

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Eyyüp Direk 156396

**COMPARISON OF RELATIONAL AND NON-
RELATIONAL DATABASES ON THE
EXAMPLE OF PROPERTY INFORMATION
MAP**

Master's thesis

Supervisor: Vladimir Viies
Co-supervisor : Lembit Jürimägi

Tallinn 2017

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Eyyüp Direk

Abstract

Nowadays almost every information can be reached through internet. Applications, internet, smart devices are everywhere and they are simplifying our lives. Data is crucial for these applications so are the Databases, with the increasing usage of internet of things even it will be much more crucial. Based on this demands, our perspective to databases have also changed through years. Many years ago when we started using databases we mostly care about the size of the data. Mainly this memory concern has driven us to develop relational database and its normalization. Even though we still care about the size today, we have much more memory space than we used to have. As users, we always want to be informed much faster and reliable data and with the growing popularity of social networks, users query a huge amount of data, since there is a trade-off between query speed and the data stored size in the database, it has driven us to find different solutions than regular SQL and the way we store the data in databases. In the last decade this demands has steered the wheel of the database structures to NOSQL and rather than having a strict dependencies of data, data relationships is required to be more flexible and lightweight.

One of the most prominent applications that we use in our daily life are map applications which guides, informs and even based on traffic jam they can suggest us less congested routes to our home. Data is crucial for these applications so are the Databases. We may not be even aware of that in our daily lives because as users of those applications we mostly interact with the user interface and don't know much about how we are delivered data through. As Users of those applications, we always desire and care about more reliable, faster and more informative data.

The main purpose of this thesis is to research and experiment the differences between relational and non-relational databases on the map application of property information.. This paper sets out to explore the particular types of SQL and NOSQL databases. The research topic has wide variety of different databases comparison. Mostly two types of those databases are focused on this paper. The empirical analysis focuses on the structure of databases used and implemented for the property information map.

The Applications developed uses modern ASP.net web pages, Google Maps API, Oracle SQL developer, IIS 10 Express, Oracle database express edition 11g and

MongoDB 3.4.4 which the last 2 ones are the members of relational and non-relational databases.

As a conclusion, it will be discovered both implementation of the property information map and the differences of databases which are used to feed information for the map.

Keywords: SQL(Structured Query Language), NOSQL(Non-structured Query Language), Database models, relational database, non-relational database, Google Maps Api, Oracle, MongoDB, IIS(Internet Information Service)

This thesis is written in English and is 63 pages long, including 8 chapters, 24 figures and 1 table.

Annotatsioon

Relatsiooniliste ja mitterelatsiooniliste andmebaaside võrdlus kinnisvara kaardirakenduse näite

Internetist võib tänapäeval kätte saada peaaegu iga teabe. Kõikjal on rakendused, internet, nutiseadmed ja need lihtsustavad meie elu. Andmed, sealhulgas andmebaasid, on nende rakenduste jaoks väga olulised ning üha suureneva interneti kasutamisega muutuvad veelgi olulisemateks. Sellest tulenevalt on aastate jooksul muutunud ka ootused andmebaasidele. Aastaid tagasi, kui hakkasime kasutama andmebaase, oli meie jaoks oluline andmete maht. Peamiselt antud mälumahu probleem viis edasi relatsioonandmebaaside ja nende normaliseerimise arendamiseni. Isegi kui me ka tänapäeval endiselt selle pärast muretseme, on meil palju rohkem mäluruumi kui varem. Kasutajatena soovime me alati palju kiiremalt ja usaldusväärsemaid andmeid ning üha suureneva sotsiaalmeedia kasutamise tõttu tehakse tohutul hulgal andmepäringuid, tasakaalu otsimine andmebaasi talletatud andmete ja päringu kiiruse vahel on sundinud meid otsima teisi lahendusi kui tavaline SQL ning viise, kuidas säilitada andmeid andmebaasides. Viimasel kümnendil on võetud suund andmebaaside NOSQL struktureerimisele ning andmete range sõltuvuse asemel nõutakse seostes rohkem paindlikkust ja kergust.

Ühed populaarseimad igapäevaselt kasutatavatest rakendustest on kaardirakendused, mis juhendavad, teatavad ja isegi põhinevad liiklusummikutel ning oskavad soovitada meile vähemhõivatud marsruuti koju. Nimetatud rakendustele on olulised nii andmed kui ka andmebaasid. Oma igapäevaelus ei pruugi me olla sellest isegi teadlikud, kuna kasutajatena suhtleme me enamasti kasutajaliidesega ja ei tea, kuidas andmed meieni edastatakse. Nende rakenduste kasutajatena soovime me aga alati veel usaldusväärsemaid, kiiremaid ja informatiivsemaid andmeid.

Antud lõputöö põhieesmärk on uurida ja testida kinnisvarainfo kaardirakenduste relatsioonandmebaaside ja mitterelatsioonandmebaaside erinevusi. Töö eesmärk on uurida konkreetseid SQL ja NOSQL andmebaaside tüüpe. Uurimisteema sisaldab palju erinevaid andmebaaside võrdluseid. Antud töös on keskendunud põhiliselt kahele andmebaasi tüübile. Empiiriline analüüs keskendub andmebaaside struktuurile, mida kasutatakse ja rakendatakse kinnisvarainfol põhineval kaardil.

Arendatud rakendused kasutavad kaasaegset ASP.net veebilehte, Google Maps Api, Oracle Sql arendust, IIS 10 Express, Oracle database express edition 11g ja MongoDB

3.4.4, milledest kaks viimast kuuluvad relatsioonandmebaaside ja mitterelatsioonandmebaaside hulka.

Kokkuvõtteks leitakse, kuidas teostada kinnisvarainfo kaarti ja millised on kaardile teabe sisestamiseks kasutatavate andmebaaside erinevused.

Märksõnad: SQL(Structured Query Language), NOSQL(Non-structured Query Language, struktuur), Database models, relational database, non-relational database, Google Maps Api, Oracle, MongoDB, IIS (Internet Information Service)

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 63 leheküljel, 8 peatükki, 24 joonist, 1 tabelit.

List of figures

| | |
|--|----|
| Figure 1: RDBMS Popularity Rankings. | 20 |
| Figure 2: Document based databases sample structure. | 23 |
| Figure 3: Wide column databases sample structure. | 24 |
| Figure 4: Graph store database sample structure. | 25 |
| Figure 5: NOSQL Databases popularity rankings pointed out in yellow. | 26 |
| Figure 6: Estonia Administrative units and settlements. | 28 |
| Figure 7: Tallinn districts. | 28 |
| Figure 8: Estonia Land Board Administrative and settlement unit structure. | 29 |
| Figure 9: General Structure and components of Property Information Web Page. | 31 |
| Figure 10: Google Maps API Key. | 33 |
| Figure 11: E-R diagram of Information Map Database. | 35 |
| Figure 12: Property Information Database Model and Relations. | 36 |
| Figure 13: Property Information Database Model and Relations. | 37 |
| Figure 14: Property Information Map implemented in Oracle. | 38 |
| Figure 15: Property Information Map with Google Street View implemented in Oracle. | 39 |
| Figure 16 : Property Information Map Implemented in MongoDB. | 41 |
| Figure 17 : Property Information Map Implemented in MongoDB with Google Street View | 42 |
| Figure 18: Joined Tables. | 43 |
| Figure 19: MongoDB data query and Representation of a document in a collection. ... | 44 |
| Figure 20: Execution plan for the Query. | 45 |

| | |
|---|----|
| Figure 21: Query Result of Necessary Information for the Map..... | 46 |
| Figure 22: Size of Tables List..... | 46 |
| Figure 23: Size of Collection in MongoDB. | 47 |
| Figure 24: Execution Plan for Collection in MongoDB. | 48 |

List of tables

| | |
|---|----|
| Table 1: SQL vs NoSQL Terminologies | 43 |
|---|----|

List of abbreviations and terms

| | |
|--------|---------------------------------------|
| RDBMS | Relational Database Management System |
| DBMS | Database Management System |
| SQL | Structured Query Language |
| NoSQL | Non-Relational or Not Only Sql |
| JSON | Java Script Object Notation |
| BSON | Binary JSON |
| CSS | Cascading Style Sheet |
| HTML | Hyper Text Markup Language |
| DCL | Data Control Language |
| DML | Data Manipulation Language |
| DDL | Data Definition Language |
| TCL | Transaction Control Language |
| API | Application Programming Interface |
| ER | Entity Relationship |
| ADSOID | Address Object ID |
| UI | User Interface |
| CRUD | Create, Read, Update, Delete |

Table of Contents

| | |
|--|----|
| Author’s declaration of originality | 2 |
| Abstract | 3 |
| Annotatsioon | 5 |
| List of figures | 7 |
| List of tables | 8 |
| List of abbreviations and terms | 9 |
| 1. Introduction | 12 |
| 2. Databases History | 13 |
| 3. Background Information | 15 |
| 3.1. Relational Databases (SQL) | 15 |
| 3.2. Relational Database Management System (RDBMS) | 16 |
| 3.2.1. Oracle as an RDBMS | 19 |
| 3.2.2. MSSQL as an RDBMS | 19 |
| 3.2.3. MYSQL as an RDBMS | 20 |
| 3.3. Non-Relational Databases | 21 |
| 3.3.1. Key-Value databases | 22 |
| 3.3.2. Document databases | 22 |
| 3.3.3. Wide Column Store / Column Families | 23 |
| 3.3.4. Graph stores | 24 |
| 4. Case study: Property information map | 26 |
| 4.1. Classification of Estonia administrative units and settlements | 27 |
| 5. Implementation of Property Information Map | 30 |
| 5.1. General Structure of the Implementation | 30 |
| 5.2. Using Google Maps API for Property Information User Interface | 32 |
| 5.2.1. Google Maps API | 33 |
| 5.3. Implementation with Oracle Database | 33 |

| | | |
|------|--|----|
| 5.4. | Implementation with MongoDB..... | 40 |
| 6. | Analysis and Comparison of Results | 42 |
| 6.1. | Terminology, Implementation and Concept differences | 42 |
| 6.2. | Analyzing Oracle on Querying Property Information..... | 45 |
| 6.3. | Analyzing MongoDB on Querying Property Information..... | 46 |
| 6.4. | Comparison of Results | 48 |
| 7. | Suggestions for Future Work | 49 |
| 8. | Conclusion | 49 |
| | References..... | 51 |
| | Appendix 1 – Source code for Oracle Database connected | 52 |
| | Appendix 2 – Source code for Mongo Database connected | 58 |

1. Introduction

Our lives are flooded with all kind of information. Owing to databases we interact with information through internet or web applications, easily and seamlessly on a daily basis. When we look at the database definitions more or less we would see something similar to these definitions below.

“A database is an organized collection of information treated as a unit. The purpose of a database is to collect, store, and retrieve related information for use by database applications.” [1]

“A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.” [2]

As it can be understood from definitions, storing data is not enough to be a database. Databases shouldn't only store but also should be able to manipulate, respond and interact with the users who query them. Since we are in an era which extremely requires to instant access, store, update or delete the data we provide, databases play extremely important role in our interaction with Internet, smart device applications and with the growing popularity of IOT, basically everything we are surrounded and use in our daily life. This interaction between the user and database are done through the DBMSs which allows multiple users to interact with data stored by database. Moreover, DBMSs manage all the processes like where to and how to store data. While it allows to multiple users to modify the data stored by the database, DBMS ensures the data consistency at the same time. Otherwise, the data provided to users, couldn't respond the demands and it would be nothing but failure. Since these requirements, databases have evolved through the years. Maybe not the first but the most influential database model is relational-database model which comes with the term of relational database management systems and ensures the data integrity with the ACID characteristics. In the Last decade , with the increasing amount of data demands over the web, technology companies has been driven to search for different solutions to supply more reliable ,consistent ,fast , maintainable and secure data. Thus today it is getting more popular to use non-relational databases. As its name suggests, without having a relation it targets to have a faster response time than relational databases and at the same time intends to be very lightweight and consistent. Even though this paper concerns more about the

querying data, knowing the general database models and databases history would give a better overview for the this thesis' case study.

2. Databases History

if we go back to first ages of human civilization, just after the invention of writing, we could easily see people using the clay tablets or carvings on walls to be able to record daily data specially by farmers. After people changed their lifestyle from nomadic to more settled life. We needed to keep records of the trades and taxation. Basically, we needed to keep track of our account. This meant not only recording data but also deleting, updating and retrieving records. In the end, this lead to the development of double entry book keeping which emerged in the 13th and 14th centuries. Indexes were used to ease the process of retrieving data.

If we come little bit closer to near past we would encounter first stages of modern times databases ,which examples are ship manifests, card catalogs and product inventories, libraries, governmental records and statistics. The Reporting was another issue that requires fast and impeccable data management which was made manually, and this was quite cumbersome, time consuming, and possible to have error during that process.

Although mechanical calculators were used by the mid nineteenth century to fasten the process, it was far from the desired level of ease and processing times of reporting.

Computers allowed us to automate our databases. Early computer databases were constructed with flat file model a single consecutive list of records but when it comes to search, this was an inefficient way to solution. Over time, we have needed to search and maintain large volumes of records, which have to be faster, reliable and safe.

That requires to retrieve and reach the data in a random order rather than in consecutive. Undoubtedly, this was also flaw of early stage computers storage disks which stores data on magnetic tape.

After IBM introduced hard disk drives in 1956 which allows user to access data randomly, In 1960s IBM has used a hierarchical model for their information management systems. This was constituted by tree structural systems which every node has pointer to its child nodes it has been used successfully by NASA for the lunar lander. After some time of that a more flexible database model was developed by

Charles bachman, which used the same tree structure but every node might have more than one parent nodes yet when the database get complicated it was hard to manage all the pointers between nodes.

In 1970s Ted Codd developed relational database model which was proposing to organize the data into simple tables with related information. There were no pointers to maintain, relations were maintained by having matching data fields in each table. It made it a lot easier to access, merge and change data. Several companies used it as a base for commercial products. In 1975 IBM produced an experimental relational databases named system R. it used structured query language developed by Don Chamberlain and Raymond boyce to search and modify data.

In 1977, Oracle was introduced as a first commercially available relational database compatible with SQL. Since then many companies and individuals have contributed the evolution of relational databases and structured query languages.

Parallel to SQL and relational databases, there has been another way of implementing databases but it wasn't as popular as relational–databases until big data era. Today, we are all connected any social networks and services with our smart phones, tablets and near future with internet of things. Introducing of this services into our life bring another necessity about the way of recording data which gained to NOSQL popularity. “There are a number of reasons for the rise in interest on NOSQL. The first is speed and the poor fit of traditional query languages to technologies such as in memory databases. Secondly there is the form of the data which people want to store, analyze and retrieve. Tweets from Twitter are data that do not easily fit a relational or object oriented structure. Instead column based approaches where a column is the smallest unit of storage or a document based approach where data is denormalized can be applied” [. The idea of No-SQL is on the contrary of the SQL ,it doesn't require strict rules and it is very flexible and scalable and besides dominant web and social service companies like Google, Facebook, Yahoo, it also targets the startup-companies, enterprise companies and open-source developers which requires to be very flexible in aspect of database models. As this thesis topic concerns about the relational and non-relational databases, the next chapter it will be more focused on the relational and non-relational databases' structures and the way how they store and reach data.

3. Background Information

The features and characteristics of database systems vary. These feature differences like how they store, access and manipulate the data makes the differences in the non-functional requirements like accessibility, performance, maintainability, consistency i.e. In this chapter, it is intended to give some background information on general differences of different databases pointing out relational and non-relational databases' types.

3.1. Relational Databases (SQL)

As it is mentioned in history part, relational databases exist since 1970s and E. F. Codd defined a relational model based on mathematical set theory.

A relational database is a database that conforms to the relational model. The relational model has the following major aspects:

Structures

Well-defined objects store or access the data of a database.

Operations

Clearly defined actions enable applications to manipulate the data and structures of a database.

Integrity rules

Integrity rules govern operations on the data and structures of a database.

A relational database stores data in a set of simple relations. A relation is a set of tuples.

A tuple is an unordered set of attribute values.

A table is a two-dimensional representation of a relation in the form of rows (tuples) and columns (attributes). Each row in a table has the same set of columns. A relational database is a database that stores data in relations (tables). For example, a relational database could store information about company employees in an employee table, a department table, and a salary table.

Relational Database Elements

Table

Represents the collection of data in consecutive rows and those tables row mostly connected to each other with foreign keys

Primary and Unique Keys

Primary and unique keys uniquely identifies each row of the table main difference between two keys primary key is enforced to be set default not null which means if there is a record to be inserted into a table it can't have an empty value on primary key column.

Foreign Keys

Foreign keys can be considered as a reference from one table to another table row and depends on the relation of tables it can constraint the insert, delete or update operations.

Views

Basically, Views can represent subset of one or many tables virtually. The main reason to use it is accessibility and speed of querying in case of multiple joined tables view.

Functions

As it is in the all programming languages they can return scalar or table values.

Procedures

Same as functions yet instead of returning value it generally modifies data.

Triggers

They function same as procedures but only under certain circumstances like while inserting, deleting or updating the data in a table. Generally they are used for maintaining the data consistency

3.2. Relational Database Management System (RDBMS)

Basically, RDBMS is a product or system which presents data as collection of rows and columns .Besides that it requires using SQL as a query language which allows user to retrieve, delete, manipulate, create and all kind of transactions to implement. An RDBMS Is responsible for 2 the following types of operations:

Logical operations

These type operations specify what content is required. For example, a patient requests some information regarding his medical condition in a hospital or a doctor as the user of DBMS, he can keep record of his or her patients.

Physical operations

In this type operations, the RDBMS determines how things should be done, For instance how to access data or how to modify or store data .For example, after an application queries a table, the database may use an index to fetch the requested rows, brings the data into memory, and may perform many operations before it returns the final result to the user.

The most common RDBMSs are Oracle, MySql, PostgreSql, MsSql. All these database management systems support relational model as represented by SQL language.

SQL is a set-based declarative language that provides an interface to an RDBMS such as Oracle Database. In contrast to procedural languages such as C, which describe how things should be done, SQL is nonprocedural and describes what should be done [3].

SQL is the ANSI standard language for relational databases. All operations on the data in an Oracle, MySql, PostgreSql, MsSql databases are performed using SQL statements. For example, you use SQL to create tables and query and modify data in tables.

Users specify the result that they want (for example, the names of employees), not how to derive it.

Examples of SQL statements;

```
Select employee_name ,employee_address ,employee_salary from employees;  
INSERT INTO employees  
(employee_id ,employee_name ,employee_address ,employee_salary)  
VALUES (12434, 'john doe','Camden town str. 19', 5000);
```

SQL statements have 4 main groups. The first one is called DDL (data definition language) and the other is DML (data manipulation language) which includes normal select statements. DCL helps control and access of database objects. The last one is TCL ensuring data integrity by managing transactions.

DDL

Main DDL statements are CREATE, ALTER, or DROP which really defines or totally deletes an object or type or tables structure.

DML

Main DML statements are SELECT, INSERT, DELETE, UPDATE, ALTER which query or change the contents.

DCL

Main DCL statements are GRANT, INVOKE which allow users to control their rights on database object like table, procedure and functions i.e.

TCL

Main TCL statements are COMMIT, ROLLBACK, BEGIN, which are generally used for ensuring data integrity.

The database must ensure and have the integrity which most of the time, the case must be so that multiple users can work concurrently without corrupting one another's data. In the heart of the integrity and consistency, there is a term which is called transaction which is a logical, atomic unit of work that contains one or more SQL statements. An RDBMS must be able to group SQL statements so that they are either all committed, which means they are applied to the database, or all rolled back, which means they are undone.

To be able to maintain this integrity the transactions have 4 important characteristics that is simply called ACID consists of the first characters of the terms atomicity, consistency, isolation and durability properties.

Atomicity

Transactions are all-or-nothing. Either all operations go through, or none of them get done.

Consistency

Transactions lead database from one consistent state to another.

Isolation

Transactions cannot see intermediate (not committed) results of each other.

Durability

DBMS must ensure that after committing a transaction all its changes are saved (they can't get lost, for instance, because of power failure).

Even though all RDBMSs requires those all characteristics, they have got different features and they might serve to different targeted users

3.2.1. Oracle as an RDBMS

Oracle was the first commercially available to use database. It has been implemented in C and C++ programming languages. It was the first RDBMS supports SQL. Besides being an RDBMS, “Oracle implements object-oriented features such as user-defined types, inheritance, and polymorphism is called an object-relational database management system (ORDBMS). Oracle Database has extended the relational model to an object-relational model, making it possible to store complex business models in a relational database” [1]

Another aspect of what makes one database different from the others is the way how they store and manage the data. As a database server, which manages to respond multiple users requests, Oracle consists of 2 primary architecture components Oracle Database and Oracle Instance ;

The Oracle Instance mostly composed of the memory part and it is a means to access Oracle Database, while the Oracle Database includes all the physical files on the server. While Oracle instance provides access to multiple users it has to manage all background processes and functionalities which has to comply with ACID principles. A database can be considered in both physical and logical perspective , In logical level Oracle database hierarchically consists of Data blocks ,which corresponds to specific size of memory , an Extent , which contiguous data blocks and A segment consists of Extents allocated for a user object. Over top all, Table-spaces is a container for a segment [4].

PL/SQL

Another feature of Oracle, it supports PL/SQL which is a procedural extension of standard SQL. With the PL/SQL it is possible to use variables, loops, conditions, error catching mechanisms.

3.2.2. MSSQL as an RDBMS

As it is in Oracle, MSSQL is another SQL-based relational database management systems which developed by Microsoft Corporation. It is developed based on C++ programming language.

Same as in all RDBMSs, MSSQL represents data by a table and it has the same elements as the other RDBMSs provides, like trigger, functions, procedures views i.e. What makes different MSSQL from others is its extension language to SQL which is called as TSQL and it can correspond to ORACLE's PL/SQL. It has totally different syntaxes and its own defined variables which intends to improve querying and CRUD operations

3.2.3. MYSQL as an RDBMS

MySQL is another popular RDBMS, which is developed in C and C++ programming languages, renowned with being open-source and first released in 1995 later than its peers. Yet being an open-source platform, it has gained popularity. Company later owned by Oracle corp. Main disadvantage of the MySQL is poor performance scaling in contrast with the Oracle and MSSQL. In spite of this disadvantage, MySQL is very popular as an RDBMS because of being an Open-Source Database platform (Figure 1).

131 systems in ranking, May 2017

| Rank | | | DBMS | Database Model | Score | | |
|----------|----------|----------|------------------------------|-----------------|----------|----------|----------|
| May 2017 | Apr 2017 | May 2016 | | | May 2017 | Apr 2017 | May 2016 |
| 1. | 1. | 1. | Oracle | Relational DBMS | 1354.31 | -47.68 | -107.71 |
| 2. | 2. | 2. | MySQL | Relational DBMS | 1340.03 | -24.59 | -31.80 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational DBMS | 1213.80 | +9.03 | +70.98 |
| 4. | 4. | 4. | PostgreSQL | Relational DBMS | 365.91 | +4.14 | +58.30 |
| 5. | 5. | 5. | DB2 | Relational DBMS | 188.84 | +2.18 | +2.88 |
| 6. | 6. | 6. | Microsoft Access | Relational DBMS | 129.87 | +1.69 | -1.70 |
| 7. | 7. | 7. | SQLite | Relational DBMS | 116.07 | +2.27 | +8.81 |
| 8. | 8. | 8. | Teradata | Relational DBMS | 76.32 | -0.23 | +2.58 |
| 9. | 9. | 9. | SAP Adaptive Server | Relational DBMS | 67.75 | +0.29 | -3.73 |
| 10. | 10. | 11. | FileMaker | Relational DBMS | 56.48 | -0.70 | +9.77 |
| 11. | 11. | 13. | MariaDB | Relational DBMS | 50.98 | +2.26 | +17.01 |
| 12. | 12. | 12. | SAP HANA | Relational DBMS | 49.05 | +0.90 | +7.68 |
| 13. | 13. | 10. | Hive | Relational DBMS | 43.47 | +1.82 | -4.04 |
| 14. | 14. | 14. | Informix | Relational DBMS | 28.23 | +1.44 | -2.35 |
| 15. | 15. | 16. | Microsoft Azure SQL Database | Relational DBMS | 21.55 | +0.45 | +1.87 |
| 16. | 16. | 17. | Vertica | Relational DBMS | 20.69 | +0.19 | +1.40 |
| 17. | 17. | 18. | Netezza | Relational DBMS | 19.79 | +0.19 | +0.53 |

Figure 1: RDBMS Popularity Rankings.

(Source: <https://db-engines.com/en/ranking/relational+dbms>)

3.3. Non-Relational Databases

It is a new trend in database world though, in reality it is not a totally new thing. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early twenty-first century [5], the term of "NoSQL" was in fact the first time used by Carlo Strozzi in 1998 as the name of file-based database he was developing. Ironically it was a relational database just one without a SQL interface. As such it is not actually a part of the whole NoSQL trend we see today. The term re-surfaced in 2009 when Eric Evans used it to name the current surge in non-relational databases [6].

When compared with Relational Databases, NoSQL databases don't consist of table column and row like structures. Instead of them they have got more flexible structures to store data. NoSQL can refer to any of these 3 terms; "not SQL", "not only SQL" or "non-relational".

As the technology advanced, RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Unlike RDBMS, NoSQL databases are designed to easily scale out as and when they grow. Most NoSQL systems have removed the multi-platform support and some extra unnecessary features of RDBMS, making them much more lightweight and efficient than their RDBMS counterparts. The NoSQL data model does not guarantee ACID properties (Atomicity, Consistency, Isolation and Durability) but instead it guarantees BASE properties (Basically Available, Soft state, Eventual consistency). It is in compliance with the CAP (Consistency, Availability, Partition tolerance) theorem [7].

NoSQL TYPES

NoSQL databases differ from each other by data model or another way to say how they store the data. Mainly, they can be categorized into 4 different types;

3.3.1. Key-Value databases

Key-value databases have the most simplistic approach between types of NoSQL. Data consists of 2 parts as the name suggests; Key and Value pairs. Keys are unique identifiers as it is primary key in RDBMSs. This key and value pairs are inspired from hash tables. Even though, the client can manipulate the value of key generally research is limited to one way from key to value otherwise exact-match is required. Data stored can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key, without concerning what's inside; it's the responsibility of the application to understand what was stored. Since key-value stores always use primary-key access, they generally have great performance and can be easily scaled [8].

Main popular key-value databases are Riak, Redis , Berkeley DB, upscaledb (especially suited for embedded use), Amazon DynamoDB (not open-source), Project Voldemort and Couchbase.

3.3.2. Document databases

Documents are the main concept in document databases. The database operates on documents, which can be in different formats like XML, BSON, JSON (Figure 2) and many others. These documents can contain many different datatypes and even can be nested but this totally depends on designer's choice. The documents stored are similar to each other but do not have to be exactly the same. This provides flexibility for the CRUD operations.

```

# Employee Collection (Employee is the root entity) # Client Collection (Client is the root entity)
[
  {
    "ID": 0,
    "first_name": "Eyyüp",
    "last_name": "Direk",
    "Department": [
      {
        "id": 0,
        "Department_Name": "IT",
        "Department_Supervisor": "Sergii
Spivakov",
      }
    ],
    "Sex": "Male",
    "BirthDate": "28.04.1991"
    "Title": "Developer"
    "HoursPerWeek": 40
    "Status": "Working"
  }
]

  {
    "ID": 1,
    "first_name": "Edgar",
    "last_name": "Davis",
    "accounts": [
      {
        "id": 3,
        "account_type": "Checking",
        "account_balance": "4200.00",
        "currency": "YEN"
      }
      {
        "id": 4,
        "account_type": "Investment",
        "account_balance": "3500.00",
        "currency": "YEN"
      }
    ],
    {
      "id": 5,
      "account_type": "Savings",
      "account_balance": "1500.00",
      "currency": "YEN"
    }
  }
]

```

Figure 2: Document based databases sample structure.

They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Key is a string but value can contain MongoDB, CouchDB , Terrastore, OrientDB, RavenDB are the well-known prominent document based databases.

3.3.3. Wide Column Store / Column Families

Column-family databases store data vertically in column families as rows that can have many columns related with a row key. It may seem similar to RDBMS, but names, which correspond to attributes in RDBMSs and formats of columns can vary from row to row across the table. The approach to store and process data by column instead of

row (Figure 3) has its origin in analytics and business intelligence where column-stores operating in a shared-nothing massively parallel processing architecture can be used to build high-performance applications [9].

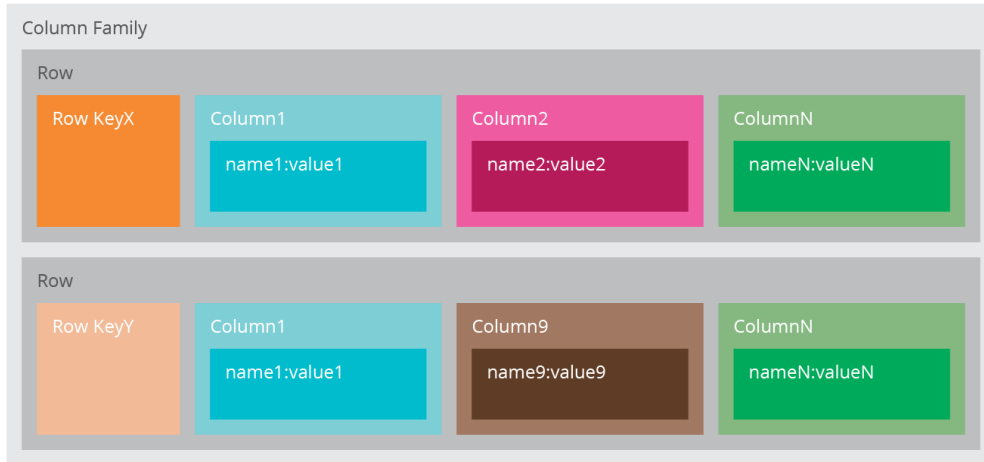


Figure 3: Wide column databases sample structure.

(Source: <https://www.thoughtworks.com/insights/blog/nosql-databases-overview>)

Cassandra is one of the popular column-family databases; there are others, such as HBase, Hypertable, and Amazon DynamoDB.

3.3.4. Graph stores

A graph database structure is built upon graph theory to store, map, and query relationships. Every graph has its own nodes, properties and edges. Data is stored on nodes edges refer to the relationships. It is not an extension of key-value pairs and it's more efficient for storing interconnected data and handling relational querying. Thus naturally they are more suitable for dependency analysis problem solving and some of the social networking scenarios [10].

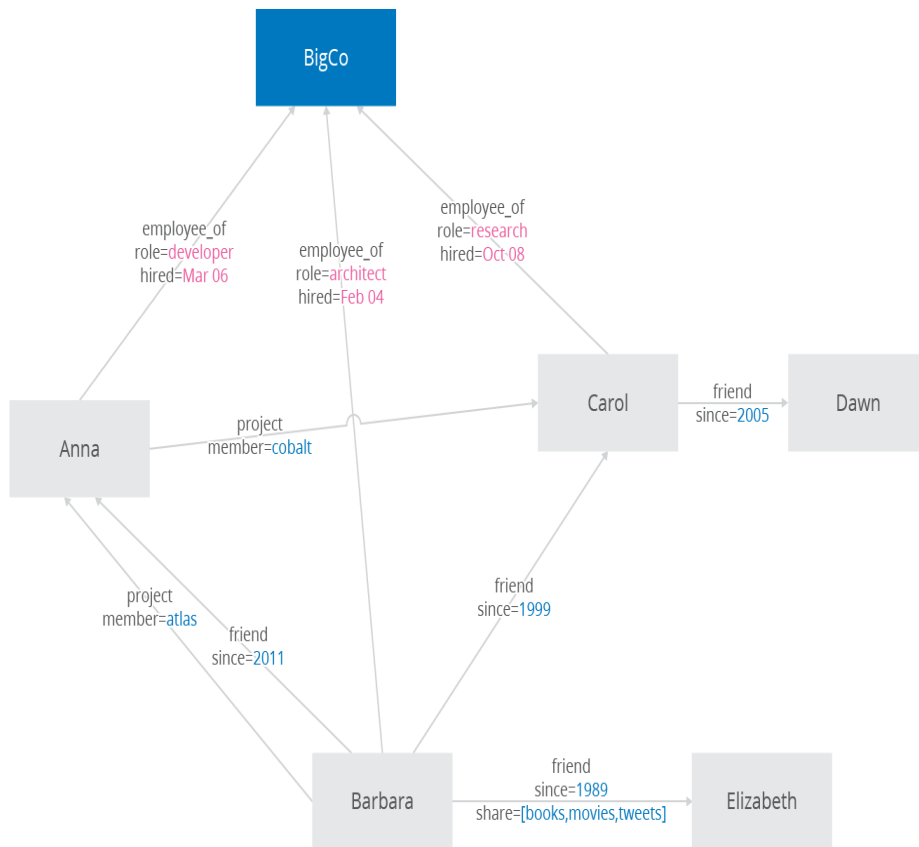


Figure 4: Graph store database sample structure.

(Source: <https://www.thoughtworks.com/insights/blog/nosql-databases-overview>)

There are many graph databases available, such as Neo4J, Infinite Graph, OrientDB, or FlockDB.

Since it is given an overview of NoSQL databases of different types, it can be viewed the ranks of NoSQL databases popularity (Figure 5).

| Rank | | | DBMS | Database Model | Score | | |
|----------|----------|----------|------------------------|-------------------|----------|----------|----------|
| May 2017 | Apr 2017 | May 2016 | | | May 2017 | Apr 2017 | May 2016 |
| 1. | 1. | 1. | Oracle + | Relational DBMS | 1354.31 | -47.68 | -107.71 |
| 2. | 2. | 2. | MySQL + | Relational DBMS | 1340.03 | -24.59 | -31.80 |
| 3. | 3. | 3. | Microsoft SQL Server + | Relational DBMS | 1213.80 | +9.03 | +70.98 |
| 4. | 4. | ↑ 5. | PostgreSQL + | Relational DBMS | 365.91 | +4.14 | +58.30 |
| 5. | 5. | ↓ 4. | MongoDB ++ | Document store | 331.58 | +6.16 | +11.36 |
| 6. | 6. | 6. | DB2 + | Relational DBMS | 188.84 | +2.18 | +2.88 |
| 7. | 7. | ↑ 8. | Microsoft Access | Relational DBMS | 129.87 | +1.69 | -1.70 |
| 8. | 8. | ↓ 7. | Cassandra ++ | Wide column store | 123.11 | -3.07 | -11.39 |
| 9. | 9. | 9. | Redis ++ | Key-value store | 117.45 | +3.09 | +9.21 |
| 10. | 10. | 10. | SQLite | Relational DBMS | 116.07 | +2.27 | +8.81 |
| 11. | 11. | 11. | Elasticsearch + | Search engine | 108.82 | +3.15 | +22.51 |
| 12. | 12. | 12. | Teradata | Relational DBMS | 76.32 | -0.23 | +2.58 |
| 13. | 13. | 13. | SAP Adaptive Server | Relational DBMS | 67.75 | +0.29 | -3.73 |
| 14. | 14. | 14. | Solr | Search engine | 63.77 | -0.60 | -1.85 |
| 15. | 15. | 15. | HBase | Wide column store | 59.50 | +1.04 | +7.67 |
| 16. | ↑ 17. | ↑ 18. | Splunk | Search engine | 56.69 | +1.19 | +12.38 |
| 17. | ↓ 16. | 17. | FileMaker | Relational DBMS | 56.48 | -0.70 | +9.77 |
| 18. | 18. | ↑ 20. | MariaDB + | Relational DBMS | 50.98 | +2.26 | +17.01 |
| 19. | 19. | 19. | SAP HANA ++ | Relational DBMS | 49.05 | +0.90 | +7.68 |
| 20. | 20. | ↓ 16. | Hive + | Relational DBMS | 43.47 | +1.82 | -4.04 |
| 21. | 21. | 21. | Neo4j ++ | Graph DBMS | 36.15 | +1.23 | +3.53 |

Figure 5: NOSQL Databases popularity rankings pointed out in yellow.

(Source: <https://db-engines.com/en/ranking>)

4. Case study: Property information map

First of all, since this thesis intends to compare two databases on an example of Property information map, it is important to comprehend what property information maps are used for. The existing web applications and websites mostly targets at real estate agencies, land registration units and aims to provide the users geological, geographical or historical data about the properties.

Generally this data are recorded by governmental units and provided users on demand. As internet and technology has spread, this kind of formal maps has been moved into the digital environments. Even though there may not be many examples in Europe, in USA many counties have its own property map websites. While the whole buildings in a city or a district taken into account, this means a huge amount of data to be managed and every units like cadastral, parcel, addresses and locations must be stated and preserved clearly to keep data records in order. General name of this systems are called Geographical Information Systems, GIS is a very broad term and they require to

manage all geographical data including roads, agricultural land areas, climate change information, navigation, earthquake information, pipeline routes, tourism information, deforestation information, land registry and many other areas. Due to being a very complex and broad applications, GISs requires different type of Database structure. Generally called GeoSpatial databases, which is more suitable and effective on GIS applications.

However, this thesis case study is more focused on property information rather than all geographical information. Nevertheless, Sample Data source being used in this research are supplied from Estonian Land Board Portal, which besides providing property information, it provides wide variety of land information such as Road Administration, GeoCoding Service, Real Estate statistics and so on. For this research, among these services, public service of the address data system and land registry query has been used in implementation. Especially for the RDBMSs, the structure of data how we use in real life affects the way how we implement our Data Model in relational databases. Considering the fact that every country has different administrative units and different settlement units which conducts the property information, as Estonian land registration is used for this research, it is essential to know how Estonian land registration structures the property's data.

4.1. Classification of Estonia administrative units and settlements

The territory of Estonia is divided into counties, rural municipalities, and towns. There are 15 counties, as it can be seen in the Figure 6 below and each County has its own cities and rural municipalities as subunits.

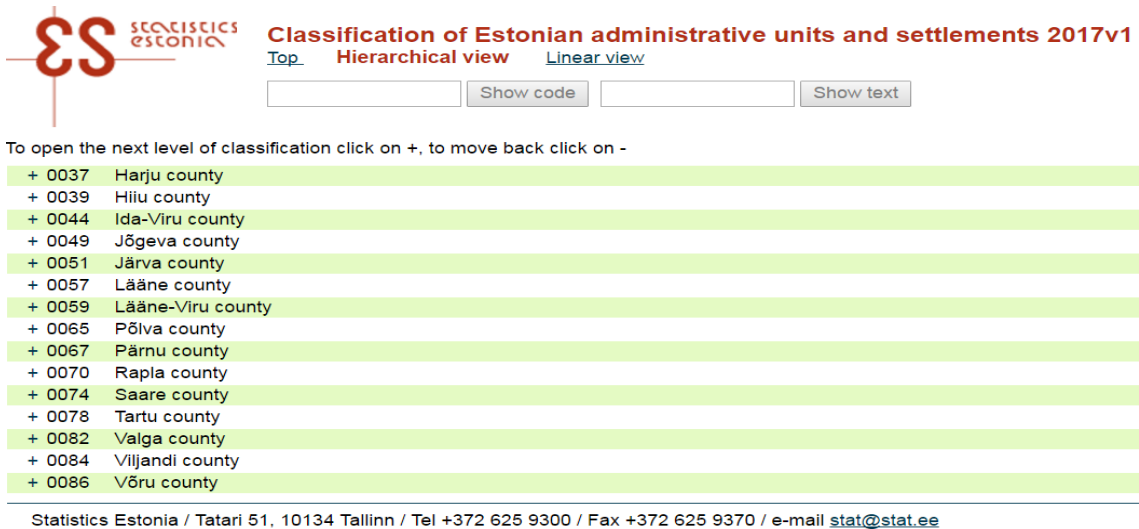


Figure 6: Estonia Administrative units and settlements.

Apart from that, Capital of Estonia; Tallinn has 8 different districts linked to City administration (Figure 7).

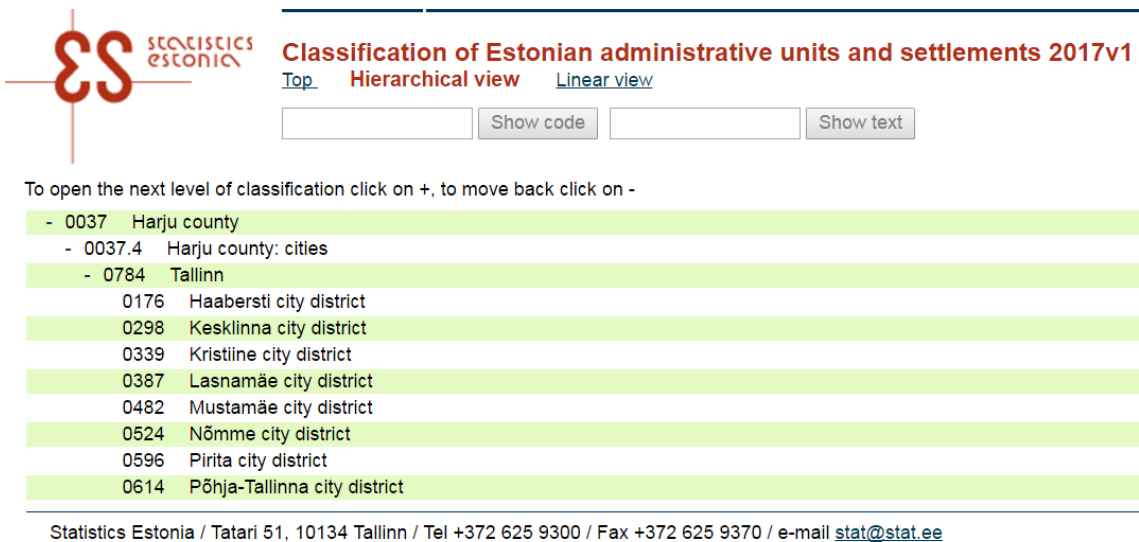


Figure 7: Tallinn districts.

Moreover, the structures of data provided by Estonia Land Board Geoportal are available at 3 levels: counties, municipalities and settlements and each entity has attributes as described in the Figure 8 below.

| Table: Counties | | Table: Municipalities | | Table: Settlements | |
|-----------------|------------------|-----------------------|------------------------|--------------------|---------------------------|
| Attribute | Description | Attribute | Description | Attribute | Description |
| MNIMI | County name | VNIMI | Municipality name | ANIMI | Settlements name |
| MKOOD | County ID (EHAK) | VKOOD | Municipality ID (EHAK) | NIMI | Settlement name with type |
| | | MNIMI | County name | TYYP* | Settlement type |
| | | MKOOD | County ID (EHAK) | AKOOD | Settlement ID (EHAK) |
| | | | | VNIMI | Municipality name |
| | | | | VKOOD | Municipality ID (EHAK) |
| | | | | MNIMI | County name |
| | | | | MKOOD | County ID (EHAK) |

Figure 8: Estonia Land Board Administrative and settlement unit structure.

All these divisions are defined in Territory of Estonia Administrative Division Act which has been accepted since 1995, chapter 1 clause 6 regarding settlement units states the rules as below;

§ 6. Settlement units

1. Settlement units are settlements and urban regions.
2. A rural municipality is divided into settlements which are villages, small towns, towns and cities without municipal status.
3. A city which is an administrative unit is also a settlement within the same boundaries.
4. A city may be divided into urban regions.
5. The types and names of and division lines between settlement units are determined on the basis of applications from rural municipality and city councils, on the bases of and pursuant to the procedure specified by the Government of the Republic.
6. State operations, including the management of statistics, the organization of address systems and the maintenance of registers and cadastres, are based on settlement units. [11]

In real life the rules that we determine, shapes the way of how we store data in a relational Databases. Hence, in relational databases all this perspective should be taken into account.

In conclusion, From Databases perspective, the requirements of property information maps can be a good opportunity and challenge to compare different databases. In previous chapters, Different NoSQL and Relational Databases is compared theoretically. Among them, two most popular and well-structured ones; Oracle and MongoDB are chosen to be compared practically. Before providing this comparison, the platforms used to implement must be understood well.

5. Implementation of Property Information Map

As mentioned in previous chapters, the structure of data to be presented must be designed depending on what exactly being represented on the map. Taking that into account, Oracle Database relational data model and the MongoDB collection structures are created so as to cover all this aspects. Asides from Database part, this particular web application uses HTML, CSS, Javascript technology with the Google Maps Javascript API for the front-end development, IIS as a web server, ASP.NET framework using Razor Markup for back-end development and finally MongoDB and Oracle 11g databases as database servers. Prior to Implementation, in this project, MongoDB 3.4 and Oracle 11g installed on a machine running an Intel i5 CPU 450M @2.40 GHz processor, RAM 4GB.

5.1. General Structure of the Implementation

As it was noted in the introduction part of this chapter, there are many components of this application. To be able to understand how it works, those components should be separated to get a better understanding. First of all, as all web applications or web pages use for their view, Javascript, HTML, CSS constitutes the most important part of the front-end development. These web technologies were used and integrated with the Google Maps Javascript API (Figure X) which shall be discussed further in next subheadings of this chapter. To mention briefly, about ASP.NET and IIS web server, Internet Information Services (IIS) 7 and later provide a request-processing architecture which includes:

- The Windows Process Activation Service (WAS), which enables sites to use protocols other than HTTP and HTTPS.

- A Web server engine that can be customized by adding or removing modules.
- Integrated request-processing pipelines from IIS and ASP.NET.

Basically IIS functions for listening requests made to the server managing processes, and reading configuration files. These components include protocol listeners, such as HTTP.sys, and services, such as World Wide Web Publishing Service (WWW service) and Windows Process Activation Service (WAS) [12].

ASP.NET is a unified Web development model that includes the services necessary for you to build enterprise-class Web applications with a minimum of coding. ASP.NET is part of the .NET Framework, and when coding ASP.NET applications you have access to classes in the .NET Framework. You can code your applications in any language compatible with the common language runtime (CLR), including Microsoft Visual Basic and C#. These languages enable you to develop ASP.NET applications that benefit from the common language runtime, type safety, inheritance, and so on [13].

In this project, ASP.NET Razor syntax was used in implementation. Razor syntax allows developers to write server side code into the client code and server checks if there is any server code in the web application or web page, it runs server code firstly then it sends the web page to browser.

Property Info Map Web Application



Figure 9: General Structure and components of Property Information Web Page.

Apart from Database part, different platforms and web technologies could have been chosen for this dissertation, but because of the personal choice and better experience in aforementioned technologies, these platforms were used in development.

5.2. Using Google Maps API for Property Information User Interface

There are some other map applications like Yandex or Bing Maps though, Google maps is the most popular application that we all use in our daily lives to find our way to home ,or less congested roads to travel faster or nearest shopping centers, pharmacies, restaurants and every kind of places that we can imagine. Since their launch in 2005, Google Maps and Google Earth have had an enormous impact on the way people think, learn, and work with geographic information. With easy access to spatial and cultural information, Google Maps/Earth has provided users with the means to understand their world and their communities of interest. Moreover, the customizable map features and dynamic presentation tools found in Google Maps and Google Earth make each one an attractive option for someone wanting to teach geographic information or make customized maps. For academic researchers, Google Mapping applications are also appealing for their powerful ability to share and host projects, create customized KML files, and to easily communicate their own research findings in a geographic context [14].

As it can be experienced in our daily life, Google API is being used by many different application services. Using the API, numerous people have created useful and interesting "mashups", combining the Google Maps interface with geographic information from other data centers. A few examples are:

Chicago Crime (<http://www.chicagocrime.org/>) - uses Google Maps to visually locate places where crimes have occurred in the Chicago area.

Housing Maps (<http://www.housingmaps.com/>)- combines Google Maps with classified ads from Craigslist to display location of properties for rent or sale.

Incident Log (<http://www.incidentlog.com/>) - combines Google Maps with crime and incident data from around the United States.

Cell Phone Reception and Towers (<http://www.cellreception.com/>) - combines Google Maps with locations of cell phone towers [15].

As it can be seen in referenced examples above, Google API provides many features for your specific target to show maps in your websites or applications. In the light of this

knowledge, it is wise to take a closer look how Google API provides this services and how its features work.

5.2.1. Google Maps API

Google Maps API provides wide variety of tools and methods to create specific targeted websites or software applications. These methods and tools are based on geo-locational coordinates, in other way to say, if someone wants to point out a location on the map, it requires knowing the geo-locational coordinates of that locations which consists of latitude and the longitude of that specific location. Besides knowing that, Google Maps API can only be used after getting an API KEY (Figure 10) which allows you to monitor your application's API usage in the Google API Console. To be able to get an API key, user must register his project on Google API Console, which is an environment provides variety of different APIs based on users' needs. After acquiring the key, it is available to use Google maps in your application or websites.

```
<script async defer  
  src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCqhcTpzfJWATABAc0zDozTtu2dGL4DM6z4&v=3&callback=initMap">  
</script>
```

Figure 10: Google Maps API Key.

To show a simple marker on the map, you have to specify latitude and longitude of that location. Depending on users' desire, it can be demonstrated many different information in different styles by using JavaScript and HTML. Turning to Property information maps it is crucial to know what data will be represented on the map and generally the data provided is strictly structured in land registry units of governments. Hierarchically, every property belongs to one sub unit of a city or a municipality.

5.3. Implementation with Oracle Database

In the first place, as Oracle being an RDBMS, to be able to create our database model it is important to design and create ER diagrams. It is common and most-chosen and well-

known way to determine the data model. The first step of ER model is to define entities and their relationships with each other. After this step, relational data model gets easier to construct. The entity-relationship (E-R) data model was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database. The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the E-R model. [13] Mainly, there are three concepts or terms that must be comprehended. First, each entity is a distinguishable object from the other entity's types. Second, attributes, which carries the data on entities and entity is represented by these attributes. Third, as it is mentioned and the name of ER suggests, relations which define associations between two entities. After briefly mentioning ER model, we can start defining our entities for property information map database. As it is addressed in chapter 4.1, Estonia Land Board has defined its main entities as settlements, municipalities and counties. For the reason that this thesis topic is more focused on property information in a specific area rather than the whole geographical information of all lands, entities defined more specific to its own types. Specially, instead of settlement units, there are three entities taking place of settlement units. Building is the main object which represents the each property. Parcels are larger area and may consist of many buildings and the city district which consists of many parcels. In Figure 11, roughly designed entity relationship diagram can be seen to address entities, their relations and some of the attributes.

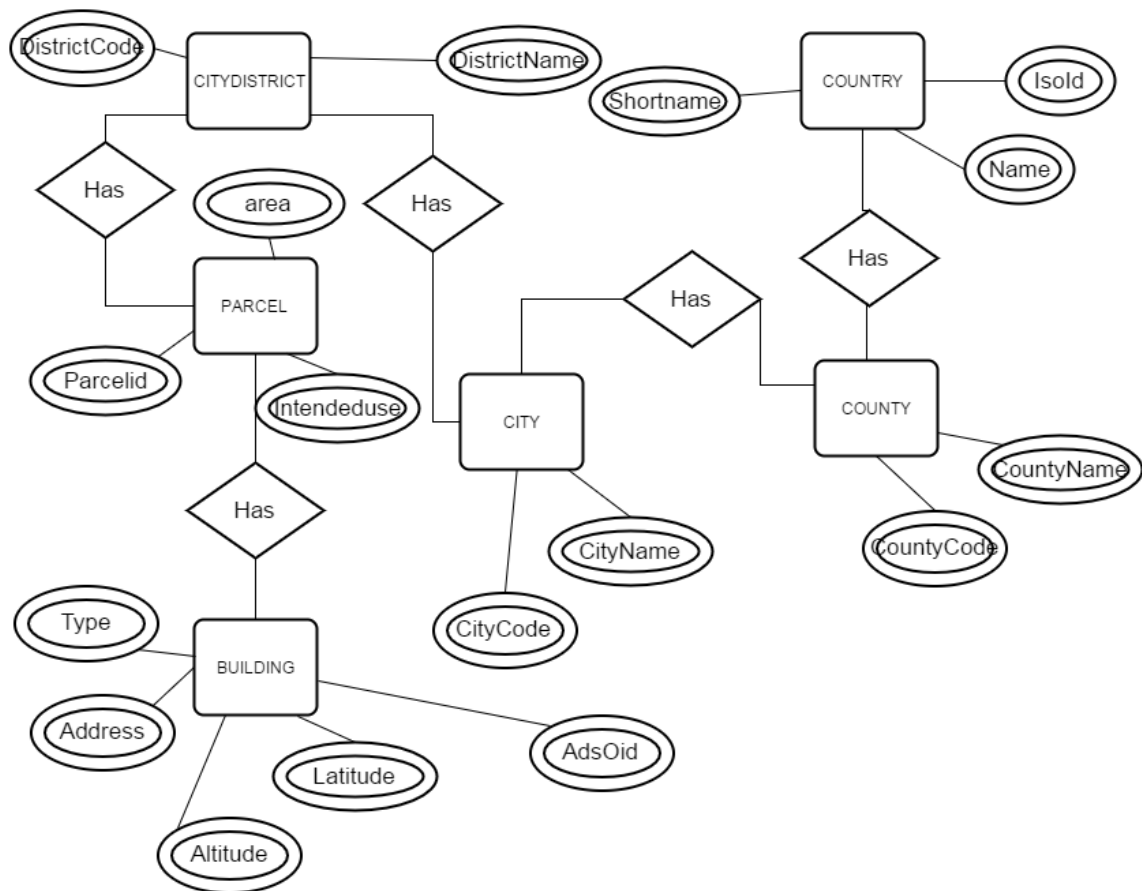


Figure 11: E-R diagram of Information Map Database.

Apart from the main entities there are high level entities described for future extension of the project as it can be applied to most of the countries. Over all, Country as an object covers all other entities, one level below that cities are defined and under cities there are counties as subunits. After defining entities, relationships and some of the attributes, as a next step before the relational model, constraints should be defined, which are mostly real life conditions that shapes our relational model.

Constraints of relations generally define the relationship types between entities. Since property map's ER diagram quite straight-forward and hierarchical, all the relationship types between entities are one to many relationships, if we start from the top level for example every countries has one or more counties and every county has at least one and more cities and every city has one or more districts and it continues so on so forth.

Normally, the structure used by Estonia Land Board includes municipalities which some of them do not comply with this hierarchical data model, the reason why it is not used in this model is municipalities' information is not required to be represented in property information map intended to be created and also some of the Estonia local administrative units that mentioned in chapter 4.1, do not comply with the general hierarchy because there are some cities and other units without municipal status. Instead of municipality entities, more straight approach is taken to keep data consistent. After completing ER diagram and knowing necessary constraints, it can be started to create relational model based on ER diagram. Starting from Building entity, it has ADSOID, which is a unique id given to every building with VERSIONID and, since it uniquely identifies every building it can be considered as the primary key of Building table and to maintain the relationship with the parcel as a foreign key PARCELID references to parcel table. The next entity relation is between parcel and city district as every city district consists of many parcels, the direction of foreign key DISTRICTID references to city district table (Figure 12).

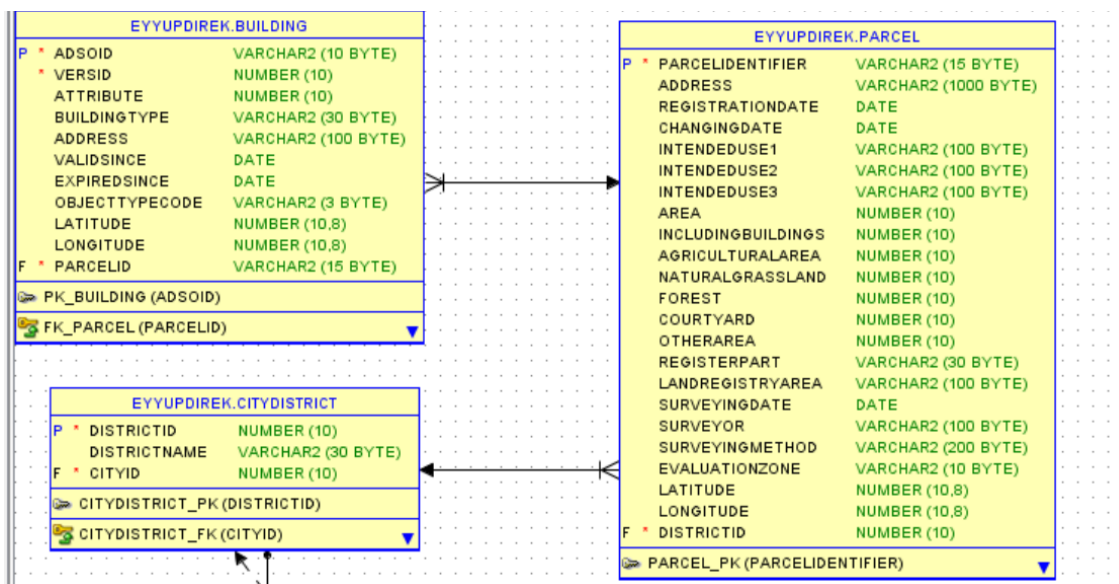


Figure 12: Property Information Database Model and Relations.

On the higher level, naturally every city district belongs to a city and one city consists of many city districts so that CITYID as a foreign key points to the city table. Same structure applies for the city table to county table and they are associated on COUNTYID as a foreign key, which can be seen in Figure 13.

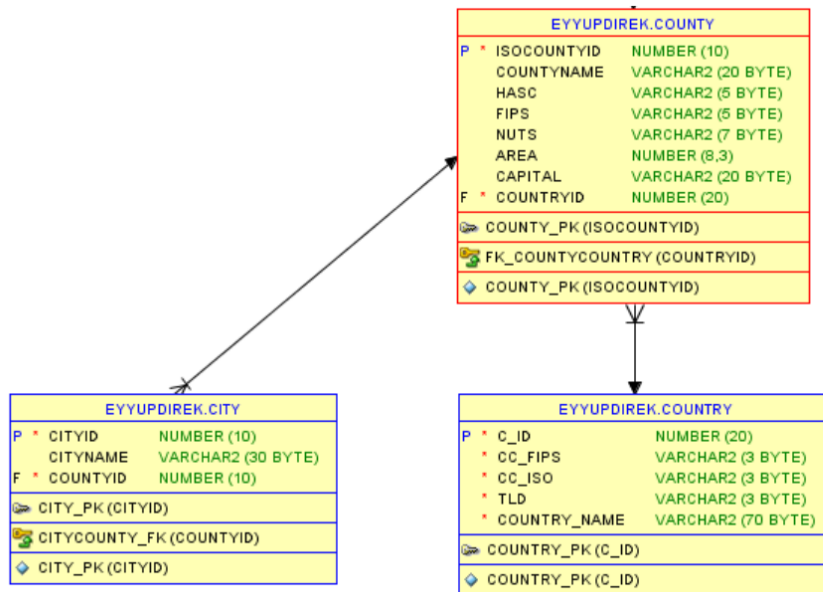


Figure 13: Property Information Database Model and Relations.

After creating database structure, the data belongs to buildings in Mustāmae district was inserted into the tables. Then as a next step, Interface of the web page was set by using HTML, Javascript and Google API. HTML design roughly has two main parts. First one is the division where general parcel information shown and the other is the google maps focused on the point where most of the buildings that planned to be shown on the map. Following the UI design, the connection between Oracle database and UI requires to be implemented by using ORACLE Data Provider for .NET 4 Managed Driver. Under the project solution, Managed Driver DLL was added as a reference to be able to use connection methods to Oracle database from ASP.NET web page project. As a

backend development language C# razor syntax used which allows to embed server based code into web pages. In the follow-up phase of the project, query string was created to retrieve data from Oracle database. Retrieving the Data from Database, for every building there was set a marker to point the building out on that location. Each marker has its own latitude and longitude information taken from database and has its own information window pops up when it is clicked on and represents information about the building. When it is clicked, related parcel information is being represented at the same time on the left pane (Figure 14).

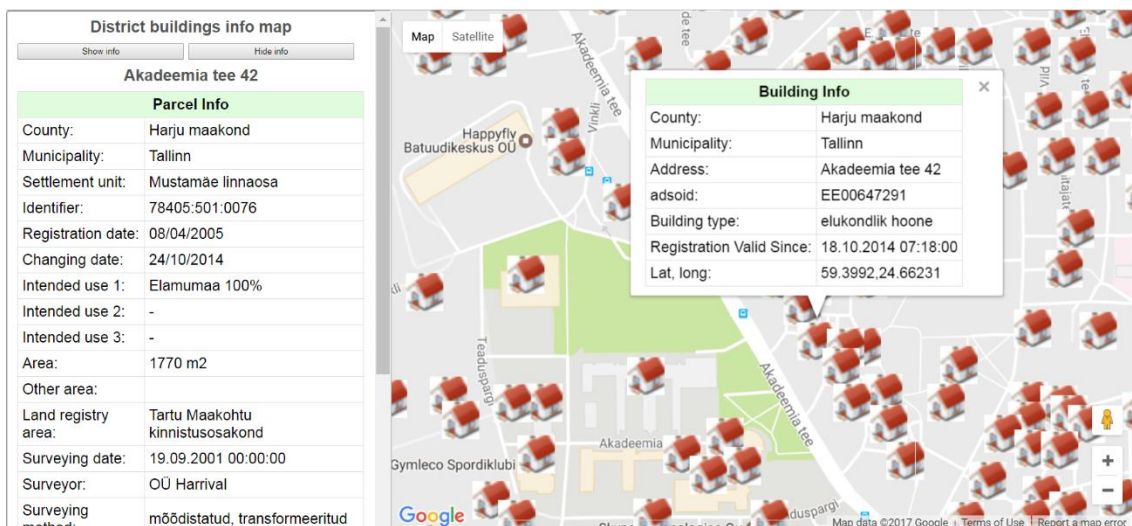


Figure 14: Property Information Map implemented in Oracle..

Another useful feature of Google Maps API is the street view (Figure 15). By using that feature users can step inside the locations nearby the buildings being represented and it shows the landmarks around the buildings and the physical information can be matched with the information provided on the map. Google Street View needs 4 elements SIZE, HEADING, FOV, PITCH to show around a specific location size specifies the output size of the image in pixels.

SIZE is specified as {width}X{height} - for example, size=600x400 returns an image 600 pixels wide, and 400 high.

HEADING indicates the compass heading of the camera. Accepted values are from 0 to 360 (both values indicating North, with 90 indicating East, and 180 South). If

no heading is specified, a value will be calculated that directs the camera towards the specified location, from the point at which the closest photograph was taken.

FOV (default is 90) determines the horizontal field of view of the image. The field of view is expressed in degrees, with a maximum allowed value of 120. When dealing with a fixed-size viewport, as with a Street View image of a set size, field of view in essence represents zoom, with smaller numbers indicating a higher level of zoom.

PITCH (default is 0) specifies the up or down angle of the camera relative to the Street View vehicle. This is often, but not always, flat horizontal. Positive values angle the camera up (with 90 degrees indicating straight up); negative values angle the camera down (with -90 indicating straight down) [16].

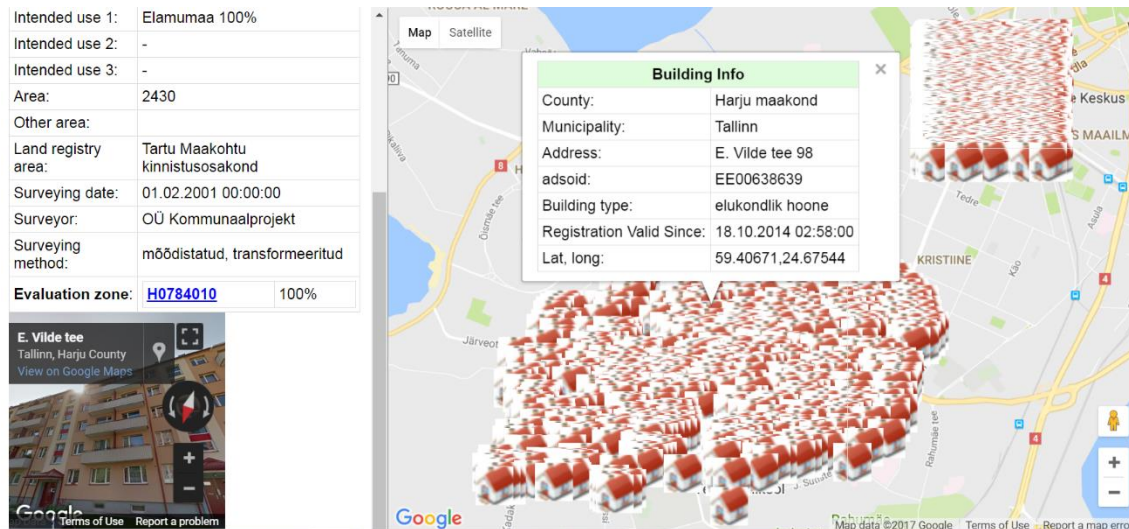


Figure 15: Property Information Map with Google Street View implemented in Oracle.

In the end, Implementation with the Oracle Database completed successfully and as it is shown figures in the Figure 15, the web application reflects the data from Oracle Database.

5.4. Implementation with MongoDB

Installation of MongoDB was the first step of this phase. MongoDB installation has 3 components to be installed, Mongo, Mongod, Mongos. Mongo is an interactive JavaScript shell interface to MongoDB, which provides a powerful interface for systems administrators as well as a way for developers to test queries and operations directly with the database. Mongo also provides a fully functional JavaScript environment for use with a MongoDB.

Mongod is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations.

Mongos for “MongoDB Shard,” is a routing service for MongoDB shard configurations that processes queries from the application layer, and determines the location of this data in the sharded cluster, in order to complete these operations. From the perspective of the application, a mongos instance behaves identically to any other MongoDB instance [17].

After completing all necessary installations the environment is ready to use MongoDB. Implementing in one Database makes easier to implement on the other, since the data model is constructed. Even though MongoDB is not a relational database the same data was used to implement with MongoDB. One of the main advantages of MongoDB is that it is possible to bulk insert into data with importing JSON files into it. By inserting the collection, Document is automatically created and it makes the developers job much easier than RDBMS do. The details about the implementation differences will be discussed further in the next chapter of this research. Turning back to MongoDB implementation, in the first place, The Data inserted in Oracle database was exported into JSON document in such a way that the structure of collections determined by the exported data. In another words, Data was shaped in Oracle Database to create documents structure. After inserting the data, the same HTML and CSS design as in Oracle was used for UI. Even though the same design was used, the packages and drivers which are used to connect UI are totally different so that it required installing MongoDB driver from Nuget Package manager, which provides packages to developers in Microsoft Developer Platforms. As a next step, installed driver was added as a reference to web page project without these references certain types of libraries and methods cannot be used specially connection string created to connect the database and

web server. To be able to retrieve data in the backend, C# Razor syntax was used, same as in Oracle application

The way MongoDB returns results to queries are different from Oracle because of that, The results returned after querying must be processed differently. MongoDB returns to queries batch by batch which means after querying for the first time MongoDB do not return whole records in the MongoDB database it only returns specific number of documents as a response. Because of that reason, in C# code there is an asynchronous method to handle each document coming from Database server and insert into the markers.

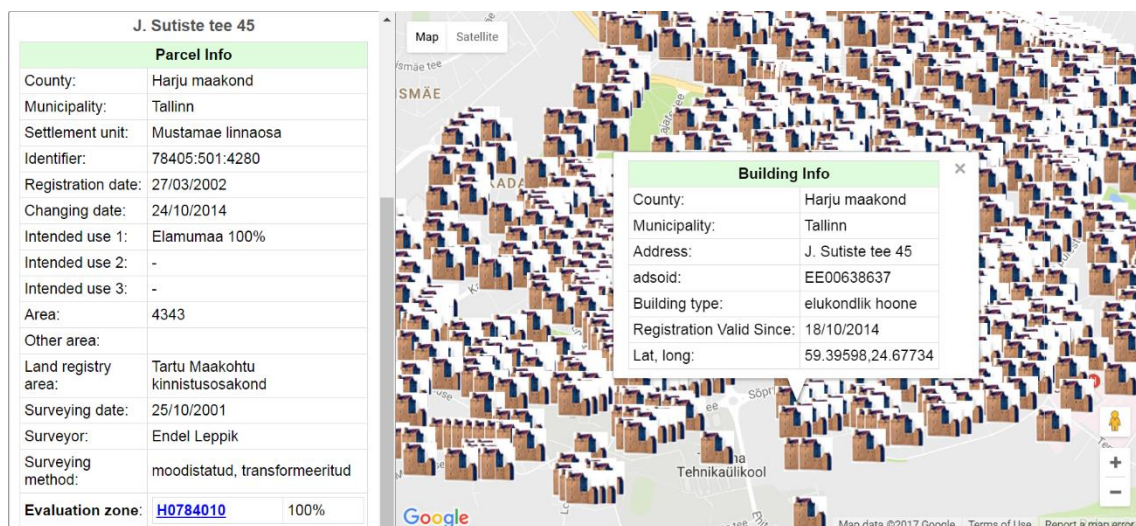


Figure 16 : Property Information Map Implemented in MongoDB.

Following all backend development process, the web page is ready to run and it creates the UI as it is the same in Oracle example, then it points out every building which locations of taken from MongoDB and the web page runs successfully showing the related building and parcel information when clicked on the markers can be seen in the figure above (Figure 16).

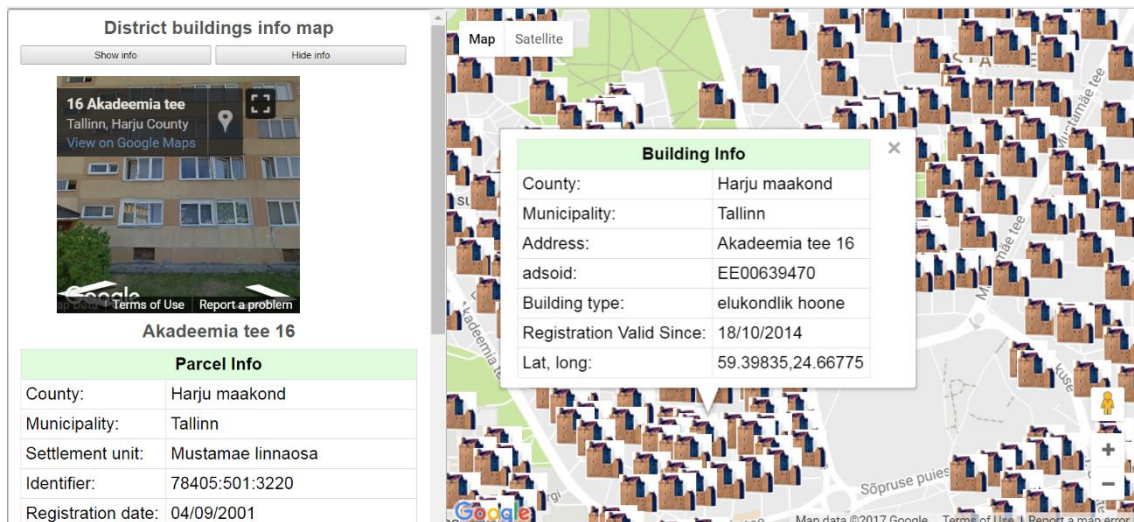


Figure 17 : Property Information Map Implemented in MongoDB with Google Street View

To be able to differentiate MongoDB implementation from Oracle Implementation, design of left pane and the markers icons was changed (Figure 17). Google street view location on the left pane moved to the upper section.

6. Analysis and Comparison of Results

Analysis of data deduced from research can be interpreted in many different ways and the comparison between these 2 databases can be done from many different aspects. Therefore, prior to comparing the research results, it can be a good starting point to compare Oracle and MongoDB in terms of database concepts and programming syntaxes as a member of RDBMSs and NoSQL databases.

6.1. Terminology, Implementation and Concept differences

During the implementation and research of the project, it turned out that even though both MongoDB and Oracle are two databases, as a terminology and the way they refer to the objects of databases are defined different, yet they can be mapped and correspond to counterparts.

| SQL Terms/Concepts | MongoDB Terms/Concepts |
|---|--|
| Database | Database |
| Table | Collection |
| Row | Document or BSON document |
| Column | Field |
| Index | Index |
| Table Joins | \$lookup, embedded documents |
| Specify any unique column or column combination as primary key. | In MongoDB, the primary key is automatically set to the _id field. |

Table 1: SQL vs NoSQL Terminologies

As it can be seen in the Table 1 above, MongoDB doesn't support joins. There are not any tables, columns or rows like structures. In spite of supporting non-structured or half-structured data, MongoDB uses primary key to uniquely identify each document in the collection and indexes are available to use. To be able to join two collections, there is a specific function called lookup, it can match the fields but it doesn't coerce to have a relation between collections. In this project, there is only one collection was created to store the data, which contains necessary information composed by 5 tables in Oracle RDBMS (Figure 18).

```

select parcelid,building.Address,buildingtype,building.adsoid,validsince
,building.latitude
,building.longitude
,cityname,countyname,districtname, registrationdate
,changingdate
,intendeduse1,intendeduse2,intendeduse3,parcel.area,otherarea
,landregistryarea,surveyingdate,surveyor,surveyingmethod,evaluationzone
from building ,citydistrict,parcel ,city , county
where parcel.districtid=citydistrict.districtid
and parcel.parcelidentifier=building.parcelid
and city.cityid=citydistrict.cityid
and county.isocountyid=city.countyid and parcelid='78405:501:1910'

```

The screenshot shows a SQL query execution window. The query text is as follows:

```

select parcelid,building.Address,buildingtype,building.adsoid,validsince
,building.latitude
,building.longitude
,cityname,countyname,districtname, registrationdate
,changingdate
,intendeduse1,intendeduse2,intendeduse3,parcel.area,otherarea
,landregistryarea,surveyingdate,surveyor,surveyingmethod,evaluationzone
from building ,citydistrict,parcel ,city , county
where parcel.districtid=citydistrict.districtid
and parcel.parcelidentifier=building.parcelid
and city.cityid=citydistrict.cityid
and county.isocountyid=city.countyid and parcelid='78405:501:1910'

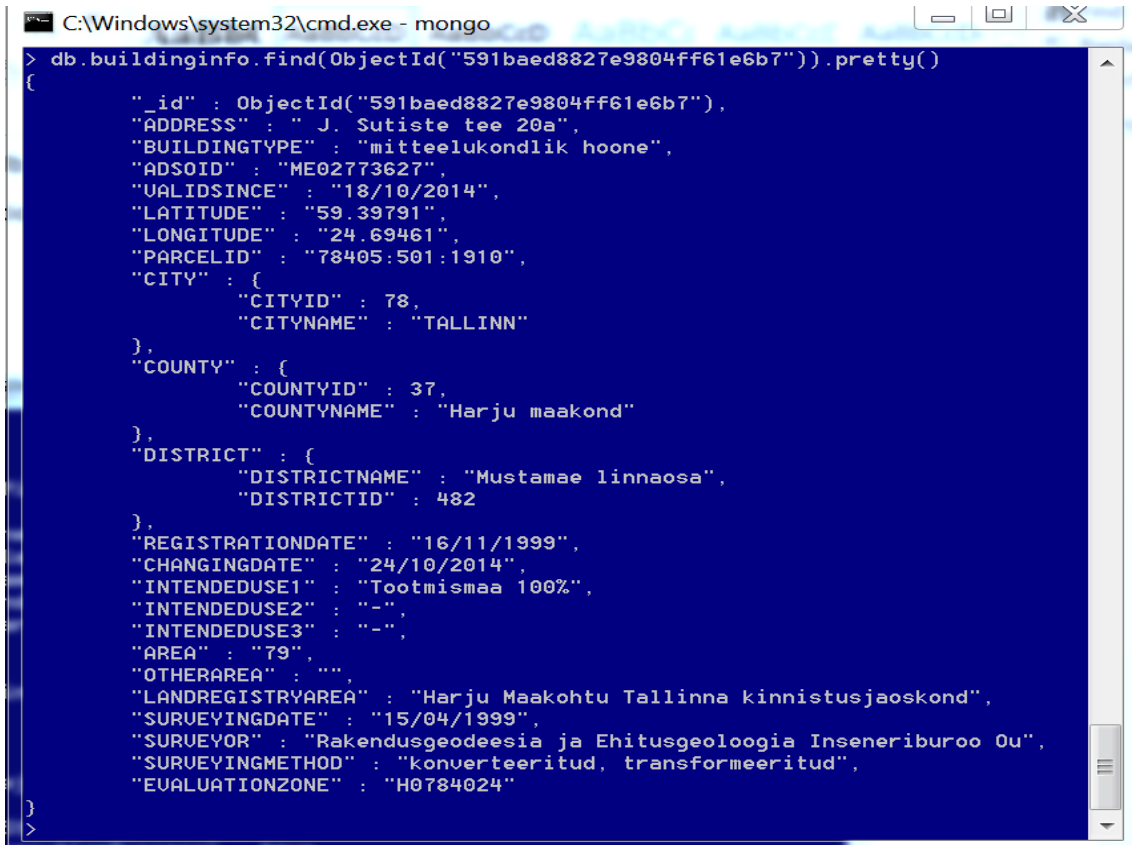
```

The interface shows the query was executed successfully, returning 1 row in 0,009 seconds. The result table has the following columns and values:

| PARCELID | ADDRESS | BUILDINGTYPE | ADSOID | VALIDSINCE |
|----------------|--------------------|-----------------------|------------|------------|
| 78405:501:1910 | J. Sütiste tee 20a | mitteelukondlik hoone | ME02773627 | 18/10/2014 |

Figure 18: Joined Tables.

The same data acquired by MongoDB in Mongo shell is more lightweight and being a single collection doesn't need to join and it can be reached just by single querying method.



```
C:\Windows\system32\cmd.exe - mongo
> db.buildinginfo.find(ObjectId("591baed8827e9804ff61e6b7")).pretty()
{
  "_id" : ObjectId("591baed8827e9804ff61e6b7"),
  "ADDRESS" : " J. Sutiste tee 20a",
  "BUILDINGTYPE" : "mitteelukondlik hoone",
  "ADSOID" : "ME02773627",
  "UALIDSINCE" : "18/10/2014",
  "LATITUDE" : "59.39791",
  "LONGITUDE" : "24.69461",
  "PARCELID" : "78405:501:1910",
  "CITY" : {
    "CITYID" : 78,
    "CITYNAME" : "TALLINN"
  },
  "COUNTY" : {
    "COUNTYID" : 37,
    "COUNTYNAME" : "Harju maakond"
  },
  "DISTRICT" : {
    "DISTRICTNAME" : "Mustamae linnaosa",
    "DISTRICTID" : 482
  },
  "REGISTRATIONDATE" : "16/11/1999",
  "CHANGINGDATE" : "24/10/2014",
  "INTENDEDUSE1" : "Tootmismaa 100%",
  "INTENDEDUSE2" : "-",
  "INTENDEDUSE3" : "-",
  "AREA" : "79",
  "OTHERAREA" : "",
  "LANDREGISTRYAREA" : "Harju Maakohtu Tallinna kinnistusjaoskond",
  "SURVEYINGDATE" : "15/04/1999",
  "SURVEYOR" : "Rakendusgeodeesia ja Ehitusgeoloogia Inseneriburoo Ou",
  "SURVEYINGMETHOD" : "konverteeritud, transformeeritud",
  "EVALUATIONZONE" : "H0784024"
}
```

Figure 19: MongoDB data query and Representation of a document in a collection.

Since having one collection, lookup functions have no use. In MongoDB every collection has 16 Mb limit, if the required data exceeds that size then user have to create a new collection, which will be the case that joins might be necessary between collections. Although not used in the MongoDB implementation because of importing the same data acquired from Oracle Database, if a user desires to create a collection simply writing name of collection and inserting in it is enough to create a new collection without describing the collections' features because of being a non-structured, shapeless BSON object and every document in the collection doesn't have to be in the same structure as the other documents in the same collection, which means, users do not have to determine what kind of document and in which order to insert into collection. Besides many other simplifying approaches, there is a CRUD like operation which is called UPSERT. It derives from combination of insert and update. In this operation,

MongoDB checks the collection if the data which is requested to be updated is available, it updates, if it is not, it inserts the new data into the collection.

6.2. Analyzing Oracle on Querying Property Information

As it is mentioned in previous chapter, Oracle being a relational database, to gather the whole information about the property, it has to reach every table to get necessary data of Property information map. Owing to the fact that normalization rules applied, on the one hand, reducing data redundancy and simplifying the database design, but on the other hand, relational databases increases the required number of access to different tables. For instance: In the database model shown in chapter 5.3. In order to get city district information for a building, it is needed to be referenced to parcel first and then necessary information can be accessed through matching parcel's DISTRICTID. Taking all these aspects into account, it is important to see execution plan of the queried information shown in Figure 20 below.

| PLAN_TABLE_OUTPUT | | | | | | |
|-------------------|-----------------------------|-----------------|------|-------|------|--------|
| Id | Operation | Name | Rows | Bytes | Cost | (%CPU) |
| 0 | SELECT STATEMENT | | 1 | 357 | 70 | |
| 1 | NESTED LOOPS | | 1 | 357 | 70 | |
| 2 | NESTED LOOPS | | 1 | 233 | 4 | |
| 3 | NESTED LOOPS | | 1 | 208 | 4 | |
| 4 | NESTED LOOPS | | 1 | 188 | 3 | |
| 5 | TABLE ACCESS BY INDEX ROWID | PARCEL | 1 | 162 | 2 | |
| * 6 | INDEX UNIQUE SCAN | SYS_C008009 | 1 | | 1 | |
| 7 | TABLE ACCESS BY INDEX ROWID | CITYDISTRICT | 14 | 364 | 1 | |
| * 8 | INDEX UNIQUE SCAN | CITYDISTRICT_PK | 1 | | 0 | |
| 9 | TABLE ACCESS BY INDEX ROWID | CITY | 30 | 600 | 1 | |
| * 10 | INDEX UNIQUE SCAN | CITY_PK | 1 | | 0 | |
| 11 | TABLE ACCESS BY INDEX ROWID | COUNTY | 15 | 375 | 0 | |
| * 12 | INDEX UNIQUE SCAN | COUNTY_PK | 1 | | 0 | |

Figure 20: Execution plan for the Query.

What is Cost? The Oracle Optimizer is a cost-based optimizer. The execution plan selected for a SQL statement is just one of the many alternative execution plans considered by the Optimizer. The Optimizer selects the execution plan with the lowest cost, where cost represents the estimated resource usage for that plan. The lower the

cost the more efficient the plan is expected to be. The optimizer's cost model accounts for the IO, CPU, and network resources that will be used by the query [18].

| | PARCELID | COUNTYNAME | CITYNAME | DISTRICTNAME | REGISTRATIONDATE | CHANGINGDATE | INTENDEDUSE1 |
|-------|----------------|---------------|----------|-------------------|------------------|--------------|-------------------|
| 10996 | 79282:767:4311 | Harju maakond | Tallinn | Mustamäe linnaosa | 30/10/2006 | 20/12/2007 | sctxaejhxbwjcdy a |
| 10997 | 78497:456:3287 | Harju maakond | Tallinn | Mustamäe linnaosa | 19/05/2004 | 09/05/1990 | ubripapezenrjm d |
| 10998 | 78953:599:4372 | Harju maakond | Tallinn | Mustamäe linnaosa | 17/03/2011 | 24/06/1991 | itymcylidvfledh g |
| 10999 | 79216:578:2159 | Harju maakond | Tallinn | Mustamäe linnaosa | 12/12/2011 | 20/03/2009 | pbzluqihlylnslh d |
| 11000 | 79178:644:1214 | Harju maakond | Tallinn | Mustamäe linnaosa | 11/04/2013 | 14/07/1992 | tqwfgblyynyparp d |
| 11001 | 78745:973:4775 | Harju maakond | Tallinn | Mustamäe linnaosa | 03/06/1996 | 25/04/1995 | pynwmkzzusstago k |

Figure 21: Query Result of Necessary Information for the Map.

There are 11001 records in total, which belongs to each building, gathered by joining the tables and it takes 3,681 seconds to retrieve and the size of the whole data in all tables is around 4 MB (Figure 21).

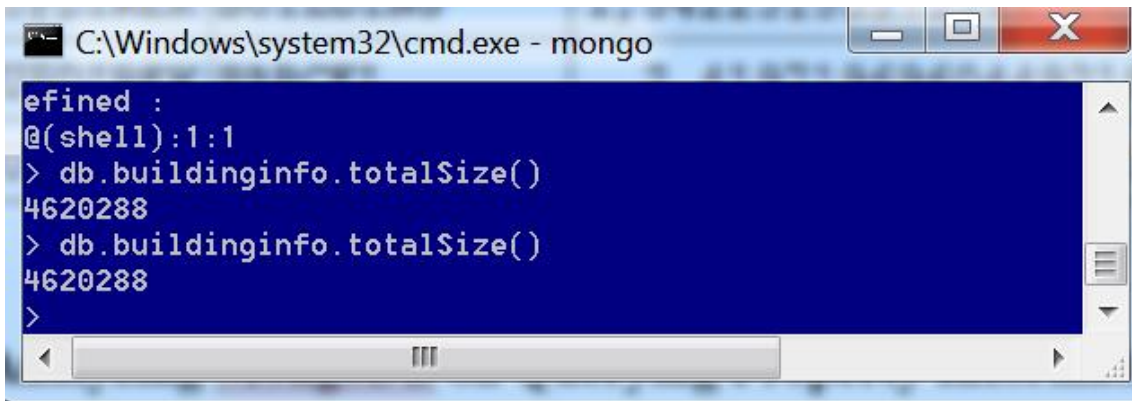
| | OWNER | TABLE_NAME | SIZEMB |
|---|------------|--------------|------------------------|
| 1 | EYYUPDIREK | COUNTRY | 0,00010013580322265625 |
| 2 | EYYUPDIREK | CITYDISTRICT | 0,000347137451171875 |
| 3 | EYYUPDIREK | CITY | 0,00057220458984375 |
| 4 | EYYUPDIREK | COUNTY | 0,000743865966796875 |
| 5 | EYYUPDIREK | BUILDING | 1,54223155975341796875 |
| 6 | EYYUPDIREK | PARCEL | 2,419719696044921875 |
| 7 | EYYUPDIREK | TOTAL | 3,963714599609375 |

Figure 22: Size of Tables List.

6.3. Analyzing MongoDB on Querying Property Information

In chapter 6.1 where stating the characteristic and implementation differences, it is mentioned that being non-structured or half-structured gives quite convenience in terms of development and flexibility. Nevertheless, this flexibility might cost more space in

the memory. As seen in the Figure 23 below, the total size of 11001 data equals to 4620288 byte which is approximately 5.5 Mb.



```
C:\Windows\system32\cmd.exe - mongo
defined :
@(\shell):1:1
> db.buildinginfo.totalSize()
4620288
> db.buildinginfo.totalSize()
4620288
>
```

Figure 23: Size of Collection in MongoDB.

Having no relations, generally, MongoDB requires to access only one collection, same case as in this project and to be able to return how collections are being reached, simply explain command has to be run on the collection. The MongoDB query optimizer processes queries and chooses the most efficient query plan for a query given the available indexes. The query system then uses this query plan each time the query runs. The query optimizer only caches the plans for those query shapes that can have more than one viable plan. For each query, the query planner searches the query plan cache for an entry that fits the query shape. If there are no matching entries, the query planner generates candidate plans for evaluation over a trial period. The query planner chooses a winning plan, creates a cache entry containing the winning plan, and uses it to generate the result documents. If a matching entry exists, the query planner generates a plan based on that entry and evaluates its performance through a replanning mechanism. This mechanism makes a pass/fail decision based on the plan performance and either keeps or evicts the cache entry. On eviction, the query planner selects a new plan using the normal planning process and caches it. The query planner executes the plan and returns the result documents for the query [19].

After running command, query planner was listed to show details as below (Figure 24). Following the list, it suggests that 11001 records were reached with a collection scan, which took 71 milliseconds.

```
Seç C:\Windows\system32\cmd.exe - mongo
},
"ok" : 1
}
> db.buildinginfo.find().explain("executionStats")
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "test.buildinginfo",
    "indexFilterSet" : false,
    "parsedQuery" : {
    },
    "winningPlan" : {
      "stage" : "COLLSCAN",
      "direction" : "forward"
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 11001,
    "executionTimeMillis" : 71,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 11001,
    "executionStages" : {
      "stage" : "COLLSCAN",
      "nReturned" : 11001,
      "executionTimeMillisEstimate" : 10,
    }
  }
}
```

Figure 24: Execution Plan for Collection in MongoDB.

6.4. Comparison of Results

Following the analysis, the query results, which retrieved the same data, of two databases can be compared. According to the analysis made previous subheadings of this chapter, both databases gave different results in terms of size and the performance. However, these results can change, based on the servers workload, which databases run on it and the execution plans` results are more like estimations rather than a real value in terms of costs and time. To be able to more accurate while comparing, the execution plans were run several times with queries. The figures representing, more reflects the average value. Apparently, there is not a big difference from memory size perspective but interestingly, in MongoDB the required data to be represented on the map is larger than the Oracle database. However, considering tables and relations Oracle database was expected to be larger in size. Even so that, since the data inserted with is relatively small in size and it may not be suggesting a general assumption. In Oracle database the

size of data was approximately 4 Mb and it is 1.5 Mb less than MongoDB's collection size. Nevertheless, MongoDB was much faster to return query results and the result execution plan suggested estimation time is 71 milliseconds which is much smaller than 3 seconds that Oracle responded for the full scanned tables. For a single query, based on altitude and latitude data, MongoDB responded in 30 milliseconds while Oracle Database corresponds in 1.4 seconds and during executions made through Property Information Map application even though the difference was not so tangible, because of once data accessed for the next time server keeps the data in memory, the response time seemed to be slightly in favor of MongoDB.

7. Suggestions for Future Work

This thesis aimed to implement a property information map application by using 2 different databases, since the framework of this thesis is rather limited, the current project should be extended in such a way that used databases can be compared from different aspects. Thus, as a future work, this project can be extended both theoretically and practically. In other words, the data size has to be increased which will entail the increasing hardware requirements as well. After doing so, the interaction between users and the application can be improved so that users should be able to update, delete or create new information regarding properties. Moreover, following these steps, there can be created more complicated data models which will be a better case to compare 2 databases under different circumstances. Property map's features can be improved by adding historical data which will show the changes about the property over time. Considering the whole properties in a country, this project's scale is rather small and if all properties' information is added into these databases then under that conditions the project and the results can provide better understanding. It should be always taken into account that this project is only focused on the property information rather than the whole geographical land information.

8. Conclusion

The research yielded that being members of relational and non-relational databases Oracle and MongoDB have certain advantages over each other. Specially, MongoDB is much more flexible and implementation in MongoDB requires less effort and querying

response time faster than Oracle, Yet, it cannot be completely concluded that MongoDB is superior of Oracle based on only this research. Also, reviewed literature suggests that if the queries had been made on the similar structure for instance MongoDB having more than one collection and relations between those collections the difference would have been much less from the performance aspect [20]. Furthermore, for this research, databases were used only for querying and there was no any CRUD operation implemented. Considering all this aspects, to sum up, it can be said that for Property Information Map which intends to give only information but not to be edited by users, can provide better performance and flexibility if it is implemented in MongoDB. Practically, the project which was implemented is more suitable for MongoDB rather than Oracle because of easy implementation, data structure, faster response time and being non-relational provides certain level of convenience. On the other hand, if the project created such a way which requires more complex transactions and strictly defined rules then it would be a better solution to use Oracle Database. DBMSs have many different aspects and database is a very broad concept. Comparing Databases from one aspect could lead a misjudgement. Based on requirements, designer or the developer of the system must pick the right DBMS. Even sometimes some applications can function better if both of relational and non-relational databases are used together. Even the MongoDB and Oracle developers are very well aware of their strong and weak sides such that MongoDB states that While most modern applications require a flexible, scalable system like MongoDB, there are use cases for which a relational database like Oracle would be better suited. Applications that require complex, multi-row transactions (e.g., a double-entry bookkeeping system) would be good examples. MongoDB is also not a drop-in replacement for legacy applications built around the relational data model and SQL. A concrete example would be the booking engine behind a travel reservation system, which typically involves complex transactions. While the core booking engine might run on Oracle, those parts of the app that engage with users – serving up content, integrating with social networks, managing sessions – would be better placed in MongoDB [21].

Furthermore, Oracle as a corporation developed its own NoSQL database in 2011 and the concepts of sharding and replication provides great flexibility and scalability when workload over a server need to be distributed on multiple servers which creates costly efficient solution for the existing but need to be overhauled database systems. Also, social web sites lead to spread the usage of MongoDB, which data doesn't have any

structure mostly and non-relational databases quickly adopted by many social websites' corporations. To sum up, this thesis doesn't give an overall view from all aspects of 2 databases and property information map tough, it certainly provides some understanding on both an instruction on how to implement and the differences of NOSQL and SQL databases on the example of Property Information Maps.

References

- [1] Oracle Corp., "Oracle Help centre," 2017. [Online]. Available: http://docs.oracle.com/cd/E25178_01/server.1111/e25789/intro.htm. [Accessed 12 03 2017].
- [2] S. B. N. Ramez Elmasri, *Fundamentals of database systems / .—6th ed.*, Boston, Massachusetts: Addison-Wesley, 2011.
- [3] Oracle Databases, "Introduction to Oracle Database," 10 04 2017. [Online]. Available: https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT001.
- [4] Oracle Corp., "Oracle Help Centre," 2017. [Online]. Available: <https://docs.Oracle.com/database/121/CNCPT/intro.htm#CNCPT88784>. [Accessed 12 3 2017].
- [5] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," *CiSE*, vol. 0018, no. 9162, pp. 12-14, 2010.
- [6] S. G. S. N. Edward, *Practical MongoDB: Architecting, Developing, and Administering MongoDB*, Apress, 2015.
- [7] A. P. P. Ameya Nayak, "Type of NOSQL Databases and its Comparison with Relational Databases," *International Journal of Applied Information Systems (IJ AIS)*, vol. 5, no. ISSN : 2249-0868, pp. 16-19, March 2013.
- [8] P. Sadalage, *NoSQL Databases: An Overview*, ThoughtWorks, Inc. <https://www.thoughtworks.com/insights/blog/nosql-databases-overview>, 2014.
- [9] C. Strauch, *NoSQL Databases*, Hochschule der Medien, Stuttgart, 2012.
- [10] Y. Gurevich, *Comparative Survey of NoSQL/ NewSQL DB Systems (Master's Dissertation)*, The Open University of Israel Computer Science Division, 2015.
- [11] Riigi Teataja, "Territory of Estonia Administrative Division Act," Riigikogu, Tallinn, 2013.
- [12] R. Templin, "Introduction to IIS Architectures," 16 11 2007. [Online]. Available: <https://docs.microsoft.com/en-us/iis/get-started/introduction-to-iis/introduction-to-iis-architecture>. [Accessed 15 1 2017].
- [13] H. F. K. S. Abraham Silberschatz, *Database system concepts*, McGraw-Hill, 2011.
- [14] A. N. Eva Dodsworth, "Academic Uses of Google Earth and Google Maps in a Library Setting," *Information Technology and Libraries* , vol. 31, no. 2, p. 102,

2012.

- [15] J. M. J. F. Nicholas C. Zakas, Professional Ajax, Wiley, 2005, p. 184.
- [16] Google, "Google Developers," [Online]. Available: <https://developers.google.com/maps/documentation/streetview/intro>. [Accessed 15 2 2017].
- [17] MongoDB, "MongoDB Documentation," [Online]. Available: <https://docs.mongodb.com/manual/reference/program/mongo/>. [Accessed 2 02 2017].
- [18] White paper Oracle, "The Oracle Optimizer Explain the Explain Plan," april 2017. [Online]. Available: <http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-explain-the-explain-plan-052011-393674.pdf>. [Accessed 5 May 2017].
- [19] MongoDB Inc., "MongoDB Documentation," 2017. [Online]. Available: <https://docs.mongodb.com/manual/core/query-plans/>. [Accessed 20 3 2017].
- [20] L. S. S. Humasak T. A. Simanjuntak, "Query Response Time Comparison NOSQLDB MONGODB with SQLDB Oracle," *ResearchGate*, vol. 13, no. 1, pp. 95-105, 2015.
- [21] "MongoDB and Oracle Compared," MongoDB Inc., [Online]. Available: <https://www.mongodb.com/compare/mongodb-oracle>. [Accessed 10 5 2017].
- [22] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

Appendix 1 – Source code for Oracle Database connected

```
1. @using WebMatrix.Data
2. @using Oracle.ManagedDataAccess.Client;
3. @using Oracle.ManagedDataAccess.EntityFramework;
4.
5. @{
6.
7.
8.     Page.Title = "Home Page";
9.     var db = Database.Open("BuildingInfo");
10.    var selectQueryString = "select building.Address,buildingtype,
        building.adsoid,validsince,replace(building.latitude,',','.')
        as latitude";
11.    selectQueryString += ",replace(building.longitude,',','.')
        as longitude ,parcelid";
12.    selectQueryString += ",countyname,cityname,citydistrict.districtname,
        to_char(registrationdate,'DD/MM/YYYY') as registrationdate,
        to_char(changingdate,'DD/MM/YYYY') as changingdate";
13.    selectQueryString += ",intendeduse1,intendeduse2,intendeduse3,
        parcel.area,otherarea ";
```

```

14.     selectQueryString += ",landregistryarea,surveyingdate,surveyor,
        surveyingmethod,evaluationzone";
15.     selectQueryString += " from building ,citydistrict,parcel ,city , county "
;
16.     selectQueryString += " where parcel.districtid=citydistrict.districtid ";
17.     selectQueryString += " and parcel.parcelidentifier=building.parcelid ";
18.     selectQueryString += " and city.cityid=citydistrict.cityid ";
19.     selectQueryString += " and county.isocountyid=city.countyid ";
20. }
21.
22.
23.
24. <html>
25. <head>
26.     <!--<script src="JavaScript.js"></script>-->
27.     <link rel="stylesheet" type="text/css" href="MapStyle.css">
28. </head>
29. <body>
30.     <div class="container">
31.         <div class="options-box">
32.             <h1>District buildings info map </h1>
33.             <div>
34.                 <input id="show-listings" type="button" value="Show info">
35.                 <input id="hide-listings" type="button" value="Hide info">
36.             </div>
37.             <div id="infowin">
38.
39.             </div>
40.
41.         </div>
42.         <div id="map"></div>
43.     </div>
44.
45.     <script>
46.
47.         var map;
48.         // Create a new blank array for all the listing markers.
49.         var markers = [];
50.         function initMap() {
51.             // Constructor creates a new map -
only center and zoom are required.
52.             map = new google.maps.Map(document.getElementById('map'), {
53.                 center: { lat: 59.396956, lng: 24.670968 },
54.                 zoom: 15,
55.
56.             });
57.
58.             var largeInfowindow = new google.maps.InfoWindow();
59.             var tr = 0; var locations = [];
60.             @foreach (var row in db.Query(selectQueryString))
61.             {
62.                 <Text>
63.                 locations.push({
64.                     title: "@row.address",
65.                     location: {
66.                         lat: parseFloat("@row.latitude"),
67.                         lng: parseFloat("@row.longitude")
68.                     },
69.                     city: "@row.cityname",
70.                     district: "@row.districtname",
71.                     parcelid: "@row.parcelid",
72.                     registrationdate: "@row.registrationdate",
73.                     changingdate: "@row.changingdate",
74.                     intendeduse1: "@row.intendeduse1",
75.                     intendeduse2: "@row.intendeduse2",

```

```

76.         intendeduse3: "@row.intendeduse3",
77.         area: "@row.area",
78.         otherarea: "@row.otherarea",
79.         landregistryarea: "@row.landregistryarea",
80.         surveyingdate: "@row.surveyingdate",
81.         surveyor: "@row.surveyor",
82.         surveyingmethod: "@row.surveyingmethod",
83.         evaluationzone: "@row.evaluationzone",
84.         buildingtype: "@row.buildingtype",
85.         adsoid: "@row.adsoid",
86.         validsince: "@row.validsince"
87.     })
88. </Text>
89. }
90.
91.     // The following group uses the location array to create an array
of markers on initialize.
92.     for (var i = 0; i < locations.length; i++) {
93.         // Get the position from the location array.
94.         var position = locations[i].location;
95.         var title = locations[i].title;
96.         var year = locations[i].year;
97.         var x = locations[i].location.lat;
98.         var y = locations[i].location.lng;
99.         var city=locations[i].city;
100.         var district = locations[i].district;
101.         var image = "/Images/Home.PNG";
102.         var parcelid = locations[i].parcelid;
103.         var registrationdate = locations[i].registrationdate;
104.         var changingdate = locations[i].changingdate;
105.         var intendeduse1 = locations[i].intendeduse1;
106.         var intendeduse2 = locations[i].intendeduse2;
107.         var intendeduse3 = locations[i].intendeduse3;
108.         var area = locations[i].area;
109.         var otherarea = locations[i].otherarea;
110.         var landregistryarea = locations[i].landregistryarea;
111.         var surveyingdate = locations[i].surveyingdate;
112.         var surveyor = locations[i].surveyor;
113.         var surveyingmethod = locations[i].surveyingmethod;
114.         var evaluationzone = locations[i].evaluationzone;
115.         var buildingtype = locations[i].buildingtype;
116.         var adsoid = locations[i].adsoid;
117.         var validsince = locations[i].validsince
118.
119.
120.
121.
122.
123.         // Create a marker per location, and put into markers a
rray.
124.         var marker = new google.maps.Marker({
125.             map: map,
126.             icon: image,
127.             position: position,
128.             title: title,
129.             animation: google.maps.Animation.DROP,
130.             id: i,
131.             lng: y,
132.             lat: x,
133.             city:city,
134.             district: district,
135.             parcelid: parcelid,
136.             registrationdate: registrationdate,
137.             changingdate:changingdate,
138.             intendeduse1: intendeduse1,
139.             intendeduse2: intendeduse2,

```

```

140.         intendeduse3: intendeduse3,
141.         area: area,
142.         otherarea: otherarea,
143.         landregistryarea:landregistryarea,
144.         surveyingdate:surveyingdate,
145.         surveyor:surveyor,
146.         surveyingmethod:surveyingmethod,
147.         evaluationzone: evaluationzone,
148.         buildingtype: buildingtype,
149.         adsoid: adsoid,
150.         validsince:validsince
151.
152.
153.         });
154.         // Push the marker to our array of markers.
155.         markers.push(marker);
156.         // Create an onclick event to open an infowindow at each
157.         h marker.
158.         marker.addListener('click', function () {
159.             populateInfoWindow(this, largeInfowindow);
160.         });
161.         document.getElementById('show-
162.         listings').addEventListener('click', showListings);
163.         document.getElementById('hide-
164.         listings').addEventListener('click', hideListings);
165.         }
166.         // This function populates the infowindow when the marker is
167.         clicked. We'll only allow
168.         // one infowindow which will open at the marker that is clicked
169.         , and populate based
170.         // on that markers position.
171.         function populateInfoWindow(marker, infowindow) {
172.             // Check to make sure the infowindow is not already opened
173.             on this marker.
174.             if (infowindow.marker != marker) {
175.                 infowindow.marker = marker;
176.                 var contentstr = '<div>' +
177.                 '<h1 id="firstHeading" class="firstHeading">' + marker.title + '</h1>' +
178.                 '</div>' +
179.                 '<table class="dogisPopupTble">' +
180.                 '<tbody>' +
181.                 '<tr><th style="text-align: center; background-
182.                 color: #dffddd;font-family: arial, sans-
183.                 serif;" colspan="2"> Parcel Info </th></tr>' +
184.                 '<tr><td>County:</td><td>Harju maakond</td></tr>' +
185.                 '<tr><td>Municipality:</td><td>' + marker.city + '</td></tr>' +
186.                 '<tr><td>Settlement unit:</td><td>' + marker.district + '</
187.                 td></tr>' +
188.                 '<tr><td>Identifier:</td><td>' + marker.parcelid + '</td></
189.                 tr>' +
190.                 '<tr><td>Registration date:</td><td>' + marker.registration
191.                 date + '</td></tr>' +
192.                 '<tr><td>Changing date:</td><td>' + marker.changingdate + '</
193.                 td></tr>' +
194.                 '<tr><td>Intended use 1:</td><td>' + marker.intendeduse1 + '</
195.                 td></tr>' +
196.                 '<tr><td>Intended use 2:</td><td>' + marker.intendeduse2 +
197.                 '</td></tr>' +
198.                 '<tr><td>Intended use 3:</td><td>' + marker.intendeduse3 + '</
199.                 td></tr>' +
200.                 '<tr><td>Area:</td><td>' + marker.area + '</td></tr>' +
201.                 '<tr><td>Other area:</td><td>' + marker.otherarea + '</td><
202.                 /tr>' +

```

```

189.         '<tr><td>Land registry area:</td><td>' + marker.landregistr
yarea + '</td></tr>' +
190.         '<tr><td>Surveying date:</td><td>' + marker.surveyingdate +
'</td></tr>' +
191.         '<tr><td>Surveyor:</td><td>' + marker.surveyor + '</td></tr
>' +
192.         '<tr><td>Surveying method:</td><td>' + marker.surveyingmeth
od + '</td></tr>' +
193.         '<tr>' +
194.             '<td><b>Evaluation zone</b>:</td>' +
195.             '<td>' +
196.             '<table>' +
197.                 '<tbody>' +
198.                     '<tr><td><a href="http://www.maaamet.ee/hv/
784.pdf" target="_blank"><u><b><font color="#0000FF">H0784010</font></b></u></
a></td><td> 100% </td></tr>' +
199.                         '</tbody>' +
200.                         '</table>' +
201.                     '</td>' +
202.                 '</tr>' +
203.             '</tbody>' +
204.         '</table>';
205.         var buildingcontentstr =
206.             '<table class="dogisPopupTble">' +
207.                 '<tbody>' +
208.                     '<tr> <th style="text-align: center; background-
color: #dffddd;font-family: arial, sans-
serif;" colspan="2"> Building Info </th></tr>' +
209.                     '<tr><td>County:</td><td>Harju maakond</td></tr>' +
210.                     '<tr><td>Municipality:</td><td>' + marker.city + '</td></t
r>' +
211.                     '<tr><td>Address:</td><td>' + marker.title + '</td></tr>'
+
212.                     '<tr><td>adsoid:</td><td>' + marker.adsoid + '</td></tr>'
+
213.                     '<tr><td>Building type:</td><td>' + marker.buildingtype +
'</td></tr>' +
214.                     '<tr><td>Registration Valid Since:</td><td>' + marker.vali
dsince + '</td></tr>' +
215.                     '<tr><td>Lat, long:</td><td>' + marker.lat + ',' + marker.l
ng + '</td></tr>' +
216.                 '</tbody>' +
217.             '</table>';
218.
219.         infowindow.setContent('');
220.         // infowindow.open(map, marker);
221.         // Make sure the marker property is cleared if the info
window is closed.
222.
223.         infowindow.addListener('closeclick', function () {
224.             infowindow.setMarker = null;
225.         });
226.
227.         var streetViewService = new google.maps.StreetViewService();
228.
229.         var radius = 50;
230.         // In case the status is OK, which means the pano was
found, compute the
// position of the streetview image, then calculate
the heading, then get a
// panorama from that and set the options
231.         function getStreetView(data, status) {
232.             infowindow.setContent(buildingcontentstr);
233.
234.             if (status == google.maps.StreetViewStatus.OK) {

```



```

236.             var nearStreetViewLocation = data.location.latLng;
237.             var heading = google.maps.geometry.spherical.computeHeading(
238.                 nearStreetViewLocation, marker.position);
239.             var streewtviewstr = '<div id="pano"></div>';
240.             contentstr = contentstr + streewtviewstr;
241.             document.getElementById("infowin").innerHTML =
contentstr;
242.             var panoramaOptions = {
243.                 position: nearStreetViewLocation,
244.                 pov: {
245.                     heading: heading,
246.                     pitch: 30
247.                 }
248.             };
249.             var panorama = new google.maps.StreetViewPanora
ma(
250.                 document.getElementById('pano'), panoramaOpti
ons);
251.             } else {
252.                 var nostreetstr = '<div>' + marker.title + '</d
iv>' +
253.                     '<div>No Street View Found</div>';
254.                 document.getElementById("infowin").innerHTML =
contentstr + nostreetstr;
255.             }
256.         }
257.     }
258.
259.     }
260.     // Use streetview service to get the closest streetview ima
ge within
261.     // 50 meters of the markers position
262.     streetViewService.getPanoramaByLocation(marker.position, ra
dius, getStreetView);
263.     infowindow.open(map, marker);
264.
265.
266.     }
267.     // This function will loop through the markers array and
display them all.
268.     function showListings() {
269.         var bounds = new google.maps.LatLngBounds();
270.         // Extend the boundaries of the map for each marker and
display the marker
271.         for (var i = 0; i < markers.length; i++) {
272.             markers[i].setMap(map);
273.             bounds.extend(markers[i].position);
274.         }
275.         document.getElementById('infowin').style.display = "block";
276.
map.fitBounds(bounds);
277.     }
278.     // This function will loop through the listings and hide them all
279.
function hideListings() {
280.     for (var i = 0; i < markers.length; i++) {
281.         markers[i].setMap(null);
282.     }
283.     document.getElementById('infowin').style.display = "none";
284.
}
285.
286.
287.     </script>

```

```

288.
289.         <script async defer
290.             src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCqhc
TpfzJWATABAc0zDozTtu2dGL4DM6z4&v=3&callback=initMap">
291.         </script>
292.
293.     </body>
294. </html>

```

Appendix 2 – Source code for Mongo Database connected

```

1. @using WebMatrix.Data;
2. @using MongoDB.Driver;
3. @using MongoDB.Driver.Core;
4. @using MongoDB.Driver.GeoJsonObjectModel;
5. @using MongoDB.Bson;
6. @using System;
7. @using System.Net.Http;
8. @using System.Net;
9. @using System.Threading.Tasks;
10. @using System.IO;
11.
12. @{
13.
14.     Page.Title = "Home Page";
15.     public async Task Mainasync(string[] args) {
16.         var connectionstring = "mongodb://localhost:27017";
17.         var client = new MongoClient(connectionstring);
18.         var db = client.GetDatabase("test");
19.         var col = db.GetCollection<BsonDocument>("buildingInfo");
20.
21.         using (var cursor = await col.Find(new BsonDocument()).ToCursorAsync()
22. )
23.         {
24.             while (await cursor.MoveNextAsync())
25.             {
26.                 var tr = 0; var locations = [];
27.                 @foreach (var item in col.Find(_=>true).ToListAsync().Result)
28.                 {
29.                     <Text>
30.                     locations.push({
31.                         title: "@item["ADDRESS"].AsString",
32.                         location: {
33.                             lat: parseFloat( "@item["LATITUDE"].AsString"),
34.                             lng: parseFloat( "@item["LONGITUDE"].AsString")
35.                         },
36.                         city: "@item["CITY"].AsString",
37.                         district: "@item["DISTRICTNAME"].AsString",
38.                         parcelid: "@item["PARCELID"].AsString",
39.                         registrationdate: "@item["REGISTRATIONDATE"].AsString",
40.                         changingdate: "@item["CHANGINGDATE"].AsString",
41.                         intendeduse1: "@item["INTENDEDUSE1"].AsString",
42.                         intendeduse2: "@item["INTENDEDUSE2"].AsString",
43.                         intendeduse3: "@item["INTENDEDUSE3"].AsString",
44.                         area: "@item["AREA"].AsString",
45.                         otherarea: "@item["OTHERAREA"].AsString",
46.                         landregistryarea: "@item["LANDREGISTRYAREA"].AsString",
47.                         surveyingdate: "@item["SURVEYINGDATE"].AsString",

```

```

48.         surveyor:"@item["SURVEYOR"].AsString",
49.         surveyingmethod:"@item["SURVEYINGMETHOD"].AsString",
50.         evaluationzone:"@item["EVALUATIONZONE"].AsString",
51.         buildingtype:"@item["BUILDINGTYPE"].AsString",
52.         adsoid: "@item["ADSOID"].AsString",
53.         validsince:"@item["VALIDSINCE"].AsString"
54.     })
55.
56.     </Text>
57.     }
58. }
59.
60.
61. }
62.
63. }
64.
65.
66. }
67.
68.
69. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
70. <meta charset="utf-8">
71. <html>
72.
73. <head>
74.     <!--<script src="JavaScript.js"></script>-->
75.     <link rel="stylesheet" type="text/css" href="MapStyle.css">
76.
77. </head>
78. <body>
79.     <div class="container">
80.         <div class="options-box">
81.             <h1>District buildings info map </h1>
82.             <div>
83.                 <input id="show-listings" type="button" value="Show info">
84.                 <input id="hide-listings" type="button" value="Hide info">
85.             </div>
86.             <div id="infowin">
87.
88.             </div>
89.
90.         </div>
91.         <div id="map"></div>
92.     </div>
93.
94.     <script>
95.
96.         var map;
97.         // Create a new blank array for all the listing markers.
98.         var markers = [];
99.         function initMap() {
100.             // Constructor creates a new map -
only center and zoom are required.
101.             map = new google.maps.Map(document.getElementById('map'), {
102.
103.                 center: { lat: 59.396956, lng: 24.670968 },
104.                 zoom: 15,
105.             });
106.
107.             var largeInfowindow = new google.maps.InfoWindow();
108.
109.             // The following group uses the location array to create an
array of markers on initialize.
110.             for (var i = 0; i < locations.length; i++) {

```

```

111.         // Get the position from the location array.
112.         var position = locations[i].location;
113.         var title = locations[i].title;
114.         var year = locations[i].year;
115.         var x = locations[i].location.lat;
116.         var y = locations[i].location.lng;
117.         var city=locations[i].city;
118.         var district = locations[i].district;
119.         var image = "/Images/Home.PNG";
120.         var parcelid = locations[i].parcelid;
121.         var registrationdate = locations[i].registrationdate;
122.         var changingdate = locations[i].changingdate;
123.         var intendeduse1 = locations[i].intendeduse1;
124.         var intendeduse2 = locations[i].intendeduse2;
125.         var intendeduse3 = locations[i].intendeduse3;
126.         var area = locations[i].area;
127.         var otherarea = locations[i].otherarea;
128.         var landregistryarea = locations[i].landregistryarea;
129.         var surveyingdate = locations[i].surveyingdate;
130.         var surveyor = locations[i].surveyor;
131.         var surveyingmethod = locations[i].surveyingmethod;
132.         var evaluationzone = locations[i].evaluationzone;
133.         var buildingtype = locations[i].buildingtype;
134.         var adsoid = locations[i].adsoid;
135.         var validsince = locations[i].validsince
136.
137.
138.
139.
140.
141.         // Create a marker per location, and put into markers
142.         // array.
143.         var marker = new google.maps.Marker({
144.             map: map,
145.             icon: image,
146.             position: position,
147.             title: title,
148.             animation: google.maps.Animation.DROP,
149.             id: i,
150.             lng: y,
151.             lat: x,
152.             city:city,
153.             district: district,
154.             parcelid: parcelid,
155.             registrationdate: registrationdate,
156.             changingdate:changingdate,
157.             intendeduse1: intendeduse1,
158.             intendeduse2: intendeduse2,
159.             intendeduse3: intendeduse3,
160.             area: area,
161.             otherarea: otherarea,
162.             landregistryarea:landregistryarea,
163.             surveyingdate:surveyingdate,
164.             surveyor:surveyor,
165.             surveyingmethod:surveyingmethod,
166.             evaluationzone: evaluationzone,
167.             buildingtype: buildingtype,
168.             adsoid: adsoid,
169.             validsince:validsince
170.
171.         });
172.         // Push the marker to our array of markers.
173.         markers.push(marker);
174.         // Create an onclick event to open an infowindow at
175.         // each marker.

```

```

175.             marker.addListener('click', function () {
176.                 populateInfoWindow(this, largeInfowindow);
177.             });
178.         }
179.         document.getElementById('showlistings').addEventListener('click', showL
istings);
180.         document.getElementById('hide-
listings').addEventListener('click', hideListings);
181.     }
182.     // This function populates the infowindow when the marker is
clicked. We'll only allow
183.     // one infowindow which will open at the marker that is clicked
, and populate based
184.     // on that markers position.
185.     function populateInfoWindow(marker, infowindow) {
186.         // Check to make sure the infowindow is not already opened
on this marker.
187.         if (infowindow.marker != marker) {
188.             infowindow.marker = marker;
189.             var contentstr = '<div>' +
190.                 '<h1 id="firstHeading" class="firstHeading">' + marker.title + '</h1>'
+
191.                 '</div>' +
192.                 '<table class="dogisPopupTble">' +
193.                 '<tbody>' +
194.                 '<tr><th style="text-align: center; background-color: #dffddd;font-
family: arial, sans-serif;" colspan="2"> Parcel Info </th></tr>' +
195.                 '<tr><td>County:</td><td>Harju maakond</td></tr>' +
196.                 '<tr><td>Municipality:</td><td>' + marker.city + '</td></tr>'
+
197.                 '<tr><td>Settlement unit:</td><td>' + marker.district + '</
td></tr>' +
198.                 '<tr><td>Identifier:</td><td>' + marker.parcelid + '</td></
tr>' +
199.                 '<tr><td>Registration date:</td><td>' + marker.registration
date + '</td></tr>' +
200.                 '<tr><td>Changing date:</td><td>' + marker.changingdate + '
</td></tr>' +
201.                 '<tr><td>Intended use 1:</td><td>' + marker.intendeduse1 + '</
td></tr>' +
202.                 '<tr><td>Intended use 2:</td><td>' + marker.intendeduse2 + '
</td></tr>' +
203.                 '<tr><td>Intended use 3:</td><td>' + marker.intendeduse3 + '
</td></tr>' +
204.                 '<tr><td>Area:</td><td>' + marker.area + '</td></tr>' +
205.                 '<tr><td>Other area:</td><td>' + marker.otherarea + '</td><
/tr>' +
206.                 '<tr><td>Land registry area:</td><td>' + marker.landregistr
yarea + '</td></tr>' +
207.                 '<tr><td>Surveying date:</td><td>' + marker.surveyingdate + '
</td></tr>' +
208.                 '<tr><td>Surveyor:</td><td>' + marker.surveyor + '</td></tr
>' +
209.                 '<tr><td>Surveying method:</td><td>' + marker.surveyingmeth
od + '</td></tr>' +
210.                 '<tr>' +
211.                 '<td><b>Evaluation zone</b>:</td>' +
212.                 '<td>' +
213.                 '<table>' +
214.                 '<tbody>' +
215.                 '<tr><td><a href="http://www.maaamet.ee/hv/
784.pdf" target="_blank"><u><b><font color="#0000FF">H0784010</font></b></u></
a></td><td> 100% </td></tr>' +
216.                 '</tbody>' +
217.                 '</table>' +
218.                 '</td>' +

```

```

219.         '</tr>' +
220.         '</tbody>' +
221.         '</table>';
222.         var buildingcontentstr =
223.             '<table class="dogisPopupTble">' +
224.             '<tbody>' +
225.             '<tr> <th style="text-align: center; background-
color: #dffddd;font-family: arial, sans-
serif;" colspan="2"> Building Info </th></tr>' +
226.             '<tr><td>County:</td><td>Harju maakond</td></tr>' +
227.             '<tr><td>Municipality:</td><td>' + marker.city +
228.             '</td></tr>' +
229.             '<tr><td>Address:</td><td>' + marker.title + '</td></tr>'
+
230.             '<tr><td>adsoid:</td><td>' + marker.adsoid + '</td></tr>'
+
231.             '<tr><td>Building type:</td><td>' + marker.buildingtype +
232.             '</td></tr>' +
233.             '<tr><td>Registration Valid Since:</td><td>' +
marker.validsince + '</td></tr>' +
234.             '<tr><td>Lat, long:</td><td>' + marker.lat + ',' +
marker.lng + '</td></tr>' +
235.             '</tbody>' +
236.             '</table>';
237.         infowindow.setContent('');
238.         // infowindow.open(map, marker);
239.         // Make sure the marker property is cleared if the info
window is closed.
240.         infowindow.addListener('closeclick', function () {
241.             infowindow.setMarker = null;
242.         });
243.
244.         var streetViewService = new google.maps.StreetViewService();
245.
246.         var radius = 50;
247.         // In case the status is OK, which means the pano was
found, compute the
248.         // position of the streetview image, then calculate the
heading, then get a
249.         // panorama from that and set the options
250.         function getStreetView(data, status) {
251.             infowindow.setContent(buildingcontentstr);
252.             if (status == google.maps.StreetViewStatus.OK) {
253.                 var nearStreetViewLocation = data.location.latL
ng;
254.                 var heading = google.maps.geometry.spherical.co
mputeHeading(
255.                     nearStreetViewLocation, marker.position);
256.                 var streewtviewstr = '<div id="pano"></div>';
257.                 contentstr = streewtviewstr + contentstr ;
258.                 document.getElementById("infowin").innerHTML =
contentstr;
259.                 var panoramaOptions = {
260.                     position: nearStreetViewLocation,
261.                     pov: {
262.                         heading: heading,
263.                         pitch: 30
264.                     }
265.                 };
266.                 var panorama = new google.maps.StreetViewPanora
ma(
267.                     document.getElementById('pano'), panoramaOpti
ons);

```

```

268.             } else {
269.                 var nostreetstr = '<div>' + marker.title + '</div>' +
270.                 '<div>No Street View Found</div>';
271.                 document.getElementById("infowin").innerHTML =
272.                 nostreetstr + contentstr;
273.             }
274.         }
275.     }
276.     // Use streetview service to get the closest streetview
277.     // image within
278.     // 50 meters of the markers position
279.     streetViewService.getPanoramaByLocation(marker.position, radius, getStreetView);
280.     infowindow.open(map, marker);
281.
282.
283.     }
284.     // This function will loop through the markers array and
285.     // display them all.
286.     function showListings() {
287.         var bounds = new google.maps.LatLngBounds();
288.         // Extend the boundaries of the map for each marker and
289.         // display the marker
290.         for (var i = 0; i < markers.length; i++) {
291.             markers[i].setMap(map);
292.             bounds.extend(markers[i].position);
293.         }
294.         document.getElementById('infowin').style.display = "block";
295.         map.fitBounds(bounds);
296.     }
297.     // This function will loop through the listings and hide
298.     // them all.
299.     function hideListings() {
300.         for (var i = 0; i < markers.length; i++) {
301.             markers[i].setMap(null);
302.         }
303.         document.getElementById('infowin').style.display = "none";
304.     }
305.
306.     </script>
307.     <script async defer
308.         src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCqhc
309.         TpzfJWATABAc0zDozTtu2dGL4DM6z4&v=3&callback=initMap">
310.     </script>
311. </body>
</html>

```