this paper (Graph Databases, authors Adrian Silvescu, Doina Caragea, Anna Atramentov) we have found that,

Current representation and storage systems are not very flexible in dealing with big changes and also they are not concerned with the ability of performing complex data manipulations. On the other hand, data manipulation systems cannot easily work with structural or relational data, but just with flat data representations. We want to bridge the gap between the two, by introducing a new type of database structure, called Graph Databases (GDB), based on a natural graph representation.

Our Graph Databases are able to represent as graphs any kind of information, naturally accommodate changes in data, and they also make easier for Machine Learning methods to use the stored information. We are mainly concerned with the design and implementation of GDB in this paper. Thus, we define the Data Definition Language (DDL) that contains extensional definitions as well as intentional definitions, and Data Manipulation Language (DML) that we use to pose queries. Then we show conceptually how to do updates, how to accommodate changes, and how to define new concepts or ask higher order queries that are not easily answer by SQL language. We also show how to transform a relational database into a GDB. Although, we show how all these things can be done using our graph databases, we do not implement all of them. This is a very laborious project that goes much beyond our class project goals. We do implement the graph databases, show how we can ask queries on them, and also how to transform relational databases into graph databases in order to be able to reuse relational data already existent.

Our interest in designing a new kind of database grew as a consequence of the problems arising in autonomous dynamic environments that are very common in a lot of scientific domains, and also business domains, Internet etc. Previous existing database systems don’t take too much into account these problems, and they are not able to deal very well with changes. Also they do not offer a friendly machine learning environment that will allow the discovery of patterns in data, new concepts, explanations for some behaviour etc. Thus, our goal was to design such a system that would be very useful for machine learning applications, and in the same time be able to efficiently and uniformly store any general information, that could also change in time. Having this goal in mind, Current representation and storage systems are not very flexible in dealing with big changes and also they are not concerned with the ability of performing complex data manipulations. On the other hand, data manipulation systems cannot easily work with structural or relational data, but just with flat data representations. We want to bridge the gap between the two, by introducing a new type of database structure, called Graph Databases (GDB), based on a natural graph representation. Our Graph Databases are able to represent as graphs any kind of information, naturally accommodate changes in data, we design Graph Databases whose main building blocks are graphs representing uniformly data, knowledge, models and queries. The preference for graph structures came from the recent tendency of using graphical representations for data with the purpose of taking into account natural structure that data usually presents, as opposed to working with flat representations. The system that we obtained is closed in spirit to OODB, but not the same. As opposed to GDB, we do not assume any encapsulation, but we pay a price by having more Ids, which is not too bad for our purpose of easily manipulating data in a natural declarative way. GDB is also closed to Data log, the main difference being in our option of using IDs (links) instead of foreign keys. This is also main difference from ORDBMs and somewhat from XML. However we do not lose anything by not using foreign keys, and we show before how we can simulate them using IDs. Obviously, there are advantages and disadvantages in using GDB as a representation/storage/reasoning/learning system, but for our purposes the disadvantages (more memory storage) are not essential and we are willing to pay this price toward the flexibility and unified view that we get. This is very important for the domains where we need to apply learning methods to extract information from data. We are aware that a lot of the existent data resides in relational databases, and in order to be able to reuse this data, an important part in our project is focused on the translation from relational databases format to our GDB input format.

CHAPTER 3

PROBLEM STATEMENT

Airline Reservation System using Graph Database

3.1 What is to be Developed

Airline reservation systems were first introduced in the late 1950s as relatively simple standalone systems to control flight inventory, maintain flight schedules, seat assignments and aircraft loading. The modern airline reservation system is comprehensive suite of products to provide a system that assists with a variety of airline management tasks and service customer needs from the time of initial reservation through completion of the flight. One of the most common modes of travel is traveling by air. Customers who wish to travel by air nowadays have a wide variety of airlines and a range of timings to choose from. Nowadays competition is so fierce between airlines that there are lot of discounts and a lot of luxuries given to customers that will give an edge to that particular airline. The World Wide Web has become tremendously popular over the last four years, and currently most of the airlines have made provision for online reservation of their flights. The Internet has become a major resource for people looking for making reservations online without the hassle of meeting travel agents. My Project intends to serve these purposes. It intends to check all the available airline databases and return a string of results, which can help them in their travel plans. The objective of this project is to create an airline reservation system where a traveler can request all flight information as per their journey dates. They can get information regarding time, cost, etc all at the same time and place. When the customer calls the Counter Assistant for his/her travel needs, the counter assistant will enter the customer's details (flight requirements) in the system. The system displays all the available airlines, schedules and prices. This system would help the airline to better serve its customers by catering to their needs. The site would use a Database to hold this information as well as the latest pricing and availability information for the airlines.

The proposed system is better and more efficient than existing System by keeping in mind all the drawbacks of the present system to provide a permanent to them. The primary aim of the new system is to speed up the transactions. User friendliness is another peculiarity of the proposed system. Messages are displayed in message boxes to make the system user friendly. The main Advantage of the proposed system is the reduction in labor as it will be possible so search the details of various 10 places. Every record is checked for completeness and accuracy and then it is entered into the database. The comments and valid messages are provided to get away redundant data. Another important feature of the proposed system is the data security provided by the system. The main aim of the proposed system are:

- Complex functions are done automatically
- Processing time can be minimized
- Simple and easy to manage
- Chances of errors reduced
- Faster and more accurate than the existing system
- Easy for handling reports

The proposed system is complete software for Airline Reservation System, Which is more efficient, reliable, faster and accurate for processing.

CHAPTER 4

CONSTRAINS AND REQUIREMENTS

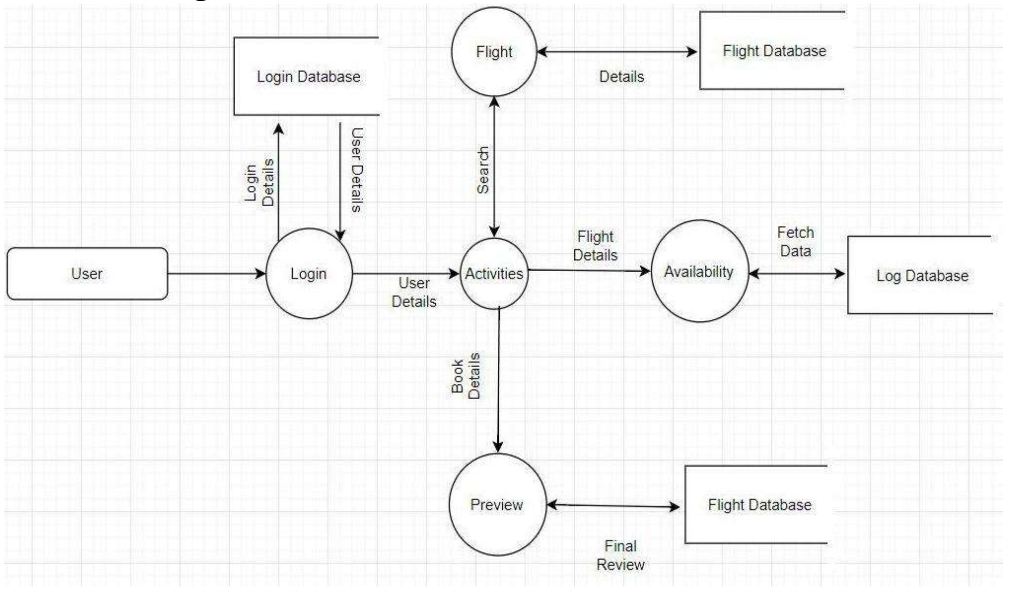
4.1 Software and hardware requirements

Processor	Intel Core i3 Processor.
Operation System	Ubuntu
RAM	4 GB
Hard Disk	5 GB
Database Engine	Titan
Backend	Cassandra

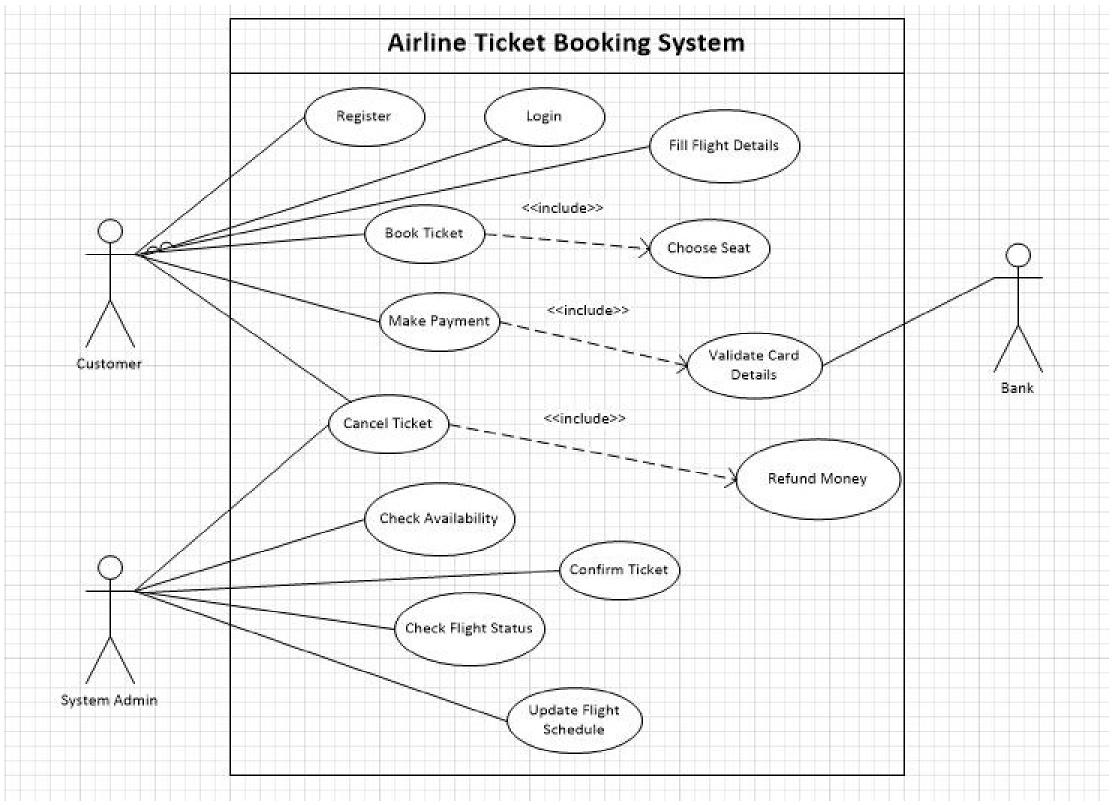
CHAPTER 5

ARCHITECTURE AND DESIGN

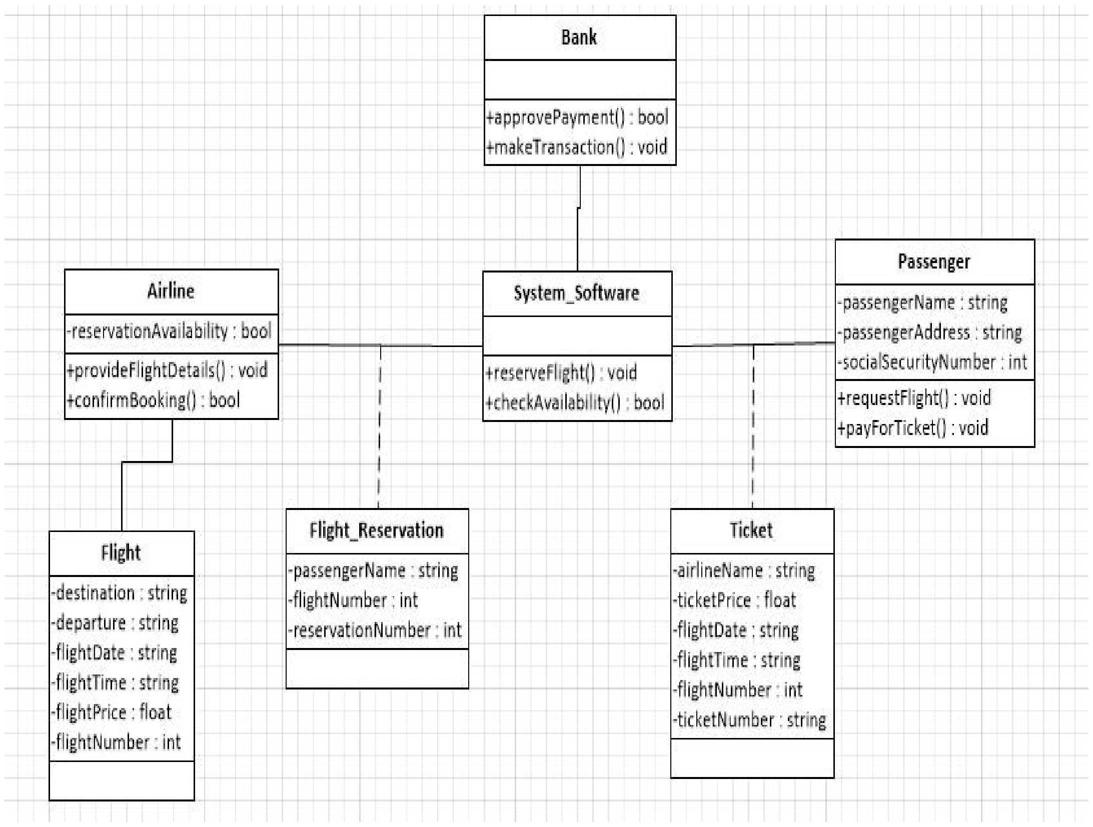
5.1 Data Flow Diagram



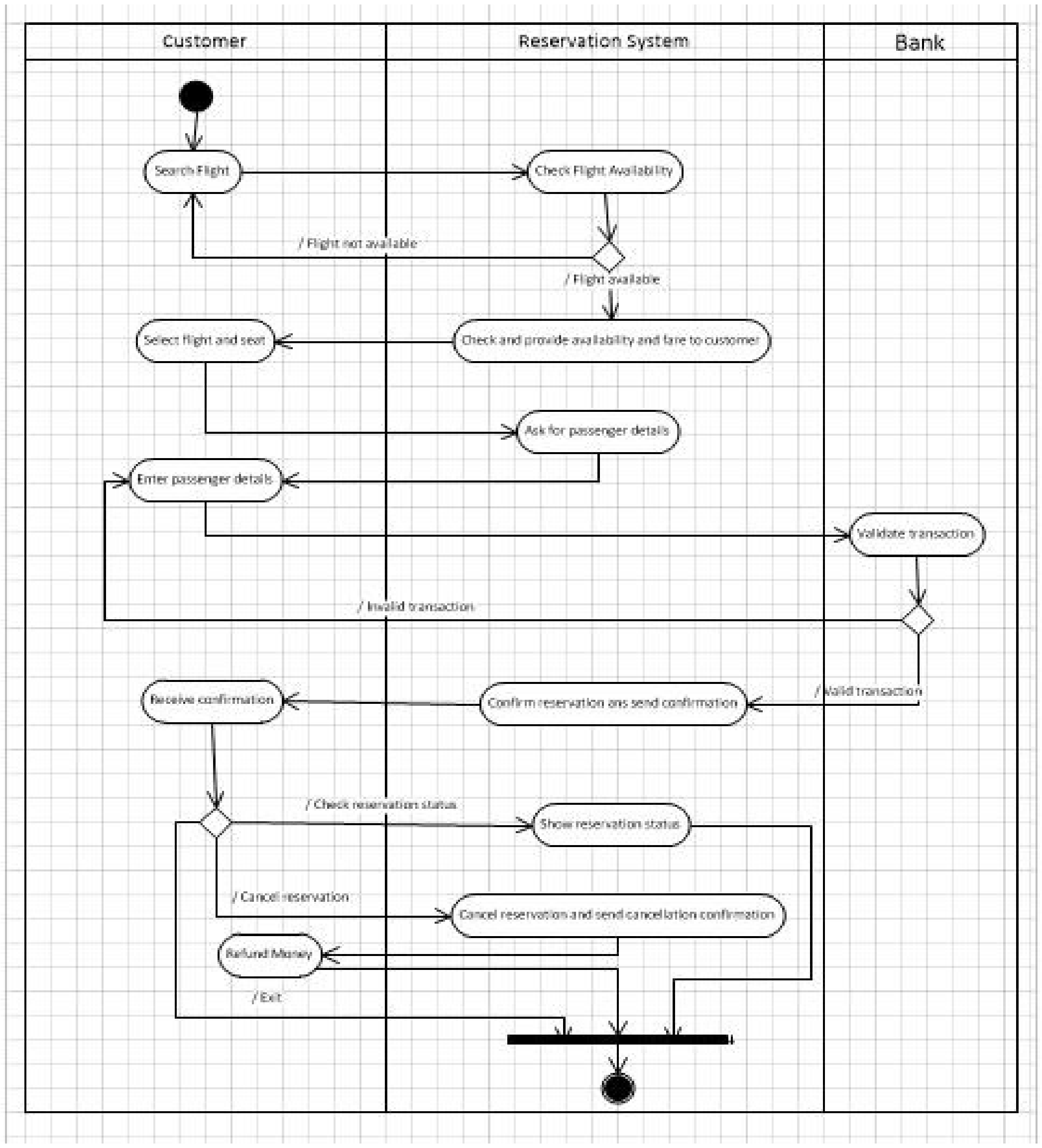
5.2 Use Case Diagram



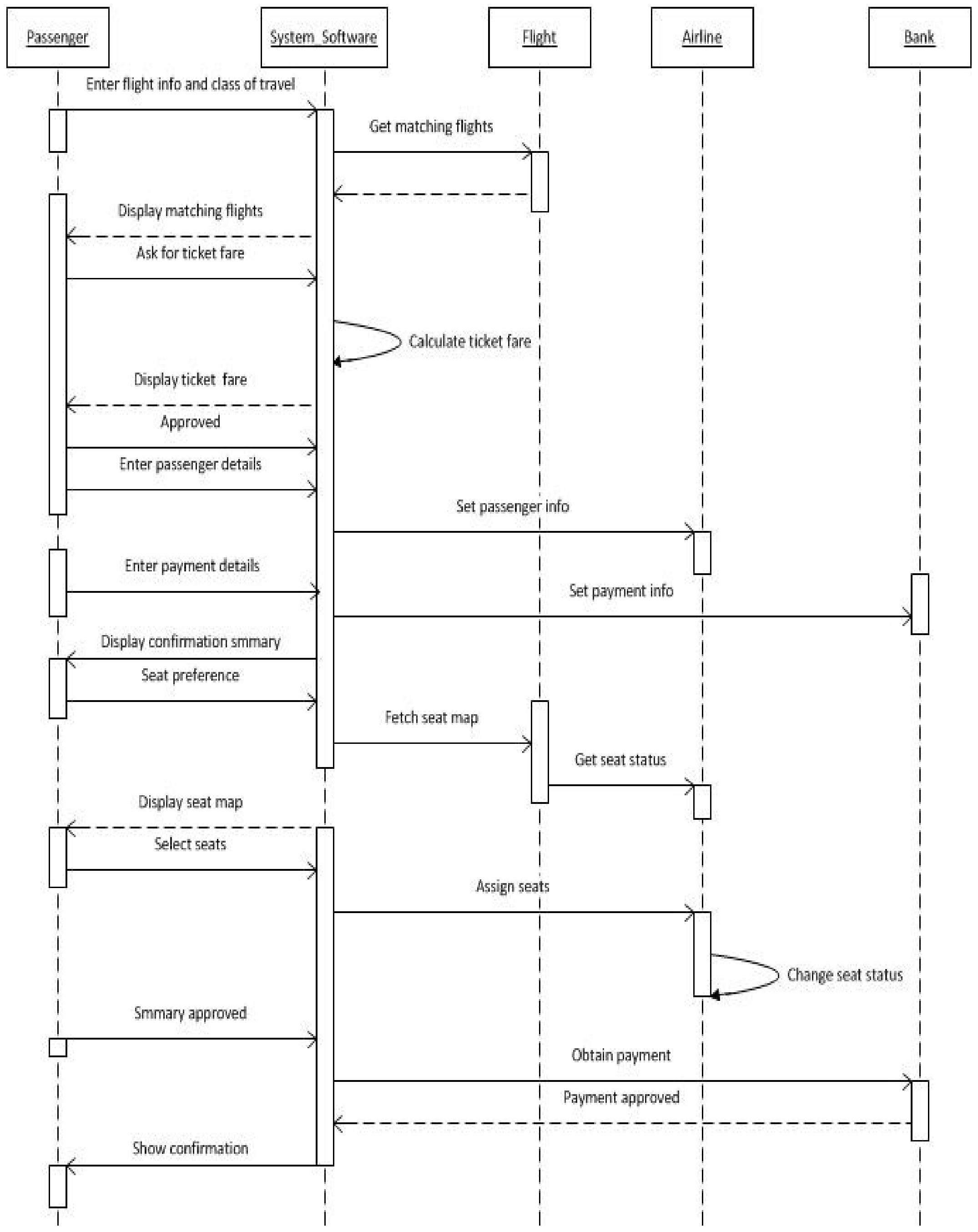
5.3 Class Diagram



5.4 Activity Diagram



5.5 Sequence Diagram



CHAPTER 6

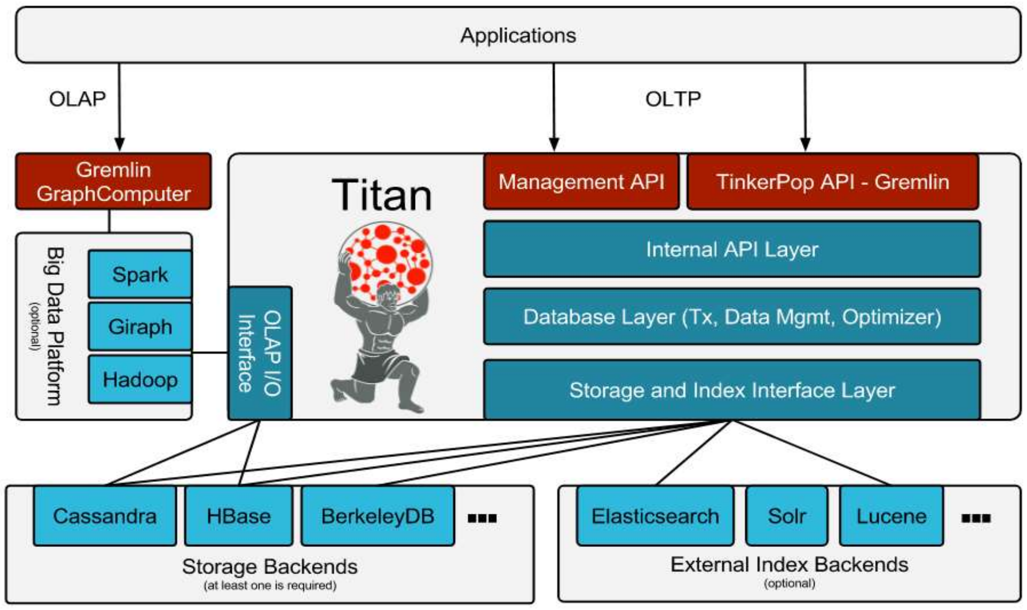
IMPLEMENTATION

6.1 Implementation procedure of the idea

1. Start Cassandra service.
2. Start elasticsearch service.
3. Enter Flights to be search.
4. Available flights will be shown.

6.2 Technologies Used

Titan:



Titan is a graph database engine. Titan itself is focused on compact graph serialization, rich graph data modeling, and efficient query execution. In addition, Titan utilizes Hadoop for graph analytics and batch graph processing. Titan implements robust, modular interfaces for data persistence, data indexing, and client access. Titan's modular architecture allows it to interoperate with a wide range of storage, index, and client technologies; it also eases the process of extending Titan to support new ones.

Cassandra :

The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data. Cassandra's support for replicating across multiple datacenters is best-in-class, providing lower latency for your users and the peace of mind of knowing that you can survive regional outages. The largest known Cassandra cluster has over 300 TB of data in over 400 machines.

Elasticsearch :

The elasticsearch is a flexible and powerful open source, distributed, real-time search and analytics engine. Architected from the ground up for use in distributed environments where reliability and scalability are must haves, Elasticsearch gives you the ability to move easily beyond simple full-text search.

Titan supports Elasticsearch as an index backend. Here are some of the Elasticsearch features supported by Titan:

- Full-Text: Supports all Text predicates to search for text properties that matches a given word, prefix or regular expression.
- Geo: Supports the Geo.WITHIN condition to search for points that fall within a given circle. Only supports points for indexing and circles for querying.
- Numeric Range: Supports all numeric comparisons in Compare.
- Flexible Configuration: Supports embedded or remote operation, custom transport and discovery, and open-ended settings customization.
- TTL: Supports automatically expiring indexed elements.
- Collections: Supports indexing SET and LIST cardinality properties.
- Temporal: Nanosecond granularity temporal indexing.

Gremlin:

Gremlin is a Titan's query language used to retrieve data from and modify data in the graph. Gremlin is a path oriented language which succinctly expresses complex graph traversals and mutation operations. Gremlin is a functional language whereby traversal operators are chained together to form path-like expression. Gremlin is developed independently from titan and supported by most graph databases.

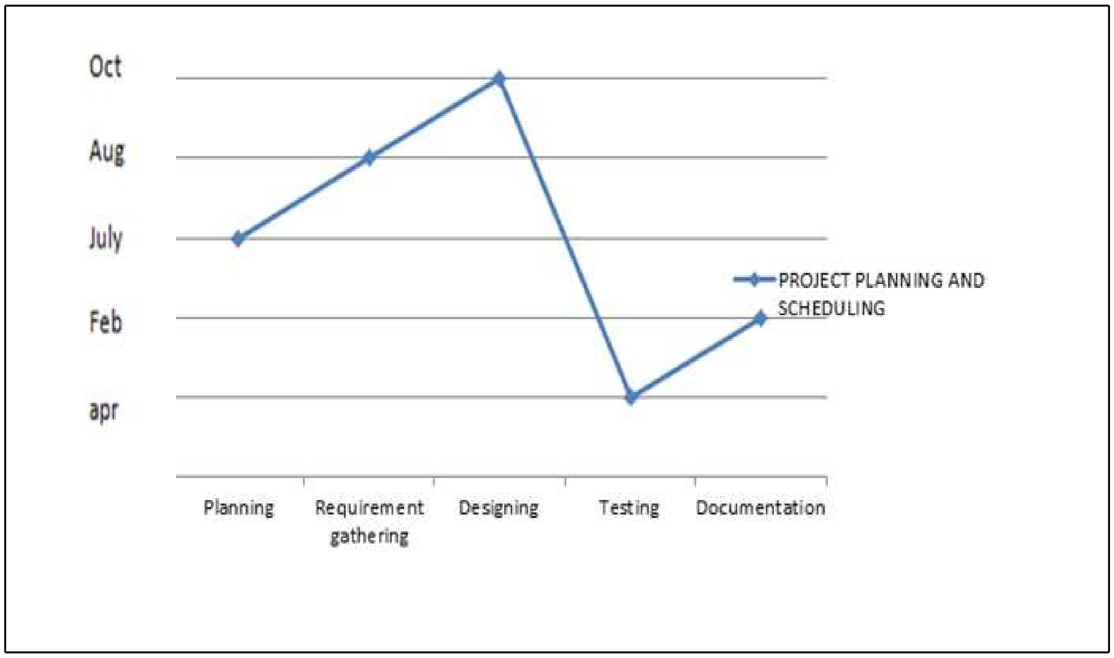
Http :

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext. HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the *client* and an application running on a computer hosting a website may be the *server*. The client submits an HTTP *request* message to the server. The server, which on behalf of the

client, returns a *response* message to the client. The response contains completion provides *resources* such as HTML files and other content, or performs other functions status information about the request and may also contain requested content in its message body.

CHAPTER 7

PROJECT PLANNING AND SCHEDULING



CHAPTER 8

TESTING

8.1 Unit Testing

Unit testing involves the design of test cases that validate that the internet program login is functioning properly, and that program input produce valid outputs. All decisions branches and internal code flow should be validate. It is the testing of individual software units of theapplication. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic rests at component level and test a specific business process, application, and/or system conguration.

Tests strategy and approach :

Field testing will be performed manually and functional tests will be written in detail .

Test objectives :

- All fields entries must work properly.
- The entry screen, messages and responses must not be delayed.

Features to be tested :

- Verify that the entries are of the correct format.
- All operations performed correctly.
- Output should displayed in correct manner.

8.2 Integration Testing

Integration tests are designed to test integration software components to determine if they actually runs as one programs. The testing is event driven and is more concerned with the basic outcomes of screen or field. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent.

Integration testing for input :

- The main class which consist of the designing of the main window should consist of linking of the sub class.
- The iteration should be called through the filter to do the operation the message window should be shown.
- The chosen of filter from the main class should be shown from the combo box.

Integration testing for output :

- The main class which consists of the designing of the main window should consist of linking of the sub class.
- The output image should be saved to the appropriate given path.

Test result :

All the test cases mentioned above passed successfully. No defects encountered.

8.3 System Testing

System testing ensures that the entire integrated software system meets requirements. It tests the configuration to ensure known and predictable result. An example of system testing is the configuration oriented system integration test.

8.4 Function Testing

Functional tests provide a systematic demonstration that functions tests are available as specific by the business and technical requirements, system documentation, and user manual.

Functional testing is created on the following items :

Valid input	:	Identified classes of valid input must be accepted.
Invalid input	:	Identified classes of invalid input must be rejected.
Functions	:	Identified functions must be exercised.
Output	:	Identified classes of application outputs must be exercised

.

8.5 Test cases

Unit test case data :

Test No.	Test case	Expected result	Actual result	Pass/Fail
1	Database connection was made	Connection should be successful	Connection is created successfully	Pass
2	Enter values into the database	Values should be stored successfully	Values are stored successfully	Pass
3	Are the values entered into the database are saved correctly.	The should be saved correctly into the database	Yes the data has been saved correctly into the database	Pass

System test case data :

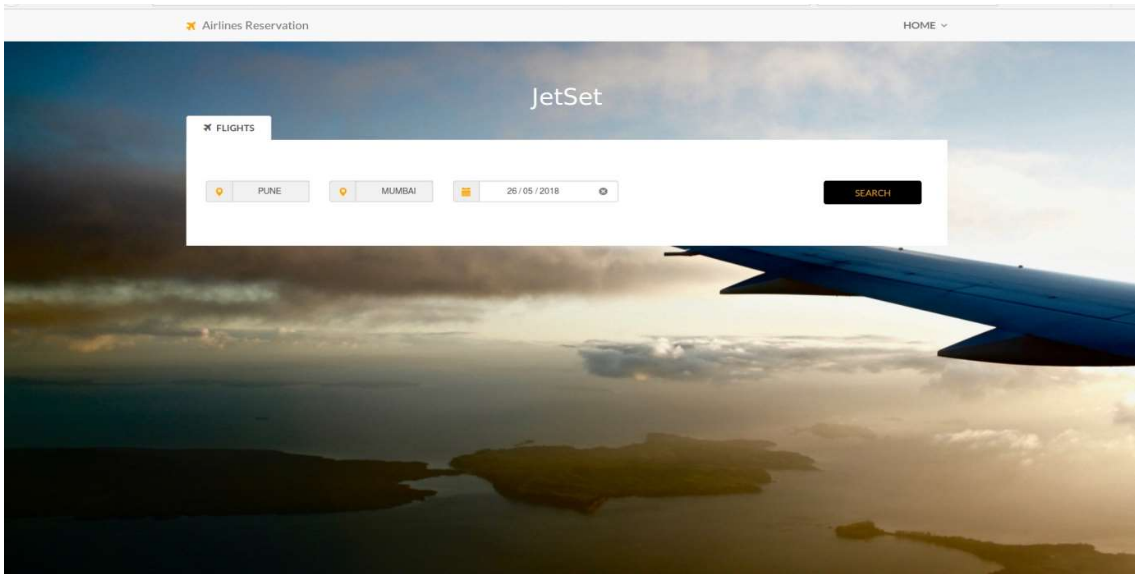
Test No.	Test case	Expected result	Actual result	Pass/Fail
1	Is cassandra running	Cassandra should be running properly	Cassandra is running properly	Pass
2	Is elasticsearch running	Elasticsearch should be running properly	Elasticsearc is running properly	Pass

CHAPTER 9

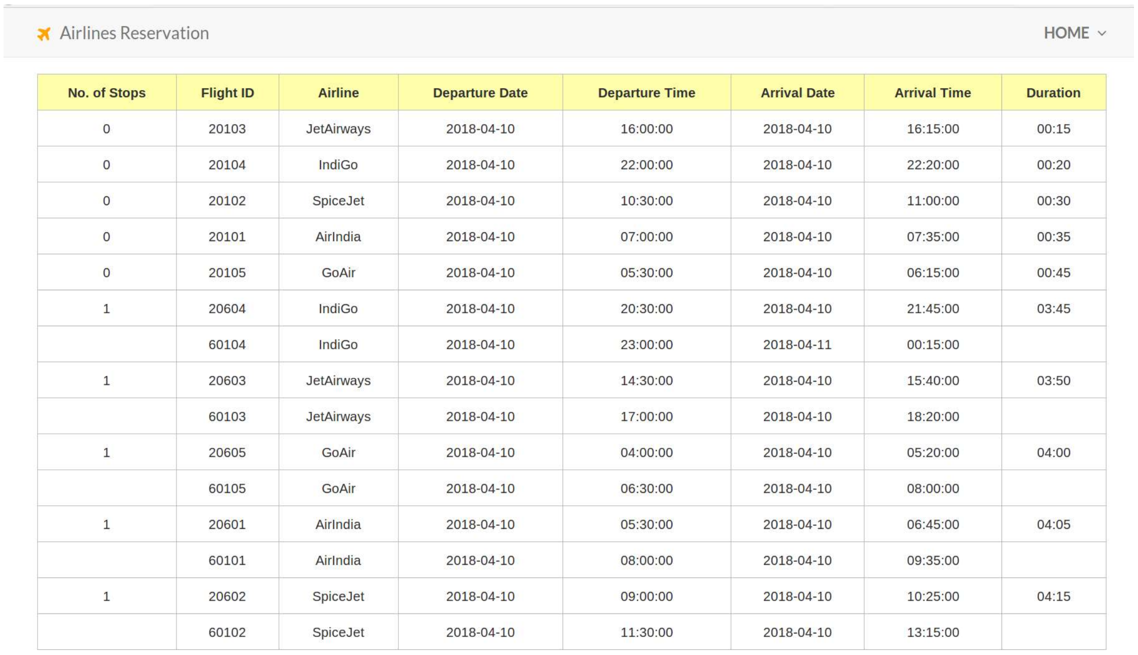
PRODUCT DESCRIPTION

9.1 Interface description for user

Home page :



Result page :



No. of Stops	Flight ID	Airline	Departure Date	Departure Time	Arrival Date	Arrival Time	Duration
0	20103	JetAirways	2018-04-10	16:00:00	2018-04-10	16:15:00	00:15
0	20104	IndiGo	2018-04-10	22:00:00	2018-04-10	22:20:00	00:20
0	20102	SpiceJet	2018-04-10	10:30:00	2018-04-10	11:00:00	00:30
0	20101	AirIndia	2018-04-10	07:00:00	2018-04-10	07:35:00	00:35
0	20105	GoAir	2018-04-10	05:30:00	2018-04-10	06:15:00	00:45
1	20604	IndiGo	2018-04-10	20:30:00	2018-04-10	21:45:00	03:45
	60104	IndiGo	2018-04-10	23:00:00	2018-04-11	00:15:00	
1	20603	JetAirways	2018-04-10	14:30:00	2018-04-10	15:40:00	03:50
	60103	JetAirways	2018-04-10	17:00:00	2018-04-10	18:20:00	
1	20605	GoAir	2018-04-10	04:00:00	2018-04-10	05:20:00	04:00
	60105	GoAir	2018-04-10	06:30:00	2018-04-10	08:00:00	
1	20601	AirIndia	2018-04-10	05:30:00	2018-04-10	06:45:00	04:05
	60101	AirIndia	2018-04-10	08:00:00	2018-04-10	09:35:00	
1	20602	SpiceJet	2018-04-10	09:00:00	2018-04-10	10:25:00	04:15
	60102	SpiceJet	2018-04-10	11:30:00	2018-04-10	13:15:00	

CHAPTER 10

SCOPE AND FUTURE ENHANCEMENT

10.1 Scope

The Airline Reservation System make the life of passengers very easy as they don't need to stand in queues for getting their seat reserved and they can easily make reservation on any airline just from a single system.

The purpose of airline reservation system project is to build an application program which airline could use to manage the reservation on airline tickets.

10.2 Future Enhancement

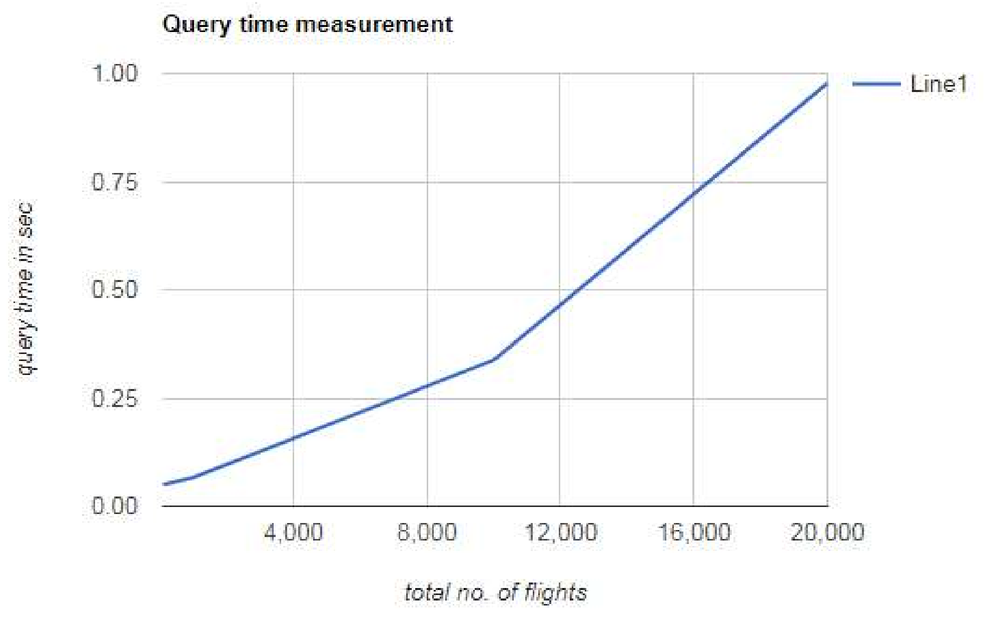
As mentioned above, although our system had completed but it is not perfect, we had planned to make some enhancement in future.

We think that our system has some potential to grow. Besides, we will include more functions and introduce more widgets to the system. Like

- Mobile app
- Call center support
- Real fund transfer through banking system we will provide customer with peer to peer tickets transfer facility.
- We also plan to enhance the interface so that it looks more attractive and interactive.

CHAPTER 11

RESULTS



CHAPTER 12

CONCLUSION

In this system we have constructed the Airline reservation System using graph database and we have analysed the time of execution required for the project. This system helps us to study and understand how to measure the query time required to execute a particular query in our project.

CHAPTER 13

REFERENCE

1. Surajit Medhi, Hemanta K. Baruah "Relational database and graph database: a comparative analysis"

2. Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, Dawn Wilkins "A Comparison of a Graph Database and a Relational Database", ACM Southeast Regional Conference, 2010.

3. Adrian Silvescu, Doina Caragea, Anna Atramentov "Graph Databases"

4. S. Batra and C. Tyagi. Comparative analysis of relational and graph databases. International Journal of Soft Computing

5. B. Iordanov. Hypergraphdb: a generalized graph database. Web-Age Information Management, pages 25–36, 2010.

6. http://s3.thinkaurelius.com/docs/titan/1.0.0/benefits.html

7. https://en.wikipedia.org/wiki/Graph_database

8. https://neo4j.com/developer/graph-db-vs-rdbms/