Student thesis series INES nr 446

Constructing and developing an integrated smart spatial database for Malmö Municipality, Case study: Västra innerstaden in Malmö

Sayed Hassan Alavi

2016
Department of
Physical Geography and Ecosystem Science
Lund University
Sölvegatan 12
S-223 62 Lund
Sweden

Sayed Hassan Alavi (2018).
Constructing and developing an integrated smart spatial database for Malmö Municipality, Case study: Västra innerstaden in Malmö
Master degree thesis, 30 credits in *Geomatics*
Department of Physical Geography and Ecosystem Science, Lund University

Level: Master of Science (MSc)

Course duration: *February* 2017 until *June* 2017

Disclaimer

Constructing and developing an integrated smart spatial database for Malmö Municipality, Case study: Västra innerstaden in Malmö

Sayed Hassan Alavi

Master thesis, 30 credits, in *Geomatics*

Supervisor: Ali Mansourian

Exam committee:
Lars Harrie
Andreas Persson

Department of Physical Geography and Ecosystem Science
Lund University, Sweden

Summary

GIS Section of Street and Park department of Malmö Municipality continuously receives new spatial data from different contractors that need to be saved to the GIS section traffic database. Meanwhile, according to interviews with GIS Section staff, due to spatial relationships and daily updates, it is evident that the traffic database requires more consistency and integration along with less redundancy. Since there is a spatial and conceptual relationship between the data, making a change in the traffic database and failing to make appropriate changes to other related features will eliminate the integrity of the data and the logical relationship of the information in the traffic database.

The traffic database of GIS Section must be able to automatically provide consistency, integrity, assigning attributes to various objects. This thesis aims at providing a solution for making the traffic database smart, so that by changing a feature, other features are automatically updated and the traffic database integrity is maintained.

In the first step, interviews were performed with street and park department staff, in particular geographical database engineers of GIS Section, in order to define the needs of the traffic database. During these interviews, some of the important needs of the GIS Section traffic database were discussed. These requirements were approved as a set of essential rules in the GIS Section traffic database. Then a conceptual model of the traffic database was designed. In this step, all object classes were specified and the attributes of each class were determined. The relationship between the objects and the degree of these relationships were also identified. Then the logical model and physical implementation of the traffic database was completed.

Triggers, which are written using the SQL programming language for the traffic database, are used to apply the essential rules to the traffic database to make the traffic database smart. Using triggers, one can run a set of SQL commands with every change in the traffic database, which increase the consistency, integrity and attribution of the quantities automatically in the traffic database. However, using triggers has some disadvantages. The most important disadvantage of trigger's use includes the loop of SQL commands due to the infinite activation of triggers. If all the defined triggers are created into the traffic database to enforce the essential rules, any change in one of the tables will activate the trigger of that table, which in turn will cause another table to be updated and its trigger will be activated; this will eventually lead to the re-update of the first table and re-activate its trigger. If this process starts, there is no end to it and the triggers will continuously activate each other.

To get rid of the problem of unlimited activation of triggers in the traffic database, the iTRIMAN user interface is designed and implemented in a desktop and a web version. iTRIMAN allows triggers to be created on the traffic database when needed and to be dropped when they are not needed. By using this method, there will only be triggers in the traffic database whose associated tables are being edited.

The tests show that using the proposed approach the traffic database will be a consistent, integrated and automated traffic database that will keep consistency and integrity of data in a better way, which will eventually save time and money for the GIS section when updating the traffic database.

Acknowledgment

I would like to express my gratitude to my supervisor Dr. Ali Mansourian for his excellent guidance and sympathetic support through this research.

I would like to express my gratitude to Joel Noren, Mozafar Veysipanah and Raymond Timlin in Malmö Municipality for all of support and advices through this research.

I would like to express my gratitude and thanks to all of my colleagues in SWECO Position, especially to Karin Neland for all of understandings, support and kindness during this research.

I would like to express my deepest sense of gratitude and love to my family, for all encouragement, support and love during this study.

Table of Contents

1 Introduction

The GIS Section of the Streets and Parks Department (Gatukontoret) of Malmö Municipality is responsible for providing, storing, and maintaining of geographical database of streets and parks department of Malmö Municipality. GIS section is also responsible for providing thematic maps, spatial analyses, and traffic plans so as to maintain an urban planning strategy. Currently, all of such tasks are performed in the PostgreSQL and Oracle databases and are displayed by the internal Web GIS. It has been decided to gradually move all the data from the Oracle database to the PostgreSQL database; therefore, the foundation should be laid for an efficient database in PostgreSQL.

Since GIS Section receives spatial data from different contractors continuously, it is necessary to enter the data into the system constantly, something which always requires the database to be changed. Data can be entered into the database directly by a user. They can also be derived from other data of the database. Sometimes, a datum might be redundant by being stored in several locations. Furthermore, the conditions may not allow the reduction of redundancy, and the database will need to be changed in certain points after new data are received. Therefore, a user must manually change and store all of the available data in certain points. It is very time-consuming to edit, compute, and enter the new data into certain locations manually. Such tasks are also error prone. Therefore, an automated method is required for computing and updating values and attributes after a change is made to the database.

The following example can be used to better understand the problem: A bus station with *XA* attribute is located inside a polygon named *A*. The attribute or feature *XA* is extracted from *A* to be allocated to the bus station. In other words, the bus station possesses *XA* only if it is located inside *A*. This attribute is obtained from the geometric computation of the two aforesaid objects. Currently, if the municipality moves the bus station from *A* to another polygon named *B*, the attribute still remains *XA*; however, it needs to be change to *XB* in the new position. Lack of such update results in a kind of disintegration and inconsistency in the database. If it is not corrected, the database stores inaccurate data and consequently the results of queries and analyses are not reliable. The manual correction is time-consuming and may lead to miscomputation. Therefore, it is essential to devise an automatic updating mechanism, by which the bus station attribute gets updated automatically from *XA* to *XB* whenever the bus station moves from *A* to *B*. If so, the database is automatically provided with consistency, integrity, and cohesion.

The database produced by this study is characterized by higher levels of consistency and integrity in addition to a lower level of redundancy. It also contains cohesive and relevant data which are sometimes derived from other data and may often need computation. Most importantly, everything is computed and updated automatically. In other words, the database is able to update itself automatically after every change.

1.1 Statement of the Problem

It is necessary to accurately implement all the components of a database in accordance to comprehensive requirement specifications in order to design a good, efficient, and smart database. For instance, the database redundancy should be minimized. In case of persistent redundancy, the database should be consistent. In other words, the values of data should be the same in different locations. All of the values should be updated simultaneously after a change is made to the database. The data are usually related in a database, something which means a data can be computed or derived from another datum. Such a computation should be performed accurately after every change in the database. Above all, the database should maintain integrity.

The traffic database of GIS Section is changing constantly: data are added to it, data are deleted from it, or data are edited. Therefore, it is very hard and time-consuming to meet the aforesaid requirements manually. There might be computational and typographical errors, too. Thus, it is necessary to devise a mechanism, which update table/tables automatically.

There are different techniques to create a smart database. First, GIS section wants to know these techniques. Then, they want to know which technique is best according to existing knowledge in GIS section, resources, and possibilities. Since most of the GIS personals of street and park department are GIS users (except DB engineers), it is needed to introduce, present and explain advantages and challenges of selected technique.

1.2 Objectives

The main aim of this thesis is to design and develop a prototype of a smart traffic database for GIS Section of the Streets and Parks Department of Malmö Municipality in order to:

- Define GIS section requirements for the traffic database.
- Understand different techniques for implementing a smart database and selecting one for implementation based on possibilities and capabilities of GIS section.
- Developing a smart traffic database based on the selected technique.
- Evaluating the implementation of selected technique against requirement analysis.

1.3 Structure

This thesis is organized in six chapters, the first of which is the Introduction including sections named Statement of Problem, Objectives, and Structure. The second chapter deals with a review of literature in which related types of databases were analyzed in addition to smart databases. The third chapter describes the methodology for designing a smart traffic database. First, study area is described. Then geographical traffic database is designed in four phases: requirement analysis, conceptual model design, logical model, and physical implementation. Then the rules of requirement analysis are discussed. After requirement analysis selected approach is described. Finally, the iTRIMAN user interface is described. In the fourth chapter, the research results are analyzed. The fifth chapter deals with the challenges of this thesis. Then, some useful suggestions are made for the current or future use of this traffic database. Conclusions are in the sixth chapter.

2 Literature Review

In this chapter, first different related kinds of databases and spatial databases are described then smart database background are studied and finally database design steps are explained.

2.1 Databases

A database is an organized collection of data that can be easily accessed, managed and updated. Each database is connected to a database management system, which is a collection of programs that can manage, select, add, update, remove and change database content.

There are different types of databases, which are related to this study, including relational databases, object-oriented databases and object relational databases.

2.1.1. The Relational Database (RDB): The relational database was introduced in 1970. In this model, data are saved in the form of related tables. Each table is made up of columns and rows, and row represents the data value of an object, while each column represents the attributes of an object. Then these tables connected together by primary key and foreign keys (Ranjana, et al. 2015). In the relational database, are integrity and normalization concepts. Integrity means to be sure about accuracy and consistency of data, which is an important aspect to the design and implementation as attested by many publications (Decker, et al. 2008; Bernstein et al, 1982; Bernstein et al, 1980; Eswaran et al, 1975). Another word, data integrity, guarantees that data will saved exactly as intended. Integrity can be divided into physical integrity and logical integrity. One important part of integrity is consistency in database. Consistency means any updates and changes in the database must follow defined rules. It means data only can change to the permitted ways. By using normalization, a consistent and healthy database with minimum redundancy can be achieved. One important current product related to relational database is Informix (Praveen, et al. 2017).

2.1.2. Object-Oriented Database (OODB): Object-oriented databases emerged in the 1980s to overcome the problems and limitations of relational databases in managing large and complex data.

The first reason for creating object-oriented database is supporting complex structure for stored objects. An important feature of this kind of database is giving power to the designer to decide about complex objects structure and applied operations (behavior) to these objects. In this model, data is represented by object and object class. “Object” in database is real entity in the world. “Object Class” is a collection of similar objects with the same type of values and methods. Values are attributes of an object. Methods are object functions, which describe the object behavior. A unique method signature identifies each method usually by a method name, method type, method number and other parameters.

Second reason for creating of object-oriented database is supporting the concept of “object-oriented” for programming, ideas like referencing, encapsulation, inheritance and polymorphism. These four components called standard object oriented (Hardy, 2001):

- Referencing: implies that objects contain data from related items (e.g. a spring is a wellspring of a waterway).

- Encapsulation: implies that information and conduct are typified together inside articles, which react to messages sent to them.
- Inheritance: means that protest classes can inherit traits from different superclasses (e.g. a congregation is both an open building and a man-made structure).
- Polymorphism: implies that diverse protest classes answer to a similar message using legitimate conduct (e.g. by tapping on a waterway and a building can obtain a distinctive sort of feature and portrayal).

2.1.3. Object Relational Database (ORDB): Object relational databases came about in 1985 to fill a large gap between object-oriented databases and relational databases. An object relational database has the advantages of both these types of database. It combines advantage of object-oriented concept of programming and relational database. It means object relational database benefit from both methods and datatype in same time. This model benefits from simultaneous referencing, encapsulation, inheritance, and polymorphism as in relational databases. An object relational database is good for organizing complex data. Examples of this type database are IBM DB2, PostgreSQL, Oracle and Illustra (Praveen, et al. 2017).

2.2 Spatial Databases

A spatial database is a database that stores and queries data that represent spatial objects. Spatial databases represent geometric objects like point, line, and polygon. Even some complex structures, like topological coverages, linear network, 3D objects, and TIN, are supported by some spatial databases. Spatial databases use spatial indices to speed up spatial operations. Examples of spatial operations are spatial measurement, spatial function, and spatial predicates, and examples of spatial indices are Quadtree, R-tree, and KD tree.

The relational spatial database has limitations for handling geometric data. This database only store singular value, which is not good for saving geometries (in this database geometries must be saved in several columns). The other limitation is related to SQL, which is a declarative language. However, by adding a middleware software for handling the geometry functionality in relational database can overcome the limitations of the relational database.

The object-oriented spatial database store real world as objects, which can represent them as point, line and polygon. Storing attributes and geometry of objects in one layer can provide better representation results. “in-house query language” is the interface for object oriented database. SmallWorld is an example of this type. Object oriented database schema contain definitions of all of object types which can define relationships between objects and object behavior.

In the object relational spatial database, geometry functionality is added directly into the database. This type of database is based on abstract data type (ADT) or user defined type (UDT). These two (ADT and UDT) extends the SQL type system and are related to a data type. Object relational database integrate attributes and geometries within the database, which are supported by spatial data queries. Oracle spatial and PostgreSQL with PostGIS extention are example of this type of database.

2.3 Smart Database Theoretical Background

A smart database as is defined in this thesis is a type of database that can keep the consistency and integrity of the database in an acceptable level. In this section, three different techniques used for creating smart databases such as ontology and semantic in database, active database and deductive database. An example database has also been introduced for each technique.

2.3.1 Ontology and Semantics in Databases

Ontology, meaning "conception of a thing," can define a specific vocabulary in a related domain. Using ontology, one can define objects, properties, and relationships between parts. Ontology is used extensively in Artificial Intelligence in the computing domain. By using ontology, one tries to define the structure of reality, which is close to the concept (Guarino, 1998).

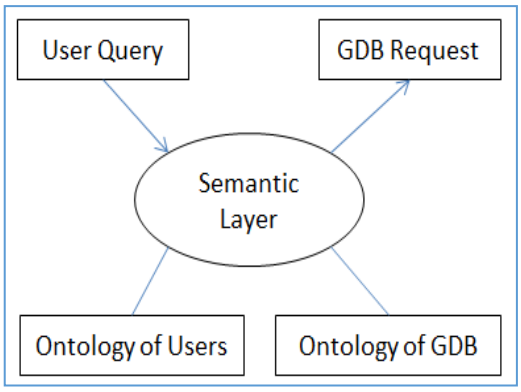

Figure 1. Structure of a system which contains ontologies and semantic layer (Viegas, et al. 2007).

Geographic ontology is the conceptualization of a geographic object that exists in the real world. It is used to find all the properties and characteristics belonging to a geographic phenomenon in order to conceptualize the geographic object well. In addition to other properties and characteristics, geographic properties like directions, coordinates, topologies, and temporal qualities are also included in geographic ontology (Viegas, et al. 2007).

The semantic layer is an intermediate layer that links or translates a user's query to the geographical database by using ontologies and returns data to the user.

The structure for a system that contains both a semantic layer and ontologies is as follows: the system gets a query from user and passes it to the semantic layer, which contains both user ontology and geodatabase ontology, and can translate user ontology to geodatabase ontology. The semantic layer then sends a request based on the geodatabase ontology to the geodatabase, and finally returns the results back to the user. Figure 1 shows the structure of the semantic layer in relation to ontologies. One example of ontology and a semantic related database, which works smartly, is the Sesame semantic database.

For more about geographic ontology and semantics, refer to Elmasri (2011), Ramez (2011), Guarino, (1998), Fonseca (2001), Egenhofer et al. (2000), and Viegas et al. (2007).

Sesame:

Sesame is a framework for analyzing and querying RDF (Resource Description framework) data. Sesame created by a Dutch software company and was part of semantic web project between 1999 and 2002. Sesame packages contain an API based on Java parsers and writers distributed along with Sesame.

Geographical resource-based and text-based search functions are added using an extension library called "USeekM" to semantic databases, which are compatible with

Sesame's API. Using the USeekM module, text data is indexed using the inverted index method and spatial data is indexed using the R-Tree or Quadtree based methods. This module uses PostgreSQL or ElasticSearch to create indexes and supports geometries (e.g. point, line, and polygon) and functions such as overlaps, within, and intersects according to the OGC GeoSPARQL standard. USeekM module is an open-source library.

2.3.2 Active Databases

Active databases can react naturally to an event that occurs either inside or outside of the database. Each time an event is recognized by the database, it checks the condition and then fires the trigger and runs the specified function.

Active databases depend on ECA (Event, Condition and Action):

- **Event**: events are operations like alter, insert, delete, or add within the database. These events fire a trigger. The occurrence of one of these activities fires a trigger that runs relevant functions.
- **Condition**: when a trigger is fired in the condition proviso, one can check the type of activity in the condition.
- **Action**: this activity results from some portion of a trigger. Ordinarily, a trigger in the activity part places significant demands on the capacity of the database

An active database requires a tool to monitor and control all triggers in the database and make sure they work correctly. Otherwise, it may result in two or more rules triggering each other, leading to non-termination or an indefinite circular triggering (Elmasri, 2011).

One example of an active database that works smartly is the Gothic database, a product from Laser-Scan company, nowadays called 1Spatial. For more about active databases, refer to Elmasri 2011.

2.3.2.1 Gothic Database (Object-Oriented Example):

Laser-Scan was the leading company in designing active databases. Founded in 1969 in Cambridge, it produced world-leading software for digital mapping, computer graphics, and geographic information handling. The Gothic object-oriented spatial database and its toolkit were among Laser-Scan's first products. The Gothic active OO database comprised two parts (Active and OO) with the following meanings and characteristics.

An object-oriented paradigm is a programming paradigm supporting the conception of "objects," which can contain knowledge within each type of field, usually referred to as attributes, and code, within each type of procedure, usually referred to as ways. It is characterized by four key idea referencing, encapsulation, inheritance and polymorphism (Hardy, 2001).

The active spatial database mapping system in a Gothic database has the following characteristics and properties (Hardy, 2001):

1. It is allowed real-world modeling and a relationship through schema that makes the action performed by the active database seem more realistic (Object data model).

2. Spatial objects are retrieved and efficient storage (an object database) is provided.
3. A map of the area of interest is provided with the requested scale (seamless mapping).
4. Connectivity and adjacency are known for both active databases and spatial data bases (topological structure).
5. Proper cartography is provided by active representation.
6. A proper map feature for the current scale is derived by active object generalization methods, which also tend to make the final representation realistic and perfectly proportional to the input.
7. Data integrity is ensured by validation methods.

Another key concept of the object-oriented database is methods, which are defined for objects. By writing methods on object classes for the Gothic object-oriented database, there are possibilities for creating smart/active database. These methods are related to behaviors. By sending a message to the object, methods will be invoked and then behavior will be executed. Defining behavior as part of the database instead of the application is a fundamental principle of the object-oriented database.

Methods are central to the object-oriented database. In the Gothic database, object classes can have both methods and behaviors. It can also inherit methods from its parent class. There are several type of methods in Gothic database, such as:

1. *Value methods:* the results of these methods are attributes like area, description, and length.
2. *Agent methods:* the result of these methods is intelligent generalization, which allows objects to think for themselves. One example of agent-based generalization in combination with other generalization techniques is providing a production flow line for generalization.
3. *Validation reflex methods:* these methods allow users to apply their own rules/logics to each object class, after which the database then ensures the method's integrity by enforcing this logic when objects are entered (e.g. when digitizing counter lines, if they cross each other, it informs user by sending an error message).
4. *Process methods:* these methods do data checking, data cleaning, and polygon formation on a defined set of objects. These methods will occur at the request of the operator.
5. *Reflex methods:* these methods occur automatically in the creation, deletion or modification of an object.
6. *Display methods:* these methods are used to provide active representation. They can produce the best cartographic visualization for maps. For example when labeling highways on a map, it can set the color of the highways' labels. For existing bridges, it can change the direction of a bridge to match the direction of the highway (Hardy, 2000).

In conclusion, active databases improve the accessibility of the database and can react to different transactions carried out by users. The software can relate well with to the rest of the applications in real time.

The Laser-Scan company produced IGIS, LAMPS2, and VTrack (not existed nowadays) based on smart databases. These are automatic software programs using smart databases designed especially for LAMPS2.

In 2007, after 38 years, the Laser-Scan Company changed its name to 1Spatial. For more about Gothic databases, refer to Hardy 2000 and Hardy 2001.

2.3.2.2 Triggers in Relational Database:

Triggers can create an automatic database. Each trigger can be activated by an event in associated table and update another table/tables based on its function.

Trigger (similar to reflexes in Gothic) is an automatically fired function associated with a table, triggered by the occurrence of any event, such as insert, update, or delete in the table, the trigger will fire and run the attached function on the relevant table. See Figure 2.

Function, also known as stored procedure, carries out the operations, which can include several queries within one function. A function can be created in a language like SQL, Python, PL/pgSQL, C, etc. See Figure 3.

SQL (Structured Query Language) is a query language, used for manipulating and accessing databases.

For more studies regarding trigger and function, refer to the PostgreSQL documentation at www.postgresql.com.

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
ON table_name
[ FROM referenced_table_name ]
[ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } ]
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name (arguments)
```

Figure 2. *CREATE TRIGGER: create a trigger, run a specified function before or after certain events. Events are operations like insert, edit, delete, or update within a database. The trigger can execute a function once for each row or for any given operation.*

```
CREATE [OR REPLACE] FUNCTION function_name (arguments)
RETURNS return_datatype AS $variable_name$
DECLARE declaration;
BEGIN
< function_body >
RETURN { variable_name | value }
END; LANGUAGE plpgsql;
```

Figure 3. *CREATE FUNCTION: Create or replace a new function, which is written in a specified language like SQL, C or Python. All of the commands take place within the function body.*

2.3.3 Deductive Database

When artificial intelligence and databases come together, they create a deductive database. The two main components of deductive databases are facts and rules. In a deductive database, rules are specified through declarative language. A deduction mechanism can deduct new facts by interpreting current facts and rules. A deductive database is related to the PROLOG language and logic programming. Rules use declarative language like PROLOG or DATALOG (a variation of PROLOG).

Facts are specified in such a way that relations are specified. Rules are virtual relations. There are two main steps for creating each deductive database. The first step is storing facts and rules in the database and the second step is the interpretation step, which interprets new facts based on current rules and facts.

One example of a deductive database that works smartly is spatial Yap.

For more information, refer to Elmasri 2011, Kumar et al. 2015, and Lausen et al. 1997.

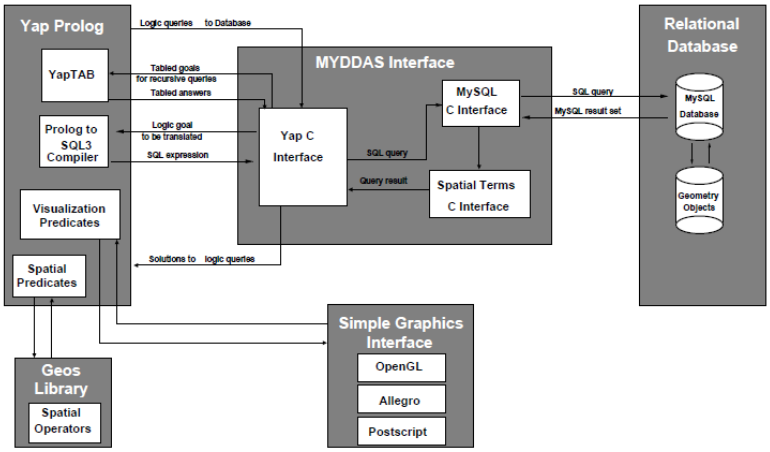

Figure 4. Structure of MYDDAS interface.

Spatial Yap:
Spatial Yap is a spatial deductive database system that is the result of several components. The Yap PROLOG system and MySQL RDBMS are the two main components of Spatial Yap. The other component which couples with these two parts is MYDDAS (Mysql/Yap Deductive Database System).

MYDDAS translates logic queries to an SQL statement and converts MySQL attributes to Yap terms. It is supported by MySQL Geometry types in the MYDDAS interface to build the spatial deductive database system. Two fundamental components for Spatial Yap are visualization component and spatial operators library. Visualization component graphically represent spatial data (Figure 4). Spatial operators library is used to extend the MYDDAS interface to support spatial terms (Vaz, et al. 2007).

2.4 Database Design

Database design includes requirement analysis, conceptual model, logical model and physical implementation.

2.4.1 Requirement Analysis

A geographical database should be able to model information from the real world. Geographical or spatial databases have different users, providers, and applications. The most critical step in database creation is requirement analysis; all other steps when depend directly on it. When the quality of this step is good, a good outcome for the database can be

expected. In other words, a good view of the real world can result in a good model made from it (Tveite, 1997).

Requirement analysis as a first step is conducted based on interviews with those who work with the database. During the interviews, one must know the query, search, and analysis they need to do on the database, as well as the kind of data processing they are doing, which functionality they are using, and what operation they are performing on the database (Mansourian, Harrie, 2012). Generally, one must know about relationships, linkages, and rules in the database.

2.4.2 Conceptual Model using UML

The second step in database design is forming the conceptual model, which is based on the required specification from the requirement analysis step. In the conceptual modeling phase, one must define the object class, relationships, cardinality, etc. The most important reason for modeling is to better understanding the system being developed. The conceptual model is a small model of the real world and it illustrates a model for the database. In other words, the model is a simplification of reality (Grady, et al. 2005).

UML (Unified Modeling Language) is used to construct, specify, visualize, and document systems. UML is used for object-oriented modeling and it is a very expressive language that is easy both to understand and to use. To form a conceptual model, one must have the elements of the UML model.

2.3.2.1 Object and Object Class

The first part of the conceptual model comprises objects in the UML model. Objects are entities in the model. Objects are independent in the real world, and object in the model represent the feature/entities in real world.

Each object in the model has the following properties: ID, state, and behavior. **ID** is a unique number for each object that enables the identification of an object in the database. **State** is information about the object, like attributes and property values, in the database. The **behavior** of an object is controlled by methods. Each object may have different behaviors based on its attributes. Each object belongs to only one object class. The object class contains a set of objects with the same attributes and methods (Elmasri, 2011).

2.3.2.2 Relationships

In the UML, object classes connect and form relationships, which are physically or logically available in the UML model. The most important relationships in object-oriented modeling are associations, dependencies, and generalization.

Association: this is a simple relationship between two object classes. It is *described by a verb*. A line with an arrow at its end shows

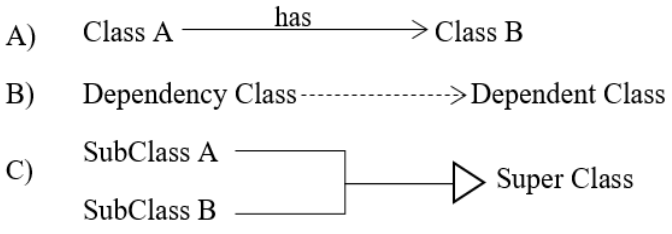

Figure 5. Shows association (A), dependency (B) and generalization (C).

this relationship in the association relationship in the UML model (Figure 5. A).

Dependency: this relationship exists between two objects, one of which uses information from another object. A dashed line with an arrow at its end shows this relationship while pointing from a dependent element (the client) to the independent element (Figure 5. B).

Generalization: this relationship relates several specific objects to one general object. Generalization can be described by an "*is-a-kind-of*" relationship. For example, commercial building is a kind of building. A line with a triangle at its end shows this relationship (Figure 5. C).

2.3.2.3 Cardinality (multiplicity)

Cardinality ratios are integer numbers with minimum and maximum ranges that show the range of related objects in the association relationships. The important thing in multiplicity is the number of objects related to each other. This number shows the cardinality between two objects.

2.4.3 Logical Model and Physical Implementation

In the **logical model** (internal model), a database schema is created. Logical model is used for relational database or object relational database. To create a database schema, conceptual models are translated to the database schema in 7 steps:

- The *first step* is to create tables for each object in the database.
- The *second step* is to find related attributes for each object.
- The *third step* is to have a unique identifier (ID) for each object of table as the primary key.
- The *fourth step* is to add a table for each many-to-many relationship in the database.
- The *fifth step* is to add both of the primary keys as attributes in the many-to-many table.
- The *sixth step* is to find the association for each one-to-many and one-to-one relationship and add the foreign key to each table in the association.
- The *seventh step* is to define relationship roles and constraints (This step is optional).

The other important aspect of the logical model is normalization, which keeps the database healthy and decreases redundancy and inconsistency within it. There are about six normalization forms, but in most cases, when creating a good conceptual model, applying only three of them is sufficient to create a good database. The first normalization form says: only one value per each cell. The second normalization form says: each table should represent one type of feature in the database. The third normalization form says: all table attributes are directly dependent on an ID.

In the **physical implementation** phase, the real database is created according to the database schema from a logical model.

3 Methodology

This section describes the procedure for designing and implementing a smart spatial traffic database for a better management of spatial traffic based data in GIS Section. For this purpose, a requirement analysis was conducted first. Then, the conceptual model of the traffic based database was designed. After that, the traffic database structure was completed by defining a logical model and implementing it physically. Finally, the necessary triggers were defined and implemented in traffic database to make it smart. An iTRIMAN user interface was designed, developed and integrated into current Web GIS system, which is used in GIS Section, to present the functionality of the prototype smart traffic database.

3.1 Case Study

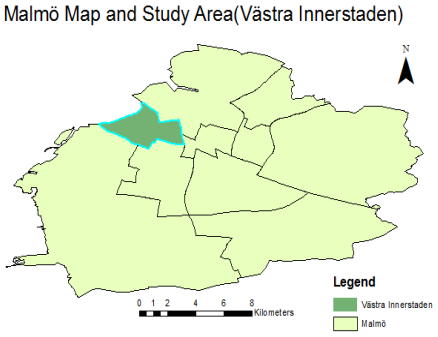

Figure 6. Shows study area.

The City of Malmö is located in the southernmost part of Sweden (Figure 6). Malmö is the third metropolis of Sweden, with nearly 300,000 people. The study area is located between (55.58 °N, 12.94 °E) and (55.61 °N, 12.99 °E) in the northwest of Malmö, named Västra Innerstaden with an approximate area of 46,530 square meters. It appears to be an ideal region for this study because it contains compressive information on traffic data like cycle ways, cycle pumps, cycle stations, bus stations, etc. This region is characterized by an appropriate rate of information dispersion representing the entire City of Malmö. In addition, the area of this region is small enough to meet all the analytical and computational requirements and save computer memory and analysis time for computation. Finally, when the programming is done, the resulting application can be used to analyze other parts of Malmö.

In this study, ten geographical layers were used in the shapefile format with SWEREF99 13 30 coordinate system (EPSG 3008).

3.2 Requirement Analysis

The first step in designing the traffic database involved performing a requirement assessment to determine how GIS section could benefit from this traffic database. For this purpose, a series of interviews was conducted with the GIS section geographical database, Mr. Mozafar Veysipanah and Mr. Raymond Timlin. According to the interviews, data relationships were among the most important aspects which needed to be developed within the traffic database. Internal data relationships indicate that an attribute of an object is assigned to another object, or an attribute of an object is derived from another object. They also show that an attribute of an object is computed and analyzed according to its geographical position. Generally, data are related spatially or conceptually in the traffic database. Therefore, it is necessary to increase consistency and integrity in order to reduce redundancy as much as possible. Since data are related in different ways in the traffic

database, a change in one data item will result in a change in another data item. If such changes are not made, the traffic database will not be consisted anymore. Moreover, it is time-consuming and error-driven to make such changes manually. Therefore, it is essential to design a mechanism for automatically updating the traffic database after every change in traffic database.

This traffic database consists of ten geographical information layers for which ten rules were defined. The geographical layers are as follows: Bus station, speed bump, cycle pump, car barriers, traffic signs, address, cycle ways, cycle station, middle of streets, and allowed areas. The rules stated below in sections denoted A to J:

A. Assigning Allowed Area Attributes to Point Objects

It is necessary to know attributes of allowed area in which a point object such as cycle pump or bust station is located. Then, the attributes of that piece of allowed area should be extracted and assigned to the point object. The allowed area attributes are computed, extracted, and assigned to point objects automatically. After a change is made to point objects or allowed area, the attributes are computed and assigned to them automatically. If a point object is not located inside an allowed area, the allowed area attribute should be null (or empty of a value) in the point object. The traffic database contains the following point object classes: bus station, speed bump, cycle pump, car barriers, and traffic signs.

In addition to computing, analyzing, and assigning piece of allowed area attributes to point objects automatically, it is mandatory to maintain traffic database consistency. For this purpose, if a change is made to a point object, only that point object needs to be updated automatically. However, if a change is made to a allowed area, several point objects might be affected and changed. Thus, the automated updating mechanism should be designed to update several different objects simultaneously.

To reduce redundancy, only key allowed area attributes (such as ID and type) were assigned to point objects. The entire allowed area attributes were not allocated to point objects.

B. Assigning Allowed Area Attributes to Linear Objects

According to the requirement specifications, it is necessary to determine the attributes of allowed area of linear objects. In other words, it is essential to know in which allowed area linear objects are located. For this purpose, every segment of a linear object is computed separately to specify in which allowed area is located. Then the attributes of that piece of allowed area is extracted and assigned to the linear object. A linear object may exist in two different piece allowed area at the same time. In this case, the linear object should be cut off on the boundary between the two piece of allowed area. Then the relevant attributes of piece of allowed area are assigned to each half. The cycle ways layer is the only linear object in the traffic database. After computing and extracting allowed area attributes and assigning them to cycle ways, a mechanism is required for performing computations automatically. This mechanism should be able to update allowed area attributes and cycle ways after every change. Since these two objects have a many-to-many relationship, every change of allowed area attributes or cycle ways can affect several objects at the same time. Therefore, it is

necessary to update all of them automatically and simultaneously. Otherwise, the traffic database will be disintegrated.

To reduce redundancy, only the important allowed area attributes were assigned to cycle ways. The entire allowed area attributes were not assigned to cycle ways.

C. Assigning Cycle way Attributes to Addresses

Cycle ways are linear objects, whereas addresses are point objects. Thus, it is necessary to determine which points these ways connect. For this purpose, certain cycle way attributes are extracted on a specific distance from addresses (20 meters for example). Then, they are assigned to addresses. Since this is a many-to-many relationship, every change of cycle ways or addresses may cause several changes in other objects. Therefore, the automated updating mechanism should be designed to update several objects simultaneously (after every change) so that the traffic database integrity can be maintained.

D. Assigning Street Names to Cycle ways

In GIS Section traffic database, every street has a specific name. However, cycle ways do not have specific names. A system is required for naming cycle ways to give every piece of cycle ways an appropriate name. For this purpose, GIS section does not intend to use a new naming system. Therefore, it is necessary to devise a system which can make the most of available information to do the naming. Accordingly, the best method is to use the names of streets for cycle ways. This method of naming is acceptable if a cycle way is located within a reasonable distance from the intended street. To extract street names and attribute them to cycle ways, it is necessary to determine two parameters: distance between lines and the length of every segment. The distance between lines indicates that a street and a cycle way should be close enough so that the street name can be attributed to the cycle way. The second parameter is the length of every segment meaning that the two nearby segments should be long enough so that the street name can be attributed to the cycle way. Therefore, if the two segments are close when they are shorter than the expected length, the street name cannot be attributed to the cycle way. According to these two conditions, it is not expected that all cycle ways can have names. However, since most of them will have names, the method is desirable for GIS section.

Since this is a many-to-many relationship, a change in every street or cycle way can affect several other objects in the created many-to-many table. Thus, it is essential to update the many-to-many table simultaneously and automatically in order to maintain the traffic database integrity.

E. Assigning the Attributes of the Closest Cycle Station to a Bus Station

GIS section needs to have information on the closest cycle station to a bus station. The information should be updated constantly after a change is made to every object. There is a cycle station for every bus station at the closest distance. According to the current requirements, certain attributes of the closest cycle station to every bus station should be extracted. Then they should be assigned to the bus station including the distance between the two objects. According to the one-to-many relationship between the two objects, a change to a bus station or a cycle station needs to update one or many objects so that the

traffic database integrity is maintained. The updating process should be automated and simultaneous in order to maintain integrity and facilitate maintenance and further updates. With the presence of bus station objects displaying attributes and the distance to the closest cycle station dynamically, it is not necessary to do the calculations manually anymore. The traffic database is updated at once after every change, and GIS section can use it in the management and decision-making process.

F. Assigning the Attributes of the Closest Address to Point Objects

In this traffic database, giving an address to every point object is one of the most important requirements. Since an address is a point object itself, all of the analyses and computations are done between two of the point objects. For every point object, the closest address should be computed. Then certain attributes of the closest address are assigned to that point object. If the distance between the closest address object and another point object exceeds a certain length (100 meters), the address attributes should not be assigned to the point object. In this case, the attributes of the closest address should be null in point objects, something which indicates the lack of an address. There is a one-to-many relationship between the two objects. After every change, the traffic database needs to be updated automatically and simultaneously in one or several locations in order to maintain integrity and facilitate further updates. For redundancy reduction, it is possible to assign the important attributes of the address object to point objects and avoid assigning unimportant attributes.

The traffic database contains the following point object classes needing addresses: bus station, speed bump, cycle pump, car barriers, and traffic signs.

G. Assigning the Attributes of the Closest Street to Traffic Signs

A street is a linear object, whereas traffic signs are point objects. In this traffic database, it is necessary to determine on which street traffic signs are located. Then certain attributes of a street are assigned to the nearby traffic signs. Attributes must be computed and assigned automatically. Since there is a one-to-many relationship between a street and traffic signs, one or many objects are affected after a change is made to the traffic database. Therefore, the traffic database should be updated automatically. According to the rules defined in the traffic database, if traffic signs are farther than a certain distance from a street, no attributes of the street are assigned to traffic signs. Thus, traffic signs hold null.

H. Presenting the Integrity of Point Objects with an Allowed Area on a Separate Column

Allowed areas are polygon objects covering small areas of the study area. An allowed area cover only certain areas such as streets and cycle ways. In many areas such as residential areas, this object is not defined. In this traffic database, all of the point objects must be located inside a piece of allowed area, something which should be shown appropriately to a user who can then make a better decision for that object. Therefore, certain computations are done first to determine the positions of point objects with respect to the allowed area. If a point object is located inside a piece of allowed area, the attribute ‘t’ (indicating that the object is inside a piece of allowed area) is assigned to it. If a point object is outside all of allowed area pieces, the attribute ‘f’ (indicating that the point object is located outside the

entire allowed area pieces) is assigned to it. Therefore, the user becomes well-aware of the position of every point object. In the next step, the allocation of attributes ‘t’ and ‘f’ must be computed automatically. Then, the traffic database should be updated automatically after a change is made to the position of a point object or the geometry of a piece of allowed area in order to compute the new values. Since there is a one-to-many relationship between the two objects, every change of a point object or a allowed area will respectively result in the update of one or many objects. Updates should be applied automatically and simultaneously in order to maintain traffic database integrity. A bus station is the most important point object which is usually located inside a piece of allowed area. Hence, the user can continuously control this attribute in the bus station objects class. If an ‘f’ value is observed in front of objects, necessary actions can be taken to correct it.

I. Presenting and Displaying the Consistency of Linear Objects with an Allowed Area on a Separate Column

This structure is very similar in structure to H (above). Here, the positions of linear objects are computed first to determine whether a linear object is located inside or outside a piece of allowed area. If a linear object is inside a piece of allowed area, a ‘t’ value is attributed to the linear object. Otherwise, an ‘f’ value is attributed to it. Furthermore, it is sometimes possible that a part of a linear object is located inside a piece of allowed area when another part of it is located outside of same piece of allowed area. In this case, the linear object is divided into two parts, each of which will receive an appropriate value. Attributes should be computed and allocated automatically. The objects should be updated automatically, as well, after a change is made to the traffic database.

J. Creating Integrity in the Cycle Pump (Accepting Changes Predefined in the Domain and Discarding Changes Not Predefined in the Domain)

A cycle pump is a point object which can usually be very useful along cycle ways. GIS Section needs to define certain rules in the traffic database so that cycle pumps can be found near and only near cycle ways (in 20 meters). According to this requirement, the user can build a new cycle pump near cycle ways or relocate it. After building or relocating the intended object, all of its attributes (such as the closest address and allowed area attributes) will be updated automatically. However, the user cannot create a new object any farther than a certain distance (20 meters) from cycle ways. It is not possible to move objects from a closer distance to a farther distance (farther than 20 meters). In addition to preventing the user from creating a new object or moving it to a farther distance than the determined domain, the system needs to warn the user that the new position is outside the defined domain in the traffic database. Nevertheless, the system does not store the new position in the traffic database and object will automatically be moved to the previous position.

3.3 Selecting Database Techniques

GIS section is interested in creating a smart database using the active database method. Since the database engineers of the GIS section are familiar with triggers and they already use the triggers to update the database, this could be a high priority option. There are no use

of ontologies and deductive techniques in this department, and engineers of this section do not have the necessary background with these methods. So using these techniques are not of interest for the moment. With these in mind, there is no proper technical background for creating a smart database based on ontology and deductive techniques. In addition, there are currently no financial resources available for creating a smart database based on deductive, ontology and semantic techniques.

Considering above, based on the interest of the database engineers of the GIS section with the triggers, and their daily use of triggers, on one side, and the lack of proper technical background and financial resources for promoting deductive databases, ontology and semantic techniques, on the other hand, the active database has been selected for further study to improve the traffic database of the GIS section.

Table 1. Shows name, attribute, geometry and behavior of each table. Points (Pt), Polygon (Pg), Polyline (Pl).

No	Table name	Geo m	ID and State(attributes)	behavior
1	Address	Pt	Idad, gem, objnr, beladress, postnr	Add point, remove point, edit point location
2	Allowed Area	Pg	Idaa, geom, typ, objnr	Add polygon, remove polygon, edit polygon
3	Middle of Streets	Pl	Idms, geom, objnr(name)	Add line, remove line, edit line
4	Bus Station	Pt	Idbs, geom, hallplats, idad, idad_postnr, idad_beladress, idad_objnr, idad_distance, idcs, idcs_distance, Idaa, Idaa_objnr, Idaa_typ, check_if_bs_in_aa	Add point, remove point, edit point location
5	Speed bump	Pt	Idsb, geom, typ, idad, idad_postnr, idad_beladress, idad_objnr, idad_distance, Idaa, Idaa_objnr, Idaa_typ	Add point, remove point, edit point location
6	Cycle pump	Pt	Idcp, geom, typ, objnr, idad, idad_postnr, idad_beladress, idad_objnr, Idaa, Idaa_typ, Idaa_objnr, check_if_cp_in_cw	Add point, remove point, edit point location
7	Car barriers	Pt	Idcb, geom, idad, idad_postnr, idad_beladress, idad_objnr, idad_distance, Idaa, Idaa_objnr, Idaa_typ	Add point, remove point, edit point location
8	Way Signs	Pt	Idws, geom, objnr, skylt_1, idad, idad_postnr, idad_beladress, idad_objnr, idad_distance, Idaa, Idaa_objnr, Idaa_typ, idms, idms_objnr	Add point, remove point, edit point location
9	Cycle Station	Pg	Idcs, geom, idad, idad_postnr, idad_beladress, idad_objnr	Add polygon, remove pg, edit pg
10	Cycle Ways	Pl	Idcw, geom, typ	Add line, remove line, edit line
11	Cycle Ways_AA	Pl	Idcw, geom, idcw_typ, Idaa, Idaa_objnr, Idaa_typ, check_if_cw_in_aa	-
12	Cycle Ways_address	Pt	Idcw, typ, geom, idad, beladress, postnr,	-
13	Cycle Ways_name	Pl	Idcw, geom, idms, idms_typ, idms_objnr	-

3.4 Conceptual Model Using UML

A model is created based on the requirement specification for traffic database to determine all object classes, relationships, type of relationship, degree of relationship (cardinality), and rules. All of the object classes are determined in this conceptual model. The attributes of every object class are defined along with relevant methods. The internal relationships between object classes are determined in accordance with the requirement analysis. The cardinality ratio of every relationship is determined, too.

3.4.1 Object Classes

This traffic database contains 10 object class and three many-to-many tables (Table 1). Two of the object classes are polygons including an allowed area and a cycle station. There are two linear object classes including cycle ways and middle of streets in addition to two many to many relationship tables (row number 11 and 13 in table 1). Finally, there are seven object classes including point object classes such as bus stations, speed bumps, cycle pumps, car barriers, traffic signs and addresses as well as the many-to-many table (row number 12 in table 1) (Figure 7).

The conceptual model includes all of the attributes, and methods of an object class. The data type is mentioned in front of every object class attribute. In addition to having attributes, all of the object classes support certain methods. However, three table (tables with many to many relationships, row number 11, 12 and 13 in table 1) do not support methods due to having many-to-many relationships in this traffic database. These object classes have a pair of external keys. Since they do not support methods, it is not possible to change them directly, add an object to them, or delete an object from them (Table 1).

3.4.2 Relationships

In this traffic database, there are binary relationships, and the notations are either “has” or “inside”. All of the point object classes have addresses (“has” notation). They are located inside the allowed area (“inside” notation). The cycle pump objects are inside the range of cycle ways (“inside” notation). The traffic signs objects are characterized by middle of streets (“has” notation). The cycleway name objects get a name attribute from the middle of street with the geometry of cycle ways. The cycle way_aa (name of M:M table) objects are located inside the allowed area with a geometry received from cycle ways. Furthermore, the cycleway address objects have an address with a geometry received from cycle ways (Figure 7).

3.4.3 Cardinality

In this traffic database, all of the relationships are characterized by one-to-many and many-to-many cardinality. There are three many-to-many relationships for which separate tables were created (row number 11, 12 and 13 in table 1). Other relationships are one-to-many (Figure 7).

3.4.4 Rules

In the requirement analysis, 10 different rules were defined for the traffic database. These rules were used for 18 relationships in the traffic database. In other words, some of these rules were used more than one time. Now, the predefined rules of every object class are described. According to the rules, attributes and values are assigned from one object to another after computation. All of the following rules were completely described in the requirement analysis. Here, it is sufficient to mention the names of rules used in every object class. In this traffic database, all of the objects are related to each other. They assign their values and attributes to each other. Only three object classes, i.e. allowed area, address, and middle of streets, assign their attributes and values to other objects. However, they do not receive any attributes from other objects at all. Thus, if a change is made to one of these three object classes, many objects will be updated in the traffic database. However, changing one of the other objects will not update these three object classes.

Cycle Station: According to the only rule defined for this object class, the closest address is computed and assigned to every cycle station. Moreover, every cycle station object can only receive certain attributes of an address object (the closets address object). The computation and attribution processes are completely automatic. After a change is made to a cycle station or an address, the cycle station is automatically updated.

Bus Station: There are four rules defined for this object class. The first rule is to compute and assign the closest address to a bus station. The second rule is to compute and allocate the certain attributes of a cycle station to a bus station. According to this rule, the distance to the closest cycle station is calculated and assigned to the bus station. The third rule is to compute and allocate the attributes of a piece of allowed area to a bus station. The fourth rule is to determine the position of a bus station in relation to a piece of allowed area shown as true or false (‘t’ or ‘f’) on a separate column. Finally, the automated updating mechanism is used to compute all of the abovementioned attributes and assign them to the bus station. The updating mechanism was designed to calculate the values and attributes automatically and allocate them to the bus station after every change is made to the bus station, address, allowed area, middle of streets, or cycle station classes.

Speed Bump: Two separate rules were defined for this object class. According to these rules, it is necessary to compute the attributes of the closest address and the attributes of the piece of allowed area in which a speed bump is located. These attributes are then assigned to speed bump objects. Therefore, the attributes are computed and assigned to speed bump objects automatically after a change is made to the speed bump, address, and allowed area object classes.

Cycle Pump: There are three rules for the cycle pump. The first rule is related to cycle ways. Accordingly, a cycle pump can be located at a certain distance from cycle ways. A mechanism was also designed for changes made to the position of the cycle pump. The

spatial changes of cycle pumps should be in a certain domain so that they can be recorded. Otherwise, the user is faced with an error, and it will be impossible to save recent spatial changes defined outside the domain. The second and third rules are related to the computation and allocation of address and piece of allowed area attributes to the cycle pump. All of the aforesaid rules must be computed and attributed to the cycle pump automatically after every change.

Car Barriers: Two rules were defined for this object class to compute and assign the address and allowed area attributes to car barriers. They are computed and updated automatically.

Traffic Signs: This object class has three rules, the first of which is related to the computation and attribution of name from the closest street to traffic signs. The second and third rules pertain to the computation and attribution of the closest address and allowed area attributes to traffic signs. They are computed and updated automatically.

Cycle ways: In this object class, there are three separate rules for the address, allowed area, and middle of street object classes. Since there is a many-to-many relationship between cycle ways and each of the above object classes, a new table is created for each of them. These tables cannot be changed directly. However, the resulting tables of many-to-many relationships will be updated automatically after a change is made to cycle ways, allowed areas, address, and middle of streets.

Allowed Area, Address, and Middle of Streets: These three object classes assign attributes to nearly all of the other object classes after computation. However, no attributes of other object classes are assigned to these three. Thus, they do not need an automated updating mechanism. Since every change of these three object classes results in changes in many other object classes, it is necessary to devise a mechanism which can update other relevant object classes after a change is made to these three.

Cycleway Name: This table was created as a result of many-to-many relationships between cycle ways and streets. According to the rule of such relationships, the attributes of street names are assigned to cycle ways. As a result of every change in the two object classes (street or cycle ways), this table is updated automatically.

Cycleway Address: This table is created as a result of many-to-many relationships between cycle ways and addresses. According to the rule of this table, a set of addresses are attributed to every cycleway and updated automatically after a change is made to cycle ways or addresses.

Cycleway Aa: This table is created as a result of defining a rule for many-to-many relationships between allowed areas and cycle ways. It indicates the allowed area of every object on cycle ways. This object class is computed and updated automatically after a change is made to every allowed area or cycleway.

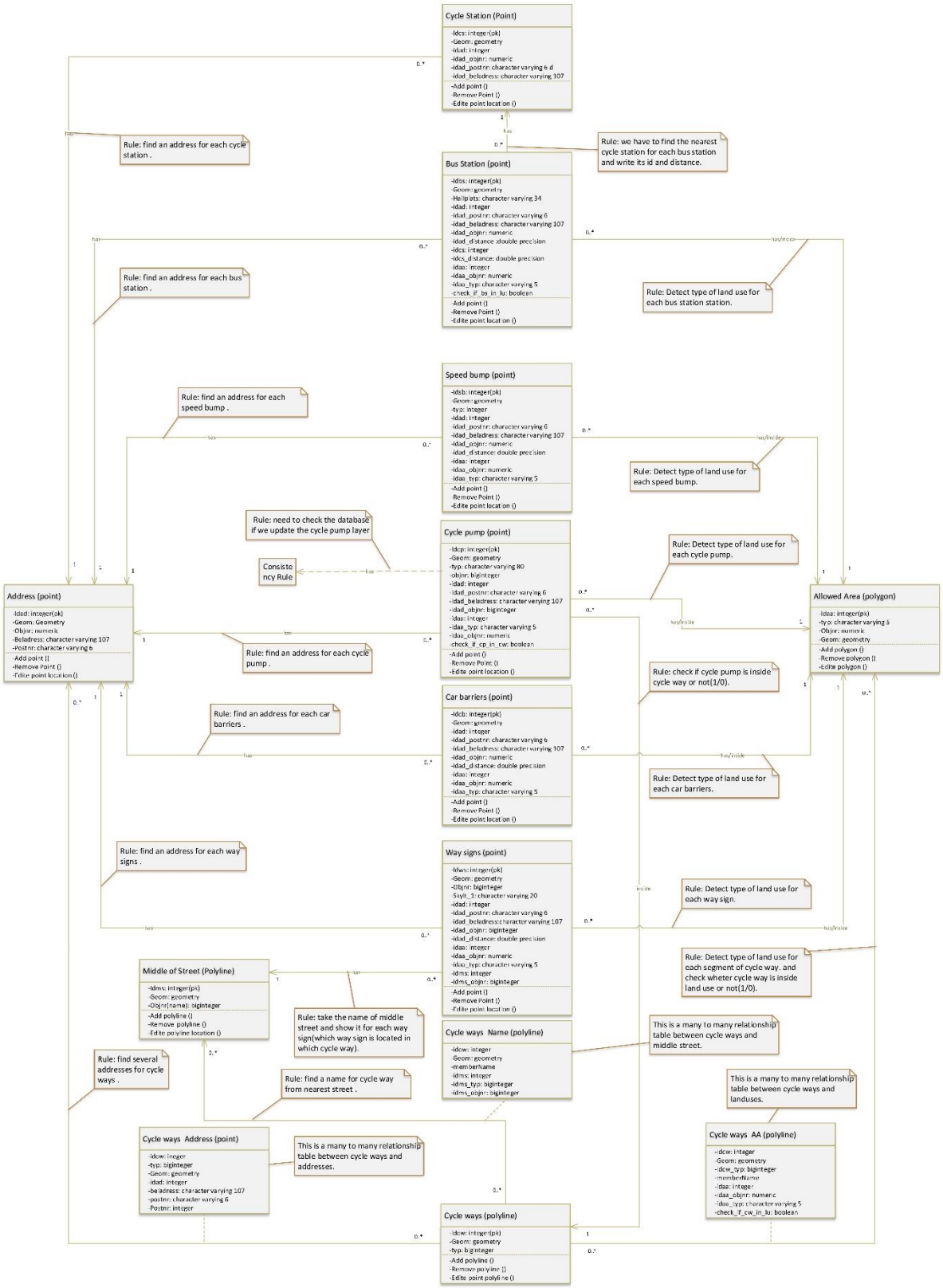

Figure 7. UML model of the database.

3.5 Logical Model and Physical Implementation

The two last steps of creating a traffic database are logical model and physical implementation. In these two steps, a real traffic database is created and by using triggers, traffic database is made automatic.

3.5.1 Logical Model

Six steps should be taken to codify a logical model for this traffic database. The first step is to create a table for every object class. Given the fact that there are 10 object classes in the conceptual model, a table is created for each one in the traffic database. Therefore, there will be 10 tables in total. All the values, and attributes of an object are defined and put on the columns in front of it in the second step. These attributes can be of the saved or derived types. In other words, the attributes can be entered directly by users or derived from the attributes of other objects in the traffic database. The attributes can also be generated by conducting spatial computations on other attributes. Then, they can be assigned to the object of interest. In the third step, the primary key is defined. This key serves as an ID giving a unique identity to every object in the object class due to the possession of a primary key. In the fourth step, a table is created for every many-to-many relationship in the conceptual model. Thus, three tables can be created in the traffic database by having three many-to-many relationships. The tables are named *cycleway name*, *cycleway address*, and *cycleway aa*. In the fifth step, the primary keys of attributes are added to tables with many-to-many relationships. One-to-many relationships are identified in the sixth step in which external keys are added to tables existing on the other side of a relation ship (Table 2).

Table 2. Shows multiplicity, name, ID and FK of each object.

No	Object or M:M table	Name of Object or M:M table	ID	Posting FK for each object 1:1 and 1:M
1	Object	Address	IDAD	
2	Object	Allowed Area	IDAA	
3	Object	Middle of Streets	IDMS	
4	Object	Bus Station	IDBS	IDCS, IDAA, IDAD
5	Object	Speed bump	IDSB	IDAA, IDAD,
6	Object	Cycle pump	IDCP	IDAA, IDAD
7	Object	Car barriers	IDCB	IDAA, IDAD
8	Object	Way Signs	IDWS	IDAA, IDAD, IDMS
9	Object	Cycle Station	IDCS	IDAD
10	Object	Cycle Ways	IDCW	
11	M:M table	Cycle way_Area	IDCW-AA	IDCW, IDAA
12	M:M table	Cycle way_Address	IDCW-AD	IDCW, IDAD
13	M:M table	Cycle way_Middle of street	IDCW-MS	IDCW, IDMS

In the physical implementation, the traffic database is created on the basis of the schematic view generated in the logical model (Figure 8). After creating the traffic database, special attention should be paid to the defined needs in the requirement specifications. Since there are spatial and conceptual relationships between the traffic database objects, a change in one object can change other objects. Therefore, it is necessary to establish consistency and integrity in the traffic database. It is difficult to provide the traffic database with consistency and integrity manually; thus, an automated mechanism is required. Such a mechanism can be created by using triggers in the traffic database. The triggers are defined to be fired after every change/addition/deletion/update in the traffic database in order to execute a set of SQL commands on the traffic database. The triggers can be used to create an automated, consistent, and integrated traffic database with low (defined) redundancy to provide all the

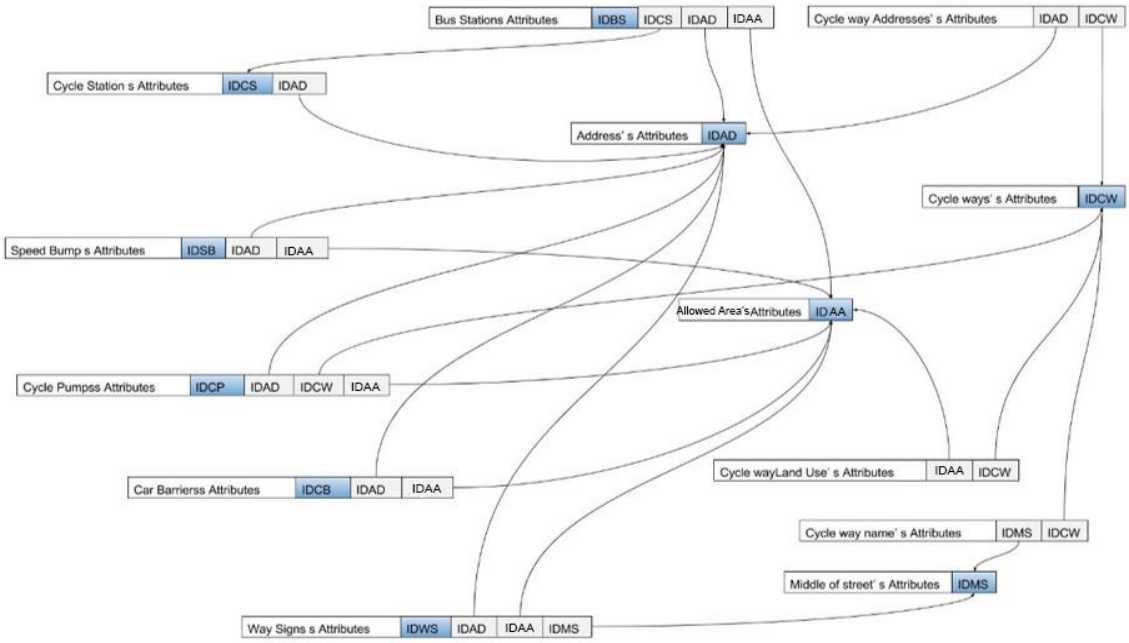

Figure 8. Database schema of implemented database.

internal relationships. There were 26 triggers created in this traffic database to execute 10 rules defined in the requirement analysis. Generally, the defined triggers were divided into two types based on the load of work and type of update. The first type includes the triggers which are fired after a change is made to a table and update the same table of trigger. The second type includes the triggers which are fired after a change is made to a table and update many other tables. In Table 3, the number of triggers and functions defined in the traffic database can be observed. It shows the name of every table trigger and the table updated by that trigger. It also indicates the duration required for executing every trigger and the duration needed to update a table in the traffic database completely (Table 3).

Table 3. Shows name of each table with related triggers, function and updated tables. Here we can see the time of execution for each trigger and for each table separately.

No	On Table	Name of Triggers	Name of Functions	Updated Tables	Execution Time for each trigger(Sec)	Execution Time to update table (Sec)
1	Cycle station	Trig1_cs	Func1_cs	Cycle station	1>	4
		Trig2_cs	Func2_cs	Bus Station	4	
2	Bus Station	Trig2_bs	Func2_bs	Bus Station	1>	1>
3	Speed bump	Trig3_sb	Func3_sb	Speed bump	1>	1>
4	Cycle pump	Trig4_cp	Func4_cp	Cycle pump	1>	1>
5	Car barrier	Trig5_cbar	Func5_cbar	Car barrier	1>	1>
6	Way signs	Trig6_ws	Func6_ws	Way signs	1>	1>
7	Cycle way	Trig4_cw,	Func4_cw,	Cycle pump	1>	35
		Trig71_cw,	Func71_cw,	Cycle way_aa	33	
		Trig72_cw,	Func72_cw,	Cycle way_ad	1>	
		Trig73_cw	Func73_cw	Cycle way_name	1>	
8	Address	Trig1_ad	Func1_ad	Cycle station	4.9	24
		Trig2_ad	Func2_ad	Bus Station	4	
		Trig3_ad	Func3_ad	Speed bump	1>	
		Trig4_ad	Func4_ad	Cycle pump	1>	
		Trig5_ad	Func5_ad	Car barrier	1>	
		Trig6_ad	Func6_ad	Way signs	12	
		Trig72_ad	Func72_ad	Cycle way_ad	1>	
9	Allowed Area	Trig2_lu	Func2_lu	Bus Station	4	52
		Trig3_lu	Func3_lu	Speed bump	1>	
		Trig4_lu	Func4_lu	Cycle pump	1>	
		Trig5_lu	Func5_lu	Car barrier	1>	
		Trig6_lu	Func6_lu	Way signs	12	
		Trig71_lu	Func71_lu	Cycle way_aa	33	
10	Middle Street	Trig6_ms	Func6_ms	Way signs	12	13
		Trig73_ms	Func73_ms	Cycle way_name	1>	

Triggers of the First Type: These triggers update only changed object. They compute new values for the attributes of that object. In other words, the trigger of a table is activated after a change is made to the table. Then, the trigger updates only the row (object) where the change occurred. Since such triggers update only one object or row after activation, they take very little time to perform the update. Hence, they are very fast. In this traffic database, such triggers were used in the following tables including bus station, cycle station, cycle pump, speed bump, car barriers, and traffic signs.

Triggers of the Second Type: These triggers update the entire object class and several dependent object classes (if necessary) after a change is made to an object. In other words, the trigger updates the entire table (every row) after a change is made to that table. This trigger can update one or more tables at the same time. Since doing update using these triggers takes a large size, they need more time to update all of the required tables. For instance, if a change is observed in the address table, other tables are updated. It might be time-consuming to update all of these tables having the address attribute with a large number of rows. In traffic database, such triggers were used in the tables including address, allowed areas, middle of streets, and cycle ways.

Challenges with triggers:

Triggers are very useful to apply the essential rules to a database and make the database smart. Using triggers, one can run a set of SQL commands with every change in the database, which increase the consistency, integrity and attribution of the quantities automatically in the database. However, using triggers has some disadvantages. The most important disadvantage of trigger's use includes the loop of SQL commands due to the infinite activation of triggers. If all the defined triggers are created into the database to enforce the essential rules, any change in one of the tables will activate the trigger of that table, which in turn will cause another table to be updated and its trigger will be activated; this will eventually lead to the re-update of the first table and re-activate its trigger. If this process starts, there is no end to it and the triggers will continuously activate each other (Figure 9). The difficulty with active database is no easy-to-use techniques for modeling, designing and writing active rules. It means it is quite hard to guarantee a termination for two rules in active database, which activate each other.

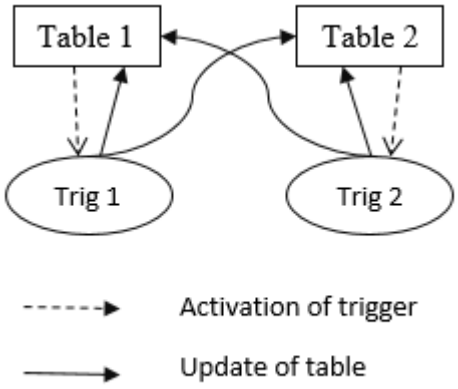

Figure 9. Shows the loop challenge.

3.6 System Architecture of User Interface

It is very effective and efficient to use triggers to perform computations and analyses, establish consistency and integrity, and data relationship in the traffic database automatically. However, the use of triggers is error prone. A major disadvantage of using triggers is the loop of SQL commands caused by firing triggers infinitely.

To resolve the infinite activation of triggers in the traffic database, the iTRIMAN user interface was designed. It can create necessary triggers on the traffic database and delete them if they are not required. This interface allows a trigger to exist in the traffic database only if its relevant table is being edited. Different applications such as QGIS, ArcGIS, and Internal Web GIS of Malmö Municipality can be used to edit the spatial data existing in the PostgreSQL traffic database with PostGIS extention. When data are edited, only relevant triggers should be created by the iTRIMAN user interface. There should not be any triggers in the traffic database by default when data are not edited.

The iTRIMAN user interface was designed in two platforms: desktop and web. The web-based platform was designed exclusively for Internal Web GIS of Malmö Municipality. However, the desktop platform can be used in desktop applications.

3.6.1 Desktop iTRIMAN

Since desktop applications such as QGIS and ArcGIS are used widely in GIS Section to display and edit spatial data, the iTRIMAN user interface is employed simultaneously with desktop applications. The applications are responsible for displaying and editing data, and the iTRIMAN user interface is responsible for creating and deleting triggers. With this combination, the data of interest can be displayed first. Then, the iTRIMAN user interface can be used to activate relevant triggers and update them automatically after every change. Finally, the results can be displayed in desktop applications. Following is described architecture of desktop iTRIMAN:

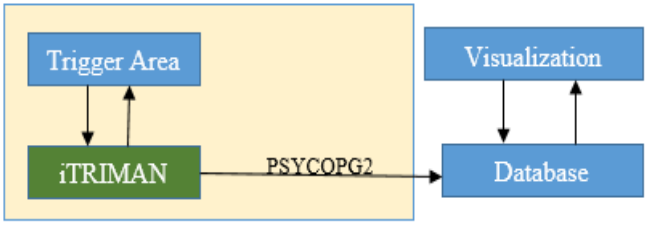

Figure 10. Shows architecture of desktop iTRIMAN.

Trigger Area: In fact, this is a folder in which all 26 triggers are written inside it. Each of these triggers is associated with one specific rule (section 3.2) and stored inside of this folder (Trigger Area) in form of separate file. This folder, along with the triggers which are inside this folder, is a part of the iTRIMAN which has a two-way relationship (send command and receive data) with iTRIMAN. These triggers are written in both Python and SQL languages. In case of need for editing, the administrator can edit the triggers directly in this folder.

iTRIMAN: iTRIMAN is an interface which is written using the Python programming language with help of Tkinter library. Tkinter is a Python library for creating GUI (Graphic User Interface). This interface, on the one hand, it is connected to trigger area (the folder of triggers) and on the other side with the PostgreSQL traffic database. All of the buttons are also located inside this interface. This interface, with every click on the buttons, call the related triggers from trigger area and run them on the traffic database. To implement the iTRIMAN commands on the traffic database, the PSYCOPG2 adaptor (connect Python language to PostgreSQL traffic database) is used.

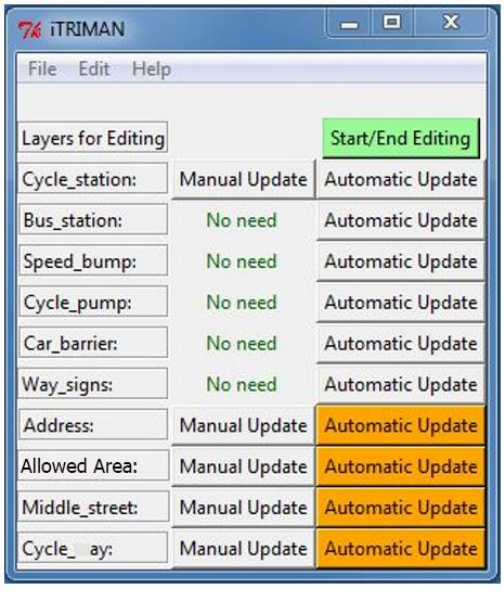

Figure 11. Shows interface of desktop iTRIMAN.

Database: This is a PostgreSQL traffic database with installed PostGIS extension which can support geographic data. This traffic database receives commands from the iTRIMAN Interface and displays the results in the *visualization*.

Visualization: This is for displaying geographic data. The information displayed in this section is received from PostgreSQL traffic database. In addition to displaying, data can be edited and the results are saved to the traffic database. For the visualization, can use available desktop software such as QGIS and ArcGIS.

Followings are instructions, which show how iTRIMAN works: Three menus (file, edit, and help) and several other important built-in buttons were designed for this user interface. The following buttons have the key roles in updating the traffic database automatically:

- ***Start/End Editing*:** This button is used before starting and after finishing editing the spatial data to delete all of the traffic database triggers and ensure the user that no other triggers exist in the traffic database. Therefore, it can prevent performance interferences and unwanted results in the traffic database (Figure 11).
- ***Automatic Update*:** This button creates the triggers of interest in the traffic database. After they are created, the editing can be started. With every change in the traffic database, the entire relevant data are updated automatically. This button was designed in two colors, i.e. gray and orange, indicating the update time. In other words, gray buttons need a very short time for updating (nearly a few milliseconds), whereas orange buttons need more time for the automatic update. The two colors are meant to notify the user of the time to update different tables so that the right buttons can be selected when the user interface is used (Figure 11).
- ***Manual Update***: It is sometimes possible that many changes should be made to the traffic database. Moreover, the time of these changes may entirely belong to the second-type triggers updating many tables. Therefore, it can be very time-consuming to update a large traffic database. In this case, manual update buttons are designed. These buttons can be used only by pressing the start/end editing button to make start editing session. Finally, the manual update button can be pressed to apply all the changes and update the traffic database (Figure 11).

3.6.2 Web iTRIMAN

Since GIS Section use the internal Web GIS of Malmö Municipality which is used constantly by the GIS personnel and engineers, it is essential to integrate web iTRIMAN with the

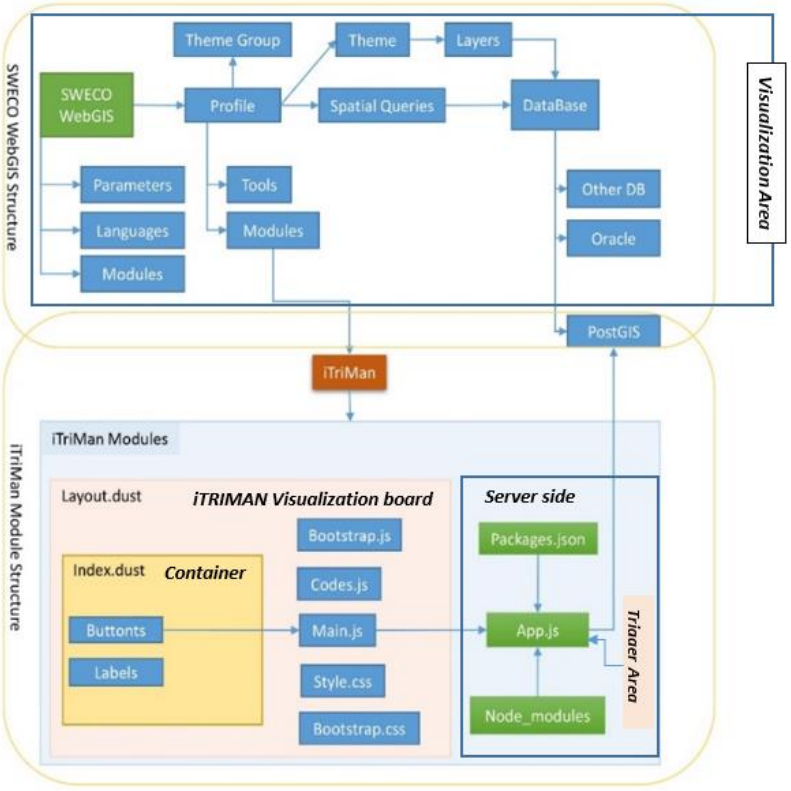

Figure 12. Shows architecture of web iTRIMAN in relation with SWECO Web GIS.

Web GIS of Malmö Municipality. For this purpose, web iTRIMAN was designed as a module inside Web GIS of Malmö Municipality. JavaScript, HTML, CSS, Node.js, Ajax, and jQuery were used to design this module. Web iTRIMAN is exactly the same as desktop iTRIMAN in structure and performance. The only difference is that desktop iTRIMAN was programmed in Python. It is also used with desktop applications, whereas web iTRIMAN is web-based and used with Web GIS (Figure 12).

There are three necessary steps in designing and implementing web iTRIMAN in Web GIS of Malmö Municipality. In the first step, an iTRIMAN button should be defined in the XML structure of Web GIS of Malmö Municipality. In the second step, a container should be designed so that iTRIMAN module can run in Web GIS of Malmö Municipality after clicking on the iTRIMAN button. The container includes all the necessary buttons and tags to receive the commands in the exact same way as desktop iTRIMAN. Finally, the third step is to establish the relationship between web iTRIMAN and PostgreSQL traffic database so that SQL commands can be sent to this traffic database.

System architecture of web iTRIMAN (Figure 12) is as follow:

iTRIMAN Module: This module is displayed on the Web GIS of Malmö Municipality by iTRIMAN button. This module has four different parts including container, visualization board, server side and trigger area, which is connected to the traffic database.

Container: Container is a part of the iTRIMAN module, which includes all of buttons, tags, labels and icons. But this section does not display for the user. With each click on the buttons in this section, a message is sent to the visualization board for further processing. The most important file in this section is called "index.dust".

Itriman Visualization Board: This section act like a monitor or display board for the iTRIMAN interface. In other words, the user sees only the result of this section. This section includes CSS files, Javascript files and "layout.dust". In addition, the most important activity of this section is to receive clicks from the container section. Each click on a button is detected and then is sent to the visualization board with a special ID. Then visualization board sends the related commands to the server side to be applied on the traffic database with the help of jQuery and Ajax. It is worth to mention that the two sections of the container and the visualization board are located in the user side.

Trigger Area: All 26 triggers are located in the trigger area. The Trigger area is located on the server side. Each button which is clicked on the user side, based on its ID number, it sends specific commands to the server side. Depending on the commands sent to the server side, the related triggers are selected from the trigger area and sent to the traffic database for execution. Depending on the commands are sent to the server side, the related triggers are selected from the trigger area and are sent to the traffic database for execution.

Server Side: One of the most important parts of the iTRIMAN is server side. All commands are sent from the user side with a special ID to the server side. The server side is designed using Node.js server framework and all required libraries are also installed on server side. The server side has trigger area and related files for connecting with both user side and traffic database. Server receive commands from the user side and then based on the ID of the received commands, select the relevant trigger and run it on the traffic database.

Database: The traffic database of this section, like iTRIMAN desktop, is a PostgreSQL traffic database with PostGIS extension which can support geographic data. This traffic database receives commands from the iTRIMAN server side. Then displays the results in the visualization area (Web GIS of Malmö Municipality).

Visualization area: The visualization area is for displaying the geographic information, which is received from the PostgreSQL traffic database. The Visualization area of web iTRIMAN is interal Web GIS of Malmö Municipality. The user can edit the data in addition to the display and the result is saved again in the traffic database.

4 Results

In this section smart traffic database will be tested which is created according to the requirement analysis and based on the selected technique (active method). For this purpose, triggers and iTRIMAN interface were used for the automatic update. Then desktop and Web GIS applications were employed to edit and display a spatial traffic database. As a result of all the aforesaid steps, an integrated traffic database was created in accordance with the requirements of GIS section. The traffic database can automatically update itself after every change. The traffic database should be tested practically to analyze the results and observe its integrity, behavior, and performance. Various requirements were defined in the requirement specifications in addition to designing two different types of triggers to design this traffic database. Therefore, the good tests include all or most of the requirements, both triggers, and their applications in both web and desktop GIS environments. Hence, the accurate functions of triggers and the fulfillment of requirements can be guaranteed.

This section is going to demonstrate three test scenarios as the result of the work. Bus station and cycle pump tables were used to test the first-type triggers. These tables include a number of defined requirements. Other tables include certain requirements tested and evaluated for bus station and cycle pump object classes. Therefore, it is not necessary to retest requirements with other tables. The address table was used to test the second-type triggers. Therefore, every change in this tables results in the update of a large number of other tables.

4.1 Bus Station

This table uses the first-type triggers for updating. It meets the following requirements for GIS section: 1) the closest address, 2) the type of allowed area , 3) inside or outside of the allowed area, 4) the closest cycle station, and 5) the distance from the closest cycle station.

The above requirements are computed and assigned to the bus station automatically by using the first-type trigger. For this purpose, the bus station layer is displayed in Web GIS first to activate its geometric editability. Then the iTRIMAN button is clicked to run iTRIMAN user interface (Figure 13). In this step, the start editing button is clicked first to ensure that there are no triggers in the traffic database. Then the automatic update button, designed for the bus station layer, is clicked to create the respective trigger in the traffic database （Figure 13). Then a new object is created for the bus station using Web GIS interface. After that, the entire attributes are computed and assigned to the bus station automatically. According to the Figure 14, then, they are displayed in the right place. Moreover, all of the attributes and values are recomputed and allocated to the bus station

automatically after the object is moved from one position to another (Figure 15). If the new position is empty or undefined, a null value is allocated to the respective attribute. According to the Figure 16, the moved bus station is outside of allowed areas. Therefore, no allowed area attributes are displayed for this section. The value shown in the table is null. The respective cell, indicating whether the bus station is located inside or outside a piece of allowed area by unticked or false column. In this case, a geometric serial ID is entered by the user. However, other attributes are computed automatically. Since only a changed object is updated, computations and attributions occur very fast.

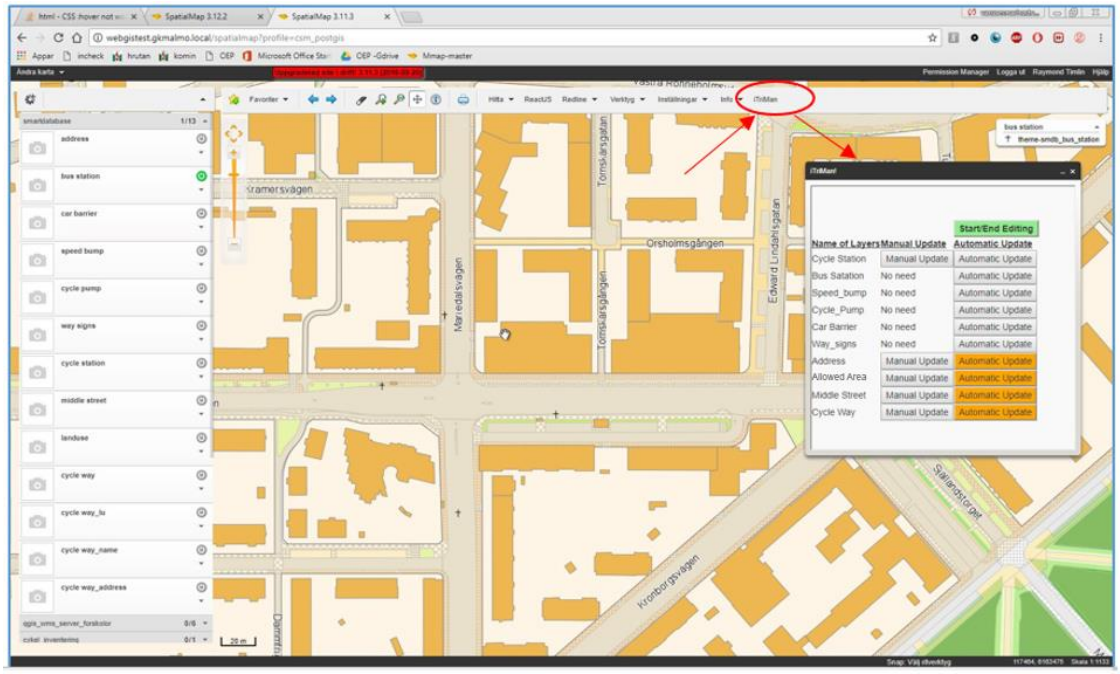

Figure 13: Open iTRIMAN interface by clicking on “iTRIMAN” module button on the top right side.

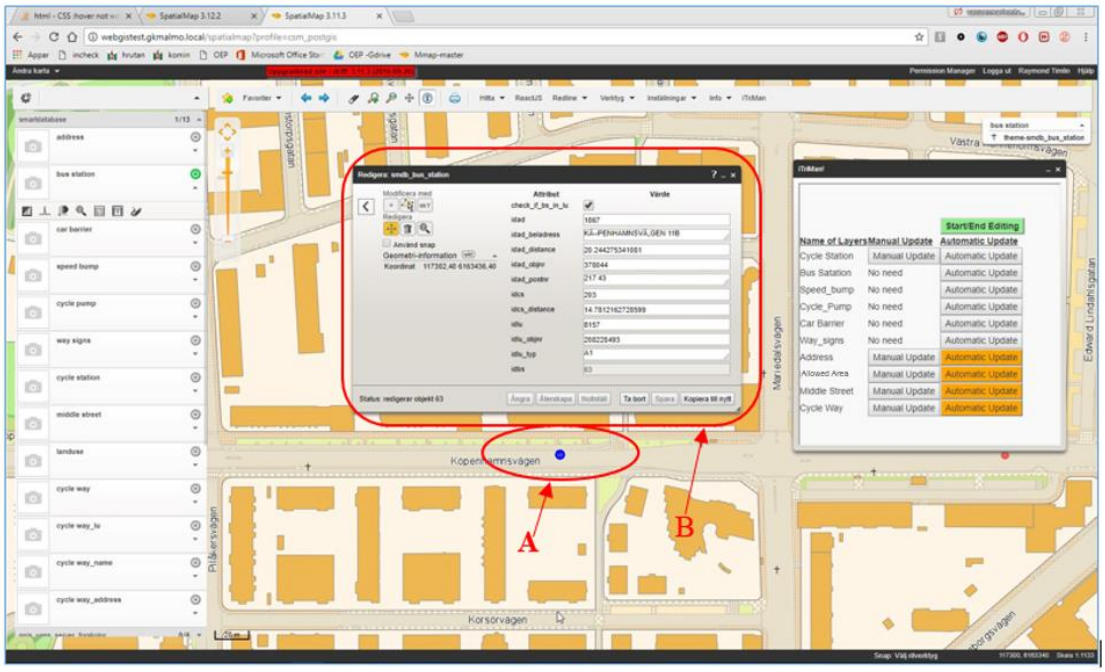

Figure 14: Shows iTRIMAN automatically updates attributes of created bus station object. (A) Shows the created new bus station, (B) Attributes of created new bus station.

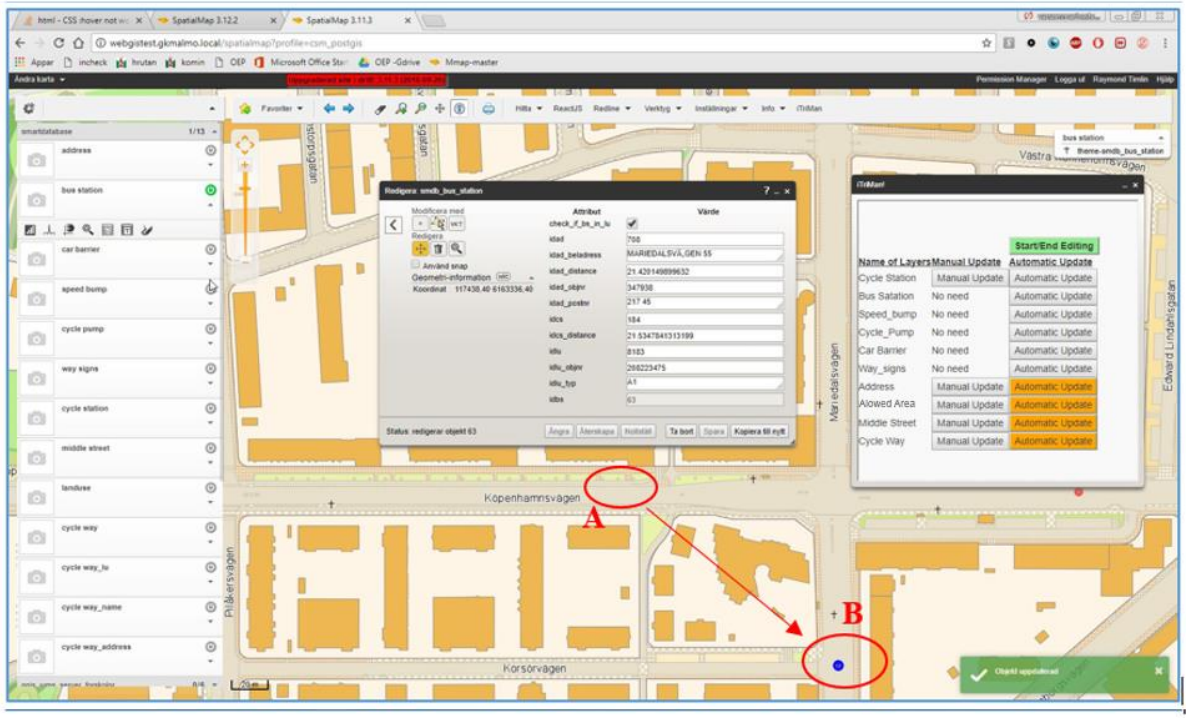

Figure 15: Shows iTRIMAN automatically updates attributes of bus station object, when its geometry is changed (bus station is moved from location A to location B).

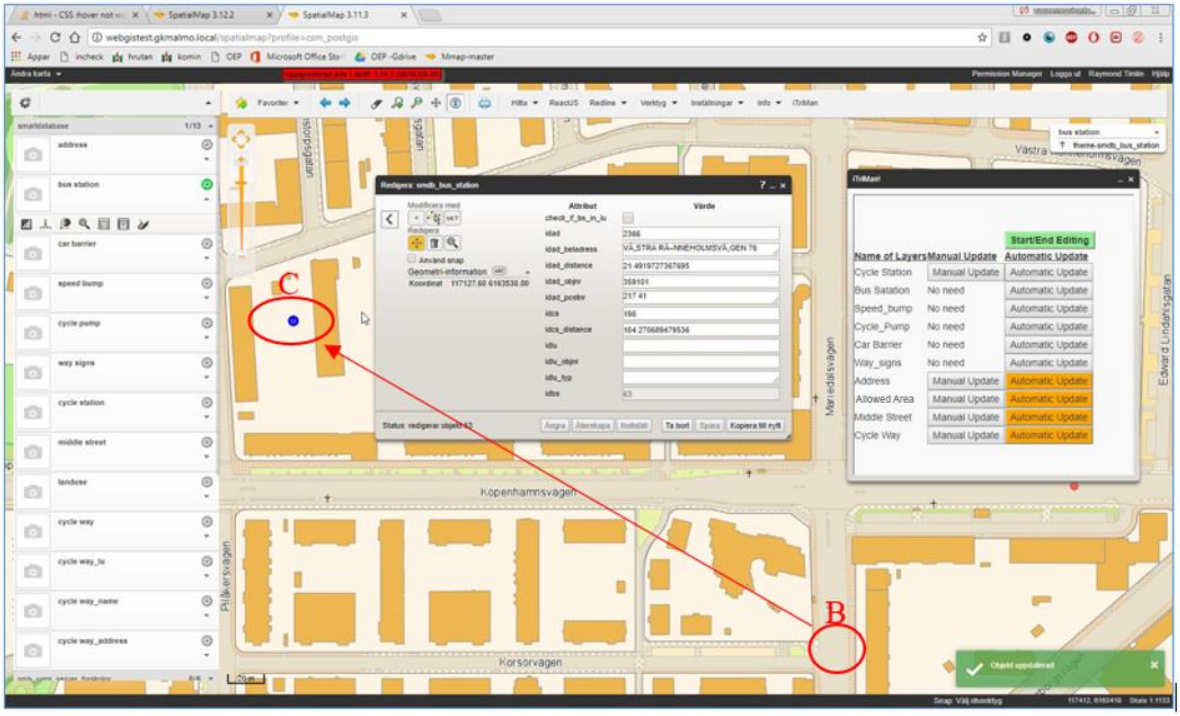

Figure 16: Shows iTRIMAN automatically updates attributes of bus station object, when its geometry is changed (bus station is moved from location B to location C).

4.2 Cycle Pump

This table also uses the first-type triggers for updating. The requirements of this table are as follows: 1) a mechanism for preventing objects from being saved and send a message to the user if the new geometry is outside the defined domain, 2) a column which can show the proximity of cycle pumps to cycle ways as true or false, 3) the closest address, and 4) the type of allowed area.

A cycle pump is a point object which should always be near cycle ways. QGIS was used to edit and display this object class. To test this object class, it should first be opened in QGIS. Then the start editing button should be activated. After that, the iTRIMAN user interface should be opened, and the start editing button should be activated again to delete any other triggers. In the next step, the automatic update button of the cycle pump layer should be clicked to create the respective trigger in the traffic database (Figure 17). Now a new object can be created for the cycle pump (Figure 17). Since the position of the created object is inside the predefined domain, all of its attributes and features are assigned to the respective object automatically (Figure 17). In the next step, the object created near cycle ways is moved to a farther position to observe how the traffic database function towards it (Figure 18). After sending the request to save the moved object, the system notifies the user of an error to indicate that the new position is outside the predefined domain for the cycle pump (Figure 18). Therefore, the system does not accept to save the new position. Furthermore, the system sends the moved object to the initial position after sending the error (Figures 19). The Boolean column, indicating the correctness of cycle pump position, is always true by definition if there are some changes made to the cycle pump. It will never turn to false unless the changes are made to the layer of cycle ways. In this case, the user will become aware of the cycle pump status and position and can edit them appropriately. Regarding a cycle pump, attributes such as object number, object type, and geometry are entered by the user. However, other attributes such as address and allowed area type are computed and allocated by the system after an acceptable change is made to the traffic database.

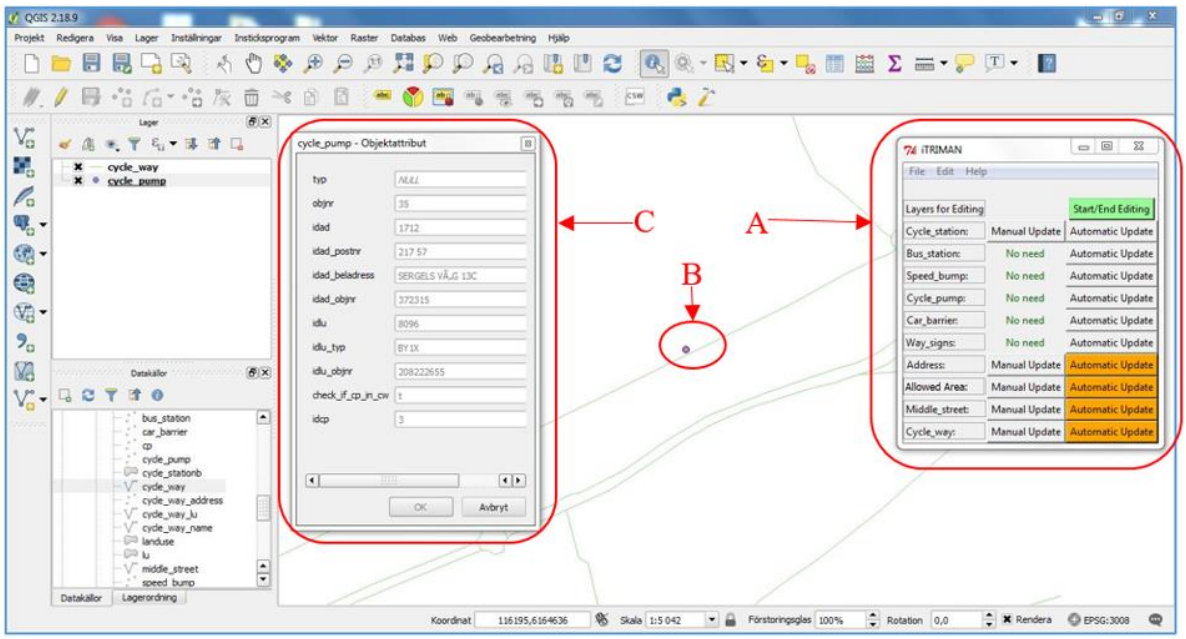

Figure 17: Both iTRIMAN desktop and QGIS are open. It shows also attributes of a cycle pump object, which is near to a cycle way object. (A) iTRIMAN interface, (B) Created new cycle pump, (C) Attributes of new created cycle pump.

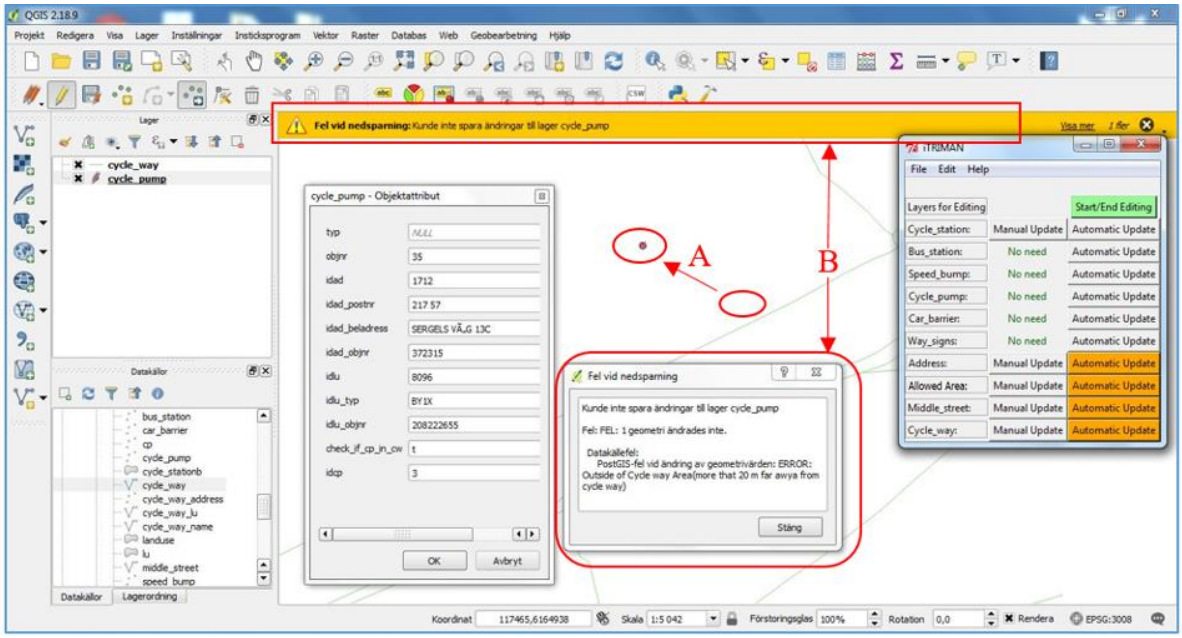

Figure 18: when want to save the new location of cycle pump, user will face with an error message, which prevent him from saving new location to the cycle pump object. (A) Shows changing the geometry of a cycle pump, (B) Shows error message, which prevent user to save new location.

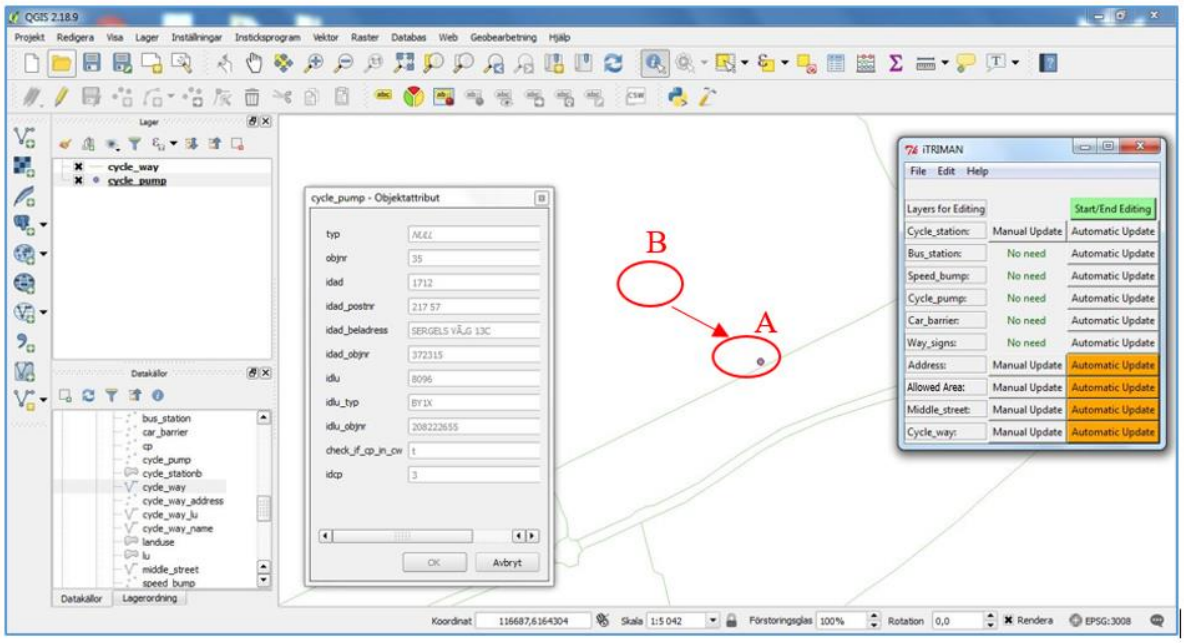

Figure 19: iTRIMAN (trigger) will not accept new location, which far away from cycle way and return back cycle pump object to the original location. (A) Original location, (B) New location which is not accepted.

4.3 Address

Since this table has second-type trigger, many of other tables should be updated as a result of a change in this table. The traffic signs, bus station, speed bump, and car barriers tables were used with the address table to observe their automatic updates after every change in the address table. The abovementioned layers were opened with their tables in QGIS to display the automatic update (Figure 20). First, the start editing should be activated in QGIS and iTRIMAN user interface (Figure 21). Then, the respective triggers should be created in iTRIMAN by pushing the automatic update button (Figure 22). All of the attributes and features are entered by the user into the address table. However, the address attributes of other tables are computed and assigned automatically in accordance with their positions (Figure 23). Therefore, an object is selected in the address layer by changing its geometry. After applying spatial changes to the object address, the attributes are recomputed and assigned to the abovementioned objects automatically.

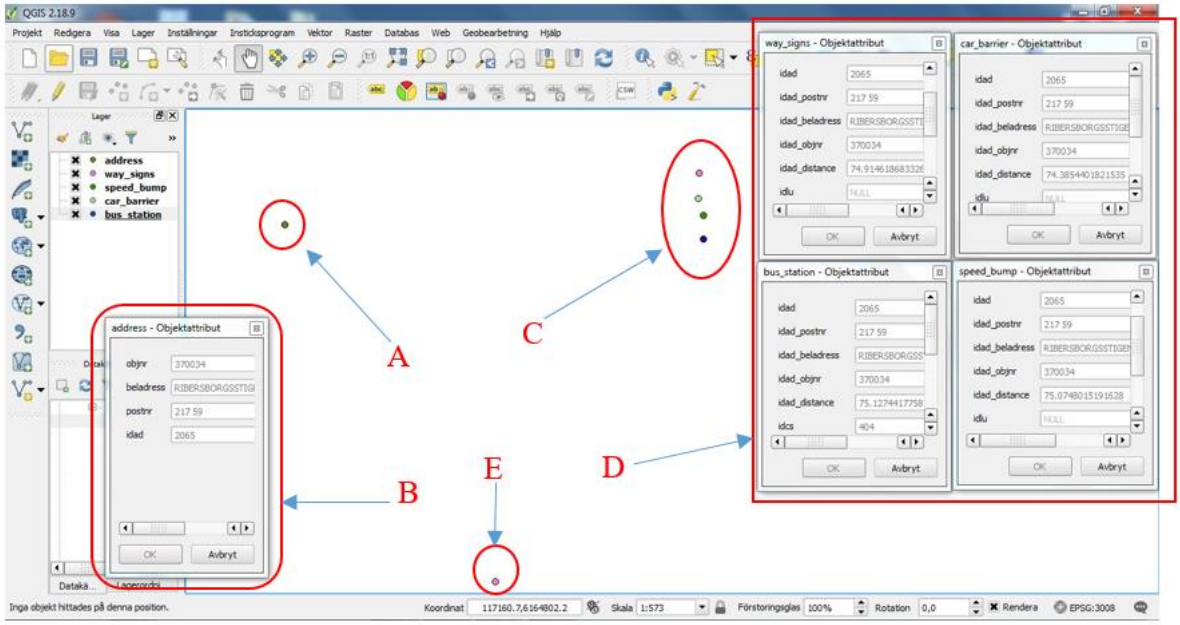

Figure 20: Shows attributes of four objects, which have a nearest address.(A) Address object,(B) Attribute of address object A,(C) Four different object (way signs, car barrier, bus station and speed bump) ,(D) Attributes of four mentioned objects, (E) Address object.

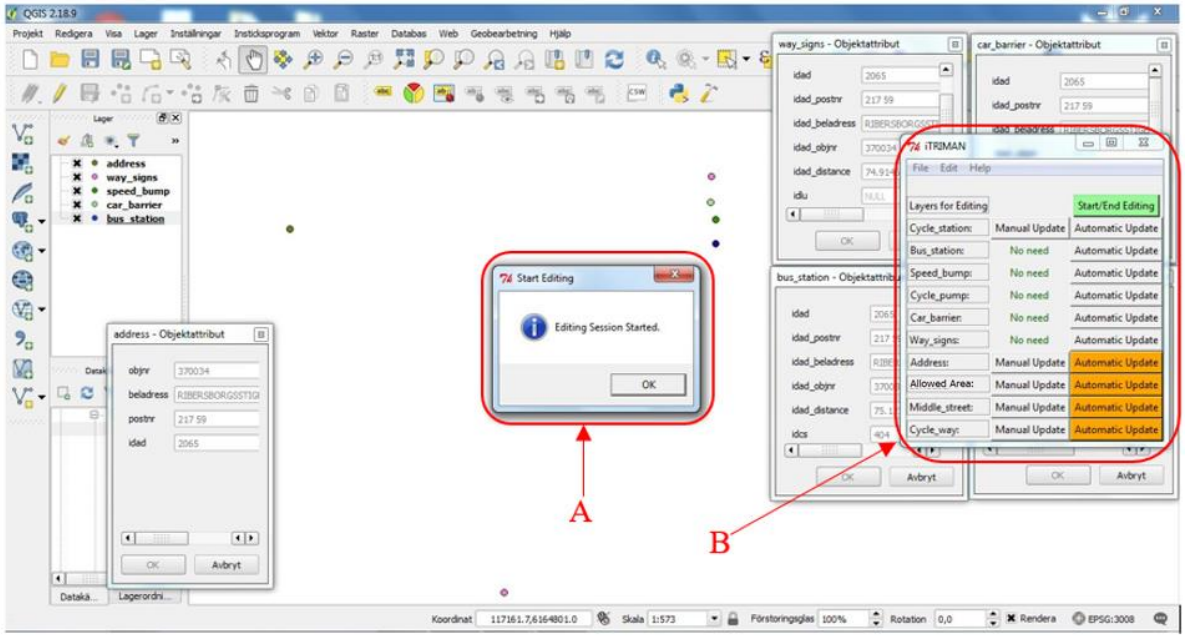

Figure 21: Shows iTRIMAN is open and “Start/End Editing” button is clicked. (A) iTRIMAN interface, (B) message shows editing session is started.

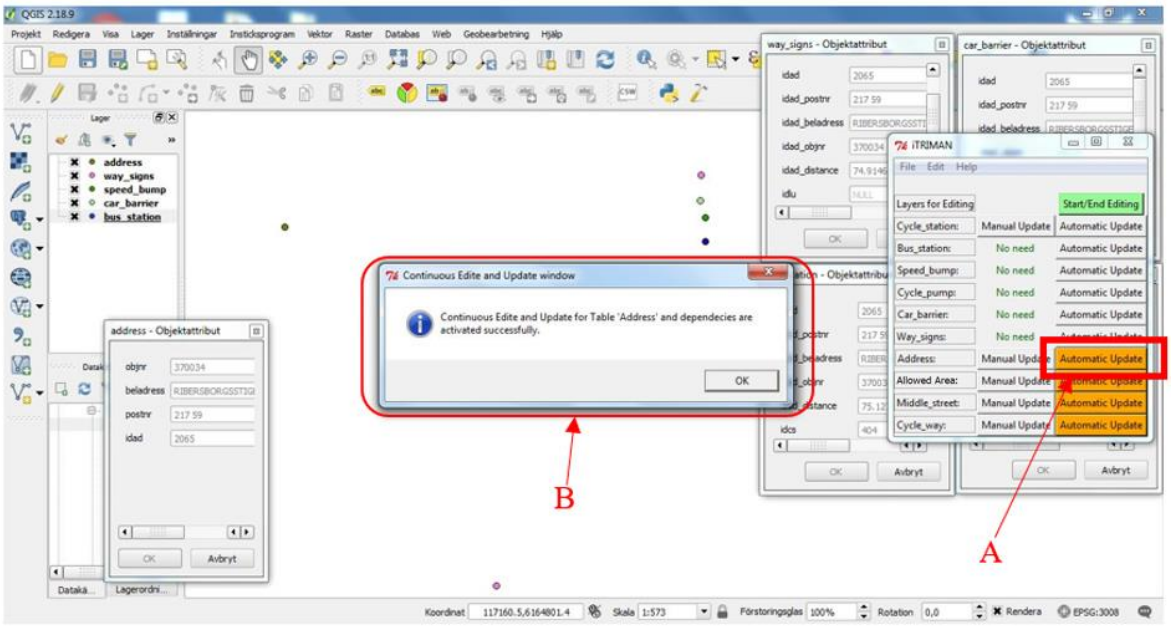

Figure 22: shows “Automatic Update” button is clicked and automatic update for address table is activated. (A) Automatic update button is clicked, (B) Message showing automatic update is activated.

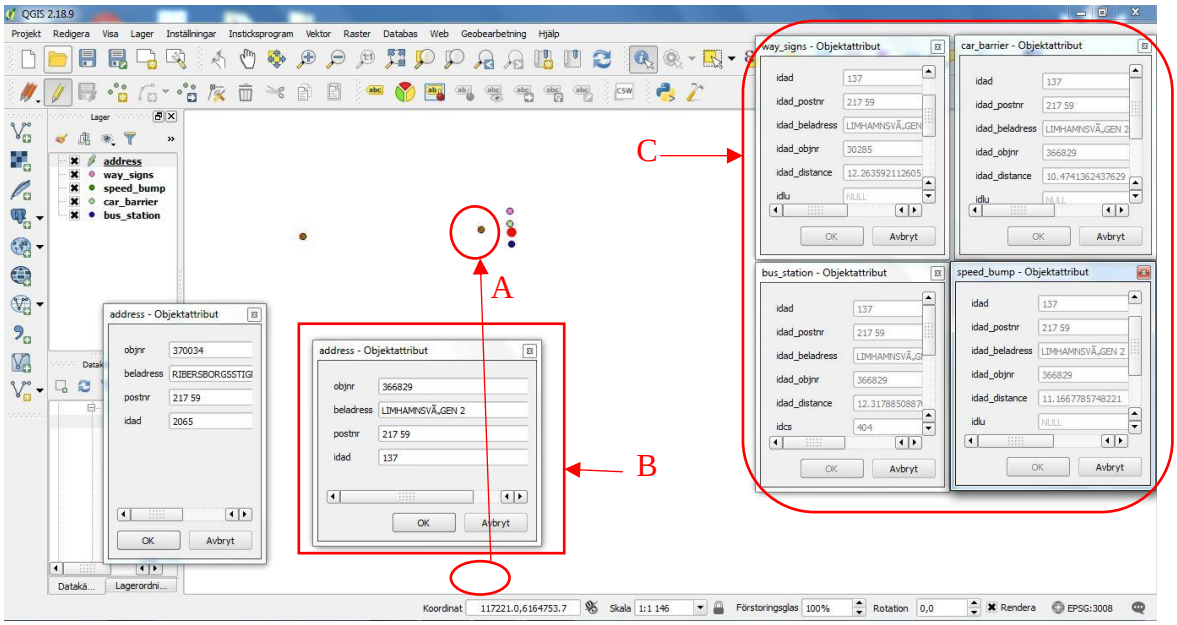

Figure 23: shows new nearest address abject for four objects by moving closer an address object to four objects. (A) Address object moved from its old position to a new position, (B) Shows attributes of moved address object (C) All of address attribute of four different objects (way signs, car barrier, bus station and speed bump) are changed.

5 Evaluation

There are 10 different requirements that were identified for the creation of the traffic database of GIS section. Triggers have been used to assign the attributes automatically and to satisfy the requirements. Triggers can calculate and assign attributes automatically from an object to another object after any change but have infinite loop problem. To resolve the problem, the iTRIMAN interface was designed in two versions: desktop and web. These interfaces create the triggers on the traffic database when needed and delete them when they are not needed.

The first and the second requirements (A, B) mentioned in the requirement analysis section are related to the assignment of the attributes of the area objects to point and linear objects. Using the first and the second types of triggers and the iTRIMAN interface, attributes are calculated and assigned from allowed area objects to the point and linear objects, which are within the allowed area. If the point and linear objects are outside of the allowed area, the value of the null is assigned to them which will be very fast when using the first type of triggers.

The third requirement (C) mentioned in the requirement analysis is the assignment of cycle way attributes to the address objects. This requirement is satisfied by creating a many to many relationship and the relevant table, which is created based on the two object classes of the cycle way and the address. This table is updated automatically by the use of triggers and the iTRIMAN interface after any changes in these object classes.

According to the fourth requirement (D), street names should be assigned to the nearest cycle ways. This requirement is satisfied in the traffic database as a many to many relationship and the relevant table, which is constructed based on street and cycle way. This table is automatically updated by using the triggers and the iTRIMAN interface. Then, this table is updated automatically after any changes in each of the two object classes.

The fifth requirement (E) is to assign the cycle station's attribute to the nearest bus station along with the distance between the two. For this purpose, the triggers and iTRIMAN interface have been used to calculate nearest cycle station for each bus station and then assign cycle station's attributes along with the distance between these two objects to the bus station. This process is completely automatic and after any changes of the two object classes, the bus station will be automatically updated.

The sixth requirement (F) in the requirement analysis section is related to the nearest address. Based on this requirement, the attributes of the nearest address are extracted for any given point object and get assigned to that point object. To assign the attributes of the nearest address to the point object, the distance between these two objects must be less than 100 meters. Triggers and iTRIMAN interface calculate the nearest address attributes and assign them to the point object and then update it after any changes.

The seventh requirement (G) is to assign the street attributes to traffic signs. To satisfy this requirement, the attributes of the nearest street to all traffic signs will be assigned to them. Triggers and iTRIMAN interface have been used for this purpose to automatically assign attributes from the nearest street to traffic signs and after any changes, it will be automatically updated.

The eighth and the ninth requirements (H, I) are related to the integration of point and linear objects with an allowed area and showing it in a separate column. Based on this requirement, a separate column has been created for each point and linear object class, which indicates the position of the point and linear objects in relation to the allowed area. This is an automatic process, which is made by triggers and iTRIMAN interface and with any change in the mentioned objects; the column will automatically be updated.

The tenth requirement (J) is related to the integration between the cycle pump and the cycling ways. According to this requirement, cycle pumps can only be created at a certain distance from the cycle ways and the cycle pumps created outside the specified distance are not accepted by the system. This requirement is automated with the triggers and iTRIMAN interface, which updated automatically with any change in the cycle pumps or the cycle ways.

Considering above, all the requirements are satisfied. Our test shows that an active traffic database that works based on triggers could be a possible solution for making smart the traffic database of the GIS section in Malmö municipality. The use of triggers increase the consistency, integrity, and automatic attribution in the traffic database.

6 Discussion

This study reached the first objective by identifying the requirement for creating a smart traffic database. The second objective was achieved by studying three techniques for creating smart databases including: active database, deductive database, ontology and semantics techniques. Considering the current technical and financial capabilities and preferences of the GIS section, the active database method was selected for the traffic database.

The third objective was achieved by developing a smart traffic database based on the selected technique. The fourth objective was met by evaluating the implementation of the selected technique according to the requirement analysis. There were certain challenges in the design and completion of this traffic database. In this study we could provide address some of them. However, other challenges still remain for future works. Both groups of these challenges should be taken into account when the traffic database is used. Triggers are used to update an automated traffic database (the main advantage). But still there are some disadvantage or challenges for active method like triggering loop, triggers runtime, new user definition and edited v.s. saved attributes.

6.1 Challenges

A challenge of designing the traffic database was the infinite fire of triggers. It was met by designing and completing iTRIMAN user interface. However, in the current implementation, several triggers relevant to different tables cannot be triggered at the same time. The long execution time of the triggers is another challenge which still persists in some tables after completing the traffic database. This traffic database is also characterized by the ability of edit saved attributes and the inability to edit derived attributes.

6.1.1 The Execution time of Triggers

There are two types of triggers in the traffic database. The first type updates an object instantly and very quickly, whereas the second type updates other tables which may include a large number of objects. The second-type triggers are very time-consuming to run due to the large size of updates (Table 3). Many tables in the traffic database use the first-type triggers. However, tables such as addresses and allowed areas use the second-type triggers. If there is a small number of edits in these tables, automatic update can be applied. If there are a large number of edits, real-time update will be time consuming. Thus, the update should be run once after applying the entire changes.

6.1.2 The Infinite Firing of Triggers

Using many triggers simultaneously can result in infinite loop of triggers. The reason is that updating the values of one table, say table 1, may also update in another table, say table 2 and accordingly updating table 2 may change some values in table 1. Such a situation may create infinite loop. It is also worth to mention the infinite loop is not because of error in modeling. iTRIMAN user interface helps in managing this situation. However, if the interface is used inappropriately (such as the creation of all triggers at the same time), the infinite loop of triggers will still remain in the traffic database.

6.1.3 New Users Definition and Their Concurrent Access to the Database

In the current traffic database, each new user needs to be individually defined. All the relevant parameters are defined separately for each user to access the traffic database. Then, while a table is edited by a user in the traffic database, only the trigger which belongs to this user is able to execute the function. Other triggers which are related to other users for this table will not execute the functions.

Users are divided into traffic database administrator (DBA) and ordinary users. The traffic database admin is able to use both types of triggers but ordinary users are limited to use only first-type of triggers. This limitation has been made because of the second-type of triggers are time consuming and result to time interference.

6.2 Suggestions and Future Works

Since the triggers of the first type are very effective and quick, it is suggested to develop and use them. Regarding the second type of triggers, it should be noted that running the triggers is time consuming for a large size of data.

iTRIMAN has a significant role in generating a smart traffic database. It is suggested to develop iTRIMAN and turn it into a professional user interface so that it can control trigger parameters in addition to creating and deleting triggers. Then the user can enter necessary parameters into iTRIMAN user interface without referring to SQL codes.

When the traffic database is used, a mechanism is required for rolling back the traffic database if any unwanted edits occur. It is also necessary to design a mechanism, which can automatically make periodic backups.

It is also recommended to consider testing and using ontologies and semantic approaches for creating smart traffic databases as future studies.

7 Conclusions

This thesis had four objectives which were achieved successfully. Requirements analysis was conducted and ten rules were identified. Three different kinds of smart database including ontology and semantic in database, active and deductive database
es were studied, and then active database was selected for test in this study. Then implemented techniques were evaluated against requirement analysis.

The traffic database was designed and implemented in the first step in which certain interviews were conducted with GIS section personnel to define all of the requirements as essential rules. Then a conceptual model was designed. After that, a logical model was devised. Finally, the traffic database was implemented practically.

In the second step, triggers were used to implement the rules and update the traffic database automatically. The use of triggers increased consistency, integrity, and automatic attribution in the traffic database. Triggers are mainly based on ECA (event, condition, and action). The trigger-generated smart traffic database is an active traffic database which can automatically compute values and attributes for objects and update them after every change. There were 26 triggers designed and implemented in this traffic database. They were divided into two types. In the first type, triggers were executed on the same tables of those triggers. On the other hand, the triggers of the second type were executed on the other tables. The triggers of the first type are very fast. They can update respective tables very quickly.

References

1) Bernstein, P.A., and B.T. Blaustein, 1982. Fast methods for testing quantified relational calculus assertions. In Proceedings of the 1982 ACM SIGMOD international conference on Management of datapp, 39-50.
2) Bernstein, P.A., B.T. Blaustein., and E.M. Clarke, 1988. Fast maintenance of semantic integrity assertions using redundant aggregate data. In Readings in Artificial Intelligence and Databases, pp.457-467.
3) Decker, H., and D. Martinenghi., 2008. Database Integrity Checking. In M. Khosrow-Pour, editor, Encyclopedia of Information Science and Technology (Second Edition), volume II, pp.961–966.
4) Egenhofer, M., F. Fonseca, C. Davis, and K. Borges, 2000. Ontologies and Knowledge Sharing in Urban Gis. CEUS - Computer, Environment and Urban Systems, Volume 24, Issue 3, pp.251-272.
5) Elmasri, R., Navathe, Sh.B., 2011. Fundamentals of database systems, 6th edition, ISBN-13: 978-0-136-08620-8.
6) Eswaran, K., and D. Chamberlin, 1975. Functional specifications of a subsystem for database integrity. In Douglass S. Kerr (Ed.), Proceedings of the First VLDB, Framingham, MA: ACM Press, pp.48-68.
7) Fonseca, F., 2001. “Ontology-Driven Geographic Information Systems”. Phd Thesis. University of Maine.

8) Friis-Christensen, A., N. Tryfona, and C.S. Jensen, 2001. November. Requirements and research issues in geographic data modeling. In Proceedings of the 9th ACM international symposium on Advances in geographic information systems, pp. 2-8.
9) Gaurav J., B. Simmi, 2012. Hierarchical Model Leads to the Evolution of Relational Model, International Journal of Engineering and Management Research, Vol. 2, ISSN No.: 2250-0758, pp.11-14.
10) Grady, B., I. Jacobson, and J. Rumbaugh, 2005. Unified Modeling Language User Guide, The (2 ed.). Publisher: Addison-Wesley, ISBN 0321267974.
11) Guarino, N., 1998. "Formal Ontology and Information Systems". In: Formal Ontology and Information Systems (FOI´S 98). Italy, 1998.
12) Hardy, P. G., 2000. Multi-scale database generalisation for topographic mapping, hydrography and web-mapping, using active object techniques. International Archives of Photogrammetry and Remote Sensing, 33(b4/1; part 4), pp.339-347.
13) Hardy, P. G., K.r. Haire, R. Sheehan, and P. A. Woodsford, 2001. Mobile Mapping On-Demand, Using Active Representation and Generalisation. In Proceedings of the 20th International Cartographic Conference, Beijing, China, pp.3239-3247.
14) Jha, A. K., 2015. An introduction to deductive database and its query evaluation. International journal of advanced computer technology, volume 3, number 3.
15) Lausen, G., B. Ludäscher, and W. May, 1997. On active deductive databases: The statelog approach. In Workshop on (Trans) Actions and Change in Logic Programming and Deductive Databases, Springer, Berlin, Heidelberg, pp. 69-106.
16) Mansourian, A., L. Harrie, 2012. Geographical Databases, Lecture Notes, GIS Center, Lund University.
17) Muñoz, A., and J. Aguilar, 2007. Ontological scheme for intelligent database. In Proc. of the 11th WSEAS Int. Conf. on Computers, ICCOMP, pp.1-6.
18) Praveen, S., Chandra, U. and Wani, A.A., 2017. A Literature Review on Evolving Database. International Journal of Computer Applications, 162(9).
19) Ranjana I., R. Khandal, R. Mohare, 2015. Comparison of HDBMS, NDBMS, RDBMS and OODBMS, International Journal of Advance Research in Computer Science and Management Studies, Volume 3, Issue 6, ISSN: 2321-7782 (Online)
20) Tveite H., January 1997. Data Modelling and Database Requirements for Geographical Data.
21) Vaz, D., M. Ferreira, and R. Lopes, 2007. Spatial-yap: a logic-based geographic information system. Logic Programming, pp.195-208.
22) Viegas, R., and V. Soares, 2007. Querying a geographic database using an ontology-based methodology. In Advances in Geoinformatics. Springer Berlin Heidelberg, pp.165-182.

Online References

1) Paul and Margaret Hardy's home page at www.pghardy.net, derived at 2017-08-30.
2) PostgreSQL documentation at www.postgresql.com, derived at 2017-09-05.

Institutionen för naturgeografi och ekosystemvetenskap, Lunds Universitet.
Studentexamensarbete (seminarieuppsatser). Uppsatserna finns tillgängliga på institutionens geobibliotek, Sölvegatan 12, 223 62 LUND. Serien startade 1985. Hela listan och själva uppsatserna är även tillgängliga på LUP student papers (https://lup.lub.lu.se/student-papers/search/) och via Geobiblioteket (www.geobib.lu.se)

The student thesis reports are available at the Geo-Library, Department of Physical Geography and Ecosystem Science, University of Lund, Sölvegatan 12, S-223 62 Lund, Sweden. Report series started 1985. The complete list and electronic versions are also electronic available at the LUP student papers (https://lup.lub.lu.se/student-papers/search/) and through the Geo-library (www.geobib.lu.se)

408 Julia Schütt (2017) Assessment of forcing mechanisms on net community production and dissolved inorganic carbon dynamics in the Southern Ocean using glider data
409 Abdalla Eltayeb A. Mohamed (2016) Mapping tree canopy cover in the semi-arid Sahel using satellite remote sensing and Google Earth imagery
410 Ying Zhou (2016) The link between secondary organic aerosol and monoterpenes at a boreal forest site
411 Matthew Corney (2016) Preparation and analysis of crowdsourced GPS bicycling data: a study of Skåne, Sweden
412 Louise Hannon Bradshaw (2017) Sweden, forests & wind storms: Developing a model to predict storm damage to forests in Kronoberg county
413 Joel D. White (2017) Shifts within the carbon cycle in response to the absence of keystone herbivore *Ovibos moschatus* in a high arctic mire
414 Kristofer Karlsson (2017) Greenhouse gas flux at a temperate peatland: a comparison of the eddy covariance method and the flux-gradient method
415 Md. Monirul Islam (2017) Tracing mangrove forest dynamics of Bangladesh using historical Landsat data
416 Bos Brendan Bos (2017) The effects of tropical cyclones on the carbon cycle
417 Martynas Cerniauskas (2017) Estimating wildfire-attributed boreal forest burn in Central and Eastern Siberia during summer of 2016
418 Caroline Hall (2017) The mass balance and equilibrium line altitude trends of glaciers in northern Sweden
419 Clara Kjällman (2017) Changing landscapes: Wetlands in the Swedish municipality Helsingborg 1820-2016
420 Raluca Munteanu (2017) The effects of changing temperature and precipitation rates on free-living soil Nematoda in Norway.
421 Neija Maegaard Elvekjær (2017) Assessing Land degradation in global drylands and possible linkages to socio-economic inequality
422 Petra Oberhollenzer, (2017) Reforestation of Alpine Grasslands in South Tyrol: Assessing spatial changes based on LANDSAT data 1986-2016
423 Femke, Pijcke (2017) Change of water surface area in northern Sweden
424 Alexandra Pongracz (2017) Modelling global Gross Primary Production using the correlation between key leaf traits
425 Marie Skogseid (2017) Climate Change in Kenya - A review of literature and evaluation of temperature and precipitation data
426 Ida Pettersson (2017) Ekologisk kompensation och habitatbanker i kommunalt planarbete
427 Denice Adlerklint (2017) Climate Change Adaptation Strategies for Urban Stormwater Management – A comparative study of municipalities in Scania
428 Johanna Andersson (2017) Using geographically weighted regression (GWR) to explore spatial variations in the relationship between public transport accessibility and car use : a case study in Lund and Malmö, Sweden
429 Elisabeth Farrington (2017) Investigating the spatial patterns and climate dependency of Tick-Borne Encephalitis in Sweden

430 David Mårtensson (2017) Modeling habitats for vascular plants using climate factors and scenarios - Decreasing presence probability for red listed plants in Scania
431 Maja Jensen (2017) Hydrology and surface water chemistry in a small forested catchment : which factors influence surface water acidity?
432 Iris Behrens (2017) Watershed delineation for runoff estimations to culverts in the Swedish road network : a comparison between two GIS based hydrological modelling methods and a manually delineated watershed
433 Jenny Hansson (2017) Identifying large-scale land acquisitions and their agro-ecological consequences : a remote sensing based study in Ghana
434 Linn Gardell (2017) Skyddande, bevarande och skapande av urbana ekosystemtjänster i svenska kommuner
435 Johanna Andersson (2017) Utvärdering av modellerad solinstrålning i södra Sverige med Points Solar Radiation i ArcGIS
436 Huiting Huang (2017) Estimating area of vector polygons on spherical and ellipsoidal earth models with application in estimating regional carbon flows
437 Leif Holmquist (2017) Spatial runner: environmental and musical exposure effects on runners through an idealized routing network
438 Adriana Bota (2017) Methodology for creating historical land use databases – a case study for ICOS-station Hyltemossa, Sweden
439 Michael Araya Ghebremariam (2017) Urban flood modelling: a GIS based approach in Lomma, Skåne region
440 Stina Sandgren (2017) Climate change impact on water balance and export of dissolved organic carbon - a sub-catchment modelling approach
441 Karla Münzner (2017) Variability and regulation of the planktonic respiratory quotient in a eutrophic lake (Lake Vombsjön) in summer 2016
442 Bastian Berlin (2017) Modeling the Weibull shape parameter to improve estimates of the annual wind energy potential in Sweden
443 Christine Walder (2018) Humpback whale (*Megaptera novaeangliae*) location in Southeast Alaska: modeling the influence of mesoscale krill (*Euphausiacea*) patch depth and size
444 Astrid Zimmermann (2018) Projecting invasive species using remote sensing and spatial explicit models
445 Linnéa Larsson (2018) Identifiering av riskområden för extremer av markfuktighet med Soil Topografic Index : Jordbruksmarken i Helsingborgs kommun i nutida och framtida perspektiv
446 Sayed Hassan Alavi (2018) Constructing and developing an integrated smart spatial database for Malmö Municipality, Case study: Västra innerstaden in Malmö