AALBORG UNIVERSITY
Department of Computer Science

Real-Time Queries and Analysis on Moving Cars

MASTER THESIS

Johann Gunnar Hermannsson
Thorir Olafsson

June 2005

Aalborg University
Department of Computer Science

RESEARCH AREA:
Database and Programming Technologies

TITLE:
Real-Time Queries and
Analysis on Moving Cars

PROJECT PERIOD:
1. February 2005 -
1. Juni 2005

PROJECT GROUP:
d634a

GROUP MEMBERS:
Johann Gunnar Hermannsson
Thorir Olafsson

SUPERVISOR:
Janne Skyt

CENSOR:
Martin Jensen, Oracle Denmark

NUMBER OF COPIES: 6

REPORT PAGES: 77

APPENDICES:
19 pages
Attached CD-ROM

ABSTRACT:

This report documents the design and empirical evaluation of a system architecture, where the driving history of personal cars is stored in a data warehouse. Along with road network information, this driving history is used to answer real-time queries, triggered by incoming GPS signals. These real-time queries check whether the sender is heading towards a traffic delay.

Experimental results are reported, evidencing the reasonable performance of the proposed architecture.

Preface

This report is a documentation of a M. Sc. project which was conducted during the spring semester of the study year 2004/2005.

This Master Thesis is the final part of a larger project which has elapsed from the spring of 2004. The main purpose of the project was to design and verify the possibility of building a system where GPS signals from moving cars are stored in a data warehouse, which supports real-time queries, triggered by incoming GPS signals, as well as general traffic analysis.

We would like to thank our supervisor, Janne Skyt, for her invaluable support, guidance and valuable comments during the project work.

1. June 2005

_____

Johann Gunnar Hermannsson

_____

Thorir Olafsson

Contents

Chapter 1

Introduction

This report introduces the "Moving Objects Data Warehouse" architecture, which has a dimensional model at its core, and the report documents its properties and experimental results. The dimensional model stores the history of personal cars and the architecture supports both traffic analysis and near real-time queries based on incoming GPS signals.

In areas where many cars are traveling through a road network at the same time, e.g., in cities, there can be many situations where drivers of cars are delayed in traffic because of general traffic delays, traffic accidents, construction work or other similar reasons.

Our main goal is to help drivers to avoid delays as much as possible, by informing them of delays beforehand. In order to fulfill this goal, the "Moving Objects Data Warehouse" architecture is proposed. A system, based on this architecture, could log GPS signals from moving cars and notify drivers about possible traffic delays in their probable future paths. When receiving an incoming signal, the system would activate a process, which results with the sending of a notification to the driver. Naturally, it is desirable that the driver receives the notification about traffic delays before he is stuck in one.

The system stores the history of the movement of cars in order to be able to predict the future path of cars. Furthermore, if driver is close to a traffic delay, the system shall also notify the driver.

The main users of this system are drivers of cars, which send their GPS position to the system. The system aims to help them to avoid the traffic delays. Besides helping the drivers, the system supports traffic analysis queries, which can be used to observe traffic patterns and behavior.

Our goal is to design and verify the possibility of building a system, able to serve Aalborg city in terms of number of cars and size of an area. Relevant numbers regarding Aalborg city are presented in Chapter 3.

The remainder of this paper is organized as follows. In Chapter 2, the evolution of the project is presented. The first of two parts of this report is dedicated to discussions on the proposed architecture. In Chapter 3, initial system requirements

are specified as well as detailed assumptions, in order to compare different model candidates. The comparison is then documented in Chapter 4. In Chapter 5, the system architecture of the Moving Objects Data Warehouse is described with respect to components and functionality. The second part consists of a discussion on model experiments and its evaluation. In Chapter 6, the design and purpose of experiments are introduced and in Chapter 7, the data set which is used in the experiments is introduced while experimental results are presented in Chapter 8. The evaluation of the experiments is then discussed in Chapter 9. Finally, Chapter 10 concludes this report. Appendices A, B and C present calculations, related to the comparison of models (discussed in Chapter 4), while source code is in Appendix D (on attached CD-ROM).

Chapter 2

Project Process

While the work documented in this report was conducted in the spring of 2005, it is a part of a larger project, which has elapsed from the spring of 2004. In this chapter, we briefly describe the evolution of the project in order to clarify the main challenges of our current work.

The larger project was initiated by an idea of using data warehouse technologies to store and query the history of moving objects. We concentrated on moving cars on a road network. A dimensional model was built without thorough research on this subject. We got data from cars in Aalborg city with GPS receivers and we made some first experiments in a prototype way.

The results from these experiments (presented in [5]) showed that storing the history of moving cars in a data warehouse was possible and with some adjustments, we believed that answering history-related near real-time queries could be a possibility. In order to build a foundation for these additional functionalities we took a step back and made a thorough analysis on issues, relevant to our problem. That was the focus of the next part of the project.

In the second part of the larger project (presented in [6]), we concentrated on making thorough theory research. Based on our analysis, we adjusted the requirements for the system. The near real-time queries on the system were now top priority, i.e., to be able to notify a specific driver about possible traffic delay in his route based on the history of his routes. In order to be able to map match GPS signals to specific roads (or road segments) and generally match the information to real-world locations, a road network representation was required. The research focused mainly on three different issues that were most relevant to our problem. These issues were; moving objects, data warehousing techniques, and road network representations.

Based on our analysis, we proposed a system architecture (see Figure 2.1) with three main components, i.e., a logging database, a road network database, and a data warehouse. The Logging database is responsible for receiving the incoming GPS signal from the moving cars and forward some signals to check whether there is a traffic delay in the drivers' route. Furthermore, the Logging database stores all the incoming signals until an ETL procedure loads them into the Data warehouse.

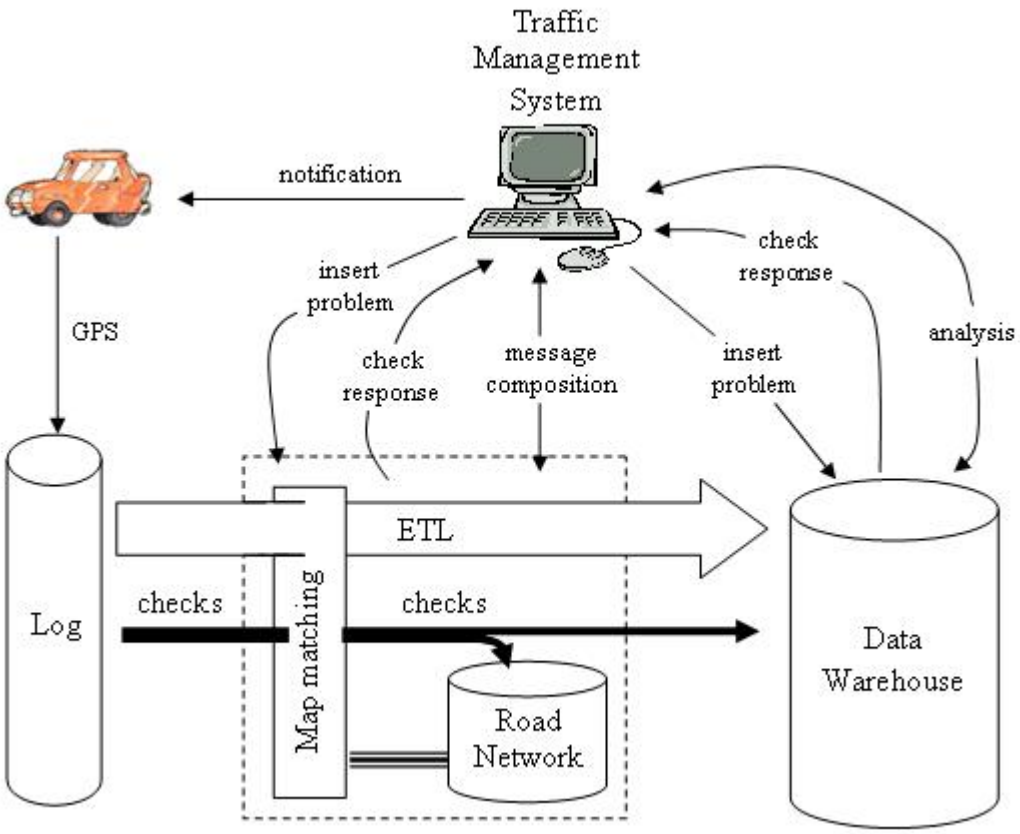

Figure 2.1: The system architecture proposed in [6].

The Data warehouse stores the history of the movement of cars in a suitable way for retrieving the data efficiently, both with respect to the near real-time queries and general traffic analysis. The Road network is outside the data warehouse and combined with Map matching is responsible for matching each GPS signal to a specific road segment. Furthermore, the Road network handles a part of the near real-time queries as well as being used when composing a descriptive driver notification. The overlaying Traffic management system takes care of driver communication and traffic analysis control.

With this architecture, there were open issues that we needed to examine in more detail before selecting the final architecture/models to be implemented for experiments. These issues were:

- **How to make a real-world relevant evaluation?** The real-world settings have to be specified in order to create criteria for the other open issues to build on.
- **How to store incoming signals and process them for querying?** The GPS signals that are used for near real-time queries have to be retrieved fast

from the incoming signals without risking slowing down of the initial signal logging.

- **What course should be taken concerning the map matching procedure?** Today there exists map matching algorithms that match GPS signals to a specific road segment. The question that arose was whether we should tailor-make a map matching algorithm or leave the implementation of a map matching procedure for future work.

- **What data warehouse model should be chosen?** In our first approach, we used a technique of storing subroutes as facts. Another approach could be to store all signals as facts, decreasing the calculations of the ETL procedure and possibly increasing the information in the data warehouse.

- **Should the road network representation be inside or outside the data warehouse?** This question has been with us since we started out with the second part of the larger project. In the architecture that we previously proposed (see Figure 2.1), we decided to have the road network stored in a database outside the data warehouse, while the dimensional model of the data warehouse would include a small *location dimension*. In our theory research, we analyzed a model, proposed by Jensen et al. in [8], where a road network is stored as one dimension, using partial containment in locational hierarchy. This was to be explored in more detail along with other possible solutions to connecting the road network to the dimensional model of the data warehouse.

All these issues had to be dealt with before doing implementation and experiments based on the theory research. We begin the first part of this report on presenting system requirements, before comparing different methods and models.

Part I

Moving Objects Data Warehouse

Chapter 3

System Requirements

Before different models can be compared, some initial set of requirements for the system is needed. In this chapter, the system requirements are presented and, furthermore, detailed assumptions are introduced in order to support the comparison of models.

3.1 Initial Requirements

The initial requirements for the system reflect what we expect from the system. The requirements are divided into two sets. First, *System Scalability*, i.e., how large an input do we want the system to serve? Second, *System Functionality*, i.e., what requirements have to be met so that the system is able to serve its users? This section covers the first of the five open issues, presented at the end of Chapter 2, concerning real-world relevant evaluation.

3.1.1 System Scalability

The system is supposed to serve Aalborg city, both in terms of number of cars and size of a road network. Important numbers concerning Aalborg city are:

- Total number of motor vehicles in January 2005 was 65.531 vehicles, there of 52.264 passenger cars (according to [4]).
- Total size of Aalborg city is around 81 km^2.
- Aalborg city contains around 1.500 streets (according to [9]).
- These 1.500 streets can be divided into around 5.200 road segments.[1] A road segment is considered a directed part of the street, ranging between start/end points or larger crossroads of the street. For a by-directional street, two road

[1] This division is not based on statistical information. By examining the road network and overall layout of Aalborg city the numbers of segments, areas, and cells were estimated.

segments cover the space between points A and B (one for each direction), while, for a one-directional street, only one road segment covers the same space. Road segments are covering and non-overlapping, i.e., each location on the street is within one, and only one, road segment. Road segments are referred to as segments in the remainder of the report.

- Aalborg city can be divided into seven areas, each around 12 km^2 (areas not necessarily equal in size). [1]
- Aalborg city can be divided into around 300 equally sized cells. [1]

Based on the numbers above, we refine the exact numbers to be used when calculating the blow-up of different models.

- Up to 30.000 personal cars travel along the road network and send signals to the system at the same time.
- 50.000 personal cars in total are registered to the "service".
- 100.000 drivers in total are using these 50.000 cars (two drivers per car on average).

3.1.2 System Functionality

Some specific requirements are made for the GPS signals, sent to the system. Apart from including basic GPS information, i.e., time, date, coordinates, car, driver, and speed, the signals will have to match the three requirements below.

- Each driving car sends a signal every tenth second.
- When a car is ignited, a labeled signal is sent.
- Each signal includes the driving direction of the car in degrees.

Concerning the basic functionality of the system, below are requirements related to data processing and query execution within the system.

- The history in the data warehouse is stored for one year.
- The system shall be able to receive continuously incoming signals without any delays.
- Routine check is taken with 30 seconds intervals to check whether problems exist in a possible near-future path of a car, with respect to the cars position and driving direction.

  - A routine check consists of four steps; retrieval of relevant signals from all the incoming signals, map matching the signal to a road segment, querying the road network data, and returning relevant information about problem(s), if any, in the possible near-future path of the car.
  - A routine check comes out positive when a car can drive upon a problem by taking up to two segments from their current location. Two segments are used in order to increase the possibility that the problem existence is relevant to the driver.
  - Each routine check shall not take more than 10 seconds in the system, i.e., from the time the signal is initially received until the four steps of the routine check are completed. As the routine check returns positive results when a car is getting close (two segments) to a problem, the response must be fast in order to help the driver. That is why the 10 seconds are chosen.

- A history-based check is taken every time a driver starts his car.
  - A history-based check consists of four steps; retrieval of start signals from all the incoming signals, map matching the signal to a road network, querying the history of the driver, and returning relevant information about problem(s), if any, in the estimated future path of the car. The estimated future path of the car is based on the routes, which the driver has traveled frequently when starting at a specific segment, weekday, and time of day period. A route is considered frequent if it has been taken more than 50 times over the last year.
  - Each history-based check shall not take more than 90 seconds in the system, i.e., from the time the signal is initially received until the four steps of the history-based check are completed. As the history-based check looks for problems in the whole route of the driver, the response does not need to be as fast as for the routine check. In most cases, 90 seconds should be fast enough to be useful for the driver.

- Real-time queries (i.e., routine and history-based checks) shall return enough information to be able to notify drivers of problems, i.e.; the street where the problem is located, nearest junctions (when applicable) in order to pinpoint the problem to a specific location inside the street, description, criticality, and estimated end time of the problem.

- Several different traffic analysis queries can be made on the history of drivers. These queries can be made with respect to cars, drivers, days, routes, etc. Traffic analysis queries can vary with respect to how often they are used and how much calculations are required. This is the reason for the fact that traffic analysis queries, in general, are not assigned a specific upper limit for execution time.

- The ETL procedure, loading data into the data warehouse, shall be executed on 24-hour basis and shall take at most 24 hours. That time includes the map matching of signals to segments. As the history-based check is made on a whole year of data, it is not considered important whether one-day-old information is included. That is the reason for the fact that 24 hours are chosen as an upper limit for the execution time of the ETL procedure.
- Inserting information on a problem to the system shall take at most 10 seconds.
- Disabling a problem in the system shall take at most 10 seconds.

These requirements are used for calculating which models are best suited for our purposes. Furthermore, these requirements are used when evaluating experimental results, in Chapter 9.

3.2 Detailed Assumptions

In addition to the requirements of Section 3.1, detailed assumptions are essential in order to calculate the scalability and blow-up of the different models we need to choose between. These assumptions give a clearer image on what data our system is supposed to handle. These additional assumptions are:

1. 5% of all segments are contained in two areas. [2]
2. 25% of all segments are contained in two cells. [2]
3. Each segment S is connected to six other segments, on average. [2]
    - Three segments that can be driven from and onto S, on average.
    - Three segments that can be driven onto, from S, on average.
4. Up to 300 cars start their car trip every 30 seconds. [2]
5. Cars travel for 32 minutes per day, on average. [3]
6. Cars travel four routes per day, on average. [4]
    - Each route consists of seven streets and 24 segments, on average.
    - Cars travel 1 previously untaken route (by him or any other car) per day, on average, for the first year.
    - Cars travel 0,5 previously untaken route (by him or any other car) per day, on average, for the second year.

[2] Estimation based on the real-world statistics on Aalborg city given in Subsection 3.1.1, where areas, cells and segments are introduced.

[3] Found from a real-world data set gathered from twelve personal cars driving in Aalborg city in a period of 112 days.

[4] Qualified guess.

- Cars travel 0,25 previously untaken route (by him or any other car) per day, on average, for the third year (and so on).

7. The equation for untaken routes is:

$$\sum_{i=0}^{m-1}(365 * n * \frac{1}{2^i})$$

where n = number of cars, and m = number of years. [4]

8. For each untaken route, the driver takes eight new subroutes, on average. [4]

- Many untaken routes consist of old subroutes, therefore we say that, on average, the driver takes eight new subroutes per untaken route, or 1/3 of the untaken route is a new route.

9. 1000 problems are inserted in one year. [4]

As we have tried to find realistic numbers for the assumptions above, it is not important if they are completely correct, as the same numbers are used for calculating the scalability and blow-up of all models.

Chapter 4

Model Comparison

Before deciding on the final architecture for implementation, several different design ideas had to be evaluated and compared to each other. This chapter documents the main issues when comparing different models and concludes on which design ideas were considered appropriate. Related to the open issues discussed in the end of Chapter 2, there are three main issues that need to be taken into account, i.e., the Logging Database, the Data Warehouse, and the Road Network representation. Regarding the other two open issues of Chapter 2, i.e., real-world relevant evaluation and the map matching procedure, they are covered in Chapters 3 and 5, respectively.

This chapter is divided into three sections, one for each of the above issues. When comparing different models, three aspects were in focus; size of tables, consequences for the ETL procedure, and complexity of queries. We use the initial system requirements and assumptions from Chapter 3 when calculating the size of tables, the amount of data to be used in queries, and the ETL procedure. We describe the calculations in this chapter and discuss the results of the calculations, while calculations are shown in detail in Appendices A to C, and source code (for queries, procedures, and triggers) is shown in Appendix D.

4.1 The Logging Database

The Logging database serves as an entry point for GPS data, on which the ETL procedure can be based. We compared two possible models to use for the Logging database, *The Interval model* and *The Triggering model* (both shown in Figure 4.1). Here, we introduce these two models, focusing on table blow-up and data flow within the models. Then, we compare the models and give arguments for the model we choose. We start by looking at the Interval model.

4.1.1 The Interval Model

The idea behind this model was that the data would travel between tables on regular intervals, mainly in order to control the frequency of routine checks.

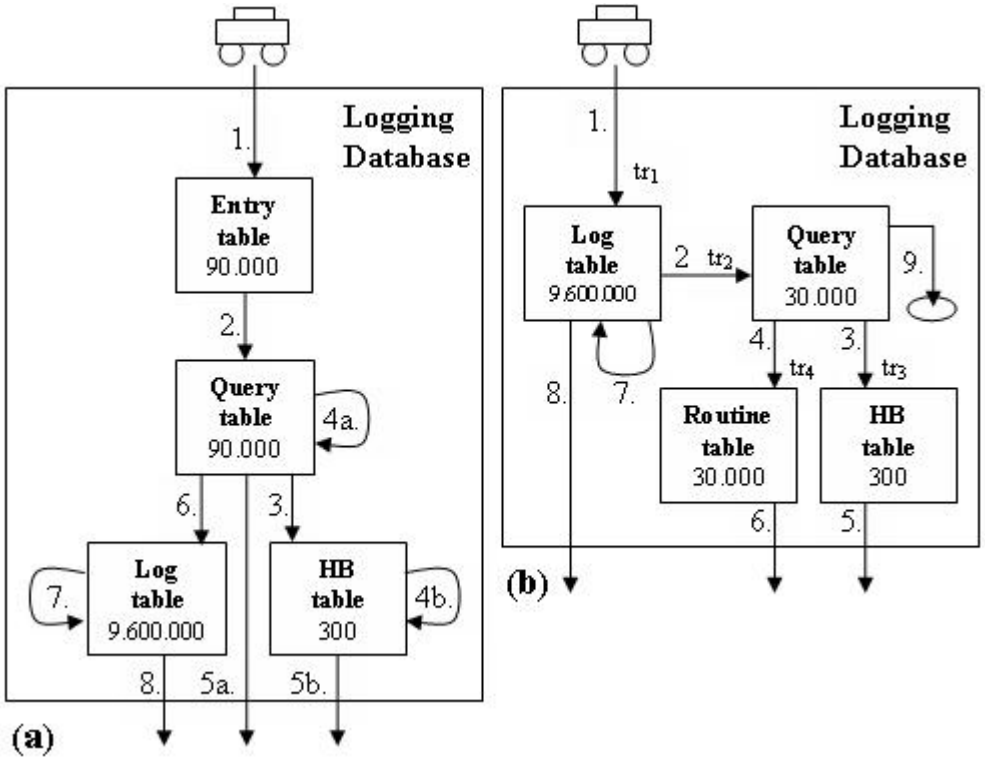

Figure 4.1: Architecture and maximum table sizes (in records) for (a) the Interval model and (b) the Triggering model.

Data flow

The flow of data inside the Interval model is described as this (numbers relate to arrows in Figure 4.1 a): (1) Continuous signals from cars are received at the *entry table*. (2) Signals are flushed from the *entry table* to the *query table* at 30 second intervals where (3) all start signals are forwarded to the *HB table* (HB stands for history-based). (4a) All signals in the *query table* are area-based map matched (see Subsection 5.2.2) and furthermore segment-based map matched when signals comes from problem areas. (4b) All signals in the *HB table* are map matched directly to segments (segment-based map matched). After being map matched, (5b) signals in the *query table* are used for routine checks (except for signals from outside problem areas). (5a) Signals in the *HB table* are used for history-based checks. At 30 second intervals, (6) all signals are flushed from the *query table* to the *log table* where (7) all signals, not previously map matched, are segment-based map matched. (8) The records of the *log table* are then extracted from the logging database by an ETL procedure on 24-hour basis.

Size of tables

The numbers inside the tables in Figure 4.1 show the maximum number of records for the tables, i.e., the estimated peak for number of records before being extracted from the tables or deleted. All numbers are found by using the numbers presented in Chapter 3. For all tables, except *log table*, we use the worst case numbers which state that up to 30.000 cars travel along the road network and send signals to the system at the same time and that up to 300 cars start their car trip every 30 seconds. Using those numbers and the fact that a signal is sent every 10 seconds, we get the numbers for the *entry table*, *query table*, and *HB table*. As an example, the *entry table* contains 90.000 records after having received signals from 30.000 cars over a 30 second period where each car sends one signal every 10 seconds.

As these numbers relate to heavy traffic, it is unrealistic to use them to calculate the size of the *log table*. For that, we use the assumed average traffic rate for one day which is 32 minutes per car. Every car would then send 192 signals per day (32 minutes * 60 seconds per minute * 0.1 signal per second) and as we use the number of 50.000 cars for our model calculations, this means that 9,6 million signals are logged every 24 hours (192 signals per car per day * 50.000 cars). All tables of this model have the same attributes and therefore it is enough to find the size of one record (in bytes) and then multiply it with the sum of all records in the database. The combined size of all tables of the Interval model is approximately 300 Mb (see Appendix A.1).

Now, we look at the other candidate, i.e., the Triggering model.

4.1.2 The Triggering Model

The idea behind this model was to use triggers to select signals to be used for real-time queries, supporting direct flow of data inside the model.

Data flow

The flow of data inside the Interval model is described as this (numbers relate to arrows in Figure 4.1 b): (1) Continuous signals from cars are received at the *log table* where (2) signals that shall be used for queries are sent to the *query table*. According to Trigger 2, signals are sent to either (3) the *HB table* or (4) the *routine table* where they are used for (5) history-based checks and (6) routine checks, respectively. (7) All signals in the *log table* are segment-based map matched and then (8) extracted from the logging database by an ETL procedure on 24-hour basis. (9) All records are deleted from the *query table* on thirty second intervals.

The direct flow of data in the Triggering model is based on the use of triggers. First, Trigger 1 goes through all incoming signals (tr_1) and filters out the ones that are to be used in real-time queries. A signal is not to be used for real-time queries when a signal from the same car exists in the *query table*. As the *query table* is emptied every 30 seconds, this means that, for one car, a real-time query is executed with around 30 second intervals. Second, Trigger 2 to chooses whether to make a

history-based or a routine check (tr_2). Finally, Triggers 3 and 4 execute the 'area-based' (tr_3) and 'segment based' (tr_4) map matching procedures on signals that are going to be used for real-time queries. The source code for the triggers is shown in Appendix D.

Size of tables

The size of the tables is found using the same assumptions as for the Interval model. The main difference is that the *log table* of the Triggering model takes over the responsibilities of both the *entry table* and *log table* of the Interval model. Furthermore, the *query table* has 30.000 records instead of the 90.000 stored in the *query table* of the Interval model. This is due to the fact that the first trigger (tr_1) sorts out only those signals that need to be considered for real-time queries into the *query table*. The calculations are done in the same way as for the Interval model as the tables have the same attributes in all tables in both models. The combined size of all tables of the Triggering model is also approximately 300 Mb (see Appendix A.2).

4.1.3 Choosing a Model

In order to decide on a model to implement there are four main aspects that have to be considered, i.e., the service the models provide to outside components, the blow-up of tables, consequences for the ETL procedure, and support for real-time queries.

Services

The logging database shall receive incoming signals and sort out signals for the real-time queries while all signals are made available for the ETL procedure.

Both models meet these basic requirements. While the signals are processed differently inside the models, they end up being in the same state and format when available for the real-time queries and the ETL procedure. Simply put, the models have the same inputs and outputs.

Size of tables

The two models are very similar in size and the blow-up is based on the numbers from Subsection 3.1.1, where we introduced the 50.000 cars as the total number of cars in Aalborg city. These numbers gave the overall size of both databases as around 300 Mb. An overall size of around 300 Mb can not be considered too large for a database and by far decisive for which model to choose, especially as the size of the two models are almost the same.

The ETL procedure

Before running the ETL procedure on the data in the *log table* in either model, all signals have to be segment-based map matched. As some part of the signals, which

are sent to the *log table* in the Interval model, have already been map matched, the time from when the *log table* is filled with signals until the ETL procedure can be executed is slightly less than in the Triggering model. Apart from this small difference in map matching time, the two models support the ETL procedure in exactly the same manner.

Real-time queries

Here we touch upon the main difference between the two models. In the Interval model, signals are kept in the *entry table* up to 30 seconds before being processed further. The Triggering model, on the other hand, processes and map matches the signals as they arrive to the Logging database. When considering that it is essential to have the time from when a signal is received until it is used for a real-time query as short as possible, the Triggering model provides a much better solution.

Conclusion

Based on the importance of processing signals for real-time queries efficiently, the Triggering model is a better choice than the Interval model. While the Interval model avoids redundant map matching, this small redundancy is insignificant with respect to getting the incoming signals ready for querying as fast as possible.

4.2 The Data Warehouse

With respect to the Data warehouse, two different models are compared to each other, i.e., *The Signal Star model* and *The Subroute Star model*. Here, we introduce these two models and describe their functionalities where four issues are in focus, i.e., the blow-up of tables, consequences for the ETL procedure, support for the history-based check and support for traffic analysis. The calculations of the blow-up of tables is based on the real-world numbers from Aalborg city, given in Subsection 3.1.1. After having introduced the two models we compare them and give arguments for the one chosen. We start by looking at the Signal Star model.

4.2.1 The Signal Star Model

The idea behind the Signal Star model, shown in Figure 4.2, was to keep the model simple. Each signal that arrives to the system is stored in the *signal fact table*. There are five dimension tables in the model which store relative information for the signal and are near-static, i.e., tables that are infrequently updated. Finally, the model contains the *route dimension*, which stores each route, i.e. each car trip, that has been taken within the road network and this dimension becomes near static after few years.

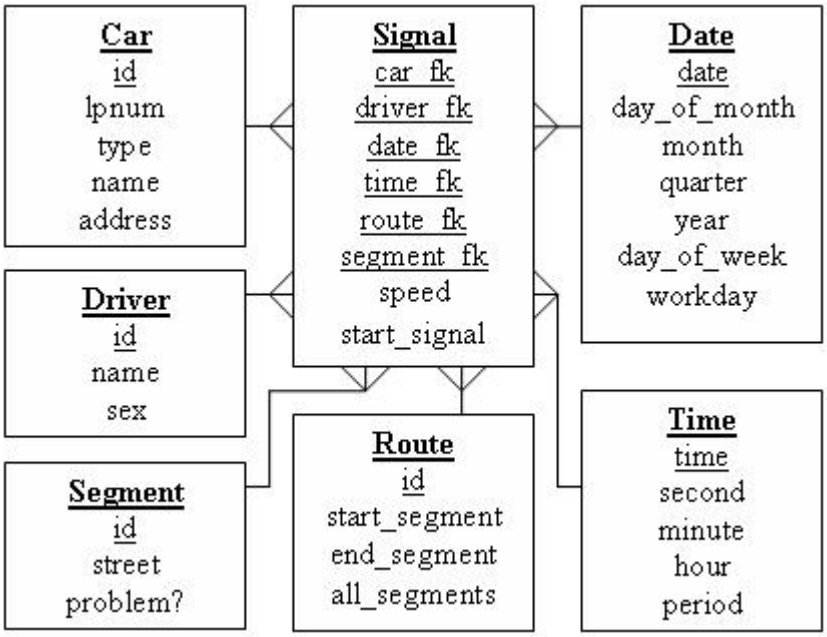

Figure 4.2: The Signal Star model.

Size of tables

Based on the real-world numbers from Subsection 3.1.1, after one year the *signal fact table* is around 70 Gb and the *route dimension* is around 2 Gb which is quite big for a dimension table, however, this is due to the size of a single attribute, i.e., 'all_segments'. The table contains few records compared to its size in bytes. Other dimension tables range from few Kb up to around 3 Mb.

In Appendix B.1 the calculation on the size of the tables are shown, both with respect to number of records and in bytes.

The ETL procedure

In order to discuss the ETL procedure without going into unnecessary details we show source code examples from where the procedure is different between models. First, an example where the route, taken in a single car trip, is selected into a variable. This variable is then used when checking whether a new record is inserted into the *route dimension*. Second, the insertion into the *signal fact table*. For each signal, the record is forwarded to the signal fact table, without modifying the record.

```
1   ROUTE OF A CAR TRIP FOUND
2
3   select distinct segment into #segments from #trip
4   select @segment = (select top 1 segment from #segments)
5   select @all_segments = @segment
6
```

```
7     delete top 1 from #segments
8
9     while exists (select top 1 segment from #segments)
10    BEGIN
11        select @top_seg = (select top 1 segment from #segments)
12        select @all_segments = @all_segments + ',' + @top_seg
13
14        delete top 1 from #segments
15    END
16
17    drop table #segments
```

After running through all segments taken in a car trip, the variable *@all_segments* contains all segment ids. Then, by using a simple *if - else* sentence, the route id for the route containing identical segment values is selected, possibly requiring the insertion of a new route. The route id is then used when inserting the signals of the car trip into the *signal fact table* (see *@route* in line 6 below).

```
1     SIGNAL FACT TABLE INSERT
2
3     while exists (select top 1 car from #trip)
4     BEGIN
5         insert into signal values (@car, @driver, select top 1 date from #trip,
6                                    select top 1 time from #trip, @route,
7                                    select top 1 segment from #trip,
8                                    select top 1 speed from #trip,
9                                    select top 1 start_signal from #trip)
10
11        delete top 1 from #trip
12    END
```

The #trip is a temporary table containing all signals of a single car trip. The ETL procedure puts all start signals into a temporary table, #start_signals, and sorts them by driver and time. Signals are then loaded into the #trip temporary table from the *log table* based on the #start_signals temporary table.

The procedure runs through the signals of #trip and inserts them into the *signal fact table*. While @car, @driver, and @route are constant for all signals of a single car trip, other attributes are selected specially for each signal inserted.

The benefit of this ETL procedure is that the transformation of signals is very simple when inserting them into the data warehouse.

The history-based check

Before choosing an appropriate data warehouse model we needed to look at how the model supports real-time queries. From the two real-time query types of the system, i.e., history-based and routine checks, only history-based checks are made on the history of cars, i.e., the data warehouse. Therefore, the history-based check is in focus when discussing the data warehouse with respect to real-time queries. Using

every-day language, the query would be:

- "For driver D, segment S, weekday W, and time of day period T, what segments SGM (which currently hold a problem) has D taken inside T on W more then 50 times, where SGM is a segment inside a route R starting on street STR, and S is on STR?"

The Signal Star model supports the history-based check, where it can be evaluated, containing simple joins between the fact table and its dimension tables.

Traffic analysis

When looking at traffic analysis queries we used three different traffic analysis queries to check the model.

- "What route is driver D most likely to take when driving in time of day period P on weekday W?"
- "What is the traffic rate [1] in Aalborg city with respect to time of day periods?"
- "When driving from segment A to segment B, what route R is the most popular?"

The first and second traffic analysis queries can be evaluated on this model, containing simple joins between the fact table and its dimension tables. The model, however, does not give full support to the third query. When calculating the most popular route between segments A and B, the model is limited to only looking at routes that start at A and end at B, not routes that start at A, goes through segment B and ends in another segment.

Now, we look at the other model, the Subroute Star model.

4.2.2 The Subroute Star Model

The idea behind the Subroute Star model, shown in Figure 4.3, was to store subroutes as facts. This way, the signals of a single car trip would be used to find the complete route of the car trip, which then again consists of several subroutes, stored as facts. There are five dimension tables in the model which store relative information for the subroute and are near static, i.e., tables that are infrequently updated. The *route dimension* stores each route, i.e. each car trip, which has been taken within the Road network and furthermore, stores all subroutes of each route. Finally, the model contains the *route dimension*, which stores both each full route, as in the Signal Star model, as well as all subroutes that have been taken within the road network. The *route dimension* becomes near static after few years.

[1] Percentage of traveling cars.

Having subroutes as facts should decrease the size of the data warehouse and give a better analytic support of routes. Routes are represented as a string of segments (e.g., from segment A to segment D), and by further storing subroutes (i.e, segment A to segment B, A to C and A to D), traffic analysis queries that relate to routes are given extra support.

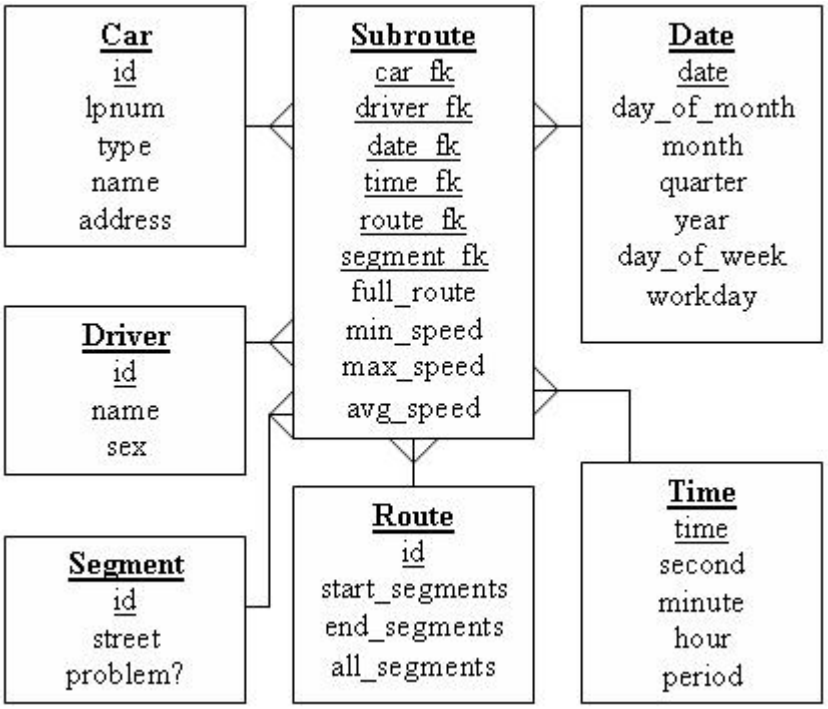

Figure 4.3: The Subroute Star model.

Size of tables

The *subroute fact table* is around 44 Gb for one year of data. In the *route dimension*, both full routes and subroutes are stored. That causes the size of the *route dimension* to be around 9,5 Gb which is large for a dimension table. This is, as for the Signal Star model, due to the size of a single attribute, i.e., 'all_segments', and the table contains few records compared to its size in bytes.

In Appendix B.2 the calculation on the size of the tables are shown, both with respect to number of records and in bytes.

The ETL procedure

The ETL procedure for the Subroute Star model finds the subroutes of a car trip before loading records into the fact table. Furthermore, it inserts a single record for each of the subroutes in the *route dimension* when not already existing. As for the

Signal Star model, we show examples from the source code of the ETL procedure where records are inserted into the *subroute fact table* and the *route dimension*. These examples are chosen as they present the main difference between the procedures for the two models presented in this section. Furthermore, these examples relate to the examples given in Subsection 4.2.1. The *while* sentence in line 3 (in the code example below) is taken for each signal of a single trip before finishing the insertion of records from one car trip into the fact table.

```
1   ROUTE DIMENSION AND SUBROUTE FACT TABLE INSERT
2
3   while exists (select top 1 car from #trip)
4   BEGIN
5       if(@counter = 1)
6       BEGIN
7           select @full_route = true
8       END
9
10      select @trip_seg = (select top 1 segment from #trip)
11
12      if (@trip_seg != @segment)
13      BEGIN
14          if exists (select id from route where all_segments = @all_segments)
15
16          BEGIN
17              select @route = (select id
18                               from route
19                               where all_segments = @all_segments)
20          END
21
22          else
23          BEGIN
24              insert into route values(@segment,@trip_seg,@all_segments)
25
26              select @route = (select id
27                               from route
28                               where all_segments = @all_segments)
29          END
30
31          insert into subroute values(@car, @driver, @date, @time,
32                                      @route, @segment, @full_route,
33                                      @min_speed, @max_speed,
34                                      @total_speed, @signal_count)
35
36          select @all_segments = @all_segments + ',' + @trip_seg
37      END
38
39      select @date = (select top 1 date from #trip)
40      select @time = (select top 1 time from #trip)
41      select @trip_speed = (select top 1 speed from #trip)
42
43      if (@trip_speed > @max_speed)
44      BEGIN
```

```
45              select @max_speed = @trip_speed
46          END
47
48          if (@trip_speed < @min_speed)
49          BEGIN
50              select @min_speed = @trip_speed
51          END
52
53          select @total_speed = @total_speed + @trip_speed
54
55          delete top 1 * from #trip
56
57          select @counter = @counter - 1
58          select @signal_count = @signal_count + 1
59          select @segment = @trip_seg
60      END
```

Before inserting into the *route dimension*, the ETL procedure calculates the subroutes of the car trip, checks whether they exist in the *route dimension*, and then inserts the subroutes which do not exist. Each subroute of a car trip is inserted into the *subroute fact table*, along with the corresponding route id. Furthermore, average, maximum, and minimum speed of the subroute is calculated, and inserted along with the fact table record. The calculation of subroutes and speed values makes the procedure more complex than if signals were inserted directly into the fact table.

The history-based check

When written in every-day language, the history-based check looks the same as for the Signal Star model. As for the Signal Star model, the history-based check can be evaluated in the Subroute Star model. It also contains simple joins between the fact table and its dimension tables.

Traffic analysis

In order to support comparison, the same traffic analysis queries were used to check the model as for the Signal Star model. The first and third traffic analysis query can be evaluated on the model, containing simple joins between the fact table and its dimension tables. However, the second query is not fully supported by the model. The second query relates to traffic rate at certain time periods, and returns in percentage how many cars were driving between 12 pm and 3 am, 3 am and 6 am and so on. For all subroutes, stored in the fact table, that have start and end times in distinct time of day periods, the query returns inaccurate results, as only one relation is between the fact table and the *time dimension* (i.e., for the start time of subroutes).

4.2.3 Choosing a Model

In order to decide on a model to implement we focus on the four aspects that were especially discussed when introducing the models, i.e., the blow-up of tables, the

difference between the ETL procedures, support for the history-based check and support for traffic analysis.

Size of tables

Size after one year	**Signal Star model**		**Subroute Star model**	
	Fact table	*Route dimension*	*Fact table*	*Route dimension*
in records	3.504.000.000	18.250.000	1.752.000.000	146.000.000
in bytes	68,53 Gb	2,2 Gb	44,06 Gb	9,52 Gb

Table 4.1: Estimated size of the fact tables and the *route dimension* for both models, when containing data from one year.

The main difference in size between the two models (see Table 4.1) lies in the size of the *route dimension*. The Subroute Star model stores all subroutes in the *route dimension* as well as all full routes. This leads to a much bigger dimension, i.e., around 9.5 Gb instead of 2 Gb. Concerning blow-up of tables, the Signal Star model is a better choice than the Subroute Star model, due to the size of the *route dimension*.

The ETL procedure

The ETL procedure for the Subroute Star model includes far more calculations behind each record loaded into the data warehouse, than the ETL procedure for the Signal Star model. How each GPS signal is stored as a single record in the fact table of the Signal Star model makes it a better choice, concerning the processing of data from the logging database.

The history-based check

Regarding the history-based check, it can be evaluated on both models. In order to further verify that both models give same results to the query, calculations were done which confirmed their equality (see Appendix B.3). There it showed that, for a specific case, the query returned result sets, proportionally equal in size to the fact table sizes. Both models support the history-based check and one model is not considered a better choice than the other.

Traffic analysis

Regarding traffic analysis, three focus queries were generated on the models. For the Subroute Star model, one of these queries (i.e., ("What is the traffic rate in Aalborg city with respect to time of day periods?") did not return completely accurate results. However, the Subroute Star model offers a different support for traffic analysis as, e.g., in the query ("When driving from segment A to segment B, what route R is the most popular?"), which is not fully supported by the Signal Star model. With

respect to the traffic analysis queries both models can be used, but have different support.

Conclusion

The model we choose is the Signal Star model. There are three reasons for choosing the Signal Star model (ordered by importance):

1. The ETL procedure is simpler for the Signal Star model because of the calculation of the subroutes in the Subroute Star model.

2. The *route dimension* table of the Subroute Star model is over four times the size of the *route dimension* of the Signal Star model.

3. The design of the Signal Star model is simpler.

4.3 The Road Network

After having decided on both the Logging database model and the Data warehouse model, it is time to fit the Road network database into the big picture. Earlier during the project process we introduced the idea of having the Road network database outside the data warehouse (see [6]). The model, shown in Figure 4.4, stores segments as polylines in the *segment dimension*. The *connection table* stores information on how the segments are connected, *problem* and *problem_area* store information on problems in the road network, and *cell*, *area*, and *area_seg* store information to support the area based map matching procedure.

As we have previously decided on the Signal Star model as the basic model for the Data warehouse, this Road network database model would be used in correlation with the dimensional Signal Star model, where the *segment dimension* (see Figure 4.2) would relate to the segments of the model in Figure 4.4. This modeling technique raises the question of consistency.

Having the same data in two separate database models increases the risk of inconsistency and would require some additional mechanism to make sure the data keeps consistent or that the data would be fed to one database by the other. Therefore, a more optimal solution would be to connect the two models through the *segment dimension*, i.e., the data which would otherwise be stored in both models. In this section, we move away from the design of having the Road network in a separate database (as proposed in [6]), due to the problem of inconsistency, and focus on representing the Road network within the Data warehouse.

Here, we compare two possible models, *The Connected model* and *The Internal model*. These two models are compared with three main issues in focus, i.e., the blow-up of tables, consequences for the ETL procedure, and support for various queries (e.g., real-time and analysis). We start by looking at the Connected model.

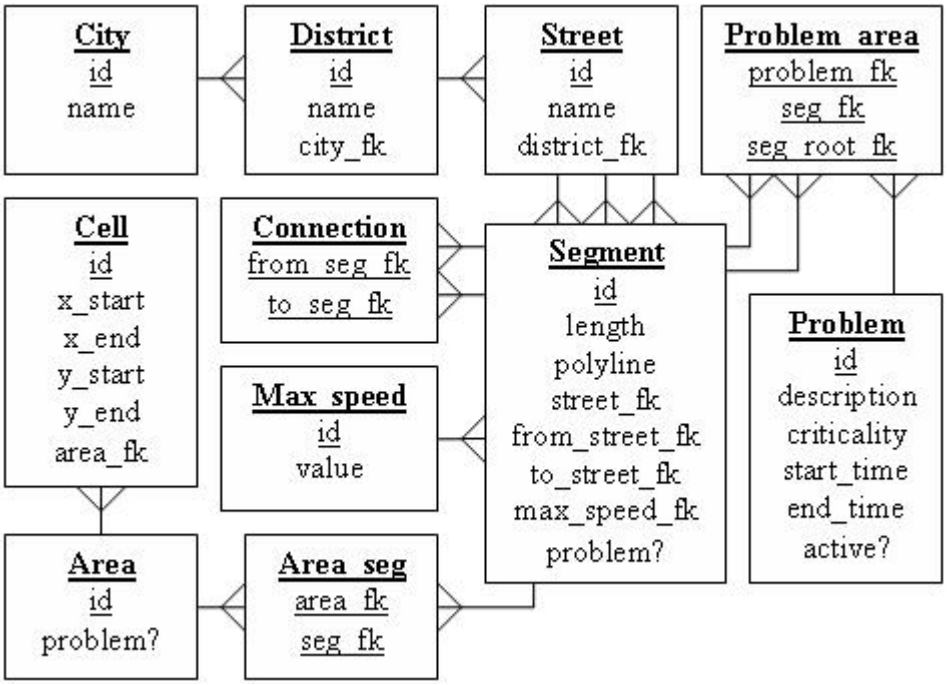

Figure 4.4: The External road network database model

4.3.1 The Connected Model

The idea behind the Connected model, shown in Figure 4.5, was to simply connect the road network model to the Signal Star model through the common *segment* table. The model is, basically, a merge of an entity-relational model (the Road network model) and a dimensional model (the Signal Star model). The *segment dimension* connects these two sides of the model and it now includes attributes for 'street', 'district', and 'city', slightly increasing the redundancy of data but instead making the corresponding tables of the road network model unnecessary.

Size of tables

The dimensional side of this model has the same size of tables as the Signal Star model (see Subsection 4.2.1). The rest of the model would contain small tables (measured in Kb) where the largest table, i.e., the *segment dimension*, would have 5.200 records, resulting in a size around 600 Kb (see Appendix C.1). The size of the *segment dimension* stays constant trough time except for topology changes.

The ETL procedure

The ETL procedure for the Connected model has the same basic functionality as the one for the Signal Star model.

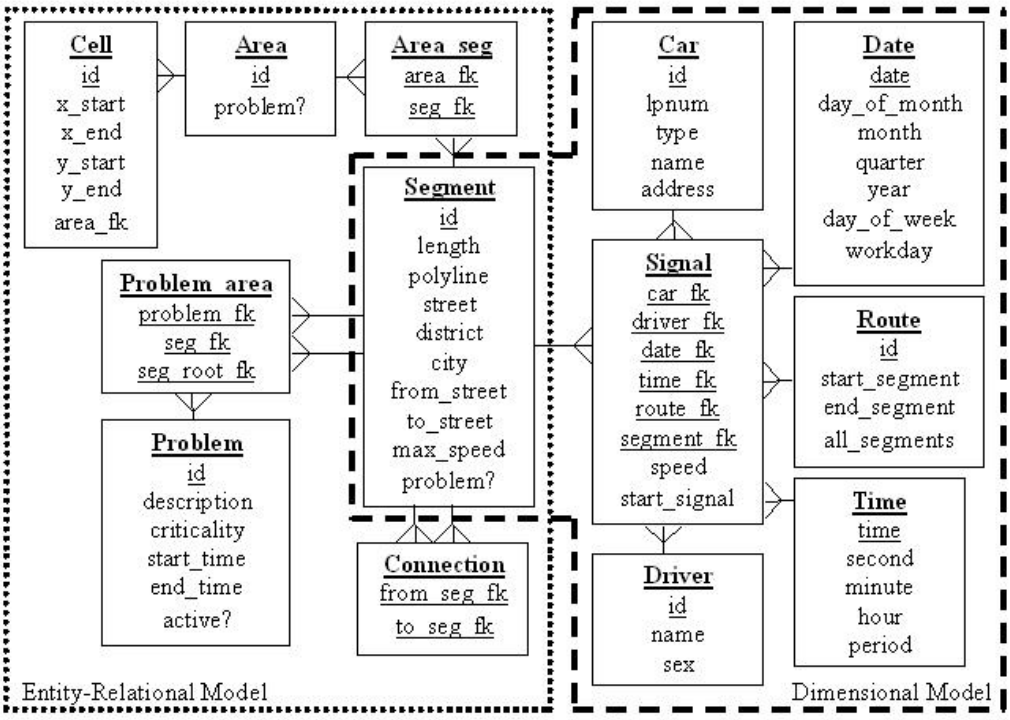

Figure 4.5: Architecture of the Connected model.

Query support

By connecting the road network to the Signal Star model this model gives a better support for the history-based check than the Signal Star model alone. While the history-based check on the Signal Star model returned values that would then be used to query the road network for problem information, this model can return all necessary information in one query. The other real-time query, i.e., the routine check can be evaluated on the Connected model, containing simple joins between the *segment dimension*, the *problem table* and the *problem_area table*. The model supports traffic analysis queries in the same way as the Signal Star model (see Subsection 4.2.1). Map matching queries can be evaluated on the model, where look-ups are made on tables and no joins are required. Note that the segment based map matching procedure will look the same for all models as it compares the 'polyline' attribute of the *segment dimension* to information gathered from GPS signals (i.e., coordinates and driving direction).

Inserts and updates

The insertion and disabling of problem information in the model are the only basic system queries (except for the ETL procedure) that either update or insert into a table. While these queries might make a lock on the *segment dimension*, they are rel-

atively uncommon compared to both the real-time and map matching queries. When traffic delay occurs, problem information is inserted into the model and even though 50 problems would be inserted to the model for one day, the number of real-time and map matching queries can range up to many hundreds of thousands.

Now, we look at the other model, the Internal model.

4.3.2 The Internal Model

The idea behind the Internal model, shown in Figure 4.6, was to store the road network as a single dimension of a dimensional model, otherwise identical to the Signal Star model. In [8], Jensen et al. proposed a dimensional model with a location dimension. When designing the Internal model we analyzed their location dimension and explored whether their ideas could be used. Even though being inspired by their work, a different solution was chosen. When comparing their location dimension with the information, which had to covered in the Internal model, the difference of the two problems became evident. We had to cover the connection between segments, areas, cells, and problem information, none of which was covered in their model. Instead of modifying their dimension, we created the *segment dimension* of the Internal model. Attributes 'to_segment', 'area', 'cell', and attributes with 'problem_' as prefix in the *segment dimension* (see Figure 4.6) were added in order to meet the system requirements.

Size of tables

The dimensional side of this model has the same size of tables as the Signal Star model (see Subsection 4.2.1). The *segment dimension*, however, is different from the Connected model. In this model, *segment dimension* is forced to store more than one record for each actual segment ('segment id' in Figure 4.6 are identical for several records) causing the many-to-many relationship between the *signal fact table* and the *segment dimension*.

Storing several records in the *segment dimension* for each actual record has negative affects on the blow-up of the table. After the first year, the *segment dimension* contains 63.350 records (based on the numbers from Chapter 3), and each time a new traffic delay occurs and a new problem information is inserted, approximately 37,5 records are inserted (on average) into the *segment dimension* (see blow-up calculations in Appendix C.2).

The ETL procedure

The ETL procedure is built up in the same way as the one for the Signal Star model, however, the many-to-many relation between the *signal fact table* and the *segment dimension* adds to the calculations of the procedure. Since the *segment dimension* stores several records for each actual segment, the ETL procedure needs to select the 'segment id' distinctively.

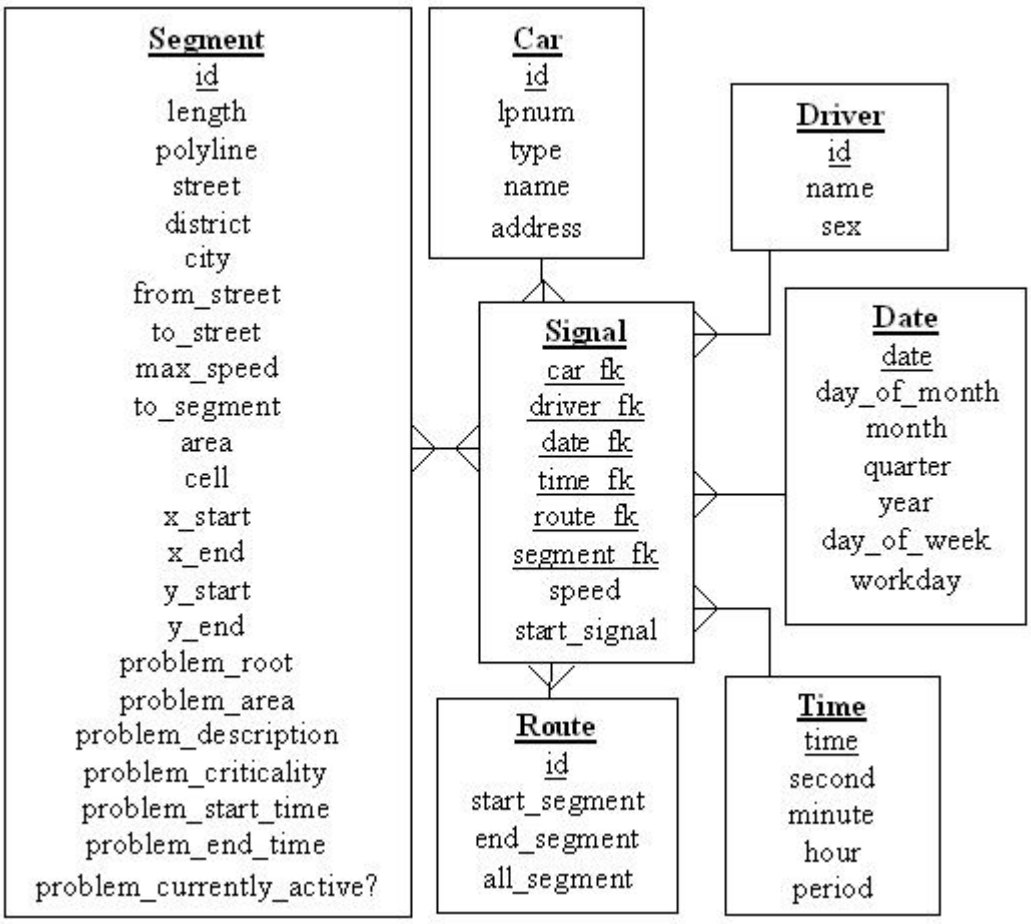

Figure 4.6: Architecture of the Internal model.

Query support

As discussed before, the Internal model requires a many-to-many relation between the *signal fact table* and the *segment dimension*. This affects all queries/procedures that select from the *segment dimension* as a distinctive selection is forced. As for the Connected model, the history-based check can be evaluated in the same way as the Signal Star model (see Subsection 4.2.1). However, when evaluating routine check in this model, self joins of the *segment dimension* are needed, which cause the routine check to run slower as the *segment dimension* grows. Concerning traffic analysis queries, they look, in general, the same as for the Signal Star model. Map matching queries (i.e., area- and segment-based) can be evaluated on the model, where look-ups are made on tables and no joins are required.

Inserts and updates

The redundancy of data in the *segment dimension*, causes the insertion of problems to be a complicated procedure where several records are inserted into the *segment dimension* for each new problem.

4.3.3 Choosing a Model

In order to decide on a model to implement, there are three main aspects that have to be considered, i.e., the blow-up of tables, consequences for the ETL procedure, and query support.

Size of tables

The overall size of the tables of the two models is almost the same. That is due to the fact that the models use the same fact table, which contains the vast majority of the data. The main difference, concerning size, is the size of the *segment dimension*, which is near constant in size (5.200 records) for the Connected model, while it would be around 63.350 records after one year in the Internal model (given the assumption that 1.000 problems are stored per year) and the table would grow nearly linearly as new problems are inserted (around 37.5 new records per problem). Assuming that 1.000 problems would be inserted each year, the growth of the *segment dimension* would be 37.500 records per year (or 6,25 Mb, see calculations in Appendix C.2). Concerning the size of tables, the Connected model is a better choice than the Internal model, due to the blow-up of the *segment dimension*.

The ETL procedure

For the Connected model, the ETL procedure is almost the same as for the Signal Star model. For the Internal model, it would have to be modified, due to the many-to-many relation between the *signal fact table* and the *segment dimension*, selecting distinct values from the *segment dimension*. Concerning the ETL procedure, the Connected model is a better choice than the Internal model, due to the need for distinctive selections in the procedure for the Internal model.

Query support

The constant growth of the *segment dimension* of the Internal model causes the time-critical routine check to run slower by the years. This is a major drawback for the Internal model since a real-time query gets slower as the *segment dimension* gets bigger. The traffic analysis queries look, in general, the same as for the Signal Star model. However, the many-to-many relationship between the *signal fact table* and the *segment dimension* forces distinctive selections for traffic analysis queries referring to segments. Overall, the Connected model provides a better support for queries.

Inserts and updates

In the Internal model, inserting a problem is more complicated than for the Connected model. The reason is the redundancy of data in the *segment dimension*, which causes several records to be inserted into the *segment dimension* for each new problem.

Conclusion

From the two models compared, we choose the Connected model. The fact that the Internal model has a many-to-many relationship between the fact table and the *segment dimension* is the main reason for considering the Connected model a better solution. This many-to-many relationship forces distinctive selections in both queries and procedures when selecting from the *segment dimension*. Furthermore, the growth of the *segment dimension* in the Internal model causes the routine check to run slower by the years.

In this chapter we have documented the comparison of different models, which were candidates for the three main components of the system architecture. Concerning the Logging database, the Data warehouse, and the location of the road network representation, the models chosen for implementation were the Triggering model, the Signal Star model, and the Connected model, respectively.

Chapter 5

System Architecture

In this chapter the overall architecture and the data flow of the system is described. As Chapter 4 provides a description of the two main components of the server architecture, i.e., the Logging database and the Data Warehouse (including a Road network representation), this chapter gives an overview of the overall system architecture and its basic functionalities.

To this point, the report has focused on the server-side of the system. Furthermore, the earlier parts of the larger project have mainly discussed database modeling and design. While that remains the main aspect of this report, the client-side is discussed in order to cover the overall system architecture and to build a foundation for experiments. Based on the responsibilities of each system component, experimental processes can be defined (see Chapter 6).

Section 5.1 discusses the overall architecture of the system, for the server-side and the client-side, respectively. While the data flow between components has been, for most parts, covered in Chapter 4, Section 5.2 fills in the missing pieces.

5.1 Component Architecture

The proposed system architecture is shown in Figure 5.1. The main changes from the architecture proposed in [6] (see Figure 2.1) are that the road network is now inside the data warehouse, connected to the dimensional model through the *segment dimension table*.

With respect to real-time queries, the communication flow in the system starts when a car sends GPS signals to the Logging database (each car, connected to the system, sends one signal every 10 seconds). The Logging database sorts out signals, which are then used for near real-time checks, i.e., both history-based and routine checks. The GPS signals, which are used for history-based and routine checks are map matched and a query is executed on the Data warehouse and Road network, respectively. The map matching procedure uses the road network to match each signal to a specific road segment. If a query from either a history-based or a routine check returns a positive result, a message is composed and the driver is notified

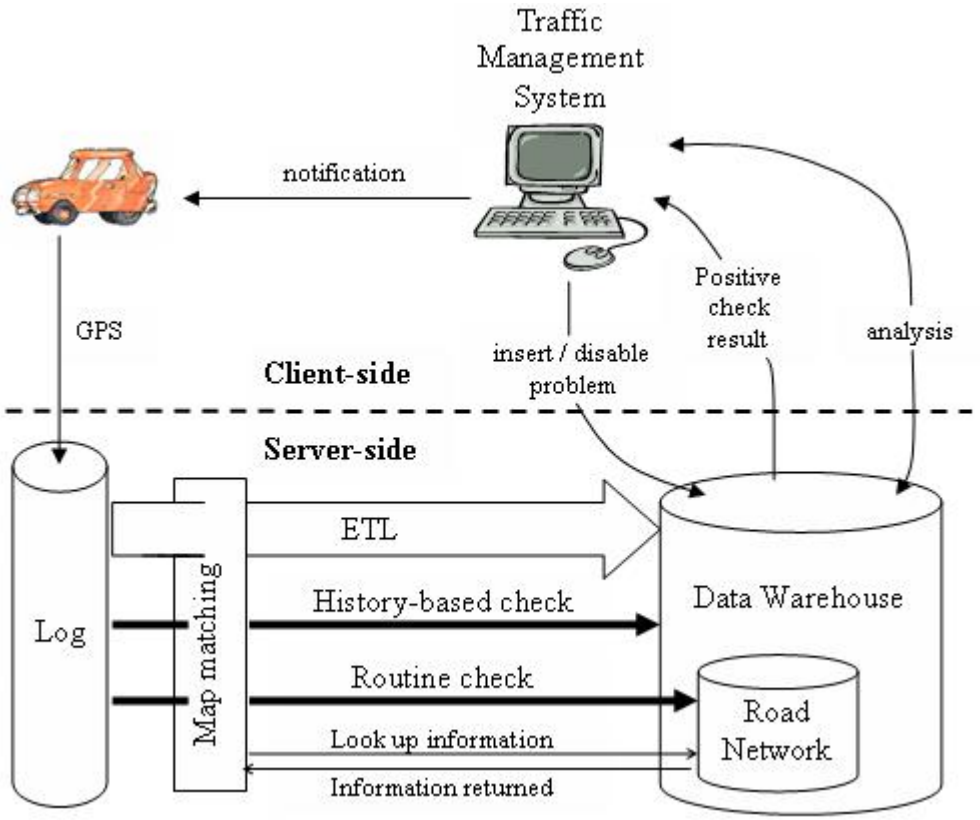

Figure 5.1: The system architecture.

about relevant traffic delay(s). At 24 hours intervals, the ETL procedure is executed, loading signals from the Logging database into the Data warehouse. The Traffic management system is used to compose messages (as a result of a positive real-time query) and notify the drivers of relevant traffic delays. Furthermore, traffic analysis queries are made on the Data warehouse through the Traffic management system.

5.1.1 Server-Side Components

Presentation of the server-side components is done in Chapter 4. The Logging database, the dimensional Data warehouse model, and the Road network representation are discussed in detail in Subsections 4.1.2, 4.2.1, and 4.3.1, respectively. All server-side components are stored on the same machine, as separate parts of a single database.

Instead of repeating what is introduced in Chapter 4, we revisit four of the five open issues which were introduced at the end of Chapter 2 and state the solution for each issue. The issue concerning real-world relevant evaluation was covered in Section 3.1. All of these open issues are focused on the server-side of the system architecture.

- **How to store incoming signals and process them for querying?** For the Logging database we chose an approach that uses triggers to choose appropriate GPS signals for near real-time querying. The triggers are activated when data arrives to the system, which enables the system to retrieve the GPS signals for the near real-time queries efficiently. The design of the Logging database expects a direct link between the GPS devices of cars and the database, where signals are logged and processed concurrently.

- **What course should be taken concerning the map matching procedure?** As map matching has been the issue of existing research, we do not consider it important to tailor-make a specific map matching procedure during this project work. Furthermore, the real-world data set that we use for the experiments consists of signals that have already been map matched. Instead, we focus on what is new in our project. However, we minimize the frequency of map matching in the system by proposing a so called area-based map matching procedure (see Subsection 5.2.2). We propose that in order to complete the system, existing map matching algorithms would be implemented, such as the one proposed in [1]. From the experimental results for the real-time queries, we might be able to estimate the maximum time allocated to the segment-based map matching procedure. With respect to the ETL procedure, all signals must by segment-based map matched before being extracted by the procedure. We propose that several identical map matching processes will run on the data in the *log table*, going through each record and assigning segment identities. This way, the ETL procedure could be executed as soon as all signals in the *log table* are map matched.

- **What data warehouse model should be chosen?** We chose a dimensional model where each GPS signal is stored as a fact in the fact table. The model supports the history-based check, the traffic analysis queries and a simple ETL procedure design.

- **Should the road network representation be inside or outside the data warehouse?** We propose a solution where the road network is represented inside the data warehouse, connecting a entity-relational road network database model to the dimensional traffic history model. The two parts are connected through a common table, i.e., the *segment dimension table*.

5.1.2 Client-Side Components

On the client-side of the system architecture, there are two types of clients, i.e., the single Traffic management system and the many GPS devices of cars. The Traffic management system is a thick client, located on the same machine as the server. It is used for making traffic analysis queries on the data warehouse, updating problem information in the road network side of the data warehouse, and sending notifications to the devices of cars when a problem exists in the probable future path of the car.

The GPS devices of cars are thin clients, sending GPS signals to the Logging database on regular intervals. The GPS devices are also capable of receiving notifications from the Traffic management system about upcoming traffic delay(s).

Having the many GPS devices as a data source for the server, requires concurrent processing of signals. The Logging database is responsible for the initial reception of signals, where signals that shall be used for real-time queries are selected as they arrive. This solution could result in a problem, where the database would have performance problems, receiving signals from many sources. The probability of that problem occurring will be discussed when evaluating experimental results (see Chapter 9). While this solution is based on a connection between the GPS devices and the server, an alternative solution is considered, where the logic of selecting signals for real-time queries is pushed into the thick client, i.e., the Traffic management system.

The alternative solution is to move the logic of selecting signals for real-time queries into the thick client, expanding the responsibilities of the Traffic management system. This technique does not alter the components of Figure 5.1. However, the communication flow between system components changes. The signals, sent from the thin clients, would be received by the Traffic management system, which would forward the signals to the Logging database. Signals which were to be instantly used for real-time queries would be sent directly to the *query table* of the Logging database while other signals would be bulk-inserted into the *log table* of the Logging database on regular intervals. With respect to Figure 5.1, this changes the current circular communication flow (i.e., thin client → server → thick client → thin client) into a more layered architecture (i.e., thin client → thick client → server, and back). This alternative solution would decrease the workload of the Logging database (in terms of signal processing), however, it could result in delaying the execution time of real-time queries by adding more components for the signals to travel through.

5.2 Component Communication

In Figure 5.1, the communication links between system components are represented with arrows, one for each of the main system functionalities. Regarding the client-side, the Traffic management system interacts with the server when; analysis is made on the driving history, problem information is updated, and a real-time query returns positive results. When the Traffic management system receives a positive result from a real-time query, it further interacts with a single GPS device, informing a driver that a traffic delay exists in his probable future path.

Regarding the server-side, the interaction between components is based on queries and procedures. As can be seen in Figure 5.1, the interaction between server components is fourfold, relating to; the ETL procedure, the history-based check, the routine check, and the map matching procedures. While the history-based check is fully covered in Chapter 4, this section takes a closer look at the three other queries / procedures.

5.2.1 Routine Check

A routine check takes place once every 30 seconds for each car driving. As these checks are not related to the history of the driver, they only use the information from the signals and on the current state of the road network. The input for the query is the 'driver id' and the 'segment id'.

When problems are stored, they may relate to more than one segment. For each segment where the problem is located, the problem area is found and stored. This problem area is the union of segment sets S1 and S2, where segments in S1 have a non-starting point connected to a non-ending point of the problem segment, and segments in S2 have an non-starting point connected to a non-ending point of a segment in S1. In Figure 5.2 an example is shown where one segment has a problem and 10 segments are in the problem area.

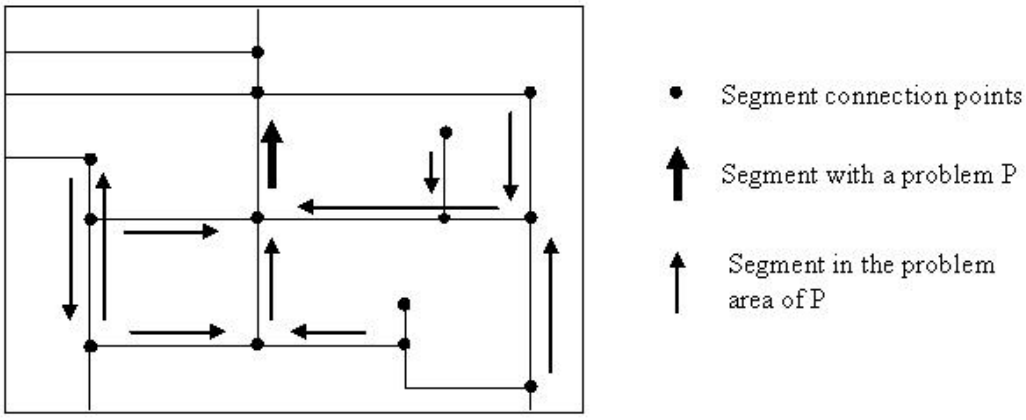

Figure 5.2: Segment with a problem and its problem area.

Based on the primary solution to signal logging, GPS signals arrive to the system at the Logging database, which sorts out the signals to be used for routine checks. When signal is in a problem area, the query returns the 'driver id' along with relevant problem information. This information includes the problem description, criticality, location and estimated duration. Note that a driver can be approaching more than one problem at one time and therefore the query may output a list of problems.

5.2.2 Map Matching

As discussed before, we will skip map matching signals to a specific segment in the road network. However, prior to routine checks, an area-based map match is used to sort out signals that are more likely to be in a problem area. The area-based map matching procedure takes the coordinates of a signal and makes a look up in the *cell* and *area tables* to see if the signal comes from a city area where a problem is active. If the signal comes from a "non-problematic" area, a routine check is not made for that signal. This way the area-based map matching procedure filters out signals that

do not need further attention concerning routine checks.

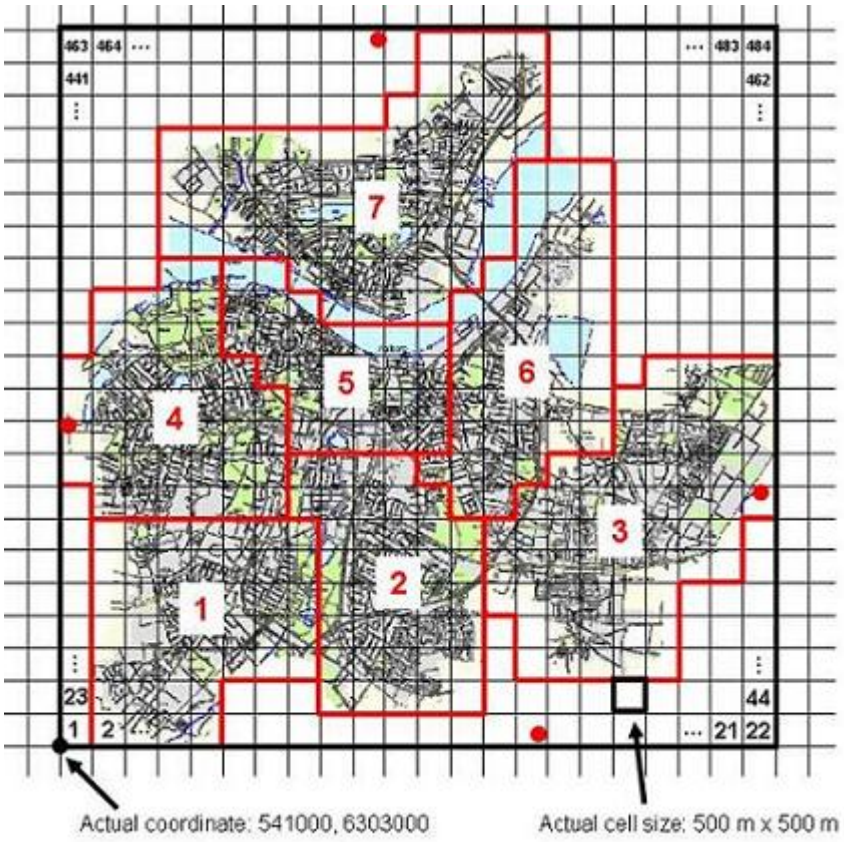

Figure 5.3: Aalborg city divided into cells and areas.

The three tables; *cell*, *area*, and *area_seg* are used to support the area-based map matching procedure. Prior to loading data into these tables, Aalborg city was divided into cells and areas. Figure 5.3 shows the division of the city. From the available data set (see Chapter 7) we found the maximum and minimum x and y coordinates (shown as red dots in the figure) and that way the range values of the Aalborg city region was found (the outer black square in Figure 5.3). The city region is 121 km^2 in size (11 km * 11 km) and it was divided into 484 equally sized cells, each 500 m * 500 m in size, and these cells were then used to split the city into 7 areas. Two of the range values (red dots) are outside the seven areas of Aalborg city as the data was collected from Aalborg city along with areas slightly outside city borders.

The *area_seg table* is a many-to-many relation between *area* and *segment*. The reason for the fact that this is a many-to-many relation is that an area contains many segments and a segment can be contained in more than one area.

5.2.3 ETL Procedure

The ETL procedure is an internal process, which takes care of loading data into the data warehouse. All signals that are sent to the logging database are extracted, transformed, and then finally loaded into the data warehouse. In Figure 5.4 the data flow in the ETL procedure is shown.

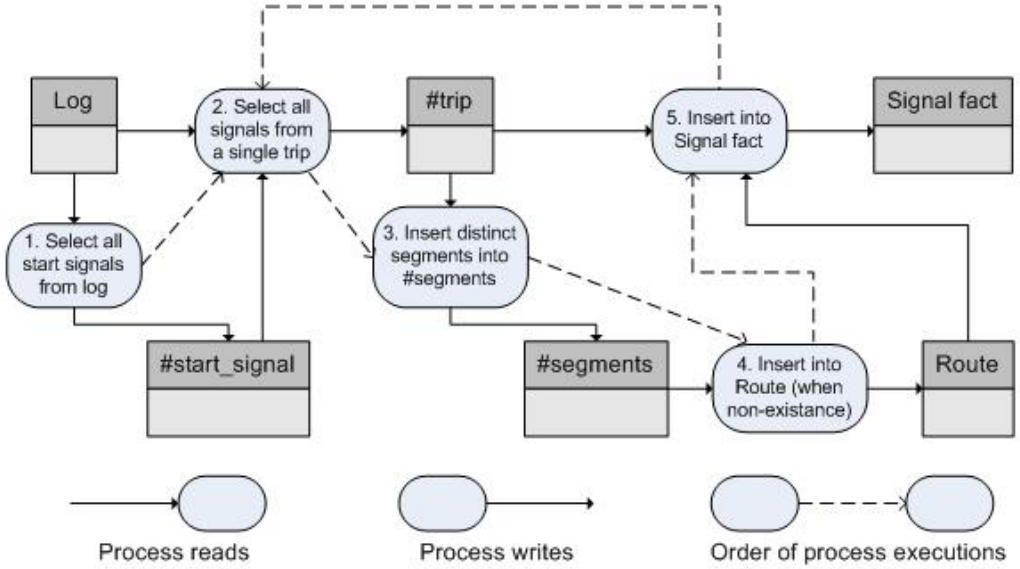

Figure 5.4: The data flow in the ETL procedure.

The flow starts in the logging database where signals that have arrived are stored. (1) The ETL procedure selects all start signals (i.e., the first signal of a car trip) from the *log table*, grouped by car and time, into the *start_signal* temporary table. (2) Based on the start signals, all signals from a single car trip are inserted into the *trip* temporary table. The GPS signals, which are time-wise between two start signals from the same driver are the signals from a single car trip. (3) Segment identities from *trip* are moved distinctively into another temporary table, *segments*, which is used to find the route the driver took. (4) The ETL procedure checks whether this route (sequence of segment identities) exists in the *route dimension* of the data warehouse. If it does not exist, the route is inserted into the *route dimensions*. (5) All signals in the *trip* temp table are then inserted into the *signal fact table* along with the route identity. (6) Finally, the ETL procedure extracts the next car trip from the *log table* and starts the process again, until all signals have been loaded into the data warehouse.

As can be seen in Figure 5.4, the actual insertion into the fact table is done once for each car trip taken. Each insertion is a single transaction and while the transaction holds a lock on the fact table, each transaction has an execution time of around 50 milliseconds and does, therefore, not cause significant delays for, e.g., the

history-based check. In order not to delay real-time queries with respect to processor time, it is suggested that the ETL procedure runs on a separate processor. This issue is further discussed in Section 9.2.

Having presented the system architecture in this part of the report, the next part documents the experimental design and results, followed by an evaluation on the outcome of the experiments.

Part II

Experiments

Chapter 6

Design and Purpose of Experiments

Having designed the "Moving Objects Data Warehouse", experiments had to be defined. In this chapter, the experimental categories are presented, expected results are discussed and required test data.

The main functionalities of the system can be divided into four categories, i.e., signal logging in the Logging database, the real-time queries, traffic analysis queries, and the ETL procedure. The experiments to be conducted are based on these four categories. Each experiment is not depending on any other experiment, i.e., the experiments can be conducted independently from the others.

6.1 Logging Database

As covered in Section 4.1, the Triggering model was chosen as the Logging database model. The Logging database is responsible for the logging of GPS signals and selecting signals for the real-time queries.

The experiments chosen to be conducted on the Logging database were load experiments, i.e., how well the Logging database handles receiving many signals over a specific time period (e.g., 30 seconds).

As stated in Section 3.1, the Logging database must cope with logging up to 30.000 GPS signals per 10 seconds. In order to simulate the situation where multiple GPS devices send signals to the system, a thin client application must be made for each of the GPS devices. All available resources were to be used for this experiment, reaching the limit of the possible client applications, i.e., the number of applications that the available computers can handle.

The expected results from this experiment are that the Logging database is able to log a N number of signals S, where S is a set of signals $S_1, S_2, ..., S_N$, and the size of N does not influence the time of logging a single signal in S. The requirements regarding N are that: $0 < N \leq M$, where M is the maximum number of client applications. It is then depending on the size of M, whether the results allow us to conclude that the

Logging database meets the system requirements.

Concerning the required data for this experiment, the inserted signals have to be with as many distinct driver identities as the number of clients used.

6.2 Real-Time Queries

In the proposed architecture, there are two types of real-time queries, i.e., routine checks and history-based checks. As these two types are different in terms of what data is queried, they are discussed separately.

6.2.1 Routine check

As presented in Subsections 3.1.2 and 5.2.1, a routine check is a query made on the information on problems in the road network. Based on the requirements of Section 3.1, the routine check should be made up to 30.000 times per 30 seconds and the execution of each check may not take more than 10 seconds. This execution time includes the map matching procedure, which was not considered for implementation during this project (see the introduction of Chapter 5).

In the experiments, the execution time of the routine check is in focus. As well as comparing the execution time to the requirements, the experiments include analysis on possible affects on the execution time, e.g., processor speed, memory available for the server, size of results set, and size of problem area on the road network.

The expected results from this experiment are that the routine check executes within the requirements and that the above issues have no significant affect on the execution time.

Concerning the required data for this experiment, information on problems has to be stored such that the query returns a non-empty result set. This means the the input of the query must match an existing problem information in the database, i.e., in tables *problem*, *problem_area*, and *segment*.

6.2.2 History-based check

As presented in Subsection 3.1.2, a history-based check is a query made on the driving history of cars. Based on the requirements of Section 3.1, the history-based check should be made up to 300 times per 30 seconds and the execution of each check may not take more than 90 seconds. As for the routine check, this execution time includes the map matching procedure.

While the routine check queries relatively small tables (i.e., the road network side of the data warehouse) the history-based check queries the fact table of the data warehouse. The size of the fact table is relative to the number of cars, which the history is stored for. In the experiments, the execution time of the history-based check is measured with respect to the number of cars, which the history is stored for in the fact table. As for the routine check, this experiment is done with respect to different processors and the available memory for the server.

The expected results from this experiment are that the execution time of the history-based check will increase linearly with respect to the size of the fact table. It is furthermore expected that the results will indicate that the history-based check will execute in less than 90 seconds for a fact table containing the driving history of 50.000 drivers. In combination with results from routine check experiments, these results should indicate the maximum time allocated to the segment-based map matching procedure.

Concerning the required data for this experiment, it would be ideal to have a whole year of data for, at least, few hundred cars. Unfortunately, we don't have that kind of data available. In Chapter 7, we explain how that data was simulated, in order to have realistic information in the data warehouse. As for the routine check, the input of the query must match existing information on a problem.

6.3 Traffic Analysis

For experiments on the traffic analysis, the three same queries are chosen as were used for model comparison (see Subsection 4.2.1).

For the same reasons as for the history-based check, traffic analysis queries are done on the fact table and connected dimension tables. Scalability issues, with respect to the size of the fact table, are the focus of this experiment.

As for the history-based check, the execution time of the traffic analysis is expected to grow linearly with respect to the size of the fact table.

Concerning the required data for this experiment, the same goes as for the history-based check, i.e., it would be ideal to have the data warehouse "filled" with real-world data.

6.4 The ETL Procedure

The ETL procedure extracts data from the *log table* of the Logging database and loads it into the *signal fact table* in the Data warehouse. Based on the requirements of Section 3.1, the ETL procedure should execute in at most 24 hours, including the segment-based map matching of signals. However, this map matching of signals should not add much to the actual time of the procedure, as explained in Subsection 5.1.1.

In the experiment, the scalability of the execution time of the ETL procedure is measured with respect to the size of the *log table*. The size of the *log table* is relative to the number of cars, sending signals to the system.

The expected results are that the execution time of the ETL procedure increases almost linearly as the *log table* contains more records. Furthermore, it is expected that the execution time will be at most 24 hours when loading data from 50.000 cars. Based on the numbers from Section 3.1, each car sends 192 signals per day on average. That means that 50.000 cars send 9,6 million signals per day. The reason why the results are not expected to show a totally linear growth in time is that the

size of the *route dimension* should make a small affect on the execution time. The ETL procedure makes a look-up in the *route dimension* before each car trip is loaded into the *signal fact table* (see Subsection 5.2.3) and the *route dimension* grows as routes, not existing in the *route dimension*, are loaded in.

Concerning the required data for this experiment, it would both be ideal to have real-world data in the data warehouse and real-world signals in the *log table* of the Logging database. Furthermore, the signals in the *log table* may not have been used to load data into the data warehouse, prior to the experiment. How this is solved is presented in Chapter 7.

Chapter 7

The Data Set

In this chapter, we briefly describe the data set which was used for experiments. We mention its origin, basic attributes, limitations, and the expansions we made to it.

In Chapter 3 we introduced the basic requirements to the system, i.e., the number of drivers it should serve, the size of the area it should store in its road network database, and so on. Furthermore, in Chapter 6, we discussed what requirements are made to the test data with respect to the experiments. The main requirements to the test data are related to experiments with the history-based check, traffic analysis queries, and the ETL procedure. For these three experiments, the data warehouse must contain the driving history of cars, preferably for few hundred cars and a one-year period. Furthermore, the experiment for the ETL procedure requires signal data, which does not exist in the data warehouse.

We were provided with a data set of GPS signals which was not of a size compatible to the numbers in Chapter 3. However, we were able to work around its limitations and use it as testing data for the proposed system.

In Section 7.1 we describe the data set and its limitations, while Section 7.2 describes how the data set was expanded in order to meet the data set requirements of Chapters 3 and 6.

7.1 The Original Data Set

The data set consists of 1.244.513 GPS signals from 12 personal cars, collected from 107 days between December 6, 2000 and March 27, 2001. The basic attributes of the data set are; *car identity*, *driver identity*, *date*, *time*, *coordinates*, *speed*, and *street code*. The GPS signals of the data set are registered with one-second interval and, as the system expects signals to be registered with 10 second intervals, signals from one car can be treated as signals from 10 cars.

By using this data set, we had to work around some limitations. Limitations relating to the size of the data set are discussed in Section 7.2 but other limitations are that:

1. In the proposed system, the *log table* of the Logging database receives non-map

matched GPS signals and these signals are then map matched to segments at various locations in the logging database by a map matching procedure. The data set has street codes assigned to each signal, meaning that the signals have already been map matched to streets, not segments. While using streets would not be practical in the real world, especially for the routine check, we make use of the street codes and treat them as segment ids as it does not have a significant impact on our experiments. Having the data set records assigned to specific street ids, further allowed us to leave the implementation of a map matching procedure to further research.

2. The proposed system expects the start signals of a car trip to be labeled. This, however, is not the case for the signals in the data set. The ETL procedure uses the start signals to split signals into car trips before loading them into the fact table. Therefore, it was important to have specific start signals in the data set. In order to compensate, we ran a procedure on the data set which labeled start signals based on the time between signals from specific cars, i.e., when more than one minute elapsed between two signals from the same car, the second of those two signals was set as a start signal. Having the limit of one minute, this procedure divided the data set into 3.149 distinct car trips.

7.2 Expanding the Data

In order to use the data set for our experiments on the model, relevant data was inserted into appropriate tables of the model. Furthermore, the data set was expanded in order to be of a size which met the requirements concerning experiments (see Chapter 6).

In Figure 7.1, the expansion strategy is presented. The original data set contained around 1.250.000 records from 12 cars and 107 days. The last 17 days were cut of to be used as testing data for the ETL procedure (see Section 6.4) while the first 90 days were multiplied over other two equally long time periods. Four days were left in between every period so that weekdays would stay correct for all signals. Finally, this new data set was multiplied ten times, each with twelve new car ids.

After having expanded the data set, it contained 32,5 million records where 2,5 million were not loaded into the tables. As the expanded data set did not elapse over a whole year and some cars had only data for periods, the data set corresponded to one year of data for around 500 cars. That is based on the assumption that a car travels for 32 minutes per day on average (see Section 3.2).

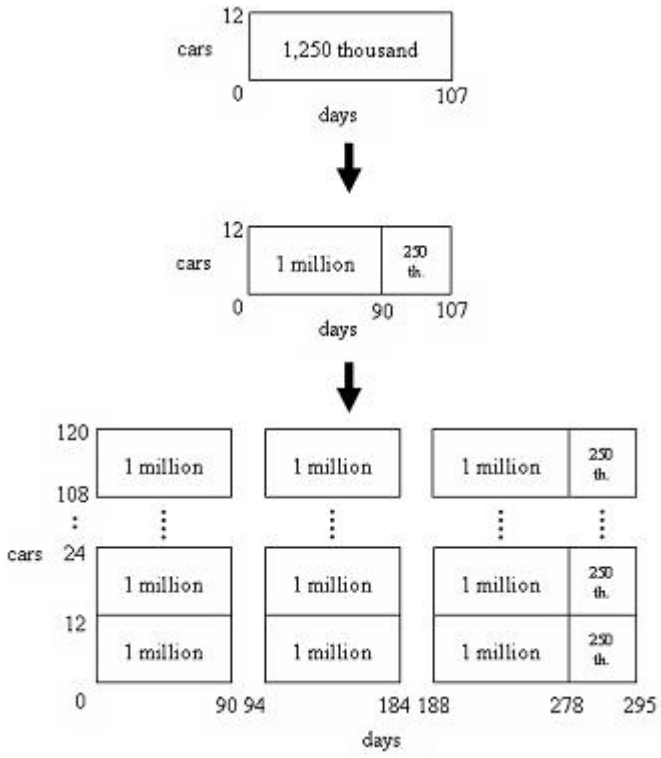

Figure 7.1: The expansion strategy of the original data set.

Chapter 8

Results

This chapter documents the experimental results. The results are presented and each experiment is evaluated, while an overall evaluation of the experimental results is left for Chapter 9.

For each experiment, before the execution of a procedure/query, the server was shut down and started again, in order to free memory. Furthermore, each experiment was conducted five times (except twice for the execution of the ETL procedure) and the average numbers are presented, excluding the best and worst case. Four computers were used for the experiments and, in Table 8.1, the hardware specifications of these computers are presented.

Computer	**Processor**				**RAM**	**Type**
	Name	*Clock speed*	*Cache*	*Bus speed*		
C1	Intel Pentium 4	2.8 GHz	512 Kb	533 MHz	512 Mb	Desktop
C2	Intel Pentium M	1.4 GHz	1 Mb	400 MHz	504 Mb	Laptop
C3	-	1.5 GHz	-	-	512 Mb	-
C4	Intel Pentium 4M	2.4 GHz	512 Kb	-	-	-

Table 8.1: Hardware specifications for the four computers used for experiments (based on [2]).

While computers C3 and C4 were only used to host clients sending signals to the Logging database, C1 and C2 were used for several experiments. The role of each computer is further explained for each part of the experiments.

The database management system used for implementation and experiments was Microsoft SQL Server 2000, Developer Edition. This SQL Server edition can manage up to 32 processors and 64 Gb of memory [3]. All computers ran on a Microsoft Windows XP Professional operating system.

Several side steps were taken for each part of the experiments where miscellaneous configurations were made, in order to observe how they affect the results. These configurations included the adjustment of; memory available to the database server, the size of the result set of queries, maximum working threads for the database server,

indexes, etc. In order to avoid the redundancy of written text and to keep focus on the experiments defined in Chapter 6, we do not mention the cases where no affect was observed. When the configurations had a significant affect, however, it is discussed where appropriate.

As presented in Chapter 6, the experiments are divided into four parts; the logging of signals to the Logging database, real-time queries, traffic analysis queries, and the execution of the ETL procedure. These four parts are the subjects of Sections 8.1, 8.2, 8.3, and 8.4, respectively.

8.1 Logging Database

The Logging database is used for receiving incoming GPS signals and sort out the signals that are used for real-time queries.

As discussed in Section 6.1, experiments on the Logging database were made to observe how it responds to heavy load, i.e., how well it handles receiving many signals over a specific time period.

In order to simulate GPS devices sending signals to the server, several client applications were created. These applications were written in Java and each represented a single GPS device. While the server was stored on C1, the other three computers hosted the client applications. Each computer was capable of, concurrently, running 80 applications. That meant that, in total, 240 client applications could be running at the same time. To realistically simulate the behavior of GPS devices, each application should only send a single signal. However, in order to have more than 240 signals sent in, each application sent three GPS signals to the server including a delay of ten seconds between signals, simulating 240 cars sending signals for 30 seconds (i.e., 720 signals in total). For each signal sent, the application opened and closed a connection to the server, where the computers were connected via a local area network. For each connection between an application and the server, a worker thread (on the server side) took care of inserting the signal into the Logging database.

Concerning indexes, one index is in the Logging database. A clustered index on the driver attribute of the Query table. Not having the index doubled the time of inserting a signal into the Logging database.

As discussed in Section 6.1, the main focus of this experiment was to examine whether the Logging database could cope with logging up to 30.000 GPS signals per 10 seconds, where none of these 30.000 signals are sent from the same source. As the available resources kept us from experimenting with more than 240 cars, the only way to do load experiments on the Logging database was to limit the amount of maximum worker threads for the server. When fewer worker threads are available to the server, it has fewer resources to use for incoming signals. When the number of available worker threads for the server is less than the number of requests for the server, MS SQL Server pools the worker threads so that the next available worker thread can handle the waiting request. If changing the amount of those threads influences the execution time of each signal logged, it points to the fact that as increasing number

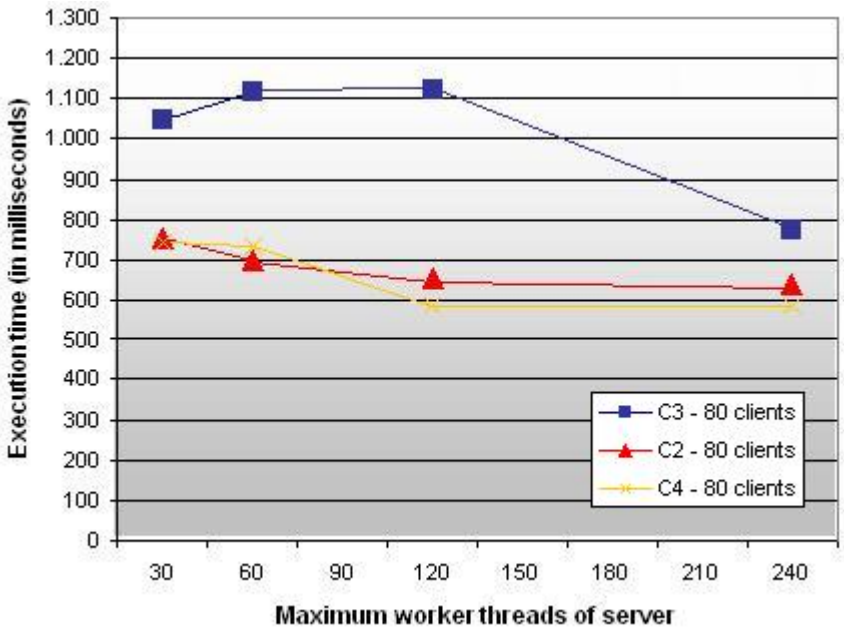

Figure 8.1: Execution time of signal logging in the Logging database with respect to available worker threads for the server.

of signals are sent to the Logging database, the execution time of logging each signal also increases.

Figure 8.1 presents the results from the Logging database experiment. The execution time presented is for a single signal logged. While each application sent three signals, the average time of these signals is presented, as we are interested in the time it takes to log a single signal. Each line represents one of the three computers, which hosted the client applications. Why computer C3 gives different results from the other computers is not known, especially as computers C3 and C2 are very similar with respect to processors while C4 has a different type of processor (see Table 8.1). In order to make sense of these results, we discuss the results from computers C2 and C4.

When looking at the numbers for computers C2 and C4, the execution time does not increase significantly (1,7% and 0,4%, respectively) when the maximum worker threads are decreased to 120, i.e., half the number of the client applications. However, when the number of maximum worker threads for the server is further decreased, the execution time increases significantly. Based on these results, it should be safe to estimate that up to 480 client applications can send signals where the execution time of inserting each signal is approximately the same as when inserting a single signal, when the maximum available worker threads for the server are set to 255. We use 255 as an example as it is the default number in the MS SQL Server. While the amount of available worker threads can be set as high as 32.767, it is likely to result in a CPU bottleneck where the processor queue length gets too long.

Based on the numbers from Section 3.1, up to 30.000 client applications should be able to concurrently send signals to the server. Based on the experimental results, shown in Figure 8.1, that number of clients would probably introduce an overhead in load for the Logging database. While the results of this experiment do not rule out having the logic of selecting incoming signals for real-time queries in the Logging database, it makes us believe that an alternative solution should be chosen.

In Subsection 5.1.2, we presented a solution where the logic of selecting signals for real-time queries is pushed into the Traffic management system, which is a thick client on the same machine as the server. This solution would decrease the responsibilities of the Logging database, and change the communication flow so that the many GPS devices would interact with the Traffic management system. However, experiments on whether the thick client (i.e., the Traffic management system) is more capable of handling the reception of GPS signals than the server, is left for further research.

8.2 Real-Time Queries

As discussed in Subsection 3.1.2, there are two kinds of real-time queries in the system, i.e., the routine check and the history-based check. The routine check is made on the Road network side of the data warehouse while the history-based check is made on the *signal fact table* and connected dimension tables. In this section, a subsection is dedicated to the experimental results, regarding both of the two query types. All experiments for real-time queries were conducted on two computers, C1 and C2, in order to analyze the difference of execution times between different processors.

Apart from conducting experiments for these two kinds of queries, the execution time of the problem insertion / disabling procedures was measured (on computer C1). When inserting a problem with a problem area of 9 segments the procedure executed in less than 500 milliseconds. A problem area of 9 segments is the average size of a problem area, based on the third assumption in Section 3.2. Disabling the same problem took less than 400 milliseconds. These times are far within the requirements, presented in Subsection 3.1.2.

8.2.1 Routine Check

As discussed in Subsection 6.2.1, this experiment had the purpose of measuring the execution time of the routine check with respect to the requirements in subsection 3.1.2. Furthermore, this experiment analyzed the influence of what can possibly affect the execution time of the routine check, such as the processor speed, the memory available for the server, the size of the result set, the size of the problem area on the road network, etc.

In Figure 8.2, the execution time is shown with respect to the size of the result set, ranging from one problem to three. As the figure shows, there was no significant difference in the execution time with respect to the size of the result set. That

was also the case for other modifications, which were made in order to see how the execution time would be influenced.

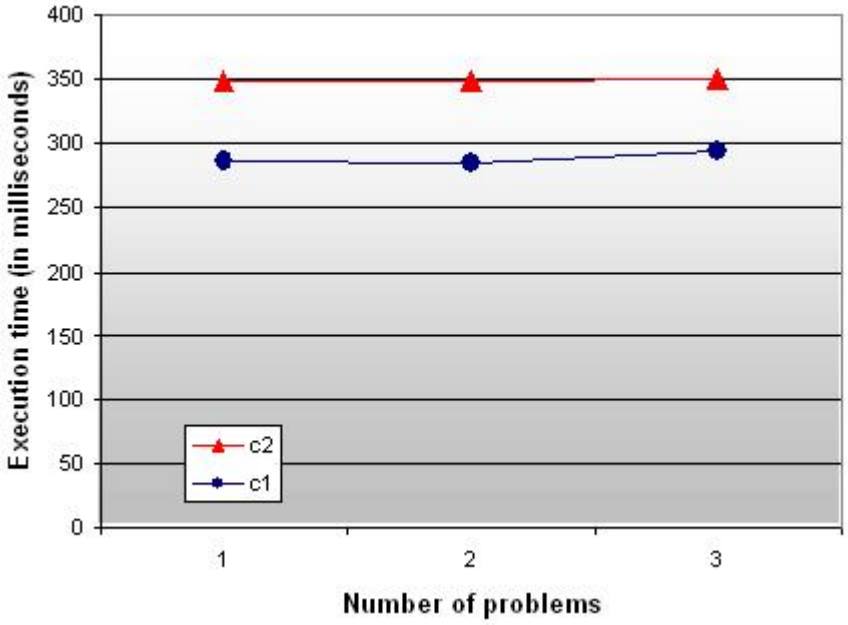

Figure 8.2: Execution time of the routine check with respect to number of problems returned.

The average execution time for computer C1 was 289 milliseconds and 349 milliseconds for C2. These times are well within the requirements of Section 3.1.2 which state that the routine check, along with segment-based map matching, may not take more than 10 seconds. As discussed in Section 6.2, the results from the experiments for the two real-time queries should indicate the maximum time allocated to the segment-based map matching procedure. With respect to the average execution time of the routine check, the segment-based map matching procedure may take up to approximately 9,7 seconds, i.e., the 10 second requirement minus 300 milliseconds.

8.2.2 History-Based Check

As discussed in Subsection 6.2.2, this experiment measured the execution time of the history-based check with respect to the size of the *signal fact table*. Experiments were done where the fact table contained up to 30 million signals, i.e., one year of data for approximately 500 cars, and the size of the fact table was altered by increasing/decreasing the number of cars for which the data was stored for. This means that the fact table always stored data from the same time period, but for different number of cars. Computers C1 and C2 were used for these experiments and the execution of the history-based check is shown in Figure 8.3.

As can be seen in Figure 8.3, the difference in execution time between the two

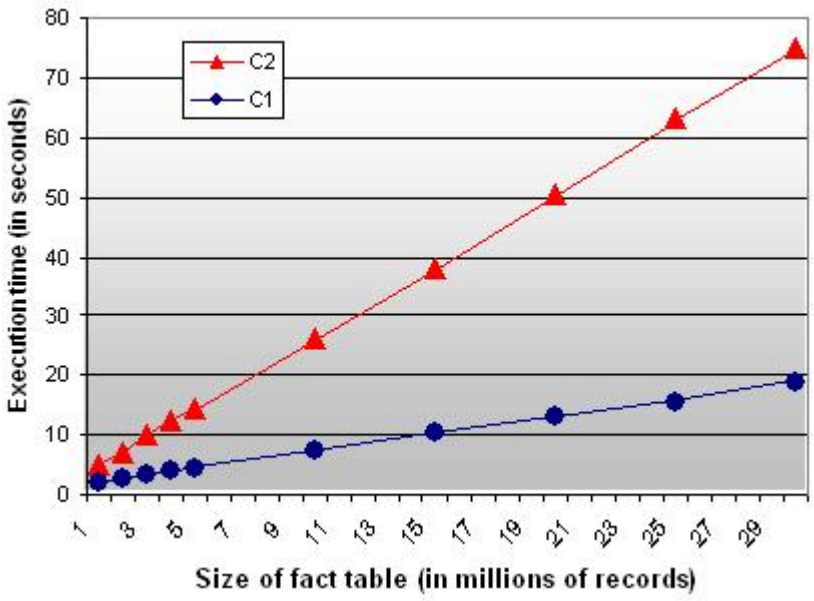

Figure 8.3: Execution time of the history-based check with respect to the size of the *signal fact table*.

computers is very much. On average, the execution time on computer C2 was more than 300% the execution time on C1. This indicates that the processor speed is highly important for queries that require many disk reads, as the history-based check does.

As default, the maximum available memory for the SQL Server is just over 500 Mb for both computers, C1 and C2. By adjusting the maximum available memory for the server, the execution time, surprisingly, decreased to approximately half its time as soon as the maximum memory was set to less than 400 Mb. Given less memory, the query optimizer seemed to choose a different strategy. The memory, used by the server during the query execution, stayed constant when less than 400 Mb was available. When more than 400 Mb were available, the memory used by the server made regular jumps from the maximum available memory down to approximately 100 Mb and back up. This indicates that the optimizer chooses to frequently fetch data to the disk when the maximum available memory is set to less than 400 Mb, while data is fetched in larger bulks having more memory available. The reason for this difference is unclear but it has a significant affect on the execution of the history-based check. The results in Figure 8.3 are from experiments where the maximum available memory for the server was set to 350 Mb. How it proved to be more efficient to have less memory available was checked for all other experiments and as for the history-based check, traffic analysis queries gave worse results when the available memory was more than 400 Mb.

If the numbers from Figure 8.3 are scaled up, we are able to estimate the size

of the fact table when the execution time exceeds the 90 second requirement from Subsection 3.1.2. That size is around 200 million records, which is approximately the number of records stored for 2.850 cars over a one year period. The system requirements state that the history-based check must execute in at most 90 seconds for a data warehouse containing data from 50.000 cars for one year. The results from this experiment show that this functionality of the system (i.e., the history-based check) does not meet that requirement. However, Figure 8.3 clearly shows the importance of the processor speed and, in Section 9.2, we discuss what alternatives to the hardware which was used for experiments are available.

8.3 Traffic Analysis

As discussed in Section 6.3, this experiment measured the execution time of different traffic analysis queries with respect to the size of the fact table. The size of the fact table was altered in the same way as for the history-based check experiment, i.e., with respect to number of cars. For these experiments, the maximum available memory for the server was set to 350 Mb, due to the same reasons as for the history-based check. The traffic analysis queries used for this experiment are the same as the ones used for model comparison in Section 4.2:

- **Query 1:** "What route is driver D most likely to take when driving in time of day period P on weekday W?"
- **Query 2:** "What is the traffic rate in Aalborg city with respect to time of day periods?"
- **Query 3:** "When driving from segment A to segment B, what route R is the most popular?"

In Figure 8.4, the results for this experiment is shown. As expected, the execution time grows linearly with respect to the size of the fact table. The reason why query 2 has a longer execution time than the other queries is the fact that it does calculation using all records of the *signal fact table*

The execution time for query 2, which has the longest execution time of the three queries, on the *signal fact table* with 30 million signals is approximately 74 seconds. While this is for 500 cars, the estimated execution time for 50.000 cars would be approximately $2\ ^1/_2$ hours, based on these numbers. That execution time meets the system requirements in Section 3.1.2 where traffic analysis queries are, in general, not assigned a specific maximum execution time.

8.4 The ETL Procedure

As discussed in Section 6.4, this experiment measured the execution time of the ETL procedure with respect to the number of cars that have sent signals for one day.

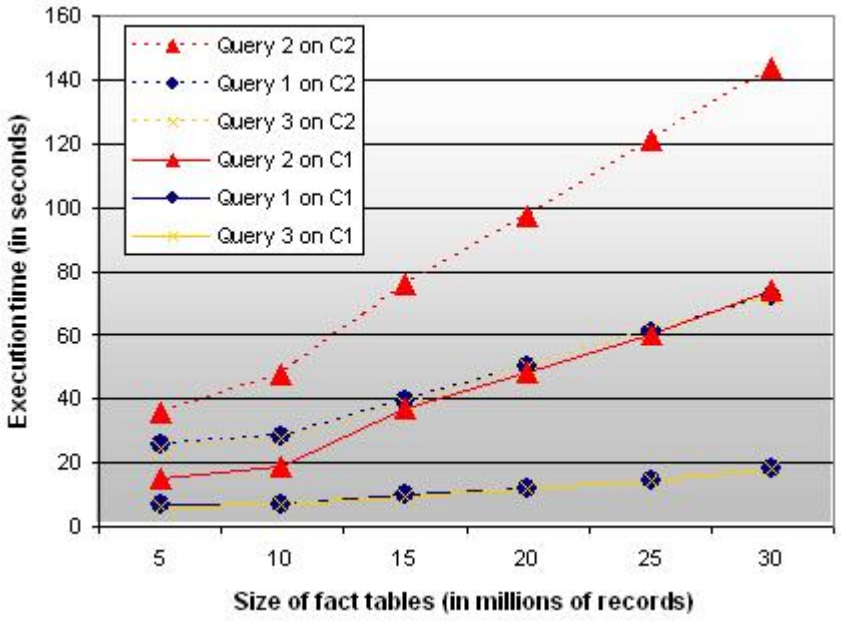

Figure 8.4: Execution time of three traffic analysis queries with respect to the size of the *signal fact table*.

Signals have been logged into the Logging database, ranging from 1.000 to 40.000 cars. The result from this experiment is shown in Figure 8.5. Concerning the amount of memory, available to the server, we examined the difference in having a maximum memory of 500 Mb and 350 Mb. The outcome was that the execution time was a little bit faster for 500 Mb as the maximum memory, e.g., for 15.000 cars the execution time for ETL was approximately 1 minute faster when the maximum memory was 500 Mb. In Figure 8.5 the execution time is shown for both computer C1 and C2 with maximum memory of 500 Mb.

The result for this experiment was almost as we had expected. The lines in Figure 8.5 do not grow as linearly as we had expected and that indicates that the growing size of the *route dimension* has significant affect on the execution time of the ETL procedure. This experiment was very time consuming and due to lack of resources, executing the ETL procedure for up to 50.000 cars was not a possibility. Based on the results we got, the estimated execution time for 50.000 cars is approximately 16 hours and 14 minutes for computer C1. This execution time of the ETL procedure meets the requirements of Section 3.1.2, where the ETL procedure is supposed to execute in at most 24 hours. That time includes the segment-based map matching of signals in the *log table*. That should, however, not add much time to the process of loading signals into the data warehouse since, as discussed in Subsection 5.1.1, signals are map matched soon after they are logged in the *log table*.

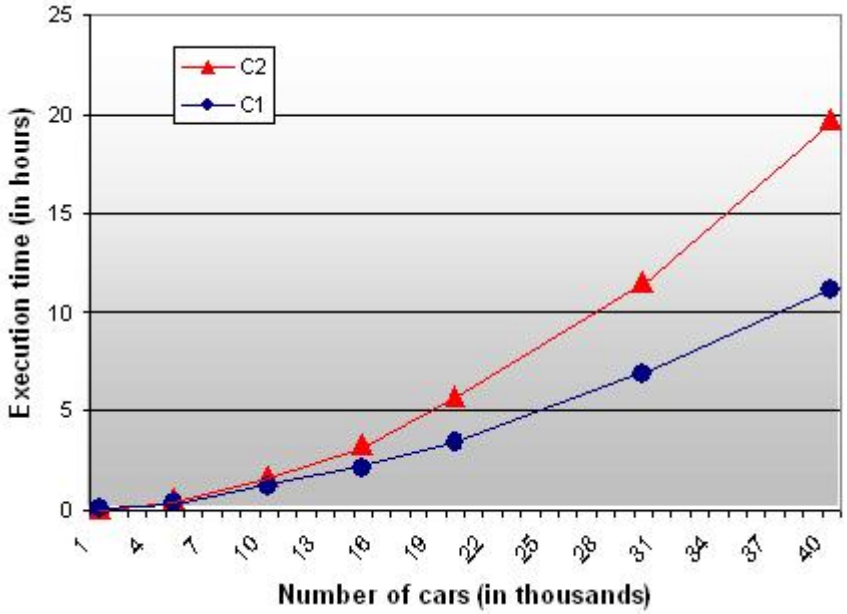

Figure 8.5: Execution time of the ETL procedure with respect to the number of cars that have sent signals to the system for one day.

Chapter 9

Evaluation

In this chapter, we analyze the experimental results of Chapter 8 and provide suggestions to possible changes in the design. Furthermore, based on the experimental results, we suggest what resources are needed when implementing the proposed system in a real-world situation.

9.1 Result Analysis

The experiments were divided into four categories; signal logging, real-time queries, traffic analysis queries, and the ETL procedure. The real-time queries were further divided into routine and history-based checks. For three of these categories, i.e., the routine check, the traffic analysis queries, and the ETL procedure, the results were as expected, and system requirements are met. For the remaining two categories, i.e., signal logging and history-based checks, the results indicated that changes have to be made to the proposed system architecture.

Concerning the logging of signals, the experimental results shows that the Logging database can handle receiving and processing signals from up to few hundred cars. As the system is supposed to be able to serve all the 50.000 personal cars of Aalborg city, we suggest a change in the overall system design. Instead of having the logic of selecting signals for real-time querying from incoming signals in the Logging database, this logic should be pushed into the Traffic management system. By doing so, the load on the Logging database would be decreased. This alternative solution was initially presented in Subsection 5.1.2, when discussing communication between the two types of clients in the system, i.e., the GPS devices of cars and the Traffic management system. However, as discussed in Section 8.1, we cannot be sure that the Traffic management system would be more capable of handling incoming signals.

The results regarding the history-based check indicate that when the data warehouse would contain the driving history of more than approximately 2.850 cars, the execution time of the query would exceed the requirement of 90 seconds. These results indicate that changes have to be made. However, the execution time of the history-based check depends heavily on the processor speed and using a faster proces-

sor, than for these experiments, would likely improve the execution time. How much is not known, but with respect to Figure 8.3, the change could be significant.

One idea for a change in the design is to let the GPS devices of cars send signals with 30 second intervals, instead of 10 second. This would decrease the number of signals that are stored in the *signal fact table* to $^1/_3$ of its size and the data warehouse could contain triple the number of cars before the execution time of the history-based query would exceed the requirement of 90 seconds. However, it would decrease the accuracy of several traffic analysis queries. For now, we accept the fact that the history-based check does only support a service to few thousand drivers.

As discussed in Subsection 5.1.1, the segment-based map matching procedure was not implemented in the system. As map matching has been the issue of existing research we considered it more important to focus on other data modeling issues in this project. In [1], Civilis et al. propose a map matching algorithm which compares the driving direction and location of a GPS signal to road segments which are represented as polylines, and assigns a segment id to the signal. The models of the proposed system architecture contain all the data needed for this sort of an algorithm.

The experimental results for the real-time queries were supposed to give indications to what time could be allocated to the segment-based map matching procedure. However, as the execution time of the history-based check proved to be beyond the system requirements, a specific time can not be allocated to the map matching procedure.

9.2 Suggested Resources

By observing the experimental results, the importance of processor speed is clearly visible. Computer C1 has Intel Pentium 4 processor, which is recommended for use in personal desktops and entry-level workstations, while C2 has a Intel Pentium M processor which is recommended for use in mobile personal computers. Neither of these processor types would be considered sufficient for the machine where the server and the Traffic management system would be stored.

Many server-class processors exist, and that type of processors would be required for the proposed system. As an example, Intel Itanium and AMD Opteron processor types are recommended for demanding enterprise-level servers and high-performance workstations (see [2] and [7]). These server-class processor types are much more powerful than the Pentium processors, which were used for experiment, especially with respect to the cache size and number of transistors. We further propose that the server machine would be a multi-processor machine, having three or more processors. One allocated to the Traffic management system, one to the ETL procedure, and one to other functionalities in the server (i.e., real-time queries, map matching procedures, etc.). That way, both the ETL procedure and the Traffic management system should not cause significant delays for other activities in the server. With respect to the database management system, having multi-processor machine does not cause a problem. As an example, MS SQL Server 2000 Developer Edition can manage up to

32 processors, and Oracle DBMS's are not bound by a specific number of processors and will handle as many processors as the operating system can support.

Regarding the client applications in cars, they would be implemented in typical GPS devices of cars. The client application would communicate with the server machine through wireless communication link.

Chapter 10

Conclusions

The main contribution of this report has been the proposal and experimental evaluation of the "Moving Objects Data Warehouse" architecture. This architecture includes a dimensional model, storing the driving history of personal cars. On this dimensional model, real-time queries can be made and drivers notified when / if there exists traffic delay in the estimated future path of the car. Furthermore, drivers are notified when approaching traffic delays.

The experimental results show that real-time queries can be made on a data warehouse, containing the driving history of cars. The results show that a data warehouse containing the history of 2.850 drivers, over a one year period can be queried in 90 seconds. This query is triggered by a GPS signal sent from a moving car, and returns information on whether the driver of the car is likely to drive upon a traffic delay, based on his driving history. While these experiments were conducted on personal computers, the performance of this kind of queries depend heavily on processor speed, and these experimental results do neither prove nor reject if history-related real-time queries scale up to the number of personal cars in Aalborg city (approximately 50.000 cars).

The proposed system architecture gives strong support for real-time queries that are triggered by GPS signals sent from moving cars, and are made on a road network database. When cars, equipped with a GPS device, are approaching a traffic delay, they can be notified of the delay in number of few seconds. Furthermore, experiments prove that the proposed architecture supports various types of traffic analysis queries, where the driving history of cars is queried. These analysis queries can relate to several different aspects, such as; drivers, cars, streets, routes, time, speed, etc.

While there is a limit to how much scalability issues regarding this architecture can be measured on personal computers, experimental results indicate that, using adequate resources, the proposed architecture can be used to build a system, which helps drivers to avoid traffic delays.

Bibliography

[1] A. Civilis, C. S. Jensen, J. Nenortaite, and S. Pakalnis. Efficient Tracking of Moving Objects with Precision Guarantees. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*. IEEE Computer Society, 2004.

[2] Intel Corporation. Microprocessor Quick Reference Guide. Published on the web, 2005. Available at `http://www.intel.com/pressroom/kits/quickreffam.htm`, accessed 23. May 2005.

[3] Microsoft Corporation. SQL Server 2000 Product Overview. Published on the web, 2005. Available at `http://www.microsoft.com/sql/evaluation/overview/default.asp`, accessed 24. May 2005.

[4] StatBank Denmark. Transport. Published on the web, 2005. Available at `http://www.statbank.dk/statbank5a/default.asp?w=1920`, accessed 22. April 2005.

[5] J. G. Hermannsson and T. Olafsson. Modelling Moving Objects using the Star Schema Design. Available at `http://www.cs.aau.dk/library/files/rapbibfiles1/1085670152.pdf`, May 2004.

[6] J. G. Hermannsson and T. Olafsson. A Foundation for a Moving Objects Data Warehouse. Available at `http://www.cs.aau.dk/library/files/rapbibfiles1/1105539328.pdf`, January 2005.

[7] Advanced Micro Devices Inc. AMD Opteron Processor. Published on the web, 2005. Available at `http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8825,00.html`, accessed 29. May 2005.

[8] C. S. Jensen, A. Kligys, T. B. Pedersen, and I. Timko. Multidimensional Data Modeling for Location-based Services. *The International Journal on Very Large Data Bases (VLDB). Vol.13, No. 1.*, January 2004.

[9] Aalborg Kommune. Traffic og Veje. Published on the web, 2005. Available at `http://www.aalborg.dk/serviceomraader/trafik+og+veje/vejnavne/gadefortegnelse.htm`, accessed 15. April 2005.

Appendix A

Logging Database Calculations

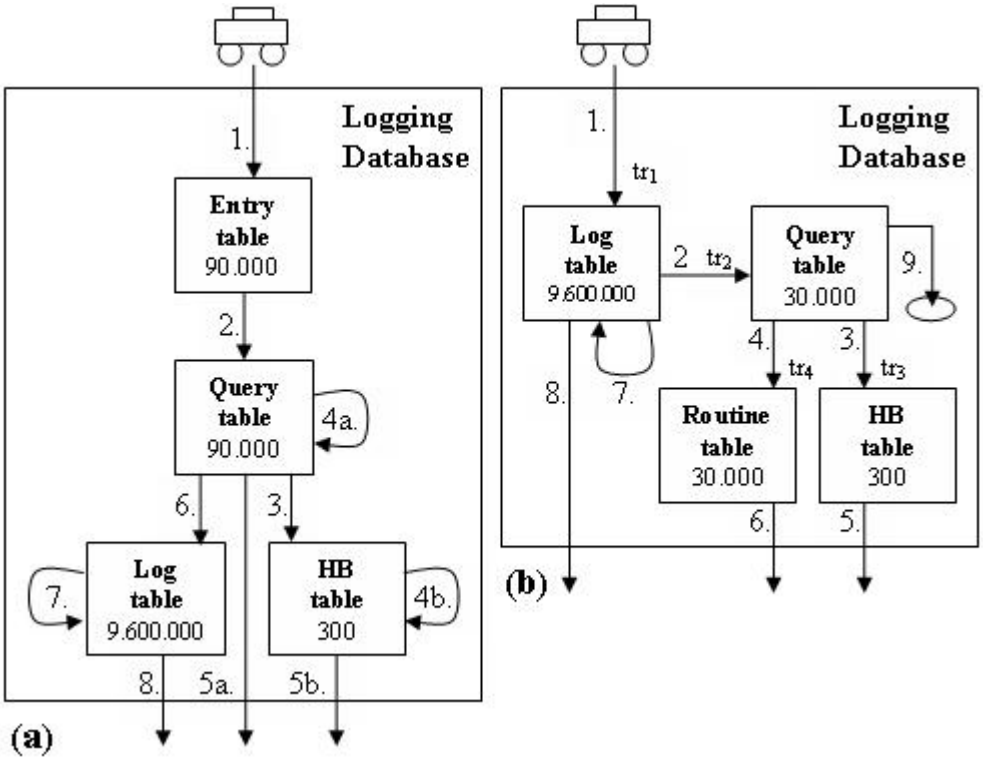

Figure A.1: Architecture and maximum table sizes (in records) for (a) the Interval model and (b) the Triggering model.

A.1 The Interval Model

Size of tables

Here is the calculation of the combined size in bytes for all tables of the Interval model. Table A.1 shows the size of each attribute of the *log table* in the Interval model. Other tables in the Interval model all consists of the same attributes.

Attributes	Data type	Size
Car	smallint	2 bytes
Driver	smallint	2 bytes
Date	integer	4 bytes
Time	integer	4 bytes
X_coordinate	integer	4 bytes
Y_coordinate	integer	4 bytes
Speed	smallint	2 bytes
Segment	integer	4 bytes
Direction	smallint	2 bytes
Start_signal	boolean	1 byte
		Total: 31 bytes

Table A.1: The size of the *log table* for both the Interval model and the Triggering model

31 bytes per record * (9.600.000 + 90.000 + 90.000 + 300) records
= 303.189.300 bytes / 1024
= 296.083 Kb / 1024
= 289,1 Mb

A.2 The Triggering Model

Size of tables

Here is a calculation of the combined size in bytes for all tables of the Triggering model. Table A.1 shows the size of each attribute of the *log table* in the Triggering model. Other tables in the Triggering model all consists of the same attributes.

31 bytes per record * (9.600.000 + 30.000 + 30.000 + 300) records
= 299.469.300 bytes / 1024
= 292.450 Kb / 1024
= 285,6 Mb

Appendix B

Data Warehouse Calculations

B.1 The Signal Star Model

Size of tables

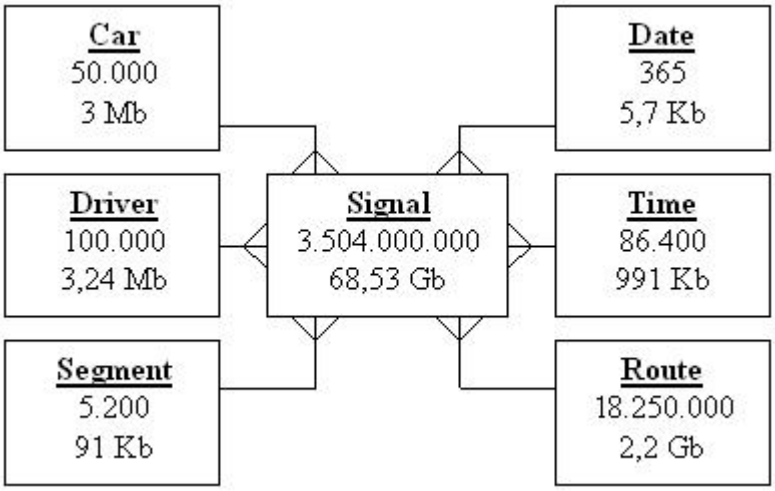

Figure B.1: The size of tables in the Signal Star model, both with respect to number of records and bytes.

These numbers are those estimated after the first year. The calculations of the tables are below:

To measure the signal table in bytes we used the following calculations

- *Size in records:* is based on the assumptions from Chapter 3.
- *Size in bytes:* Data types size for one record * all records.

- **Signal fact table**
  - *Size in records*
    - Each car sends signal every 10 seconds = 6 signal every minute
    - Each car drives on average around 32 minutes per day
    - => 6 * 32 = 192
    - Each car sends 192 signals per day
    - => 192 * 365 = 70.080
    - Each car sends 70.080 signals per year
    - There are 50.000 cars
    - => 50.000 * 70.080 = 3.504.000.000
    - There are 3.504.000.000 records in the signal fact table for one year.
  - *Size in bytes*

Attributes	**Data type**	**Size**
car_fk	smallint	2 bytes
driver_fk	smallint	2 bytes
date_fk	integer	4 bytes
time_fk	integer	4 bytes
route_fk	integer	4 bytes
segment_fk	smallint	2 bytes
speed	smallint	2 bytes
start_signal	boolean	1 byte
		Total: 21 bytes

Table B.1: The size of the *signal fact table*

23 bytes * 3.504.000.000 records = 73.584.000.000 bytes
(/1024) = 71.859.375 Kb
(/1024) = 70.175 Mb
(/1024) = 68,53 Gb
The size of the signal fact table is **68,53 Gb**

- **Route dimension**
  - *Size in records*
    - This table evolves throughout the years
      - First year: each driver takes one new route every day on average
      - Second year: each driver takes $^1/_2$ new route every day on average
      - Third year: each driver takes $^1/_4$ new route every day on average

        - and so on
    - Each car takes 4 routes every day
        - One new route
        - Three routes that someone has taken before
    - There are 50.000 cars
    - 50.000 * 365 = 18.250.000
    - For the first year 18.250.000 routes has been taken
    - (After 10 years 36.464.355 routes has been taken - by using the formula given in Section 3.2)

- *Size in bytes*

Attributes	**Data type**	**Size**
id	integer	4 bytes
start_signal	smallint	2 bytes
end_segment	smallint	2 bytes
all_segments	varchar	122 bytes
	Total:	130 bytes

Table B.2: The size of the *route dimension*

- *After 1 year:*
  130 bytes * 18.250.000 records = 2.372.500.000 bytes
  (/1024) = 2.316.895 Kb
  (/1024) = 2263 Mb
  (/1024) = 2,2 Gb

- *After 10 years:*
  130 bytes * 36.464.355 records = 4.740.366.150 bytes
  (/1024) = 4.629.264 Kb
  (/1024) = 4.521 Mb
  (/1024) = 4,42 Gb

- **Segment dimension**
  - *Size in records*
    - There are 5200 segments
  - *Size in bytes*

Attributes	**Data type**	**Size**
id	smallint	2 bytes
street	varchar	15 bytes
problem?	boolean	1 byte
	Total:	18 bytes

Table B.3: The size of the *segment dimension*

18 bytes * 5200 records = 93.600 bytes
(/1024) = 91 Kb

- **Car dimension**
  - *Size in records*
    - There are 50.000 cars
  - *Size in bytes*

Attributes	**Data type**	**Size**
id	smallint	2 bytes
lpnum	varchar	7 bytes
type	varchar	12 bytes
name	varchar	12 bytes
adress	varchar	30 bytes
	Total:	63 bytes

Table B.4: The size of the *car dimension*

63 bytes * 50.000 records = 3.150.000 bytes
(/1024) = 3076 Kb
(/1024) = 3 Mb

- **Driver dimension**
  - *Size in records*
    - There are 100.000 drivers
  - *Size in bytes*

Attributes	Data type	Size
id	smallint	2 bytes
name	varchar	30 bytes
sex	varchar	5 bytes
		Total: 37 bytes

Table B.5: The size of the *driver dimension*

33 bytes * 100.000 records = 3.400.000 bytes
(/1024) = 3320 Kb
(/1024) = 3,24 Mb

- **Date dimension**
  - *Size in records*
    - There are 365 days in one year
  - *Size in bytes*

Attributes	Data type	Size
date	integer	4 bytes
day_of_month	smallint	2 bytes
month	smallint	2 bytes
quarter	smallint	2 bytes
year	smallint	2 bytes
day_of_week	varchar	10 bytes
workday	boolean	1 byte
		Total: 23 bytes

Table B.6: The size of the *date dimension*

23 bytes * 365 records = 8.395 bytes
(/1024) = 8,2 Kb

- **Time dimension**
  - *Size in records*
    - There are 86.400 seconds in one day
  - *Size in bytes*

Attributes	**Data type**	**Size**
time	integer	4 bytes
second	smallint	2 bytes
minute	smallint	2 bytes
hour	smallint	2 bytes
period	smallint	2 bytes
	Total:	12 bytes

Table B.7: The size of the *time dimension*

12 bytes * 86.400 records = 1.015.200 bytes
(/1024) = 991 Kb

History-based check

To be able to get a clearer view on how the model deals with this query we need to give us more assumptions (The sql code for the history-based check is in Appendix D).

1) The driver is driving ten times the average. The average in this case would be that if all 100.000 drivers are driving equally much and then the signals in the signal fact table are distributed equally among the drivers. 2) We say that there is a problem on 30 segments. 3) The start segment, which the driver is starting from, is fifty times the average. The average in this case that all routes have equally many start segments, i.e., all segments are equally often a start segment. 4) We say that 1/3 of all signals comes from the time period that this history-based check is on. 5) Workdays are divided into workdays and holidays, we say that 90% comes from a workday and that this history-based check is made on a workday.

Now we can calculate the estimated hits of the history-based check on average for the Signal Star model.

We start out with 3.504.000.000 signals:

s.driver_fk = @driver: 3.504.000.000 * ((1 / 100.000) * 10) = 350.400

seg.problem = 'yes': 350.400 * (30 / 5200) = 2022

r.start_segment = @segment: 2022 * ((105 / 547.500) * 50) = 19,4

t.period = @time_period: 19,4 * (1 / 3) = 6,5

d.workday = @workday: 6,5 * 90% = 5,8

5,8 is the average estimated hits for this example, i.e., on average, there are around 6 records in the *signal fact table* that would be returned, which is reasonable since this number is the average number of hits and it is not likely that everybody in this assumption has equal number of hits. Furthermore, in this example, the traffic distributes equally throughout the segments, which is very unlikely and gives us lower number of hits.

B.2 The Subroute Star Model

Size of tables

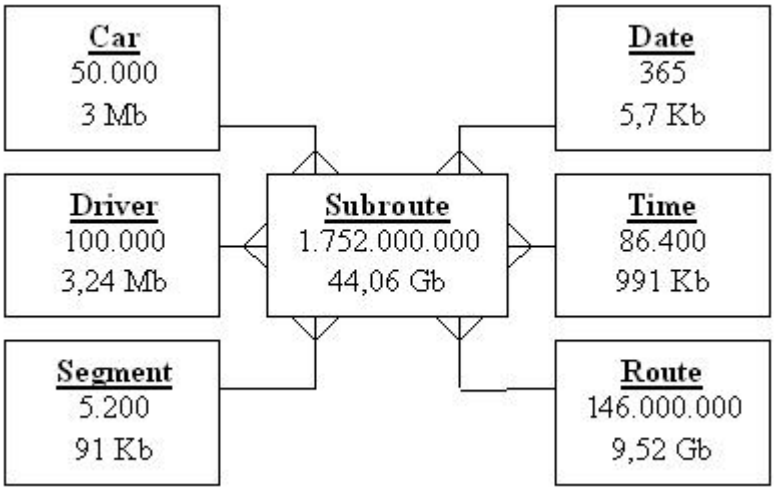

Figure B.2: The size of tables in the Subroute Star model, both with respect to number of records and in bytes.

These numbers are those estimated after the first year. The calculations of the tables are here below:

To measure the signal table in bytes we used the following calculations

- *Size in records:* is based on the assumptions from the Sections 3.1 and 3.2
- *Size in bytes:* Data types size for one record * all records

- **Subroute fact table**
  - *Size in records*
    - Each route consists of 7 streets on average
    - Each street consists of 3,47 segments on average (5200 segments/1500 streets)
    - => 3,47 * 7 = 24
    - Each route consists of 24 segments
    - => *subroute fact table* stores 24 records for each route on average
    - Each car takes 4 routes per day on average
    - There are 50.000 cars
    - There are 365 days in one year
    - => 24 records * 4 routes * 365 days * 50.000 cars = 1.752.000.000
    - There are 1.752.000.000 records in the subroute fact table for one year.
  - *Size in bytes*

Attributes	**Data type**	**Size**
car_fk	smallint	2 bytes
driver_fk	smallint	2 bytes
date_fk	integer	4 bytes
time_fk	integer	4 bytes
route_fk	integer	4 bytes
segment_fk	smallint	2 bytes
full_route	boolean	1 byte
min_speed	smallint	2 bytes
max_speed	smallint	2 bytes
avg_speed	smallint	2 bytes
signal_count	smallint	2 bytes
		Total: 27 bytes

Table B.8: The size of the *subroute fact table*

27 bytes * 1.752.000.000 records = 47.304.000.000 bytes
(/1024) = 46.195.313 Kb
(/1024) = 45.113 Mb
(/1024) = 44.06 Gb
The size of the *subroute fact table* is **44,06 Gb**

- **Route dimension**
  - *Size in records*
    - This table evolves throughout the years
      - First year: each driver takes one new route every day on average
      - Second year: each driver takes new route every day on average
      - Third year: each driver takes new route every day on average
      - and so on
    - Here in this route dimension, both full routes and subroutes are stored
    - Every route consists of 24 subroutes
    - Each cars takes 4 routes every day
      - one new route
      - For each new route, 8 out of 24 subroutes are new subroutes, i.e., 1/3 of the new route is new subroutes while 2/3 of the new route, the driver has already taken before
      - three routes that someone has taken before
    - There are 50.000 cars
    - 50.000 * 365 = 18.250.000
    - For the first year 18.250.000 full routes has been taken
    - Each new route consists of 8 new subroutes
    - => 8 * 18.250.000 = 146.000.000
    - After 10 years 291.714.844 routes has been taken - by using the formula given in Section 3.2
  - *Size in bytes*

Attributes	**Data type**	**Size**
id	integer	4 bytes
start_segment	smallint	2 bytes
end_segment	smallint	2 bytes
all_segments	varchar	62 bytes
	Total:	70 bytes

Table B.9: The size of the *route dimension*

  - *After 1 year:*
    70 bytes * 146.000.000 records = 10.220.000.000 bytes
    (/1024) = 9.980.469 Kb
    (/1024) = 9747 Mb
    (/1024) = 9.52 Gb

– *After 10 years:*
70 bytes * 291.714.844 records = 20.420.039.080 bytes
(/1024) = 19.941.444 Kb
(/1024) = 19.474 Mb
(/1024) = 19 Gb

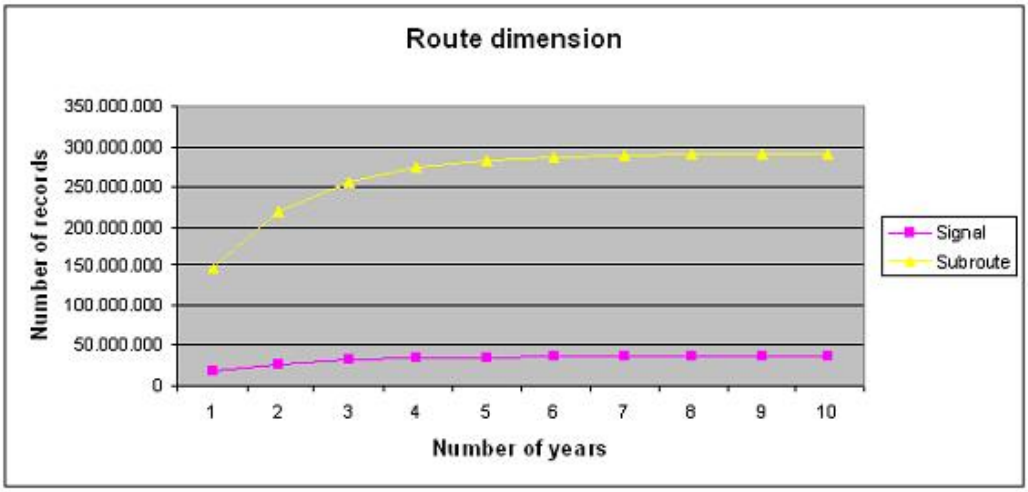

Figure B.3: The growth of the route dimension for both the Signal Star model and the Soubroute star model.

In Figure B.3 the growth of the route dimension (both for the Signal Star model and the Subroute Star model) is shown and how it will evolve in 15 years according to the formula given in Section 3.2. This means that after few years the tables will be nearly static.

- **Segment dimension** - Same as for Signal Star model
- **Car dimension** - Same as for Signal Star model
- **Driver dimension** - Same as for Signal Star model
- **Date dimension** - Same as for Signal Star model
- **Time dimension** - Same as for Signal Star model

History-based check

To be able to get a clearer view on how the model deals with this query we need to give us more assumptions (The sql code for the history-based check is in Appendix D).

1) The driver is driving ten times the average. The average in this case would be that if all 100.000 drivers are driving equally much and then the signals in the *signal fact table* is distributed equally among the drivers. 2) We say that there is a problem on 30 segments. 3) The start segment, which the driver is starting from, is fifty times the average. The average in this case that all routes have equally many start segments, i.e., all segments are equally often a start signal. 4) We say that 1/3 of all signals comes from the time period that this history-based check is on. 5) Workdays are divided into workdays and holidays, we say that 90% comes from a workday and that this history-based check is made on a workday.

Now we can calculate the estimated hits of the history-based check on average for the Signal Star model.

We start out with 1.752.000.000 signals:

s.driver_fk = @driver: 1.752.000.000 * ((1 / 100.000) * 10) = 175.200

seg.problem = 'yes': 175.200 * (30 / 5200) = 1011

r.start_segment = @segment: 1011 * ((842 / 4.380.000) * 50) = 9,7

t.period = @time_period: 9,7 * (1 / 3) = 3,2

d.workday = @workday: 3,2 * 90% = 2,9

2,9 is the average estimated hits for this example, i.e., on average, there are around 3 records in the *subroute fact table* that would be returned, which is reasonable since this number is the average number of hits and it is not likely that everybody in this assumption has equal number of hits. Furthermore, in this example, the traffic distributes equally throughout the segments, which is very unlikely and gives us lower number of hits.

B.3 Comparison of History-Based Checks

Based on the example introduced in Sections B.1 and B.2 we calculated the estimated hits of the history-based check for both the Signal Star model and the Subroute Star model. For the Signal Star model the average estimated hits were 5,8, i.e., around 6 records in the *signal fact table* that would be returned. For the Subroute Star model the average estimated hits were 2,9, i.e., around 3 records in the *subroute fact table* would be returned. These numbers are reasonable since this number is the average number of hits and it is not likely that everybody in this assumption has equal number of hits. Furthermore, in this example, the traffic distributes equally throughout the segments, which is very unlikely and gives us lower number of hits.

Both models support the history-based check. The Subroute Star model returns fewer hits in the procedure; however, that is not that big difference. While the Subroute Star model has 2,9 hits on average, according to the calculations above, the Signal Star model has 5,8 hits on average. That is in the Signal Star model there

are 2 times more hits for the history-based check. The difference can be explained by looking at how the *subroute fact table* and *signal fact table* are built up.

If we look at how many records are in these two tables for one year, we have the same difference, i.e., 52.560.000 records in *subroute fact tabl*e and 105.120.000 records in *signal fact table*. 5,8/2,9 = 105.120.000/52.560.000 = 2.

This shows that both models return proportionally equally sized result sets to the history-based check.

Appendix C

Road Network Calculations

C.1 The Connected Model

Size of Tables

Size of tables in the Connected model are calculated by using numbers from Chapter 3.

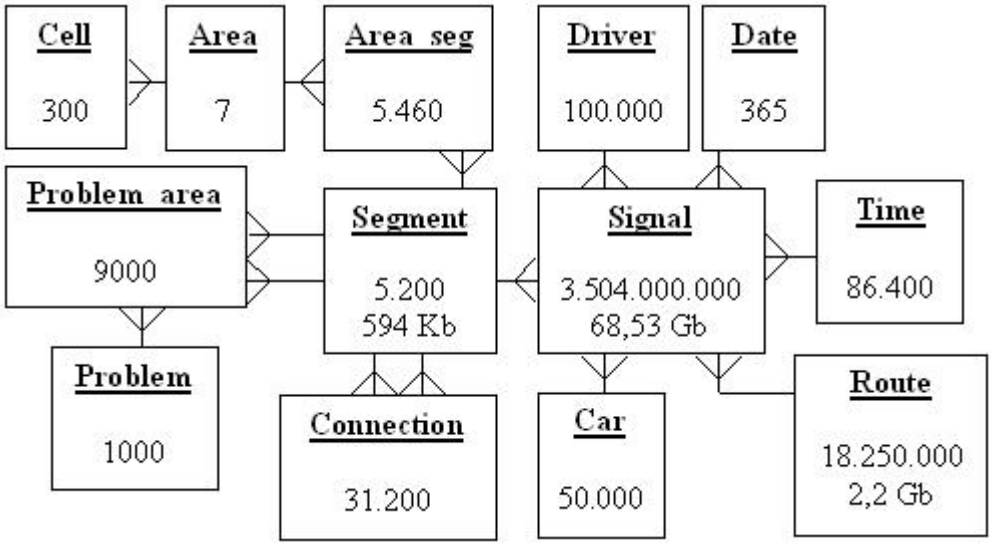

Figure C.1: Size of the tables of the Connected model.

Below is the estimated size of the tables in records three for of the largest tables of the Connected model. Furthermore, the *segment dimension* were each record is rather large, is calculated in bytes in order to show how little the addition of the road network adds to the overall size of the data warehouse. The number of records in the tables are found from the assumptions in Section 3.1 and 3.2.

- *area_seg*

  - We assume that 5% of all segments are contained in two areas.
  - 5200 segments + (0.05 * 5200) segments in two areas = 5460 'area_seg' records.

- *connection*
  - We assume that each segment is connected to 6 other segments (from and to) on average
  - 5200 segments * 6 connections per segment = 31.200 connections

- *problem_area*
  - Each segment has 3 from-segments and 3 to-segments (on average) and the problem area is the segments possible to take into the problem segment and the segments possible to take into those (see 'routine check' in the 'calc_notes- file).
  - 1000 problems * (3 * 3) problem area segments per problem = 9000 'problem_area' records.

- *segment in bytes*

Attributes	**Data type**	**Size**
id	smallint	2 bytes
length	smallint	2 bytes
polyline	varchar	35 bytes
street	varchar	15 bytes
district	varchar	15 bytes
city	varchar	15 bytes
from_street	varchar	15 bytes
to_street	varchar	15 bytes
max_speed	smallint	2 bytes
problem?	boolean	1 byte
		Total: 117 bytes

Table C.1: The size of the *segment dimension*

117 bytes * 5.200 records = 608.400 bytes
(/1024) = 594 Kb

C.2 The Internal Model

Size of Tables

The blow-up of the *segment dimension* can be roughly calculated given the assumptions in Sections 3.1 and 3.2.

- 3 segments that one can drive from each segment (on average)
- 25% of all segments are contained in two cells
- There are 5200 segments
- Around 1000 problems are inserted in one year
  - There are 9 segments that are in the problem area for each segment that has a problem
  - Around 9000 segments are inserted in one year which are in the problem area

Then we can calculate the blow-up to see how many records are inserted into the *segment dimension* for one actual segment, on average:

1 segment ∗ 3 to_segment per segment ∗ 1,25 cell per segment
∗ 1,19 (1000 problem_root / 5200 segments) problem_root per segment
∗ 2,73 (9000 problem_area / 5200 segments) problem_area per segment
= 12,18 records in the *segment dimension* per actual segment

The number 1,02 problem_root per segment is calculated from the assumption that there are 1000 problems per year. 1000 problems / 5200 segments gives us 0,2.

The number 2,37 problem_area per segment is calculated similarly. If there are 1000 problems per year and there are 3 segments connected to the segment that has a problem and 3 segments connected to each of these 3 segments that are connected to the problem, there are 9 segments that are in the problem area for each segment that has a problem. This means that for 1000 problems per year there are 9000 segments that are in the problem area. 9000 problem area / 5200 segments give us 1,73.

Note that the numbers 1,19 and 2,37 are just for one year and the number 12,18 records in the *segment dimension* per actual segment is just for the first year. Then we can calculate the size of the *segment dimension* after the first year.

12,18 records per actual segment ∗ 5200 segments = 63.350

Finally, when inserting a problem the dimension grows and based on the assumptions in Section 3.1 and 3.2, we can estimate how many records are inserted into the *segment dimension* for every new problem.

(1 actual root segment ∗ 3 to_segments per segment ∗ 1,25 cells per segment)
+ (9 actual area segments ∗ 3 to_segments per segment ∗ 1,25 cells per segment) = 37.5

Then we can estimate that around 37.5 records are inserted (on average) for every new problem. Here below is the size of the *segment dimension* in bytes in the Internal model.

- *segment in bytes*

Attributes	**Data type**	**Size**
id	smallint	2 bytes
length	smallint	2 bytes
polyline	varchar	35 bytes
street	varchar	15 bytes
district	varchar	15 bytes
city	varchar	15 bytes
from_street	varchar	15 bytes
to_street	varchar	15 bytes
max_speed	smallint	2 bytes
to_segment	smallint	2 bytes
area	smallint	2 bytes
cell	smallint	2 bytes
x_start	int	4 bytes
x_end	int	4 bytes
y_start	int	4 bytes
y_end	int	4 bytes
problem_root	int	4 byte
problem_area	int	4 byte
problem_description	varchar	10 byte
problem_criticality	smallint	2 byte
problem_start_time	datetime	8 byte
problem_end_time	datetime	8 byte
problem_currently_active?	boolean	1 byte
		Total: 175 bytes

Table C.2: The size of the *segment dimension*

175 bytes * 63.350 records = 11.086.250 bytes
(/1024) = 10.826 Kb
(/1024) = 10,6 Mb

The *segment dimension* is 10,6 Mb after the first year. According to the blow-up calculations, and if 1000 problems would be inserted each year, the size of the *segment*

dimension would grow each year by 6,25 Mb.

37,5 records $*$ 1000 problems = 37.500 records
=> 37.500 records $*$ 175 bytes = 6562500 bytes
=> 6562500 bytes / 1024 / 1024 = 6,25 Mb