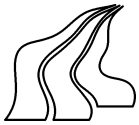
Data Warehouse Based Traffic Jam Detection

Semester:
DAT6
Project group:
E4-201
Group members:
Jan Eliasen
Casper Kjær
Helen Urban

Aalborg University
Department of Computer Science.
Fredrik Bajers Vej 7E, 9220 Aalborg Ø.

Aalborg University

Department of Computer Science.
Fredrik Bajers Vej 7E, 9220 Aalborg Ø.



Title:

Data Warehouse Based
Traffic Jam Detection

Project period:

1st February 2002 to
21st June 2002

Semester:

DAT6

Project group:

E4-201

Group members:

Jan Eliasen
Casper Kjær
Helen Urban

Supervisor:

Torben Bach Pedersen

ABSTRACT:

It would be preferable if drivers could be warned about traffic jams such that they can take another route to their destination. In this thesis, methods for detection of traffic jams are designed. A data warehouse containing historical traffic information is designed to test whether it can be used to detect non-periodic traffic jams. A prototype of the design is implemented to test the three methods on realistic data generated by a data generator. The results from the methods are compared to each other and combinations of the three methods are also considered. The experiments indicate that traffic jams can be detected and the detection process can be improved by optimizing different parameters. The force of data warehouses, namely preaggregation, is not used. Therefore, a normal relational model could have been used instead.

Number printed: 7

Number of pages: 133

Preface

This report is the masters thesis of group E4-201's and it serves as documentation for the project work on the DAT6 semester in the period from the 1st of February 2002 to the 21th of June 2002 at the Unit for Database and Programming Technologies at the Department of Computer Science at Aalborg University.

This thesis is based on the experiences from the DAT5 project during fall 2001. These experiences are documented in a report but this thesis can be read and understood independently. Knowledge of databases and data warehouses is an advantage in understanding the thesis. Chapter 3 and appendix B are from the DAT5 report written in fall 2001.

Bibliographic references are given as [author, publishing year], or [author, publishing year, page number] when referring to a special page.

We would like to thank the following persons for contributing to the project:

Professor Thomas Brinkhoff for providing a data generator, for answering questions about it, and for supplying source code.

Martin Jensen, Oracle Denmark, for answering questions about technical details in Oracle.

Mads Orbesen Troest for linguistic supervision.

Aalborg University, 14th of June 2002

Jan Eliasen

Casper Kjær

Helen Urban

Contents

1 Introduction

Traffic jams are an everyday issue. Every day many people are delayed in their driving because they are stuck in a traffic jam. Traffic jams can also be the reasons for accidents because drivers might not pay attention to the changing traffic conditions, like reduced speed and increased number of cars. As drivers approach a traffic jam they might not have time enough to stop before they hit the car in front of them. This scenario will make the traffic jam even worse. Another concern regarding traffic jams is that people that have been stuck in a traffic jam are delayed. If drivers are in a hurry, if they for instance have to catch a plane, the traffic jam will slow them down and therefore this might cause them to increase their speed once out of the traffic jam. This will endanger other drivers and the risk of accidents will increase. The most important matter, however, is the annoyance traffic jams can cause the drivers. When drivers are stuck in a traffic jam they get annoyed and frustrated because they will be late and perhaps mostly because they are wasting their time. Drivers would probably be grateful if they somehow could avoid getting stuck in traffic jams.

Traffic jams can not be fully avoided but the impact of a traffic jam can be decreased by redirecting cars heading towards it. Drivers that are heading in the direction of a traffic jam but have not reached it yet can be informed about the traffic jam such that they have the opportunity to choose a different route. If most of them choose another route, the traffic jam will increase in size more slowly or perhaps even decrease. This will cause the traffic jam to dissolve faster. A way to inform drivers about traffic jams ahead is to have a traffic jam detection service. This service must detect traffic jams and inform drivers about them. In this report the part of the service that detects the traffic jam will be designed.

In order to inform drivers about a traffic jam it must be determined that there is a traffic jam. This can be done in various ways. The approach suggested in this report assumes that a representative number of cars report their position frequently while

driving. From these positions information can be derived in order to determine where there are traffic jams. In this report three different methods for detecting traffic jams are proposed and a combined method involving all three methods is designed.

The methods are primarily designed for detecting non-periodic traffic jams which means that repeating traffic jams, for instance during morning rush hours, will not be the main issue. The design allows for detecting these reoccurring traffic jams but because these might be anticipated by the driver, the design focuses on detecting traffic jams that have suddenly occurred, for instance because of a road accident. In order not to detect the repeating traffic jams the actual situations will be compared to past behavior. Information about past behavior could be available by maintaining historical information in an ordinary, relational database. This type of database, however, supports retrieval of individual values, not quantitative, aggregate information. Therefore, the historical information is instead stored in a data warehouse.

A data warehouse is a special kind of relational database with a simple design optimized for providing fast answers to aggregate queries. In the design of traffic jam detection methods the data warehouse could provide the necessary historical information. By comparing this to the actual traffic situation it could be determined if the actual situation is normal or if there is a traffic jam. The historical traffic information is constituted by large amounts of data that must be aggregated in order to provide a general view of the traffic situation. The benefits of storing the data in a data warehouse is that required aggregated information can be retrieved fast despite the large amount of data.

Most contemporary traffic jam detection methods are based on location-dependent devices such as monitors above the road or electromagnetic sensors in the road. A consequence of this approach is that traffic jams can only be detected in places where these devices have been installed. In order to detect traffic jams in all places such devices must be installed on all roads and in all parts of the road. This can be a very expensive solution. Instead, a design is suggested that based on the locations of the cars can detect traffic jams on all roads.

Data warehousing is widespread in the area of business intelligence and On-Line Analytical Processing (OLAP). It is often used for quantitative analysis on sales with the purpose of determining increases and decreases of sales in products and stores. A relatively new area is the data warehouse based analysis of spatio-temporal objects. The design of data warehouses for spatio-temporal data has been investigated in order to determine requirements and restrictions on such designs. This investigation provides the background for designing an application that utilizes data warehouses for spatio-temporal data. The purpose of the design presented in this thesis is to investigate the applicability of data warehouses for providing sufficient historical information to detect traffic jams.

The remainder of the thesis is structured as follows. Chapter 2 gives a discussion

of traffic jams and the technology necessary for detecting them. The chapter is concluded with some requirements for a traffic jam detection service. Chapter 3 contains a brief overview of general data warehouse theory. Chapters 4, 5 and 6 each contain a different method for detecting traffic jams and chapter 7 presents a design of a method combining the three previous methods. Chapter 8 contains a description of the data generator used to generate realistic data for testing the design. In chapter 9 the results of the tests are explained. In chapter 10 the conclusion is presented together with suggestions for future work in the area. There are two appendices in the report. They contain the source code of the implementation and a short description of the Global Positioning System (GPS).

2 Problem Description

The design of a traffic jam detection service requires knowledge about traffic jams and other traffic related topics. The aim of this chapter is to examine these topics and provide the necessary information for designing a traffic jam detection service. This includes considering what a traffic jam is, how it occurs, how it is detected and suggestions for other detection methods. The chapter also considers the necessary road information and it discusses other traffic jam related problems. The chapter is concluded with some requirements for the design of a traffic jam detection service together with a scenario description.

2.1 Traffic Jam Description

In this section, a discussion of how to define a traffic jam is provided and situations causing traffic jams are discussed.

2.1.1 Definition of Traffic Jam

A traffic jam is not a concept that can be universally defined because each person has an individual opinion. People would in general agree on the existence of traffic jams in extreme cases but in the less extreme cases people might have different opinions. For a given situation some people might consider it a small traffic jam whereas others might think it is only a bit more traffic than usual. Therefore, it might not be possible to give a definition that everyone would agree upon but a

good definition would be such that most people agree in most cases whether there is a traffic jam or not. A definition could be based on the number of drivers that are stuck in the traffic jam or it could be based on the amount of time that the drivers are delayed. It could also be considered how annoyed the drivers are such that if more than a certain percentage of the drivers are annoyed by the slow traffic it will be considered a traffic jam. This is not easy to measure, however. Furthermore, the definition of a traffic jam should depend on the local conditions and it might therefore be difficult to cover all special cases. For these reasons, no definition will be given. Instead, to avoid misunderstandings, and to create a common consensus, the following statement is used to indicate a traffic jam.

A traffic jam is a situation with slow and heavy traffic.

2.1.2 Causes of Traffic Jams

An obvious reason for a traffic jam to occur is a road accident. If an accident happens, the cars involved might block the road and if the road is totally blocked, no cars will get past the point of the accident. Cars that are heading in the direction of the accident might not know that an accident has occurred. They will continue towards the point of the accident but they will not be able to get past it. Therefore, the traffic will accumulate and a traffic jam will arise.

A similar case for a traffic jam to arise is road construction work. In such cases there might be a reduced speed limit, the road might be narrow and sometimes only one lane is open such that cars can only pass in one direction at a time. If the number of cars heading for the point of the road construction is higher than the number of cars being able to pass through the road construction the cars will line up behind each other. This will increase the time it takes to pass through the area and at some point there will be a traffic jam.

Traffic jams often occur during rush hours. During this time of day the traffic is highly increased because a lot of people need to go the same way at the same time. At some point there will be too many cars compared to the size and conditions of the road. This will drastically reduce the speed of the cars, and a traffic jam thus becomes inevitable simply due to overload. When the number of cars entering the congested area decreases the traffic jam will slowly break up and the flow will again reach a normal level.

Traffic lights on a road can also be a reason for congestion. The lights might not let enough cars through so the cars will line up behind each other. This means that some cars might have to wait for several green lights before they can pass through the intersection. Even though the situation seems under control the cars might have to wait for a long time.

A traffic jam on a road can affect adjacent roads. Cars on adjacent roads might not be able to leave these roads and enter the congested road because this road is already full. This might cause a traffic jam on the adjacent roads in the direction towards the congested road. Furthermore, cars on the congested road might turn to the adjacent roads to get away from the traffic jam. This will cause an increased load on these roads and a traffic jam might occur here as well.

From these considerations a common understanding is obtained that can be used for further analysis throughout the chapter.

2.2 Traffic Jam Detection

In order to design a traffic jam service, detection methods of existing services should be considered and various new approaches should be suggested that provide other or better solutions. This section contains an overview of present traffic jam detection techniques and considerations on new methods.

In order to inform drivers about traffic jams it must be determined that there is a traffic jam on the road. One way of determining this is by having coils in the road. They can detect the speed and number of cars passing over them and they can distinguish the different lanes of a road such that only when all lanes in the same direction have slow and heavy traffic a traffic jam is detected. In Denmark, in the town of Aalborg, this system is implemented at the ends of the tunnel under the Limfjord where traffic jams frequently occur. The speed and number of cars is recorded periodically within a given time interval. If five or more cars have a low speed the situation is considered a traffic jam. A disadvantage is that the coils can only provide information about the specific point where they are placed. In order to provide information about an entire road many coils must be distributed over the road. Furthermore, the coils cannot determine the individual cars from each other.

In some places traffic jams are detected using cameras filming the road. The information from the cameras is used by a computer vision system that can distinguish the cars and their speed together with their type, such as motorcycle, car or truck.

Another way of detecting traffic jams is if people have seen the traffic jam or drivers are stuck in it they might call the police station or radio station. These stations could send out warnings on the radio such that people have the option of choosing another route.

If an accident has occurred and the police is called, they have the option of informing other drivers about the accident such that they can choose another route.

When a traffic jam has been detected, people can be informed in various ways. Some traffic jams are reported in the morning news on TV. Traffic jams can also be reported

on the radio. Several countries use a system called Radio Data System (RDS). When a traffic jam has been detected, a notice is typically sent out from special RDS radio stations. A special RDS radio can receive this information and interrupt the current radio function such that the listener is informed even though the car stereo is playing a CD or cassette.

Using these approaches, only a limited amount of information is received. They will not provide a steady, representative source of data.

The position and speed can be determined by a mobile phone if the driver has one in the car. Many people have a mobile phone so the percentage of drivers represented by this approach will be very high. Using mobile phones, the individual cars can be distinguished from each other but the position, on the other hand, cannot be determined very precisely. The positions will typically be determined with an accuracy of 100 to 500 meters. If two roads are close it might therefore be impossible to determine on which one a given position is recorded. Furthermore, if the speed is determined from these positions by computing the Euclidean distance it might differ much from the actual speed.

Another way is to use the GPS system, briefly described in appendix B. If a driver has a GPS receiver in the car this receiver can very precisely determine its position and from this position the speed can be calculated. A problem with using GPS is that it is not very widespread. Not many drivers have a GPS receiver in their car and therefore, basing a traffic jam detection service on cars with GPS receivers would not provide sufficient data to make conclusions about traffic jams. Another disadvantage is that it is up to the drivers to report their position. The GPS receivers determine their position but they do not automatically forward this information to anyone.

It might also be possible to install transmitters along the roads. If these transmitters send out signals about their position and road identity this signal can be picked up by devices in the cars. In this way cars can determine their position and speed quite accurately. This technology has not yet been developed and it could furthermore be an expensive solution.

Coils and cameras provide accurate information. In order to base the detection of traffic jams on every road on such devices, these must be installed on every road and this is an expensive solution. Relying on reportings from drivers or police is not a reliable method because there is no guarantee of how many traffic jams will be reported. Using mobile phones, on the other hand, has the advantage of widespread use, and therefore the potential of becoming a representative method of collecting data. The disadvantage is that GSM positioning is not precise enough to determine which road the car is on. GPS has the accuracy needed to determine the road, but the usage is not widespread enough, and therefore sufficient data can not be provided.

2.3 Infrastructure Description

This section contains a description of existing road information and additional information required by traffic jam detection methods.

When cars are moving they follow a network of roads and intersections. Therefore, the traffic jam detection service need only consider this network instead of the complete two-dimensional space. When analyzing traffic and detecting traffic jams it is important to be able to distinguish different roads from each other such that it can be uniquely determined on which road each car is driving. Roads have a name but all roads in Denmark furthermore have a *road code* of seven digits attached. The country of Denmark is divided into 275 smaller areas each called a kommune and each of these covers a small area usually centered around a town. The first three digits of a road code are a code for the kommune to which the road belongs, and the last four digits are a unique number within the kommune for that road. The road codes for all roads in Denmark are easily accessible information because they can be obtained from the administrating offices in each kommune.

For a traffic information service it might not be sufficient to consider roads because roads can be long and the conditions in one end of the road might differ from the conditions in the other end. If a road is longer than 100 km, for instance, it is not interesting for drivers in one end to know about a traffic jam in the other end. Furthermore, it might be difficult to detect traffic jams in such a situation, because the traffic jam conditions might only apply for part of the road. To avoid these problems, a road can be divided into smaller parts represented by straight line segments with starting and ending coordinates. The starting and ending point of a segment can be determined by an end of a road, a crossing or a curve. If the road bends there will be an increasing distance between the road and the segment. When this distance exceeds a given threshold the first segment is ended and a new one is added to reduce the distance to the road as illustrated in figure 2.1 on the following page. The two roads in the figure are represented by five segments. The intersection determines a division of segments and the curve causes an increased distance between the road and the segment. At the point of the arrow the distance exceeds the threshold and a new segment is created.

Information about road segments already exists. It is maintained by Dansk Adresse- & Vejdatabase (DAV) which is a Danish address and road database. DAV has among other things information about road segments, road codes, road names, kommune codes, road classifications and postal codes.

Another concern for a traffic information service is that traffic can be different depending on the direction. For instance in the morning, the direction towards town might be congested whereas the opposite direction might be much less crowded. It is therefore necessary to be able to distinguish the two opposite directions from each

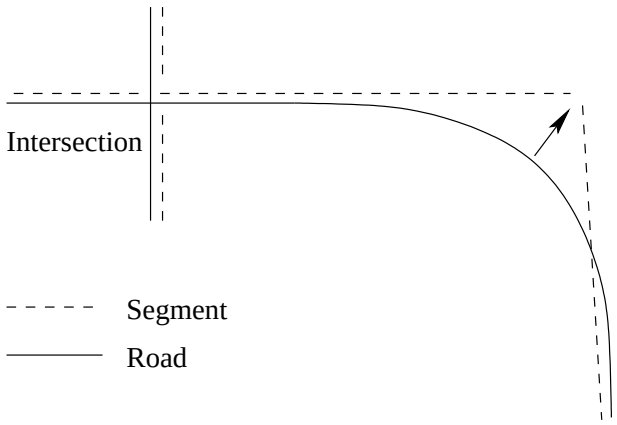


Figure 2.1: Five segments representing two roads

other. If roads have been divided into segments a further division into the two possible directions would be a simple task. A road is divided into segments and a segment is divided into two segments with opposite directions. Segments that were originally one-way segments are not divided into two segments. For the remaining part of this report, a segment is defined to be part of a road and having only one direction.

2.4 Applications of Traffic Information

This section considers the benefits of a traffic jam detection service and discusses some of the properties that such a service could have.

2.4.1 Traffic Jam Information Applications

Traffic information can be used in different ways and for different purposes. It could be used to provide valuable information about traffic conditions to drivers. This could for instance be information about the positions of traffic jams, estimated travel time through the congested area and suggestion of a different route. Traffic jam detections are currently used for reducing the allowed speed on the road while the traffic jam is present. This is to obtain a more regular traffic and to avoid that cars heading in the direction of the traffic jam drive so fast that they are not able to stop before hitting cars already in the traffic jam.

Traffic jam information can also be used for statistical information. For instance, if traffic jams are detected many times on the same road it might be beneficial to extend the road with an extra lane, change the speed limits, build a bypass road or

change the intervals between green lights at intersections.

Another application in the area of traffic jams could be to predict future traffic jams. If patterns in the driving can be compared with past information it might be possible to determine that a traffic jam will occur even before cars get stuck. If, for instance, it is observed that many cars are heading in the same direction and similar behavior has previously resulted in a traffic jam it is very likely that a traffic jam will occur again. A part of the drivers heading in the direction of the predicted traffic jam could be warned and the traffic jam might be avoided.

2.4.2 Detection Probability

A concern regarding the report of traffic jams is the certainty. A traffic jam can be reported if it is likely, that there is a traffic jam or it can be reported only if it is certain that there is a traffic jam. There are two different kinds of uncertainties that must be considered when deciding whether to report a traffic jam. As described previously, it is difficult to give a definition of a traffic jam because people have different opinions. Therefore, they also have different preferences of which traffic jams they would like to be warned about. For a given situation people that consider it a traffic jam would prefer to be warned and they might get annoyed if they are not warned. On the other hand, people that do not consider it a traffic jam might be annoyed for being warned about nothing. The second uncertainty that should be considered is the uncertainty related to the detection methods. Assuming that all people agree on a specific traffic jam definition there will still be uncertainties related to the methods. This is both because of the physical devices used to determine speed and position and because of the different conditions on the roads. It simply will not be possible to design a method that takes all road information into account.

The solution to the problem about uncertainty is to minimize the number of false positives and negatives. False positives are the cases where the methods detect a traffic jam that is not there and false negatives are cases where the methods do not detect a traffic jam even though there is one. The goal is to determine when to report traffic jams such that least people are annoyed about wrong detections. If many people are annoyed about not being informed more traffic jams should be detected. If this approach is used, it should be taken into account that people might find it very annoying to get stuck in a traffic jam without being warned. Receiving a warning if there is not a traffic jam would to most people be less annoying. Furthermore, if people are warned even though there is no heavy traffic jam they might still choose another route and this might prevent a heavy traffic jam from occurring. It might also be that the alternative route is only a few minutes longer. If this is the case, choosing this route might be preferable compared to the risk of getting stuck in a traffic jam for a long time. Therefore, instead of minimizing the number of false positives and negatives, the number of false negatives should be minimized rather

than the number of false positives. This, however, is only until the increase in the number of false positives gets too high.

A way of determining when to report traffic jams is to have test persons that volunteer to give feedback on the warnings they receive. This way, the detection process can be adjusted such that it does not detect too many or too few traffic jams. Another solution is to let people decide for themselves how certain the traffic jam should be before they are informed. When people sign up for a detection service, for instance, they could have the possibility of selecting between different types of certainties of the traffic jams. They could for instance be able to choose between three types of traffic jams, such as heavy traffic jams, high probability of traffic jams and small indication of traffic jam. This would allow people to select the type of certainty that is best suited for them.

2.4.3 Selection of Traffic Information

As described in section 2.1.2 there can be many reasons for a traffic jam to occur. These reasons can be divided into two types, those that are periodic such as daily rush hours and Christmas traffic, and those that happen at arbitrary points in time such as road accidents.

Assuming that a person leaves for work every morning at the same time and follows the same route. If there is usually a traffic jam on that route at that time every morning the person will know about this. He or she has the option of choosing another route but it might be that no other route is faster and therefore, the driver stays on the same route even though it is congested. If this person has signed up for a traffic jam detection service he or she will not be interested in being informed every morning that there is a traffic jam on the route. Only unusual traffic jams will be interesting for that driver. It should therefore be possible to distinguish between periodic and non-periodic traffic jams such that only the non-periodic are reported to drivers.

In order to detect only non-periodic traffic jams the daily traffic on the road must be known such that if the normal condition is congestion a traffic jam should only be detected if there is more congestion than usually. Information about the daily traffic can be obtained by storing incoming observations in a data warehouse. From this data warehouse the traffic for different roads and different periods of time can easily be selected. A brief overview of data warehouse theory can be found in the following chapter.

2.5 Traffic Jam Service Scenario

The traffic jam detection service developed here will focus on the detection of present traffic jams and it will aim at providing traffic jam information to drivers.

The ideas for detecting traffic jams are based on certain requirements. It should be possible to determine the position and speed of cars and the cars should be distinguishable. Furthermore, a large number of cars should be considered. It is not necessary to have information about all cars, only enough to give a representative view of the traffic jam situation.

An example of the techniques behind the traffic jam detection service is based on both GPS receivers and mobile phones. The example is illustrated in figure 2.2. Satellites send out signals as described in appendix B and from these signals a GPS receiver can determine its position. If a driver has both a GPS receiver and a mobile phone in the car and if these are connected the position determined by the GPS can be sent by the mobile phone. The position is transmitted via antennas to a stationary location where it is used for detecting traffic jams. If a traffic jam is detected a notice is sent to the driver's mobile phone.

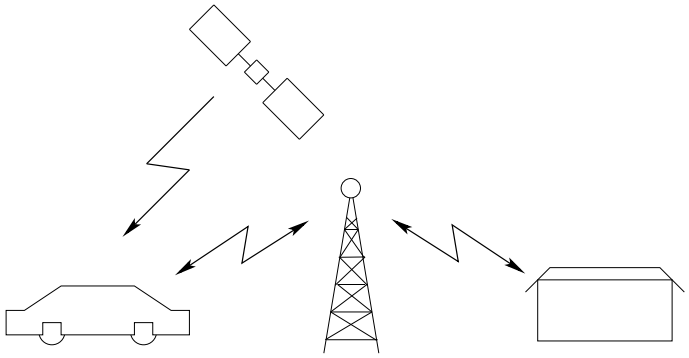


Figure 2.2: Data flow of the traffic jam detection service

First of all, it is necessary that drivers agree to report their positions because these positions are fundamental for the traffic jam detection. A way of convincing drivers to report this information is to give them something in return, for instance free or cheap traffic jam information. This could especially be necessary in the starting phase of the detection service, because in the beginning there is not a sufficient number of drivers that have signed up for the service to supply sufficient data for traffic jam detection. Furthermore, reporting their positions is a necessary condition for obtaining traffic jam information. If drivers do not report their position, the traffic jam detection service can not send relevant traffic jam information to the driver, because it cannot determine which segment the car is on.

Drivers that have agreed to report information have their position and speed recorded all the time when driving around. When a driver has reached the desired destination the information from that car is no longer interesting and should not be reported. When turning off the engine the reporting process could automatically be turned off as well.

The position, speed and id is reported from the cars very often, for instance every second. At the central place where the information about the cars is received a traffic jam test is run continuously or very often based on the data from the cars. If it is determined that there is a traffic jam, information can be sent to the radio or to people that have signed up for a traffic service, for instance as an SMS. Drivers that receive the traffic jam information can choose another route and avoid getting stuck.

People that receive traffic jam information could pay to obtain this information. This could for instance be the individual drivers that have signed up for a traffic jam detection service or it could be radio stations providing this as a service for their listeners.

2.6 Overall Design Goals

The above discussion has considered different topics that should be taken into account when designing a traffic jam detection service. From this discussion the requirements for the design can be stated. The overall goal of the design is to obtain a traffic jam detection service that is based on data warehouse design and that can detect traffic jams recently occurred. The detailed design requirements are listed below.

- The number of cars providing input data for the traffic jam service must be sufficiently high for representing the traffic.
- The input data must contain speed, identity and position in terms of the segment the car is on.
- The technique used must be precise enough to determine the segment of the observation.
- The road information must contain road code, segment id and direction.
- The input data must be reported frequently, at least every few seconds.
- It should be possible to detect both periodic and non-periodic traffic jams.
- It should be possible to detect only non-periodic traffic jams by using a data warehouse.

- A traffic jam should be reported soon enough for the drivers to choose another route.

These requirements are necessary conditions for a traffic jam detection service to provide the basic functionality. Only the last three items are requirements for the actual design. The remaining items are requirements that the design assumes fulfilled.

3 Data Warehouse Theory

As described in the introduction, the purpose of the project documented in this report is to design a traffic jam detection service based on historical information from a data warehouse. This chapter gives a brief introduction to the theory of data warehousing. The first section contains a general description of data warehouses. This is followed by two sections each describing a specific schema of a data warehouse. Finally a section describing related work in the area is presented.

3.1 General Data Warehouse Concepts

A data warehouse is a database used for OLAP. In an OLAP database the focus is on analyzing data, that is, to extract information to be used for decision support. A data warehouse can contain large amounts of data from several source databases and it is updated from these sources periodically, for instance once a day, or once a month depending on the purpose of the data warehouse. Therefore, a data warehouse need not be consistent with the On-Line Transaction Processing (OLTP) source databases. However, when analyzing data from several years to find overall trends information from a single day has hardly any influence.

A data warehouse is constituted by two elements, *facts* and *dimensions*. A fact contains the data to be analyzed, and the dimensions give a view of the data stored in the fact. A fact consists of *measures*, which are values describing the fact. An example could be a sales fact for a supermarket chain. The fact could contain the measures Revenue and Profit. The dimensions could be Customer and Product.

As an illustration, the example in figure 3.1 uses a spreadsheet metaphor, where the vertical and horizontal directions show the dimensions and their members. The Product dimension has three members Pasta, Milk, and Cola and the Customer dimension has members Jan and Jens. Each cell in the figure represents a sale. The two measures in a cell show the revenue (R) and profit (P) of the sale measured in dollars.

Facts are divided into three types according to the properties of the fact. The fact types are *event fact*, *snapshot fact* and *cumulative snapshot fact* [Pedersen and Jensen, 2001]. Event facts represent events in the real world. An example of an event fact is the sale fact in figure 3.1.

The profit and revenue of each cell represent a sale of a certain product to a certain customer. The snapshot fact represents the state of measured properties at a given point in time, for instance an inventory. The measured value represents the inventory at a given point in time, e.g. the number of colas in stock. The cumulative snapshot fact is the cumulative value of a snapshot fact. An example could be a Total Sales fact which holds the sum of all sales up to a given time.

Customer

Product	Jan		Jens		Total	
Pasta	R: 100 $	P: 30 $	R: 103 $	P: 37 $	R: 203 $	P: 67 $
Milk	R: 42 $	P: 8 $	R: 127 $	P: 46 $	R: 169 $	P: 54 $
Cola	R: 2001 $	P: 903 $	R: 242 $	P: 103 $	R: 2243 $	P: 1006 $
Total	R: 2143 $	P: 941 $	R: 472 $	P: 186 $	R: 2615 $	P: 1127 $

Figure 3.1: Spreadsheet example of a data warehouse

In some dimensions the values of the members can change over time. In a customer dimension, for instance, the age of the customers changes. Such dimensions are called *slowly changing dimensions* and can be handled in different ways. In [Kimball *et al.*, 1998, p. 180], three solutions are suggested. The simplest solution is to overwrite the old value with the new one, but this will erase the historical information of the dimension. Another solution is inserting a new record with the new values into the dimension table. This solution requires all records to have a time period associated, stating when the record is valid. The problem could also be handled by adding an old-value attribute, to which the old value is transferred when updating the attribute.

The dimensions have *hierarchies* that group facts at different levels. The hierarchy of the product dimension could be as illustrated in figure 3.2. The product level is the lowest and contains an entry for each product in the supermarket. A product, e.g. Cola, belongs to the product group Soda, which again belongs to the product category Beverages. The All level is the highest and contains only one entry. Every dimension has an All level which is the top level of the hierarchy [Thomsen *et al.*, 1999, p. 9].

The measures of a fact are typically numeric values that can be aggregated. Aggregation is used to calculate the value of the measures at any level of a hierarchy. Aggregation functions are sum, average, min, max and count. The revenue and profit measures could be aggregated by the sum function. Therefore, the aggregated sale fact at the product category level would be the sum of profits and revenues for products belonging to that category.

Measures can be additive, semi-additive or non-additive according to whether they can be added over all dimensions, some dimensions, or no dimensions, respectively. As an example, an inventory measure can be considered. This measure cannot be added over time and is therefore semi-additive. However, the remaining aggregation functions can still be applied, for instance, to calculate average inventory per week.

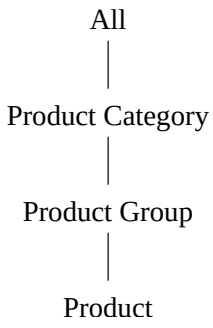


Figure 3.2: Hierarchy for a product dimension

The structure in a data warehouse is very different from an OLTP database where the focus is on the individual transaction e.g. money transfer in a bank. In an OLTP database it is important to avoid redundancy. This is achieved by normalizing the schema but normalization can lead to long execution times for queries involving large amounts of data from different relations.

An important goal in the design of a data warehouse is to achieve a low query execution-time. Therefore, unlike the OLTP database, a data warehouse permits redundancy in order to reduce query execution-time. In order to obtain fast answers to queries a technique called *pre-aggregation* is also used. The principle is to pre-calculate and store aggregate values at higher levels of the hierarchy. At run-time

these values can be accessed directly or used in further calculations thereby reducing execution-time. An example of pre-aggregation is illustrated in figure 3.1 on page 28 where the total revenue and profit per product is pre-calculated and listed at the end of each row.

Two main approaches to pre-aggregation are *full pre-aggregation* and *practical pre-aggregation* [Pedersen *et al.*, 1999, p. 1]. Full pre-aggregation means pre-aggregating the aggregate value of the fact at every combination of levels in every dimension of the facttype. However, the storage space needed to store the aggregated results increases rapidly with the number of dimensions and levels, and therefore, this strategy becomes infeasible. This phenomenon is called *data explosion* [Pedersen *et al.*, 1999]. Practical pre-aggregation, on the other hand, selects a subset of the aggregation levels. This increases the performance by decreasing response time without taking up too much storage.

Practical pre-aggregation uses the pre-aggregated values at one level to pre-aggregate the values at the levels above. For this approach to give the correct result, data of a dimension used for pre-aggregation must be *summarizable*. Summarizability is achieved if the hierarchy of the dimension members is *strict*, *onto*, and *covering* [Pedersen *et al.*, 1999]. If these requirements are not satisfied, aggregated values based on lower level aggregates can be incorrect because data at the lowest level are either counted more than once or not at all.

The following two sections each contain a description of a data warehouse schema. First, the multidimensional schema is described, and then the star schema which is an adaption of the multidimensional model to the relational schema.

3.2 Multidimensional Design

The multidimensional model, described in [Kimball *et al.*, 1998, p. 27], organizes each facttype in a cube. A cube is constructed of the dimensions of the facttype such that each combination of dimension values creates a cell where the measures of a fact can be stored. An example is the member Cola of the product dimension and the customer Jan in figure 3.1 on page 28. The two measures representing the sale of cola to Jan would be stored in the corresponding cell. The main difference between the spreadsheet example and a multidimensional cube is that the cube can have any number of dimensions. An example of a cube can be seen in figure 3.3 on the next page, where an extra time dimension has been added. In cases where no fact exists for a combination of dimension values, the cube has an empty cell [Kimball, 1996, p. 199].

In the cube the hierarchies of dimensions are modeled by having different granularities of the cells according to the level in the hierarchy. When aggregating measures to

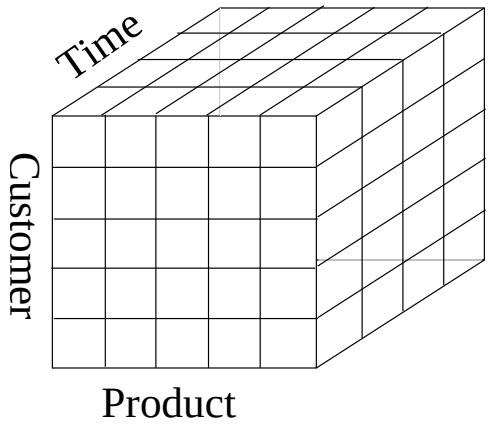


Figure 3.3: A cube with three dimensions

a higher level of a dimension the number of cells in that dimension will be reduced. For instance, as shown in part a) of figure 3.4 on the following page, the values at the lowest granularity of the product dimension, cola, milk, juice, pasta, and rice, can be aggregated to the product category level, beverages and food, resulting in the cube in part b) of figure 3.4 on the next page with only two cells in the product dimension.

In the theory of multidimensional models viewing the data at a higher level, such as part b) instead of part a) in figure 3.4 on the following page, is called rolling up. The inverse process is called drilling down.

Selecting a subset of the dimension values in the cube is called slicing. The result is the slice from the cube containing the specified value and the rest is cut away. An example could be selecting the month of May in the time dimension in part a) of figure 3.5 on page 33. As shown in part b), the result is a slice of the cube only containing measures from May.

3.3 Star Schema Design

The star schema is an alternative to the multidimensional design and is based on the relational database design. In figure 3.6 on the following page the sale example from figure 3.1 on page 28 is designed as a star schema with an additional time dimension. In a star schema the facts and dimensions are stored in tables, and there is typically a one-to-many relationship between the dimensions and the fact. By selecting a subset of the dimension tuples and joining with the fact table the star schema allows slicing in the dimensions just like the multidimensional design.

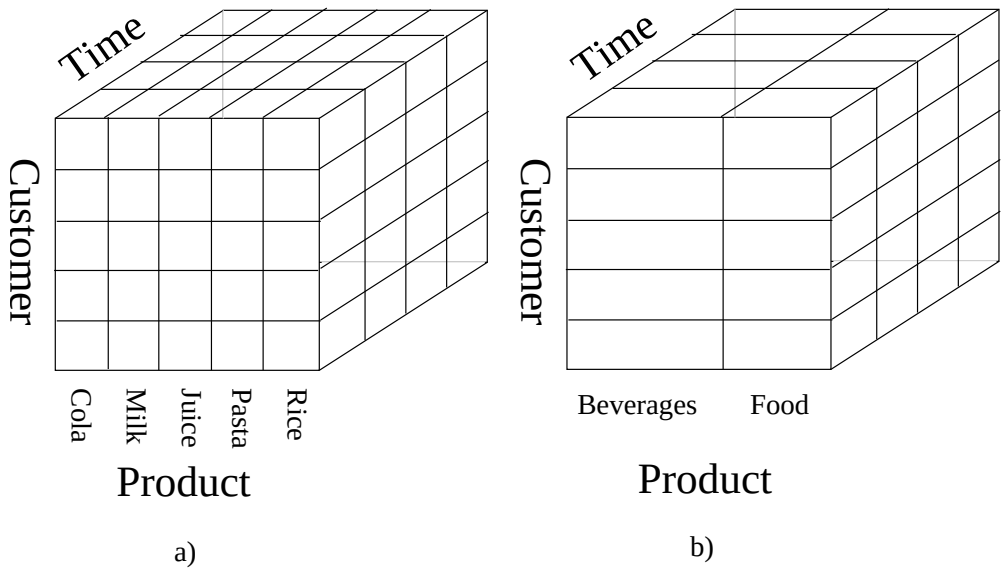


Figure 3.4: A cube before (a) and after (b) aggregation on the product dimension

The advantage of the star schema design is that it can be implemented in conventional relational databases, whereas the multidimensional designs are implemented in multidimensional OLAP systems like Microsoft SQL Server with Analysis Services.

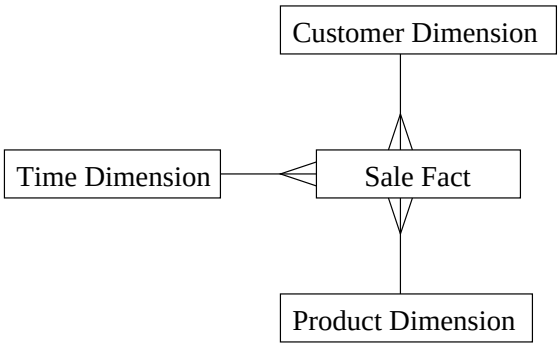


Figure 3.6: Star schema for the sale fact

The disadvantage of using a star schema is that it does not support the hierarchy structure of the multidimensional design. In order to achieve the advantage of pre-aggregation, the Oracle RDBMS has a functionality called *materialized view*. A materialized view is a table storing the result of an SQL-query. A materialized view is created for the the chosen aggregation level. The Oracle query optimizer can then rewrite a query to utilize the pre-aggregated values in the materialized view.

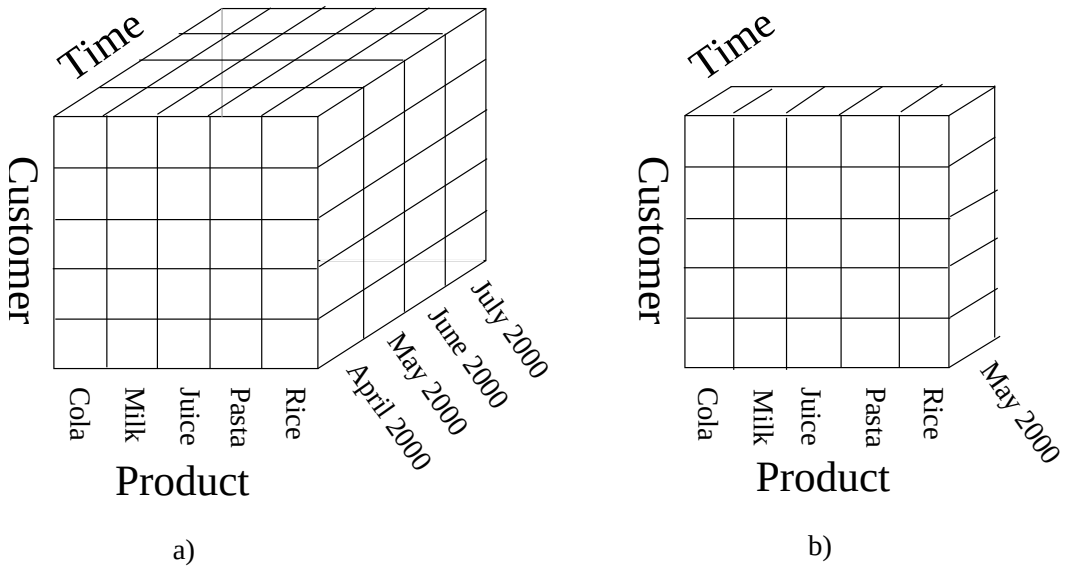


Figure 3.5: A cube before (a) and after (b) slicing on the time dimension

3.4 Related Work

OLAP systems are finding widespread use in the area of continuously moving objects and location-based services. An introduction to temporal data management can be found in [Jensen and Snodgrass, 1999], and [Papadias *et al.*, 2001a] considers the star schema and focuses on the spatial dimensions. The paper presents a grouping hierarchy based on the spatial index at the finest spatial granularity and it presents efficient methods to process arbitrary aggregations. The same applies for [Papadias *et al.*, 2001b] which describes a framework for supporting OLAP operations over spatio-temporal data and presents data structures integrating spatio-temporal indexing with pre-aggregation.

From data found in real-world applications, [Pedersen *et al.*, 2001] identifies some important modeling requirements for multidimensional data models. A thorough analysis of existing multidimensional data models reveals shortcomings in meeting these requirements and therefore lacks the ability to represent more complex data. The paper defines an extended multidimensional data model and algebraic query language that address these requirements where topics such as change, time, imprecision and data with varying granularity are handled.

In [Guting *et al.*, 2000] the goal is to provide a DBMS data model and query language capable of handling time-dependent geometries such as moving points. The paper presents an abstract data type extension to a DBMS data model and a query language where auxiliary data types are included.

Moving objects will have continuously changing positions but continuous updating and storing of such objects is not a possible solution. The positions will instead be updated at discrete time intervals. The paper [Pitoura and Samaras, 2001] contains a survey of various approaches to the problem of storing, querying and updating the location of objects in mobile computing. The article [Pfoser and Jensen, 1999] reports on the research in the representation of the positions of moving point objects such as interpolation of positions inbetween samples and the position uncertainty.

The problem of updating and querying databases representing information about moving objects is addressed in [Wolfson *et al.*, 1999] especially including information about when to update the positions. An information cost model that captures uncertainty, deviation, and communication is proposed as a solution to the problem.

OLAP systems require fast answers to queries and therefore, indexing techniques are developed for minimizing query-response time. In [Guttman, 1984] the R-tree is described which is a dynamic index structure suited for data objects of non-zero size located in multidimensional spaces. The paper also includes algorithms for searching and updating the R-tree. The article [Saltenis *et al.*, 2000] proposes an R*-tree based indexing technique that supports efficient querying of current and projected future positions of continuously moving objects in one, two- and three-dimensional space. The index accommodates a dynamic set, where objects may appear and disappear. In [Pfoser *et al.*, 2000] the focus is on the spatio-temporal sub-domain of trajectories of moving point objects. New types of queries are presented as well as algorithms for their processing. Furthermore, two trees for accessing the data are presented.

Recently, DSMSs (Data Stream Management Systems) have emerged. These systems can handle unlimited streams of data and perform queries on these. The article [Babcock *et al.*, 2002] considers fundamental models and issues in developing a general purpose DSMS.

This general theory provides the basis for designing data warehouse s. This will be used in the following four chapters where the traffic jam detection methods are described.

4 Max Speed Method

The first method for traffic jam detection uses the speed of vehicles as an indicator. The method indicates that there is a traffic jam on a road segment if the maximum speed of cars driving on the segment is low. First, a method for traffic jam detection will be developed and then, a database for supporting the method is designed. The method will be referred to as the Max Speed Method.

4.1 Description of Max Speed Method

In this section it will be described how the speed of cars is used to detect a traffic jam.

4.1.1 Discussion of Detection Method

The purpose of the Max Speed Method is to determine whether there currently is a traffic jam on a road segment. In order to do this the current speed of the cars located on the segment is registered. The principle of the Max Speed Method is that the slower the traffic is moving the greater the probability of traffic jamis. The maximum speed of all the observations of moving cars on a segment is used as an indication of traffic jam since the remaining cars must have a lower or equal speed. Therefore, if the maximum speed is low then all cars are driving slowly and this could be because of a traffic jam.

If there is in fact a traffic jam on a segment, the cars will all have approximately the same speed since a car must adapt its speed to the car in front of it. This is not the case while the traffic jam is developing. In this phase the cars joining the traffic jam

from behind will have a higher speed until they catch up with the traffic jam. When the traffic jam occupies the stretch from the beginning of the segment to the origin of the traffic jam, all cars in the traffic jam have the same speed. Therefore, the Max Speed Method will not be able to claim there is a traffic jam on a segment unless the stretch is affected. Furthermore, if cars in the front part of the traffic jam are getting out of the traffic jam they might be able to accelerate and reach high speeds. If they have not left the segment at this point in time the Max Speed Method will not detect the traffic jam. This is why the Max Speed Method should be associated with road segments and not roads as described in section 2.3. Segments are shorter and can therefore provide information faster.

The Max Speed Method might in some cases make wrong detections such that a traffic jam is detected even though there is no traffic jam on the segment. First of all, if there is little traffic on a road, a single car can have great impact on the result of this method. If only one car is driving on a road segment, the speed of this car will be the maximum speed of the segment. If that car is driving slowly the Max Speed Method would falsely detect a traffic jam on that segment. In order to avoid this situation both the maximum speed and the number of cars driving on the segment should be taken into account. If this is done, the case just described would not result in a traffic jam detection, even though the maximum speed is low. That is because at the same time as the maximum speed is low the number of cars on the segment is low which means that the traffic is too sparse for the cars to affect each others speed and therefore, it is not a traffic jam but a small number of slow drivers.

Another situation where the Max Speed Method might fail is crossroads or crossings of other sorts where even though a lot of cars are waiting in line it does not mean there is a traffic jam. The cars are waiting for other traffic to pass before they can continue. The Max Speed Method will in this case detect a traffic jam. However, the number of cars might remedy the situation. There might not be enough cars to set off the traffic jam detection. A discussion of how to solve this problem can be found in the following section.

It might also be a problem for the Max Speed Method to consider parking lots, for instance at big shopping malls, where there might be many cars. Most of them will be parked with their engine turned off and they will therefore not report any positions. But cars that are not parked will be driving slowly and therefore, a traffic jam could easily, but incorrectly, be detected. Furthermore, the parking lot might be close to a road and it might not be possible to determine the position of the cars precisely. Therefore, some of the cars in the parking lot might be recorded as being on the road and a traffic jam could therefore incorrectly be detected on the road. Another problem is that parking lots are not assigned road codes and they might therefore not be considered as a possible position when positions of cars are assigned road segment codes. This would cause all observations from cars driving on the parking lot to be mapped onto the nearest road and a traffic jam might be detected on the

road. A possible solution to this problem is to manually add segment codes to all parking lots.

Another thing to consider is that a traffic jam can be in one direction but not in the other. If there is a traffic jam in only one direction the Max Speed Method might still detect high speeds in the other direction. If a segment is not divided into two directions the Max Speed Method will fail to provide the desired information. In order for the Max Speed Method to provide traffic jam information it must be able to distinguish between the two directions of a road.

4.1.2 Calculation of Maximum Speed

This section will describe several methods of calculating the present maximum speed and discuss advantages and disadvantages of them.

The maximum speed for a segment is the maximum speed of all the observations of cars driving in the same direction on that segment. Therefore, the present maximum speed could be calculated by taking the maximum of all reported observations at a certain instant in time. The disadvantage of this approach is the reliability of the value as a description of the general behavior of cars on the road segment. That is, in the instance of the calculated maximum the cars might be driving slower than the general situation. The cars might for instance have stopped for a red light.

Instead, the maximum speed could be calculated for a time interval to obtain a better description of the general value. The time interval could for instance be 30 seconds or five minutes. For the remainder of this thesis, one minute will be used as the length of the intervals. It should be long enough that no traffic lights can remain red for the entire interval and it should be short enough for people to be informed in time. If the maximum speed is low in the entire time interval, the traffic has been moving slowly during that interval. So if only the serious traffic jams should be detected, i.e. those that last a long time and occupy a lot of space, then the time interval could simply be increased. An example could be only to detect traffic jams which last longer than ten minutes since those that last shorter dissolve too quickly to be of general interest. In such a case the time interval could be set to ten minutes. The disadvantage of prolonging the time interval is that the detection can only occur at the end of the interval. This means that cars heading in the direction of the traffic jam might reach the traffic jam before being warned about it. A solution to this problem is to choose intervals of one minute and then report traffic jams only if a traffic jam is detected several intervals after each other.

The Max Speed Method for detecting traffic jams will use intervals of one minute and it will detect traffic jams for each segment and direction individually. The method could be used in two ways. It could calculate the maximum speed for the last 60 seconds each time a request to determine traffic jam is received from a driver. It

could also calculate the maximum speed periodically every minute and use the stored value of the last calculated period each time a request is received. Calculating the maximum speed on request has the advantage of always giving the maximum value for the last minute whereas the periodic calculation method in the worst case would give the maximum value calculated 59 seconds ago for a period of one minute. The request calculation method, however, has the disadvantage that it can introduce an extra load on the request processing, since the maximum speed has to be calculated each time a request is received. If the maximum speed is calculated on request and if there are many requests the calculation might be performed many times within the same minute. Therefore, when using the periodic calculation more requests can be processed simultaneously.

Calculating the maximum speed for a minute might not be enough to determine the general behavior of cars on a segment. On the other hand, calculating the maximum speed for a longer interval might not be desirable because it postpones the result. By using periodic calculations, however, it is possible to store the last five or six results. If the values are the same in all the stored results, then the likelihood of it being representative for the general behavior is large. Comparing stored maximum speed values is not an advantage when calculating on request, since the requests are asynchronous.

In order to reduce the amount of storage in memory needed to store the necessary traffic information, the periodic calculation method could simply store the current maximum speed up till now for the current interval for each segment. This means that each time an observation is received the speed of the observation is stored only if it is larger than the current maximum speed. Therefore, it is not necessary to store the actual observations.

4.1.3 Detection by Max Speed Method

As previously described there are two kinds of traffic jams, periodic and non-periodic traffic jams. If no distinction between these two kinds is made, traffic jams can be detected by comparing the observed values with constant values. The maximum speed of the observations from a segment and the number of cars located on the segment are compared to threshold values in order to determine whether there is a traffic jam on the road segment. The threshold values for the speed could for instance be determined by the legal speed limit on the road. The threshold for the number of cars could be determined from the capacity of the road. The capacity is a value that states how many cars can be on a segment without affecting each others speed considerably. A road classification can be obtained from DAV. This road classification is independent of the length of the road and from this information an approximate capacity can be estimated.

First of all, if a traffic jam is detected on a segment there must be more than only a few cars on that segment. To ensure this, the number of cars on the segment can be compared to the capacity as in equation (4.1). The Min-cars-Quantifier could for instance be set to 0.2 which means that if the number of cars is smaller than or equal to 20% of the maximum capacity no traffic jam is detected for that segment. This equation is thus a necessary but not sufficient condition for a traffic jam to be present.

$$\text{Number of Cars} > \text{Min-cars-Quantifier} \cdot \text{Capacity} \tag{4.1}$$

If there are sufficiently many cars on the road a traffic jam is detected if the speed is low. The recorded speed can be compared to the speed limit on the segment as in equation (4.2) and if the equation is true, there might be a traffic jam. The Speed-limit-Quantifier determines how low the recorded speed should be compared to the limit before there is a traffic jam. It could for instance be set to 0.75.

$$\text{Maximum Speed} < \text{Speed-limit-Quantifier} \cdot \text{Speed Limit} \tag{4.2}$$

Thus, only if both equations (4.1) and (4.2) are true a traffic jam is detected.

If only non-periodic traffic jams should be detected the incoming values from the cars can be compared to historical recordings of the traffic situation. As an example the maximum speed could be compared to a historically recorded average maximum speed of a road segment.

The advantage of using historical data is, that if ordinarily the average maximum speed of the road is low then the situation is not classified as a traffic jam as described in section 2.6. Using historical data also means that if a road often has traffic jams, these are not detected since this is the normal situation and therefore, the average maximum speed of that road is low. If the legal speed limit of the road is used, however, every traffic jam would be detected since they are all extraordinary situations compared to the legal speed limit. Assuming that the speed is recorded periodically the average maximum speed for one direction of a road segment based on observations from the past is calculated as in equation (4.3). The value of Number of Intervals is the number of intervals that passed for which the maximum speed has been registered.

$$\text{Average Maximum Speed} = \frac{\sum \text{Maximum Speed}}{\text{Number of Intervals}} \tag{4.3}$$

The average maximum speed of the cars constitute the normal situation of the road segment. However, only situations where the traffic conditions deviate from the normal situation are interesting when detecting traffic jams. That is, the traffic on the road segment should only be identified as a traffic jam if the maximum speed of the cars on the road segment is lower than the average maximum speed on the segment. It is not sufficient that the maximum speed is only slightly lower than the average maximum speed because there will always be variations in the traffic. Sometimes, the speed will be lower than usual even though there is no traffic jam, for instance because of intersections or people driving slowly. When traffic jams are detected the speed should be even lower than in these cases. Therefore, the value Speed-Quantifier is introduced. It could for instance be set to 0.8. This means that if the recorded maximum speed is only 80% compared to the usual maximum speed, a traffic jam will be detected. Also in this case a traffic jam should only be detected is there are more than just a few cars on the road. Therefore, both equations (4.1) and (4.4) should be true before a traffic jam is detected.

$$\text{Maximum Speed} < \text{Speed-Quantifier} \cdot \text{Average Maximum Speed} \tag{4.4}$$

4.2 Design of Max Speed Method

In this section the design of the Max Speed Method is described. Based on the description in the previous section and in chapter 2 the demands for the design will now be described.

- **Present:** The method should contain information about the current traffic situation and it must therefore be able to detect traffic jams occurring in real-time. Hence, the methods should be able to consider real-time data.

- **Past:** The design must contain historical information about traffic such that the detection can be based on the past. In this way the normal deviations in the traffic patterns can be taken into account.

- **Areas:** A road can be long and there might be a traffic jam in one end and not the other. The Max Speed Method cannot detect such traffic jams. By using segments instead of roads more traffic jams can be detected. Another solution is to consider small areas such as intersections involving more than one road. However, there might be a traffic jam on one of the roads but not on the other and in such a case the Max Speed Method would not detect the traffic jam. Therefore, using segments is a better solution.

- **Direction:** A traffic jam can occur in only one direction of a road. In order to detect such traffic jams each part of a road should be divided into two directions.
- **Time of Day:** The historical data should not be calculated for an entire day because there might be differences in the traffic over the day and these differences will be blurred if all values throughout day are added.
- **Day Type:** The traffic can be different depending on whether it is weekday or weekend. Therefore, the values of the Max Speed Method should be separated depending on which part of the week they belong. There can also be differences depending on the individual day. This indicates that a further partition into the seven days of week can be advantageous.

The demands described above are the requirements for the data warehouse which will be designed in this section. This, however, does not mean that the entire method will be implemented as a data warehouse. The calculation of the present maximum speed and number of cars does not have to be stored in a fact since they will only contain one value at a time. There is no need for aggregating which is the main feature of multidimensional databases. Another possibility is to store it in an ordinary database. The benefit of this approach is not as big as the benefit of storing it in the memory, since storing it in a database means slow access because it is stored on disk. Furthermore, the relational power of the database is not utilized because only two values per segment need to be stored at a time. Storing the maximum speed and number of cars in memory on the other hand has the advantage of fast access and traffic jam queries can therefore be answered quickly. Also, since there are only two values per segment, there is no need for sorting or searching.

The historical data on the other hand is well suited for a data warehouse. Placing the average maximum speed and the average number of cars in a data warehouse enables the possibility of slicing on the time dimension and rolling up and down. For instance, determining the average number of cars on a segment in weekends is a simple task if there is a day type dimension. Therefore, a data warehouse with appropriate dimensions and measures will be designed.

4.2.1 Road Segment Dimension

Traffic jams will be detected for each road segment individually because different conditions apply for different segments. The Road Segment dimension could have two levels, an All and a Segment level. However, to enable the possibility of statistical analysis on the information in the data warehouse several additional levels are added. The first level above the Segment level is the Road level. This level could be used to determine if a road in general is overloaded. The next level is Postal Code, then

Town, Kommune, Country and finally the All level. Each segment belongs to exactly one road. A road can pass through several postal districts and towns. Therefore, the road level cannot be child of the postal code level or road level. A town can have several postal codes and a postal code can cover more than one town. Therefore, these can also not be in a parent-child relationship. A road can pass through several kommunes and even countries. Therefore, the parent to the road level should be the All level.

The hierarchy is illustrated in figure 4.1.

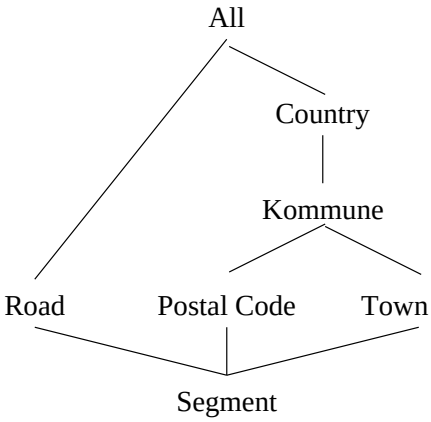


Figure 4.1: The hierarchy of the Road Segment dimension

The hierarchy is strict because each child element has only one parent and it is onto because each element has at least one child except for the segments at the lowest level. The hierarchy is not covering because a segment need not belong to a town. This means that some segments skip a level and map straight up to the Kommune level. If this part of the hierarchy is used for pre-aggregation an extra attribute called countryside can be added to the Town level such that all segments that are not in a town can map up to this value. By adding this attribute pre-aggregated values at the Town level can be used for further aggregation.

The two remaining parts of the hierarchy are covering,

4.2.2 Day Dimension

In the requirements it was stated that the traffic varies depending on the day of week. Therefore, the data warehouse must have a Day dimension, which enables slicing on weekend and weekday values.

The Day dimension could be realized as a dimension with two levels, an All and

a Day Type level where the latter contains the values Weekend and Weekday. By adding another level containing the name of the day, variations between days of the week are taken into account when calculating the measures. The consequence is that the fact table will grow. Instead of having a fact for weekends and weekdays there will be a fact for every day. This means that in the worst case there will be $\frac{7}{2}$ times as much data in the fact table. This is because the dimensions are independent so adding or deleting levels in the hierarchy of a dimension only affects the fact table and not the other dimensions.

The Day dimension could furthermore be expanded with a Date level. The increase of the fact table would depend on the period registered in the Day dimension. If the Day dimension contains dates from one year it would in the worst case increase the amount of facts by 365 times. If no observations occur for a given hour, no fact will be stored in the fact table for that hour. Therefore, the average case scenario will most likely require less storage space. This is because the probability of observations not occurring during a single hour is larger than the probability of an observation not occurring during the same hour on the same day of week.

Adding a date level to the dimension eases the addition of extra information which is dependent on the date. For instance, information about the weather like when it rains or snows or information about events such as football matches can be added to the Day dimension when it occurs. This is possible because the changes only occur in the Day dimension. They would not affect the relationship between the dimension and the facts.

Another advantage of adding a date level is that old information can be left out so that it does not affect the aggregated values. An example where this could be beneficial is when the traffic conditions change, for instance because of road construction work. In this case, the values from before the repair started should not be used when retrieving values from the data warehouse. In order to adjust to the continual changes in traffic patterns, values older than for instance a month should perhaps not be used. This is possible by slicing on the Day dimension. For the same reason, the Month and Year levels are added. They enable the user to select a specific period if certain properties depending on the period is to be investigated.

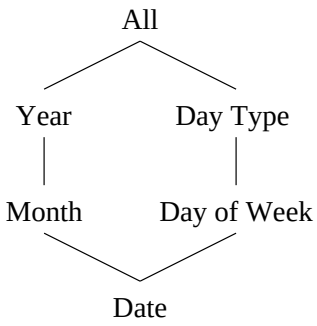


Figure 4.2: The hierarchy of the Day dimension

Both paths of the hierarchy are strict, onto and covering.

4.2.3 Time of Day Dimension

In order to comply with the requirement of separating values that occur on different times of a day, the Time Of Day dimension is added. The Time of Day dimension divides a day into intervals. The first interval is the morning traffic where the morning rush hour occurs, the next interval is day-time, the third is the evening rush hour, then late-evening, and finally night time.

As for the day type dimension if the five intervals above are the lowest granularity it would be difficult to change the time space of each of these periods since it would require changing the relations between the dimension and the fact. Instead, the dimension could be generalized by adding an hour level as the finest granularity of the Day Interval dimension as shown in figure 4.3.

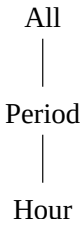


Figure 4.3: The hierarchy of the Time of Day dimension

The hierarchy is strict, onto and covering.

Adding the Hour level increases the amount of facts by $\frac{24}{5}$ times in the worst case.

The requirements stated in the beginning of this section consider two measures of time, Day and Time of Day. In a data warehouse the Day and Time of Day dimensions as described above could be implemented as two separate dimensions or one dimension containing all information. The advantage of having two dimensions is that selecting only one of the values, e.g. traffic in the weekend, requires less computation if stored in two separate fact tables. Furthermore, the storage space needed for storing the two dimensions is smaller than for one large dimension, since the combined dimension should contain the Cartesian product of the two. The disadvantage of this approach is that it increases the size of each tuple in the fact table, because there must be two attributes in the fact table instead of one in order to refer to two dimensions. Individually, these two attributes will be smaller than the large attribute, however. Another disadvantage of having two dimensions instead of one is that when slicing on both dimensions and joining with the fact table two joins must be computed instead of one.

4.2.4 Speed History Data Warehouse

The historical data that is used for comparison with the incoming observations is stored in a data warehouse. The structure of the data warehouse is illustrated in the star schema in figure 4.4 on the following page. The fact table contains five measures. The Maximum Speed is the sum of all maximum speeds recorded for the given segment within the given hour. The sum is not interesting, however, only the average. Therefore, the measure Number of Intervals holds the number of values contained in the Maximum Speed measure such that the average can be computed. The Number of Observations is an additional measure containing the number of observations received within the given hour for the given segment. This value can be used as an indication of whether there is sufficient data for making decisions based on it. The two final measures are provided for statistical analysis only. The first contains the number of non-periodic traffic jams and the other contains the total number of traffic jams, that is, both periodic and non-periodic traffic jams, detected within the hour for each segment. The Speed History fact is a snapshot fact, because the measures contain snapshots of the traffic situations on road segments.

The measures listed in figure 4.4 on the next page are physical measures. They are used to calculate the logical measure Average Maximum Speed as described in section 4.1. All the physical measures are aggregated by the sum function. Assuming that the time interval during which the maximum speed is calculated is one minute, then the value of the Maximum Speed measure at the lowest granularity is the sum of the 60 maximum speeds calculated during one hour. The Number of Observations is the total number of observations that have been received within one hour. This measure is used to determine if the quantity of the historical data is sufficient. If not, more data could be included. The Number of Intervals measure would have the value 60 since the Maximum speed measure contains the value of 60 time intervals

of one minute. If aggregating over the Road Segment dimension the value would not be of any use. Therefore, the measure Number of Intervals could instead be stored in the Day and Time of Day dimensions with the value 60 at the lowest level of the Time of Day dimension and the value $24 \cdot 60 = 1440$ at the lowest level of the Day dimension. At higher levels the aggregated values would be stored and at any level the values could be used for calculations as if it were placed in the fact table. Storing this measure in the dimensions instead of in the fact table would reduce the storage space needed. It could, however, make query writing more tedious because the query writer has to remember the right granularity for the given query. The measures Maximum Speed and Number of Intervals are only additive over the temporal dimensions. They are non-additive over the Road Segment dimension. The remaining three measures are additive over all dimensions.

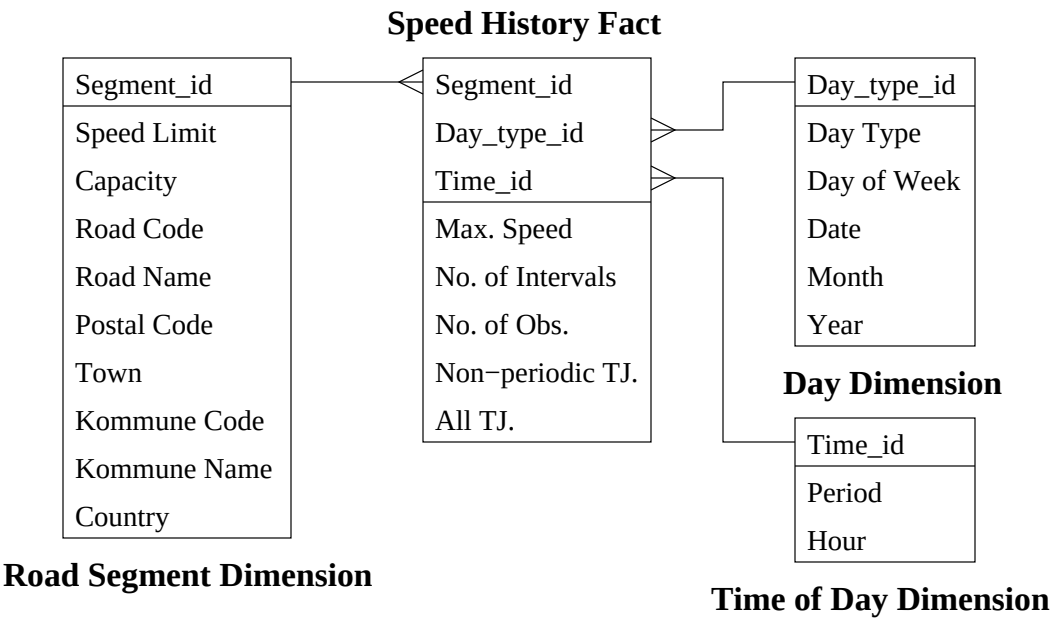


Figure 4.4: The star schema for the Speed History fact

The worst case size of the fact table depends on the extent of the geographical area covered by the traffic jam service, that is, the area which the road segments in the Road Segment dimension belongs to. This is because the larger the area the more roads and therefore the more segments. As an example, according to information from Aalborg Kommune obtained in fall 2001 the town Aalborg in northern Jylland in Denmark has 2.217 roads. Assuming each road on average is divided into 20 segments, the worst case size of the fact table for one day will be as shown in equation (4.5).

$$2.217 \text{ roads} \cdot 20 \text{ segments} \cdot 24 \text{ hours} = 1.064.160 \text{ tuples} \tag{4.5}$$

This is not a considerable size considering that it is for one town and for one day. On the other hand, there might be some roads where traffic jams are very unlikely to occur such as small roads in residential areas and remote country roads. Even if a traffic jam would occur in such a place only few cars would be affected because the road is not used by many cars. If the fact table becomes to large it could be determined that traffic jams will not be detected for these roads.

5 Traffic Flow Method

This chapter contains a description of another way to detect a traffic jam. The detection method is based on the number of cars on a segment and the traffic flow on that segment within a given time interval. The method will be referred to as the Traffic Flow Method.

5.1 Description of Traffic Flow Method

If there is a traffic jam on a segment, cars will move slowly or not at all. This will cause a decrease in the number of cars leaving the segment and perhaps also in the number of cars entering the segment. This decrease can be used as an indication of traffic jam. To determine such a decrease the continuous flow of cars can be measured. The term Traffic Flow will be used to describe this continuous flow.

The Traffic Flow of a segment is the number of cars leaving the segment within a certain time interval. The flow could be calculated by taking the number of different cars on the segment within, for instance, a minute and subtracting the number of cars on the segment at the end of that minute. This way, the number of cars that have left the segment within that minute is obtained and this will be the traffic flow for that minute. If there is a traffic jam on the segment the traffic flow will be low. The calculation of the Traffic Flow of a given segment is computed from equation (5.1).

$$\text{Traffic Flow} = \text{Number of Cars Within Minute} - \text{Final Number of Cars} \quad (5.1)$$

Sometimes, the flow could be low even though there is no traffic jam on the segment. This could for instance be the case if there are only few or perhaps even no cars on the segment. To ensure that no traffic jams will be detected in these cases the number of cars on the segment can be taken into account. Thus, to detect a traffic jam there must be both a low flow and a considerable number of cars.

Considering the number of cars alone would not be sufficient to detect a traffic jam. Even though there are many cars on a segment the flow might still be high and if the flow is high at least part of the traffic is moving and this is sufficient for the case not to be considered a traffic jam.

If a traffic jam suddenly occurs, the Traffic Flow Method will not be able to detect it immediately. If, for instance, an accident occurs and the segment is almost blocked, this will affect the flow. At first, right after the accident, cars that have already passed the place of the accident can continue unaffectedly. The number of cars leaving the segment might therefore not change immediately after the accident. The flow will not be reduced until all these cars have left the segment. This is because cars that were behind the point of the accident are not able to continue in the same way as cars in front of the accident. This will cause a reduction of the flow at the point of time where cars behind the accident were supposed to have left the segment. Thus, in this case shorter segments are able to provide faster information about traffic jams. The segments should not be too short either. Segments of only a few meters would be too short because it would be difficult to distinguish a traffic jam from normal deviations in the traffic. An appropriate size for a segment might be 200 to 500 meters.

As for the Max Speed Method, a traffic jam cannot be detected unless there is a certain amount of cars on the road and therefore, the same equation as for the Max Speed Method displayed again in equation (5.2), is used to determine if there is a sufficient number of cars on the segment for a traffic jam to be determined. The Min-cars-Quantifier need not have the same value for this method as for the Max Speed Method.

$$\text{Number of Cars} > \text{Min-cars-Quantifier} \cdot \text{Capacity} \quad (5.2)$$

In order to determine that there is a traffic jam, the traffic flow must be low. The Flow will also be low if there are only few cars on the segment because then only few cars will leave the segment. This makes it difficult to determine whether there is a traffic jam. It can only be certain that there is a traffic jam if there is a high

number of cars on the road at the same time as when the flow is low. Therefore, the Min-cars-Quantifier could be set to a higher value than in the Max Speed Method. It could for instance be set to 0.8 such that if there are less than 80% cars compared to the capacity no traffic jams will be detected.

There is no road information available about the flow on a road and it can therefore be difficult to determine if a given flow is high or low because there are no flow values to compare it with. A maximum flow might, however, be determined from the capacity and speed limit on the segment. Segments with high capacity and high speed limit can have a higher flow than segments with low capacity and low speed limit. From these values an approximate maximum flow can be computed for each segment. It can then be decided that traffic jams will only be detected if the flow drops below 50% of the maximum flow and only if there are more than 80% cars compared to the maximum capacity. The additional equation that must be true at the same time as (5.2) in order to detect a traffic jam is shown in equation (5.3). The Max-flow-Quantifier could for instance be set to 0.5.

$$\text{Traffic Flow} < \text{Max-flow-Quantifier} \cdot \text{Maximum Flow} \tag{5.3}$$

The detection of traffic jams can also be based on historical information in the same way as described in chapter 4. The traffic may differ throughout the day and throughout the week and this can be taken into account by the historical information. The traffic might also differ from road to road. Such differences are also be taken into consideration when using historical information. All this can be done by calculating different historical values depending on the segment and the time. The historical values of the traffic flow for each segment and for each time period is calculated by taking the sum of all traffic flows recorded within the last period, for instance within the last month, and dividing by the number of values summarized. The formula is given in (5.4).

$$\text{Average Traffic Flow} = \frac{\sum \text{Traffic Flow}}{\text{Number of Intervals}} \tag{5.4}$$

As described previously, there might not be a traffic jam even though the traffic flow is low. This is because the flow is automatically low if the number of cars is low. Therefore, the average traffic flow is not sufficient indication of a traffic jam. The number of cars must also be considered. By comparing the number of cars with the average number of cars, the situations where a low number of cars cause a low traffic flow are eliminated. The historical information about the number of cars is calculated in a similar way as shown in equation (5.5).

$$\text{Average Number of Cars} = \frac{\sum \text{Number of Cars}}{\text{Number of Intervals}} \tag{5.5}$$

In order to determine if there is a traffic flow on a given segment the historical traffic flow for that segment and for the current time is selected. Then the historical traffic flow is compared to the traffic flow recently recorded and if the latter is sufficiently low, there might be a traffic jam. However, considering the everyday traffic for a given period of the day there might be a varying number of cars and smaller changes in the flow, for instance caused by intersections. A traffic jam will cause a larger decrease in the flow than what happens due to normal variations in the traffic. In order to avoid that normal traffic variations are detected as traffic jams the historical information should not be directly compared to the actual flow. Instead, a quantifier called Flow-Quantifier is introduced. This quantifier is multiplied by the historical information in order to scale down the size of the historical value. The value of the quantifier must therefore be between zero and one, for instance 0.75. Furthermore, the value could depend on the segment and the time because the traffic might behave different on different segments and on different times of the day. A way of determining a good value for the quantifier is to adjust it according to information about when traffic jams have occurred. Once a value for the quantifier has been determined, a traffic jam will not be detected if the value of equation (5.6) is false.

$$\text{Traffic Flow} < \text{Flow-Quantifier} \cdot \text{Average Traffic Flow} \tag{5.6}$$

Due to the different loads and sizes of roads the number of cars that can be on a road without causing congestion will also differ from road to road. Therefore, to determine if there are enough cars on a road to cause a traffic jam historical information can be considered and a quantifier is introduced as for the traffic flow. The quantifier called Cars-Quantifier should be larger than 0 and it should be sufficiently large to detect only high numbers of cars. It could for instance be set to 0.9. The equation (5.7) is used to indicate a traffic jam.

$$\text{Number of Cars} > \text{Cars-Quantifier} \cdot \text{Average Number of Cars} \tag{5.7}$$

If both equations (5.2) (5.6) and (5.7) are all true at the same time the case is considered a traffic jam. However, if the cars have a red light, the flow can be low and a traffic jam will be detected even though the low flow is only temporary. As for the previous method, this can be avoided by not detecting a traffic jam unless all three expressions have been true for several consecutive intervals.

5.2 Design of Traffic Flow Method

In the previous section the Traffic Flow Method for detecting traffic jams was discussed. From this discussion and from the general description in chapter 2, certain requirements for the design have been made. These are the same as for the Max Speed Method and will therefore not be restated.

As for the Max Speed Method the application for detecting traffic jams is divided into two parts. One part handles the incoming observations from the cars and the other part maintains the historical information.

The incoming observations are kept in memory and periodically, for instance every minute, the traffic flow on each segment is computed from equation (5.1). The number of cars on each segment is similarly counted from the observations in memory and if equation (5.2) is false there is no traffic jam on the segment. Otherwise, the values of the computed traffic flow and number of cars are compared to historical information by equation (5.6) and equation (5.7) and if both equations are true, there might be a traffic jam on that segment.

When it has been determined for each segment whether there is a traffic jam, the observations from the cars are no longer needed. The values Traffic Flow and Number of Cars from the traffic jam detection, on the other hand, must be used to update the historical information. They cannot yet be deleted but they can be kept on disk instead of in memory. The historical information is updated periodically, for instance every hour or every day.

The historical information is contained in a data warehouse. In order to meet the requirements, the data warehouse must have certain dimensions. For the same reasons as in the previous chapter, the data warehouse can have the same dimensions and hierarchies as the data warehouse for the Max Speed Method.

The values of the historical information is maintained in a fact which has the six physical measures Traffic Flow, Number of Cars, Number of Intervals, Number of Observations, Non-periodic Traffic Jams and All Traffic Jams. The first measure holds the sum of all traffic flows on a given segment within a given hour and the Number of Cars measure holds the number of cars for that segment within that hour. The Number of Intervals measure holds the number of intervals over which the sum of traffic flow and number of cars is calculated. At the lowest granularity this number will be 60 assuming that values are calculated every minute and because there are 60 minutes in an hour. Number of Intervals is used to calculate the logical measures Average Traffic Flow and Average Number of Cars from the equations (5.4) and (5.5). The Number of Observations measure holds the number of observations recorded within an hour and the two final measures hold the number of traffic jams detected.

The fact and the related dimensions are illustrated in figure 5.1.

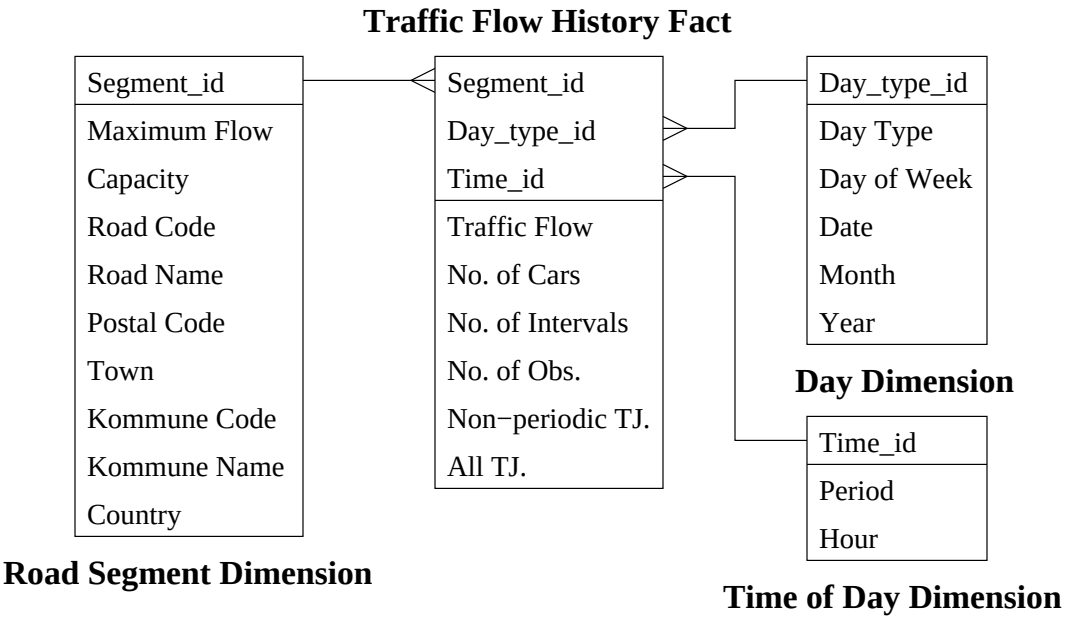


Figure 5.1: The data warehouse of the Traffic Flow Method

The fact is a snapshot fact because it is a snapshot of the traffic each minute. Each value is summed up to hours before being loaded into the data warehouse. The measures are additive over the Time of Day and Day dimensions. For the Road Segment dimension the situation is different. The flow of several consecutive segments cannot be added to produce the overall flow of the segments because cars that leave a segment might enter another segment in the area. The overall flow will therefore be lower than the sum of the individual flows. Therefore, the Traffic Flow is not additive on the Road Segment dimension in traditional sense but the values can still be used for comparison.

6 Occupation Time Method

This chapter contains the description and design of the third method for detecting traffic jams. The method considers the amount of time cars spend on a segment. This method is called Occupation Time Method.

6.1 Description of Occupation Time Method

As described in chapter 4 and chapter 5, the speed and flow of cars on a segment will be affected by a traffic jam because they will both decrease. A consequence of this is that if there is a traffic jam, cars will remain on the segment for a longer period of time than normally. A third method for detecting traffic jams could therefore be to measure the amount of time cars spend on a given segment. If several cars spend more time than usual on that segment it could be due to a traffic jam.

One way of using this method for detecting traffic jams is to assign to each segment a threshold. This threshold represents the number of cars that must have been on the segment for longer than usual in order to consider the case a traffic jam. For instance, consider a segment that is usually passed through in one minute. This segment could then be assigned two thresholds, the first equal to ten and the next equal to two. Then, when more than ten cars spend more than two minutes on the segment there is a traffic jam. This way of detecting traffic jams, however, does not resemble the two previous methods for detecting traffic jams.

Another way of using this method is to consider all cars currently on the segment

together with the time they have spent on the segment so far. If the cars on average have been on the segment for a long time it might be because they cannot leave the segment due to a traffic jam. The time that each of these cars has spent on the segment so far since they last entered the segment is summed and this sum is divided by the number of cars currently on the segment. This way, the average time spent on the segment at this given point in time is obtained. The total time that cars currently on the segment have spent on that segment since they last entered it is called Occupation Time. It is calculated as in equation (6.1).

$$\text{Occupation time} = \sum \text{Time car has spent on segment} \tag{6.1}$$

If there is a traffic jam new cars may still enter the segment as normally but only few or no cars will leave the segment. These cars have therefore been on the segment longer than normally and they will contribute to raising the Occupation time. It is not sufficient to consider the Occupation Time alone because if there are many cars on the segment the Occupation Time will be high even though there is no traffic jam. Therefore, the Occupation Time must be divided by the Number of Cars in order to obtain an average value as shown in equation (6.2).

$$\text{Current Occupation Time} = \frac{\text{Occupation Time}}{\text{Number of Cars}} \tag{6.2}$$

If the Current Occupation Time is high there might be a traffic jam. This way of detecting traffic jams will be used as the third method for traffic jam detection.

In order to determine the amount of time a car spends on a segment the identity of the car must be known. When a car enters a segment the time is recorded. At any point in time and for any car it can then be determined for how long that car has been on the segment that it is currently on.

Not all cars drive straight through a segment. Some cars might stop for a shorter period in the side of the road before moving on. This could be to drop off somebody or pick something up from a store. A car could also be parked for hours or it could turn to a side road in another direction. In any case the time that the car spends on the segment will either be longer or shorter than if it were passing straight through the segment. Therefore, it is not sufficient to consider only one car. The time spent on the segment must be longer than normally for several cars in the same period of time before a traffic jam is detected.

As for the previous methods there must be more than a few cars on the segment in order for a traffic jam to be detected. Therefore, the same equation as previously can be used to ensure a minimum number of cars. The formula is shown in equation (6.3) and it must be true before a traffic jam can be detected.

$$\text{Number of Cars} > \text{Min-cars-Quantifier} \cdot \text{Capacity} \tag{6.3}$$

In order to detect all traffic jams including periodic traffic jams, the current occupation time must be compared with a constant value describing the occupation time under optimal conditions. This value could be computed by first computing the time it would take a car to pass through the segment under optimal conditions. This is done from the length and speed limit on the segment. This time period is divided by two to obtain the average and this value is called Occupancy. The Occupancy is the time it takes to pass halfway through the segment. This corresponds to taking the average of the time cars that have just entered the segment and cars that are close to leaving the segment have spent on this segment. In average, this corresponds to all cars being halfway through the segment. A traffic jam can then be detected if the most recently recorded Current Occupation Time exceeds the Occupancy by a certain percentage, for instance 50%. The formula is shown in equation (6.4) and both this and equation (6.3) must be true before a traffic jam is detected. The quantifier Occupancy-Quantifier is used to scale the Occupancy.

$$\text{Current Occupation Time} > \text{Occupancy-Quantifier} \cdot \text{Occupancy} \tag{6.4}$$

Traffic jam detection can also be based on historical information. As for the previous methods, knowledge about the normal traffic can be obtained by maintaining historical information about the traffic. The historical data should provide information about the normal average time spent for a given segment and for a given period of time. It can be calculated by taking the sum of the occupation times and dividing by the number of cars as illustrated in (6.5).

$$\text{Average Occupation Time} = \frac{\sum \text{Occupation Time}}{\text{Number of Cars}} \tag{6.5}$$

The difference between this equation and equation (6.2) is that the latter is recorded for new values each minute, whereas equation (6.5) contains an average over a longer time, for instance several months. There will always be variations in the traffic and this will affect the average time that each car spends on a segment. Therefore, if the average deviates only slightly from the historical values it should not be considered a traffic jam. In order to ensure a sufficiently high deviation a quantifier called Occupation-Quantifier is introduced. The value of the quantifier should be above one in order to scale the historical average time spent on a segment. It could for instance be set to 1.5 such that a traffic jam is detected only if the recently recorded Occupation Time is 50% above the average.

If the Current Occupation Time contains the most recent value obtained from a segment a traffic jam will only be detected if the equation (6.6) is true for that segment.

$$\text{Current Occupation Time} > \text{Occupation-Quantifier} \cdot \text{Average Occupation Time} \tag{6.6}$$

A traffic jam is only detected when both (6.3) and (6.6) are true.

6.2 Design of Occupation Time Method

The above discussion and the discussion in chapter 2 have lead to some requirements for the design of the Occupation Time Method for traffic jam detection. These are similar to the requirements for the Max Speed Method and Traffic Flow Method except that this method has an additional requirement

- **Identity:** The detection method computes the time each car has spent on the segment. This can be done by knowing the identity of the cars.

The current traffic situation should be compared to the usual traffic situation. As for the two previous methods, the application can therefore be split into two parts, one that handles the current traffic situation and one that manages information about the past.

The current traffic situation is determined by observations from the cars. Cars that have signed up for the service report their position together with their id periodically, for instance every second. At other time intervals, for instance every minute, the occupation time and current number of cars for each segment is computed. These two values are compared with historical information and capacity as in equations (6.3) and (6.6). If both equations are true there might be a traffic jam.

The information about the past can be maintained in a data warehouse with the same dimensions and hierarchies as the previous methods.

The data warehouse contains five measures. The first measures, Occupation Time, is the sum of all occupation times computed within the given hour and similarly for the Number of Cars and Number of Observations. The two final measures have the same meaning as in the fact tables for the two previous methods. The star schema of the data warehouse is illustrated in figure 6.1 on the next page.

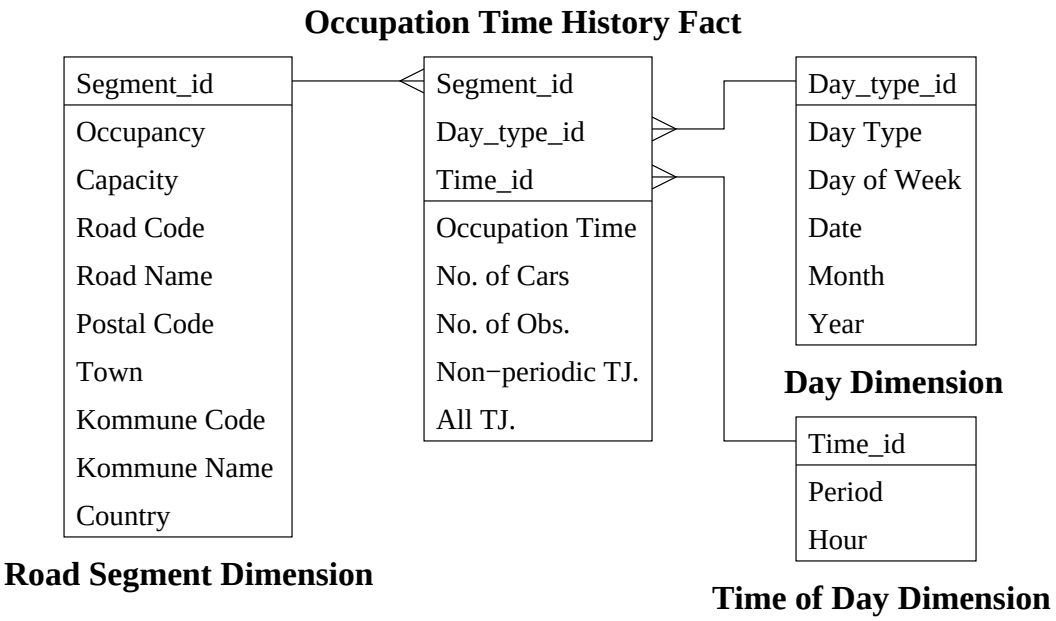


Figure 6.1: The data warehouse of the Occupation Time Method

The fact is a snapshot fact because it holds a snapshot of the sum of the values of each minute.

Except for the Occupation Time the measures are all additive because they can be added over all of the dimensions. When adding the Occupation Time for two segments the result is the total time that cars have been on the segments. However, when a car moves from one segment to the other the time that the car has been on the segment is reset when the second segment is entered. Therefore, adding values from two different segments does not provide the same value as if the two segments were combined to one larger segment.

7 Combination of Designs

This chapter contains a description of the design of a combination of the three methods for traffic jam detection described in the previous three chapters. The following sections will discuss aspects of combining the methods and advantages and disadvantages of this approach. After this, the design of a single data warehouse for all the methods is presented. The reason for having one data warehouse containing measures of all the three methods, is that they are quite similar. The data warehouse described for each of the methods share the same dimensions and some of the measures are present in all the methods. The chapter is ended with a description of the detection process starting with the incoming observation data and ending with the outgoing traffic jam information.

7.1 Combination of Methods

A method consisting of a combination of the three methods described in the previous three chapters can be divided into two parts that are not necessarily dependent. One part is a combined method that is used to determine if there is a traffic jam and another part is the data warehouse that contains the historical data. In this section the combined method for traffic jam detection is described.

In each of the previous three chapters a method for traffic jam detection has been designed. It might be, however, that a combination of these methods can provide a better result than the three methods individually. The methods could for instance be weighted differently such that if one method is close to detecting a traffic jam and

another method has a slight indication of traffic jam, a traffic jam can still be detected. The different methods could also be used in different places. On motorways, for instance, the Max Speed Method might provide the best information whereas the Traffic Flow Method might be better in towns.

Each of the three methods previously described are actually already combined methods because they each consists of different values. The Max Speed Method, for instance, considers both the number of cars and the maximum speed. Instead of the Max Speed Method two individual methods could have been designed, one based on the maximum speed and the other based on the number of cars, but none of these two methods would be good detection methods by themselves. The same applies for the Traffic Flow Method and Occupation Time Method. They are also combinations of methods that cannot stand alone.

A similar combination can be made on the three methods to obtain a combined method. This method can consider the four different values Number of Cars, Maximum Speed, Traffic Flow and Average Occupation Time. The values can then be compared to historical values as in each of the three methods and from this it can be determined if there is a traffic jam on a given segment. The result of the comparison can be multiplied by quantifiers to weigh the different values. The values of these quantifiers need not be the same as for the individual methods.

The values could be calculated as in the three previous methods. The expressions are shown below except for the number of cars. The expressions have been rearranged such that they must all be higher than the threshold for a traffic jam to be detected. This is done because the thresholds must be added in order to calculate the combined threshold. If the thresholds are not rearranged, their values will even out when they are summed. This would render it impossible to use the summed value for detecting traffic jams.

$$\frac{\text{Average Maximum Speed}}{\text{Maximum Speed}} > \frac{1}{\text{Speed-Quantifier}} \tag{7.1}$$

$$\frac{\text{Average Traffic Flow}}{\text{Traffic Flow}} > \frac{1}{\text{Flow-Quantifier}} \tag{7.2}$$

$$\frac{\text{Average Occupation Time}}{\text{Historical Average Occupation Time}} > \text{Occupation-Quatifier} \tag{7.3}$$

In order to design a combined method that considers all of the three values the sum of the fractions on the left-hand side of the equations can be calculated. If this sum exceeds the sum of the right-hand sides a traffic jam is detected. The necessary computations are illustrated below.

$$\frac{\text{Average Maximum Speed}}{\text{Maximum Speed}} = \text{Speed Result} \tag{7.4}$$

$$\frac{\text{Average Traffic Flow}}{\text{Traffic Flow}} = \text{Flow Result} \tag{7.5}$$

$$\frac{\text{Average Occupation Time}}{\text{Historical Average Occupation Time}} = \text{Occupation Result} \tag{7.6}$$

$$\text{Sum} = \text{Traffic Jam Result} \tag{7.7}$$

A traffic jam could then be detected if the sum, Traffic Jam Result, of these three equations exceeds a threshold. The computation of the threshold is illustrated in equation (7.8).

$$\text{Combined-Threshold} = \frac{1}{\text{Speed-Quantifier}} + \frac{1}{\text{Flow-Quantifier}} + \text{Occupation-Quantifier} \tag{7.8}$$

In addition, as described in the previous methods, the Number of Cars must be compared to the capacity of the segment. Only if the Number of Cars is sufficiently high compared to the capacity a traffic jam is detected, that is, if equation (7.9) is true.

$$\frac{\text{Number of Cars}}{\text{Capacity}} > \text{Capacity-Quantifier} \tag{7.9}$$

As it can be seen in the above calculations, historical information about cars is not included. If a road in general has very low traffic it can suddenly have a lot of traffic, for instance because of a detour. In this case, the fraction that computes the percentage of cars on the road compared to normally can suddenly become very high. It can easily be several hundred percent higher than usual. If the value of the fraction is included in the Traffic Jam Result the value can be dominant and the values of the remaining three fractions will be irrelevant.

A combined method can also consider the results of each of the three expressions individually. Instead of comparing the Traffic Jam Result to a threshold, the three values Speed Result, Flow Result and Occupation Result could each be compared to a threshold. The disadvantage of this is that the degree of the traffic jam cannot be measured. It cannot be determined how much the values exceeds the thresholds. Therefore, detections based on different combinations of these values cannot be made.

The advantage of having a combined method is that it allows for adapting more parameters. If a good combination of the parameters is found more traffic jams might be detected because traffic jams that are not detected by one method might be detected by another.

A disadvantage of having a combined method is that a bad combination of parameters can even out the methods such that less traffic jams are detected. If, for instance, the Max Speed Method detects many traffic jams a combined method might not detect the same because the combined method also considers the flow and occupation time. If these values do not indicate traffic jam it might be assumed that there is no traffic jam whereas in fact there might be one if the Max Speed Method is simply a better method.

Other disadvantages of a combined method is the increased complexity. In order to keep a simple design the time intervals over which the Maximum Speed and Traffic Flow are computed should preferably be of the same size. A further restriction for simplicity would be if all values are measured at the same time, that is, if both time intervals end at the same time and if the Number of Cars and Occupation time are computed at these same points in time. These restrictions are not necessities but they keep the method simple.

7.2 Combination of Historical Information

The historical information described in the previous three chapters can be integrated in a combined design. There are two independent reasons for having a combined design. An obvious reason is to support the combined method described in the previous section and another reason is the reduced storage usage regardless of whether the combined design is used to support the combined method or the three individual methods.

A combined design must consist of a combination of the three individual designs. The star schemas of the three previous methods all have the same dimensions and all have the same granularity. Therefore, it is be possible to store all values in the same fact table. Storing all measures in the same fact table reduces the storage requirements in several areas, partly because the number of tuples in a combined fact table will be the same as in each of the three fact tables.

First of all, each of the three fact tables in the individual designs has three foreign keys, one for referencing each of the three dimension tables. This corresponds to a total of nine foreign keys. Assuming the fact tables share the same dimensions, a combined fact table only requires three foreign keys, one for each dimension. Therefore, for each tuple six times the storage requirements for a foreign key is saved when having a combined fact table instead of three separate.

Furthermore, each of the three methods have the measure Number of Observations. Because each of the three designs have the same lowest granularity, Hour, at the lowest level of the Time of Day dimension, the measure Number of Observations will contain the same value for all facts. Therefore, if all measures are stored in a combined fact table the Number of Observations measure need only be stored once instead of three times. The same applies for the Number of Cars. This number can be calculated at different times for each method. It will then result in different values but if this measure is computed at the same time for all methods it will contain the same value regardless of the method. Therefore, the value need only be stored once in a combined fact table. Similarly for the Number of Intervals. Both the Maximum Speed and Traffic Flow are calculated within an interval. If they are calculated the same number of times within the same hour the Number of Intervals will contain the same value for each of these methods. In this case only one value need to be stored in the combined fact table.

For each of the three methods and for the combined method the number of traffic jams are stored in the fact table in order to provide sufficient information for statistical analysis. In a combined fact table that stores all values from all methods separate attributes must be present to hold the number of traffic jams detected by each method. If only a combined design is used and not the individual methods, the number of traffic jams need only be stored for the combined method. If this is the case, only two measures instead of six is needed.

This gives a total storage savings of nine measures and six foreign keys. The data warehouse containing only a single fact table is illustrated in figure 7.1 on the following page. The Road Segment dimension contains all the necessary attributes for each of the methods.

Another advantage of a combined fact table is the reduced query execution time. If a query slices on all three dimensions a combined fact table will have to do a join with each of the dimensions, that is, a total of three joins. If three fact tables are referenced in the query each of these three must be joined with the three dimension tables. This gives a total of nine joins. As previously described, the size of a combined fact table has the same number of tuples as any of the individual fact tables. Therefore, joining the three dimensions on the combined fact table takes the same time as joining them on any of the individual fact tables. The two remaining fact tables that must also be joined will therefore be extra overhead compared to a combined design.

A side effect of a combined design is the reduced complexity. A combined fact table provides a better overview of the design and by combining the measures Number of Cars, Number of Intervals, Number of Observations, Non-periodic Traffic Jams and All Traffic Jams less measures need to be handled.

The combined design has almost the same aggregation properties as the individual designs because the dimensions are the same. On the Road Segment dimension, how-

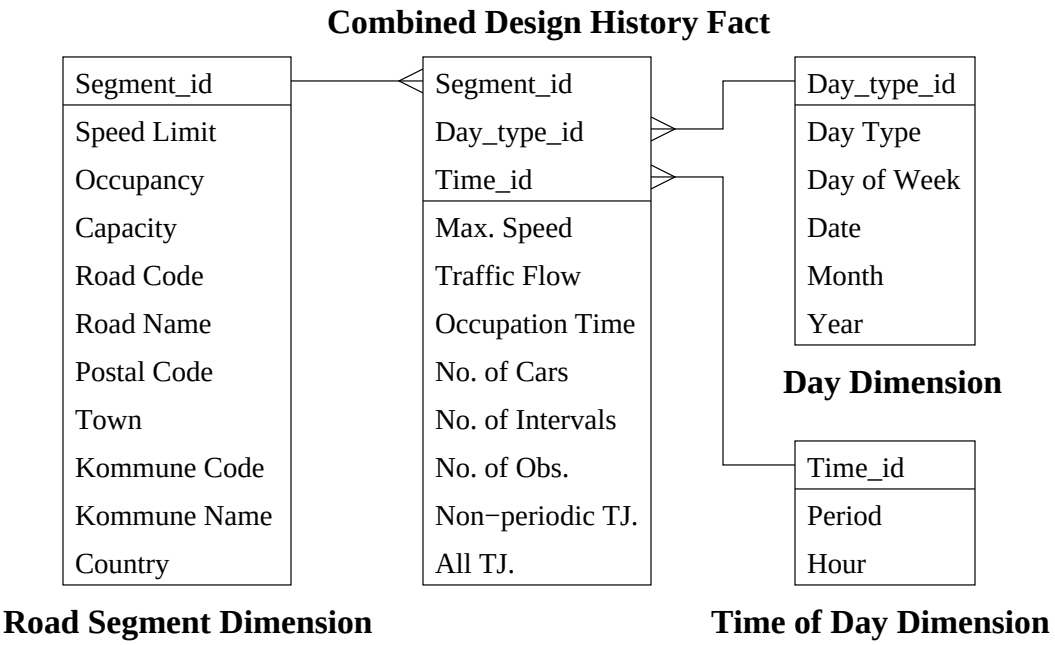


Figure 7.1: The data warehouse combining all three methods

ever, some measures cannot be added. Only the measures Number of Cars, Number of Observations, Non-periodic Traffic Jams and All Traffic Jams are additive over the Road Network dimension. This should be taken into account when aggregating such that no conclusions by mistake are made on the remaining measures.

The combined fact table is a snap-shot fact because it contains a combination of facts that are all of this type.

7.3 Design of Detection Process

This section contains a description of the information process of the traffic jam detection. First, the process for updating the data warehouse is described and then the detection process is explained.

7.3.1 Updating the Data Warehouse

As cars drive around in the streets they report their position continuously. The positions could be reported as x- and y-coordinates in a given coordinate system. As the positions are received, the id of the segment that the car is driving on is

determined for each position.

Assuming that at some times 20% of all cars in Aalborg are in use, this will roughly correspond to 15.000 cars that move around in the streets. If half of them report their position every second this means that roughly 7.500 observations are received every second from the town of Aalborg.

Figure 7.2 on the next page illustrates the data flow of the application. The observation data are continuously loaded into memory as they are received from the cars and every interval, for instance every minute, a computation process is issued. From the received observations this process computes the Maximum Speed, Traffic Flow, Occupation Time, Number of Cars and Number of Observations for each segment. These values are needed for the detection method. Within the next interval these values are kept available for the detection process described in the next section. When they are no longer needed they are moved to a temporary storage area, for instance a database. Periodically, the data from the temporary storage area are preprocessed and loaded into the data warehouse. The reason for preprocessing the data before loading it is that the values for the methods are computed every minute but the lowest granularity of the Time of Day dimension is Hours. All values for a given hour must therefore first be added before they can be loaded into the fact table. Besides this, the data warehouse need not be updated at the same frequency as the values for the detection process. This is because the lowest granularity of the Time of Day dimension is hour, and a newly calculated fact contains measures for the past hour, which will not be needed for the next 23 hours. It is also because the data from the data warehouse are being averaged over many days before being used. The data from one day does therefore not affect the average much.

The reason for having a temporary storage area is that memory is too small for storing all values for a longer time. Most important, though, had the lowest granularity corresponded to the time interval, such as one minute, it would still be infeasible to update the data warehouse at the end of every interval because there would simply be too many tuples per minute. As described previously the town of Aalborg has 2.217 roads. Assuming each road on the average is divided into 20 segments and assuming that the lowest granularity of the Time of Day dimension is minutes the fact table should be updated every minute with 44.340 tuples each containing six measures. If, within the given minute, there were some roads where no cars were driving no tuples would be added to the fact table for these segments.

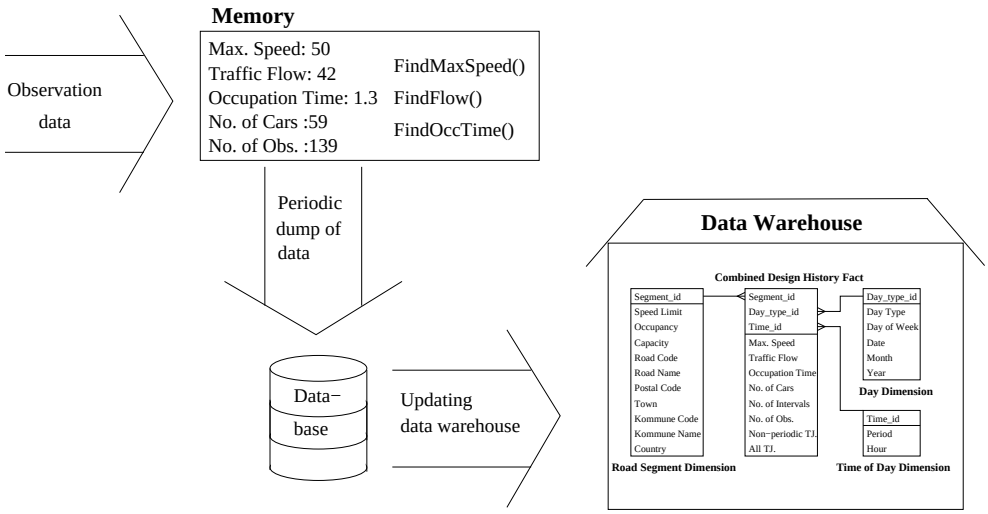


Figure 7.2: Architecture of the updating process

7.3.2 Detecting Traffic Jams

Traffic jams can be detected periodically or on request. If they are detected on request and many requests are received the detection process might have to be run many times compared to the periodic computation because the previous results cannot be reused. Therefore, the traffic jam detection process will be run periodically every

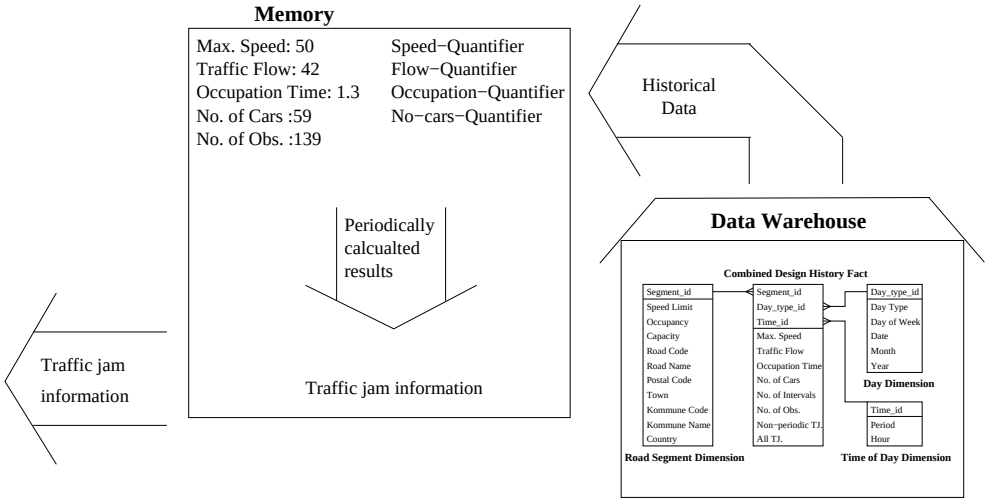


Figure 7.3: Architecture of the detection process

minute. When the detection process is issued, the current Day Type and Period are selected in the data warehouse. Every hour the values of Maximum Speed, Traffic Flow, Occupation Time and Number of Cars of each segment are fetched from the data warehouse. For each segment each of these four values are multiplied by the corresponding quantifier. Every minute the values obtained are compared to the corresponding values from the observation data and if the limits are exceeded a traffic jam might be detected.

The detection process is illustrated in figure 7.3.

8 Data Generator

In order to test the methods for detecting traffic jams, it is important to have a sufficient amount of data, and that the data is realistic. In this chapter, the data generator [Brinkhoff, 2002] is described. First, the requirements for the data generator are listed, and then a description of the data generator and the goals of the data generator are given. Finally, it is discussed how the data generator is adapted to meet the requirements.

8.1 Requirements

In order to test a system that detects traffic jams, the test should be based on realistic data. Realistic data are data that simulates the behavior of the real world. If no realistic data are at hand, it might not be possible to determine how the system will perform in the real world.
Below, the requirements for the data generator are listed together with the reason for each requirement. The requirements are aimed at generating realistic data.

1. The data generator must be customizable.
   It is important that tests can be performed with many cars on the roads as well as with few cars on the roads to explore different situations because this should affect the outcome of the methods. Other parameters such as speeds of cars and capacity of roads must also be customizable.

2. The cars must follow a network.
   Many data generators generate objects that can move freely in two dimensions. In order to detect traffic jams on a given road, it is necessary that the test objects simulate the behavior of real cars, and therefore follow a network.

3. It must be possible to specify the network to be used.
   In order to test different settings, it should be possible for the data generator to use different networks and to use networks designed by the user. This could for instance be networks where traffic jams constantly occur and never occur.

4. The cars must take each other into consideration.
   If there are many cars on a road, then, in the real world, the speed of the cars on the road will drop because the speed of a car depends on the speed of the cars in front of it. When there are so many cars on a road that their speeds affect each other, it is likely that traffic jams occur.

5. It must be possible to generate traffic jams
   In order to test the detection of traffic jams, there must exist traffic jams in the generated data. These could be created by customizing the parameters or changing the network to be used.

6. It must be possible to distinguish cars moving in opposite directions on a road.
   Since traffic jams typically only affect cars moving in one direction, it is not desirable to warn drivers moving in the other direction.

7. The variation in the speed of the cars must resemble the real world.
   If the speeds of cars in the data vary more than in the real world, the detection of traffic jams might not be accurate.

8. It must be possible to generate more traffic at specific times.
   This is necessary to simulate rush hours.

9. It must be possible to determine which road a car is driving on.
   In order to only warn cars on the same road as the traffic jam it is necessary to know which road a car is driving on. Also, the methods for detecting traffic jams depend on having this knowledge.

10. The speed of a car must be known
    The Max Speed Method uses the speed of the cars to detect traffic jams. Therefore, this speed must be known.

11. The capacity of a road must be known
    In order not to detect traffic jams with few cars on the roads, the number of cars on a road is compared to the capacity of the road. Therefore, this capacity must be known.

8.2 Description

This section contains a description of the data generator. First, the description of the data generator and its functionality is given, and then the changes made in the configuration file are described.

8.2.1 Functionality

The data generator is programmed in Java, and since it only uses classes from the SDK, it is OS independent. The source code is divided into packages. One of the packages, called "generator2", contains code from the functional component. A part of this package can be downloaded from the same web page as where the data generator can be found [Brinkhoff]. Along with the data generator is also the Javadoc documentation for the program. With this documentation and the source code it is possible to change or extend most of the functionality of the data generator.

If the data generator is started without changes in the configuration file, the user interface illustrated in figure 8.1 on the following page will appear. It displays the map of roads, on which the objects can travel. When the "Compute"-button is pressed, objects will appear and begin to move around in the network as seen in figure 8.2 on page 75. During a simulation, the progress in time is measured in timestamps. A timestamp corresponds to a cycle within each object has the opportunity to move once. The data generator is controlled both by a configuration file and by the parameters in the user interface. In the configuration file, many parameters can be adjusted. These parameters can be divided into three groups. The first group consists of parameters that determine what the user interface looks like, i.e. the size of the window, the colors of the buttons, etc. The parameters in the second group are parameters for specifying input and output files. The last group has parameters that determine the maximum number of cars initially on the road, how many timestamps to run for, the maximum number of objects that should appear in the network per timestamp, etc. In the configuration file it is also possible to switch off visualization of the objects. This allows for much faster data generation, since there can be a lot of objects that need drawing on the screen.

The input for the data generator is constituted of three files. The first file is the configuration file and the other two are network files specified in the configuration file. The first network file contains a listing of the nodes in the network, and the second consists of the edges between these nodes. The edges correspond to roads and a road is defined by its two ending points. The format of the files is a binary format described by a document [Brinkhoff, 2000]. In order to specify a user-defined network for the data generator, we have written a program that can take a text file with a textual description of a network and write it to two different binary files, containing the nodes and edges. In order to get the network into the database, we

have also written a program that takes the two binary files as input and writes them in two text files in a format that SQLLoader can manage. The source code of the programs can be seen in section A.1.

The generator will run for as many timestamps as specified in the configuration file or in the user interface and it generates a text file with the observations of the objects of the network. Each line in the output file represents one observation and consists of a list of parameters, such as the id of the object, the x-coordinate, the y-coordinate of the observation, a consecutive number for all observations and a consecutive number for each object.

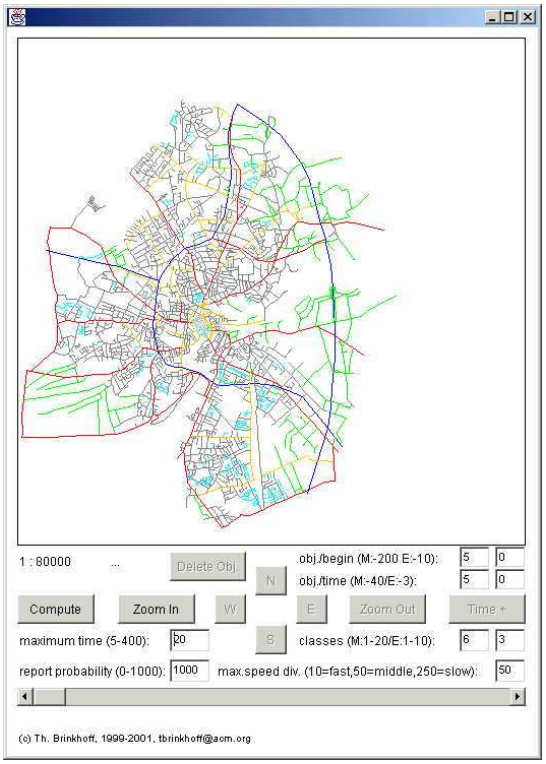


Figure 8.1: The user interface of the data generator

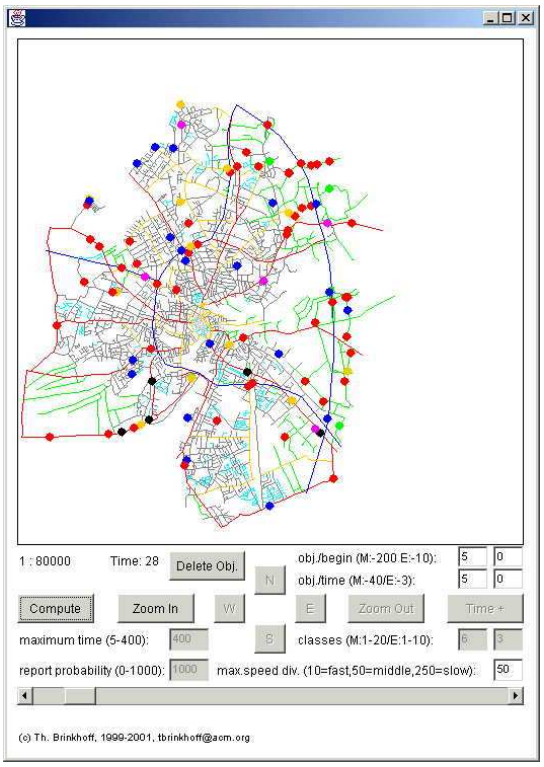


Figure 8.2: The data generator simulating moving objects

8.2.2 Configuration

The visualization of the moving objects is not important, and the visualization takes a lot of CPU time, so the visualization is switched off. There is a parameter called waitingPeriod that is the number of seconds between every cycle. Its primary goal is to let some time pass between two observations such that it is possible to watch the visualization. This parameter is set to 0. In order to generate a lot of data on different computers at the same time, six copies of the configuration file are made. In each configuration file a new output file is specified.

8.3 The Data Generators Goals

The paper that describes the data generator [Brinkhoff, 2002] contains nine statements that describe the behavior of moving objects. These statements are based on the author's experience while he was working for a company offering traffic telematics services. Next, the statements are listed together with a short description of each.

After that, a comparison between the data generator and the requirements stated in section 8.1 is given.

8.3.1 The Statements

The statements are written in small-caps, and the description follows the statement.

1. Moving real-world objects very often follow a network.
   Almost all real-world moving objects follow a network. Cars on roads, pedestrians on the pavement, even planes follow air-routes. Ships are limited by rivers, seas, etc. This corresponds to requirement number 2.

2. Most moving objects use a fast path to their destination.
   Usually, a moving object is moving from one point in the network to another point. It will not necessarily choose the fastest route between the two points, but at least a path not much slower than the fastest route. This is good for simulating realistic behavior. It is not a requirement for this project, however, since the methods for detecting traffic jams do not depend on this behavior.

3. Networks consist of classified connections, which have impact on the speed of spatiotemporal objects.
   There are different types of roads. Some roads are built for hundreds of cars passing by each minute, others are small side streets downtown. The speed an object can have on a road depends on the type of the road. This fulfills requirement number 11. It also makes it possible to generate traffic jams. To generate a traffic jam a road with low capacity can be combined with a road with high capacity.

4. The number of moving objects will influence the speed of the objects if a threshold is exceeded.
   A road has a maximum capacity, which is the maximum number of cars that can be on the road at the same time. If this number is exceeded, then there are so many cars on a road that their speed will decrease. This fulfills requirement number 4 since a traffic jam can be forced if enough objects are on the same road.

5. The path of moving objects may change if the speed on a connection is modified.
   If a driver sees that the speed on the road decreases then the driver might choose a different route expected to be faster. This is good for simulating realistic behavior, but is not needed by this project since the methods do not take routes into consideration.

6. The number of moving objects is a time-dependent function.
   During certain time intervals, there is higher traffic than in other time intervals.

Such time intervals are rush hours, holidays, etc. The data generator does not support this, but it can be simulated by generating data with different number of objects. Using this method, requirement number 8 is fulfilled.

7. The speed of moving objects is influenced by a spatiotemporal function, which is independent of the network and of the objects moving on the network.
   The weather also has impact on the routes and speeds of cars. If it is snowing, many cars wont be driving. This is good for simulating realistic behavior but it is not used in this project.

8. Moving objects belong to a class. This class restricts the maximum speed of the object.
   A truck is not allowed to drive as fast as a normal car. Cars with trailers also have restrictions on their speed. Therefore, the speed of a car depends on the type of car it is. This is good for simulating realistic behavior but it is irrelevant for testing traffic jam detection methods.

9. Time-scheduled traffic may influence the speed and the paths of moving objects.
   If cars have to take a ferry, for instance, then the time-schedule of that ferry has influence on the cars path. It will also create a temporary congestion when cars are getting on and off the ferry. The data generator does not support this feature, but if it did, this would be a way of generating a traffic jam to test the methods.

8.3.2 Comparison with Requirements

In this section, the data generator is compared to the requirements listed in section 8.1. The data generator has a configuration file, and the source code can be downloaded from the web page. This yields the data generator customizable although some of the customization requires Java knowledge. The data generator takes a network as input, and creates observations in this network. Finally, in the construction of the network, it is possible to force traffic jams. What is needed is a road with high capacity that ends in a road with low capacity. If the network has few roads, then cars will flow quickly through the road with high capacity and get onto the road with low capacity, creating a traffic jam. This customization along with the statements fulfill requirements number 1, 2, 3, 4, 5, 8 and 11. Requirements number 6, 7, 9 and 10 are still not met but in the following section it will be described how the data generator can be modified to meet these requirements as well.

8.4 Modifications to the Data generator

In order to meet the last requirements, some changes are made to the data generator. First, these changes are described and then the requirements in section 8.1 are compared to the modifications of the data generator. Finally, some desired functionality that the data generator does not support is described.

8.4.1 Changes

As previously mentioned, the source code for parts of the "generator2" package is available. The functionality of the data generator must be changed and extended in order to meet the requirements and hence to be of use in this project. These necessary changes are described below.

- Each edge in the network has a specific id, but the data generator does not per default output this id. For the traffic jam detection methods it is necessary to have the id of the edge for each observation in order to know which edge the traffic jam is on. Therefore, this id has been added to the output of the data generator.
- The Max Speed Method for detecting traffic jams described in chapter 4 detects traffic jams by considering the speed of the objects. Therefore, the data generator is modified to remember the last known position and from this position together with the current position, the speed can be calculated. The speed is also added to the output.
- The direction of an object is important for two reasons. First, it is only desirable to alert cars that have the same direction on a given road as a possible traffic jam has. Cars driving in the opposite direction need not be warned since the traffic jam does not affect them. Secondly, the methods for detecting traffic jams depend on the direction. Therefore, a method has been implemented to return the direction of an object on an edge. The data generator defines an edge by a straight line between its two ending points, node1 and node2. The method returns one value if the object is moving away from node1 and returns another value if the object is moving towards node1. This way, all objects that are moving in the same direction on a edge have the same value as their direction. The direction has also been added to the output.
- Some of the output from the data generator has been disabled. In the output file there is a text output of either "newpoint", "disappearpoint", or "point" associated with each observation depending on whether the object is observed for the first time, for the last time or in between these two, respectively. To save space, the output of this text has been disabled. Also, the x- and y-coordinate

of the observations are of no use to the traffic jam detection methods so they, too, have been disabled from the output.

- There is a method in one of the Java classes that assigns a maximum speed to each object type. This method can be altered such that objects have the desired speed.

8.4.2 Comparison with Remaining Requirements

In section 8.3.2 it was shown that the data generator itself fulfills all requirements except numbers 6, 7, 9 and 10. Due to the changes to the data generator described in this section, it fulfills these last requirements, and the data generator now meets all the requirements.

8.4.3 Desired functionality

Even though the data generator is very customizable, some features cannot be customized. For instance, the data generator does not support the changing of parameters at runtime. This could be achieved by creating an execution plan that states at which cycles the parameters of the data generator are changed. A person could look at the user interface and assess that a traffic jam could be forced if the capacity of a specific edge was reduced. Also, traffic jams could be forced if it were possible to insert a specific slow object on a given edge at a given time.

Traffic jams could also be forced if it could be specified how many of each object type can be in the network at the same time or if the capacity of roads could be set for each road.

9 Evaluation

This chapter contains a description of the experiments performed to test the methods for detecting traffic jams and the data used for the tests. First, the configuration of the PC on which the tests are performed is described and then the data used for the experiments and the test methods are described. Finally, the results from the tests are described.

9.1 Configuration of Test Machine

The tests are all performed on the same PC. The PC is a PentiumII-350MHz and it has 512MB of RAM and 95GB hard disk. Out of these 95GB hard disk, 1.5GB are reserved as swap space. The OS is Redhat Linux 7.2 with all the newest patches from Redhat's official patch-site as of 10'th of June 2002. The kernel is version 2.4.9-34.

A standard Oracle9i Database Enterprise Edition has been installed on the machine. Oracle states that the minimum system requirements for running Oracle9i are a PentiumII-233MHz with 512MB RAM and twice as much swap space or 512MB of swap space, whichever is the highest. All these demands are met by the test machine. The machine was not in use for anything else during the tests.

9.2 Implementation

In this section, the implementation is described. The generated data is described and then the process of loading the historical data into the fact table is described. Finally, the implementation of the traffic jam detection methods is given. First, the architecture of the implementation is illustrated in figure 9.1 on the following page.

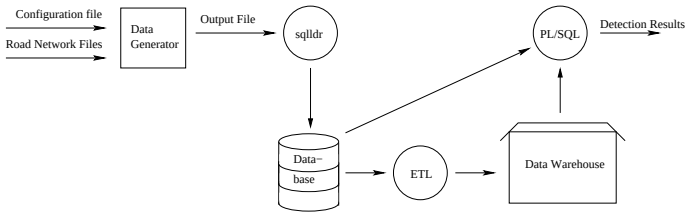


Figure 9.1: The architecture of the implementation

Input data are provided for the generator in terms of input files and a road network. The generated data are then stored in a relational database. From these data the data warehouse is updated by using the standard Extraction, Transformation and Loading (ETL) process. When simulating the traffic jam detection process, PL/SQL code is used to fetch data from the database and data warehouse into memory and it is used to obtain the necessary results.

9.2.1 Data

In order to test the methods, some historical data are needed. Therefore, ten data sets of 7200 cycles each are generated. A cycle is defined to correspond to one second, and a data set therefore contains data for two hours. The data sets consists of almost 90,000,000 observations. The data sets are loaded into a temporary table in the database using SQLLoader and the data are then loaded into the fact table using PL/SQL procedures described in the next section. In order to test the methods for detection of traffic jams, the data sets are defined to contain data for the time interval between 8 a.m. and 10 a.m. on Mondays. This way, historical data for ten weeks is available for the testing of new data.

In order to test the methods, data that simulates runtime input is also required. Therefore, seven different data sets are created, each with different capacities on the roads. Data sets with both higher, lower and the same capacities on the roads as the historical data are used. The different capacities are used to create traffic jams because the speeds of cars are reduced if more cars are on a road than the capacity allows.

9.2.2 Loading Process

Five different PL/SQL procedures are created to load the data into the fact table. Each procedure calculates one of the measures from the fact table. The measures Number Of Intervals, Non-periodic Traffic Jams and All Traffic Jams are not calculated. The PL/SQL code can be seen in section A.4. Each procedure divides the

observations into minutes consisting of 60 cycles, calculates a measure, and inserts a tuple in a temporary table for storing measures. The tuple contains the measure, foreign keys to dimensions, and null-values for the measures that are not calculated. After all five procedures have calculated their respective measures, a final procedure loads the measures into the fact table.

Only the measures for historical information are implemented. This means that tests will only be performed to check for traffic jams that do not normally appear.

9.2.3 Implementation of Methods

To test the methods, data that simulates runtime input is required. Procedures that simulate taking the runtime input are also needed. The data that is tested against the historical data are loaded into a table, and the test procedure takes the observations from this table instead of receiving runtime observations. For each interval of 60 cycles, the three methods are run and the results from each method is handled as described in the three chapters 4, 5 and 6 to determine whether each method detects a traffic jam at the current time for each road. The detection of a traffic jam by a method depends on the quantifier for that method. The final results from the methods for the given values of the quantifiers are kept in a result table. From this result table, it can be counted how many traffic jams each method has detected and how much overlap between the methods there is.

9.3 Results

In this section, the results from the experiments are presented. The seven test data sets are as follows. One data set is based on the same capacities as the historical data. Two data sets are based on different higher capacities on the roads than the historical data and four data sets are based on different lower capacities. It is expected that no traffic jams are detected for the two data sets with higher capacities and that only few traffic jams are detected for the data based on the same capacities as the historical data. For the data sets with smaller capacities than the historical data, it is expected that more traffic jams are detected for each decrease in the capacity.

To test whether or not the methods actually detect traffic jams, different approaches can be applied.

One approach is to visualize the test data and check visually whether or not there are traffic jams and make notes. These notes can then be compared to the results of the methods to check whether or not the methods detect the traffic jams that were actually there. This approach, however has some disadvantages. First of all, the visualization is difficult to do properly. The data generator does not prevent objects

from overtaking each other, making it practically impossible to determine how many objects are on a road at the same time. Secondly, the only means of determining whether a traffic jam is present or not is to assess the situation according to the same criteria that the methods used.

Another approach could be to have real test data. Then, a person could be placed on all roads and make notes about traffic jams. These notes can then be compared to the results of the methods. This requires quite a large amount of people, and that many people have not volunteered to help in this project.

The final approach is to compare the methods against each other. If two or more methods detect the same traffic jams then the probability that there is a traffic jam increases. This is the method that is used to test the methods in this report. The following histograms describe the result of the methods for each of the test data sets. For the tests, the quantifiers for the methods are set as seen in table 9.2.

Min-cars-Quantifier	1/5
Speed-Quantifier	4/5
Flow-Quantifier	3/4
Occupation-Quantifier	3/2
Cars-Quantifier	9/10
Combined-Threshold	61/20

Table 9.2: The values of the quantifiers used for the experiments

The values of the quantifiers have been set according to an estimate about how slow the traffic must be before it is annoying to the driver and how heavy it must be before it is considered a traffic jam. A preferable approach for determining the values is to have real-world data about traffic jams. For each actual traffic jam the values of the quantifiers can be determined and from these values good quantifiers can be obtained. As mentioned previously, such information is not available.

Besides the quantifiers, other parameters have been set for the experiments. The traffic jam detection process is run periodically and the interval length is the duration of the period for which each detection process runs. The number of consecutive detections indicates how many positive detections in a row the methods require before reporting a traffic jam. The values of all the parameters are given in table 9.3 on the facing page.

Description	Value
Interval length	1 minute
Number of segments in the network	62
Number of historical data sets	10 (20 hours)
Number of test data sets	7
Number of consecutive detections	1
Number of cars initially in the network	5
Maximum number of cars in the network	471

Table 9.3: The values of all other parameters for the experiments

The alternation of the capacities for each test is described in table 9.4.

Test Identifier	Description of capacities
Standard	The capacities are the same as the historical data (between 30 and 60)
Plus10	The capacity is increased by 10 on each road
Plus20	The capacity is increased by 20 on each road
Minus5	The capacity is reduced by 5 on each road
Minus10	The capacity is reduced by 10 on each road
Minus1015	The capacity is reduced by 10 on the four road classes with the lowest capacity and with 15 on the other roads.
Minus1020	The capacity is lowered by 10 on the four road classes with the lowest capacity and with 20 on the other roads.

Table 9.4: The capacities for the test data sets

The results of the tests are given in the histograms below. The x-axis displays the different methods and the y-axis shows the number of traffic jams detected. The labels Method 1, Method 2 and Method 3 refer to the Max Speed Method, Traffic Flow Method and Occupation Time Method, respectively. Method 4 refers to the combined method and the remaining labels represent the overlap of detected traffic jams between the different methods. The text below each figure determines the alternation in capacities, as it is listed in table 9.4. The histograms in figures 9.5 on the next page to 9.11 on the following page show that the Max Speed Method has detected more than 400 traffic jams in every test and the same applies for the Occupation Time Method except for two cases. This is a rather high number of traffic jams considering that it is only for two hours and 62 segments. The reason for these high numbers can be the values of the parameters. If, for instance, the quantifiers are too high or too low either too many or too few traffic jams can be detected.

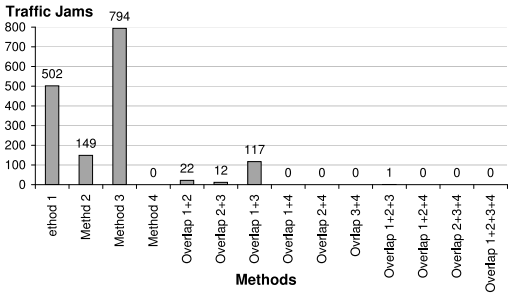


Figure 9.5: Standard

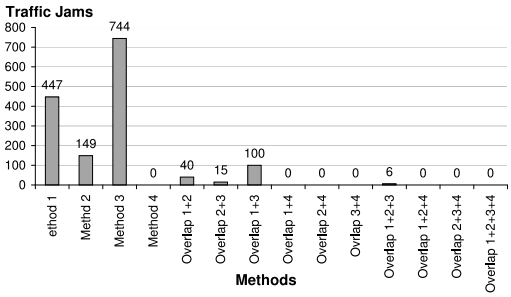


Figure 9.6: Plus10

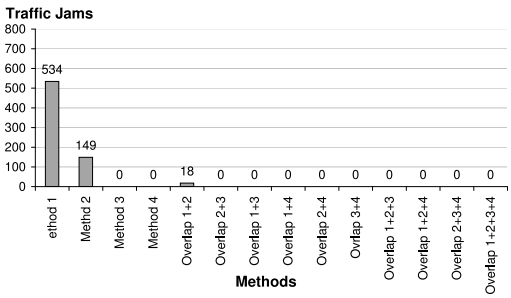


Figure 9.7: Plus20

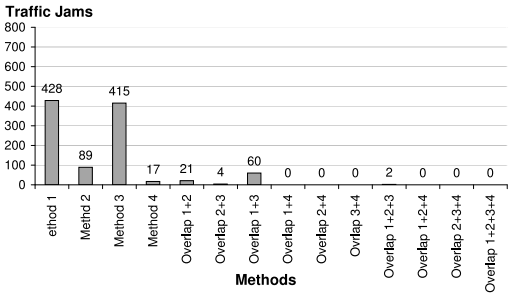


Figure 9.11: Minus1020

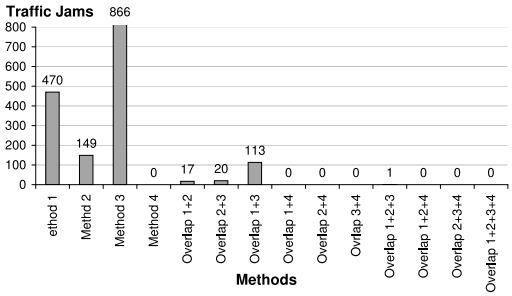


Figure 9.8: Minus5

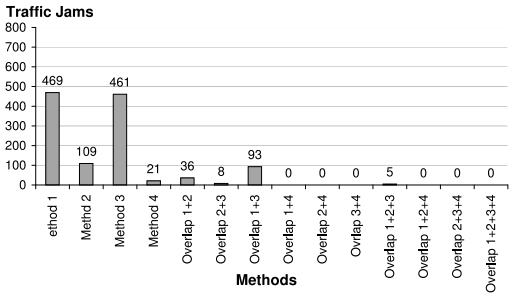


Figure 9.9: Minus10

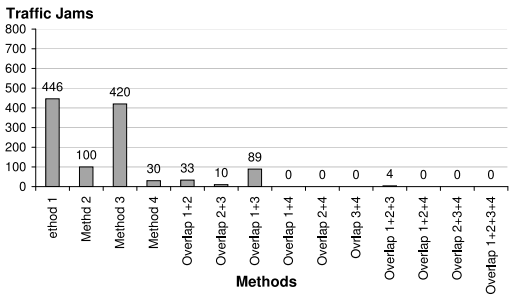


Figure 9.10: Minus1015

In the five tests where both the Max Speed Method and the Occupation Time Method have a high detection rate there is an overlap of approximately 100 traffic jams detected by both methods. The Traffic Flow Method has a lower but steady detection rate between 89 and 149. The reason for the low number of traffic jams detected by this method might be due to the requirement about the minimum number of cars on a segment. Because the traffic flow depends on the number of cars the minimum number of cars required for this method is relatively high compared to the minimum requirements for the Max Speed Method and Occupation Time Method. Due to the sparse amount of data this requirement might not be fulfilled in several cases where the other methods detect traffic jams. This could be the reason for the lower number of traffic jams detected by this method. A similar reasoning applies for the combined method. The threshold for this method is rather high compared to the other methods because the threshold is the sum of three quantifiers for the other methods. This implies that all three methods should detect a traffic jam before the combined method detects one. If not all three methods detect traffic jams one or two of them must detect a heavy traffic jam in order to provide a sufficiently high value for exceeding the threshold. A suggestion for improving the combined method is to reduce the threshold for detecting traffic jams, especially the contributions to the threshold from the quantifiers from the Max Speed Method and Occupation Time Method could be reduced.

The different capacities of the roads in the different data sets are intended at resulting in a different number of traffic jams detected such that high capacities result in few traffic jams and low capacities cause many traffic jams. This, however, is not indicated by the histograms. These show different values regardless of the increased or decreased capacity. The reason for this can be the small amount of data. Not many cars are simulated moving in the network and historical data for only ten days has been used. The variations over these ten days can easily be reflected in the aggregated values and this renders the historical data unreliable.

As a summary, traffic jams have been detected by all of the methods and by adjusting the parameters a larger overlap between the methods might be obtained.

10 Conclusion and Future Work

In this chapter, first the conclusion is given and next, some future work is described.

10.1 Conclusion

Every day, many drivers are annoyed because they are stuck in a traffic jam. If the traffic jams could be detected and the drivers warned in time to drive around it, many drivers would be satisfied and they would reach their destination faster. In this project, three methods and a combination of these three methods have been developed to detect traffic jams. The methods are based on a data warehouse that contains historical data about the general behavior of traffic on given roads. Each minute the current situation can be compared to this historical data to test if a traffic jam is present. Only traffic jams that are non-periodic are detected, although the design allows for detection of all traffic jams. The design is a star schema and this allows for fast retrieval of historical traffic information for a given road on a given time to be compared with the current situation.

A data warehouse is especially useful for fast answers to specific queries that involve a large amount of aggregated data. In a data warehouse the information is pre-aggregated in order to answer queries about data on different levels in hierarchies in dimensions fast. This pre-aggregation is not used in the design or implementation of the methods. The implementation could therefore be a normal relational model. Some of the aspects of a data warehouse are present, though. The Extraction, Transformation and Loading process is present in the design and it is possible to

aggregate over days if data from more days is needed in order to detect traffic jams. On contrary, if a road construction work has reduced the normal traffic flow on a road then for that road only historical data from the beginning of the construction work can be considered.

An advantage of the design is that it allows for detecting traffic jams on all roads that have more than just a few cars. In contradiction to other existing traffic jam detection methods the design is thus independent of the location. The disadvantage is the technology required for the design. The design assumes that a representative number of cars can report their position and it suggests a combination of using GPS and mobile phones for this matter. Only few people have GPS receivers and therefore, the design cannot be due to the rapid developments of technological devices GPS might be offered as extended equipment for mobile phones for a reasonable price within a decade.

The test of the methods show that traffic jams can be detected and that for some methods there is an overlap between the traffic jams detected. This provides a more certain indication of traffic jam. The number of traffic jams detected by the different methods can be adjusted by altering the parameters for the methods. This way, more even results might be obtained.

In chapter 2, some overall design goals are listed. Statements number 1, 2, 3, 4 and 5 are all met by the data generator, if not per default then the data generator has been altered to meet these goals. Therefore, in the experiments of the methods the preconditions for the design have been met. The methods are designed to detect both periodic and non-periodic traffic jams and it also allows for only detection of non-periodic traffic jams. Therefore, all design goals are met, either by the generator or by the design.

10.2 Future Work

The part of the design that detects traffic jams based on historical information has been implemented. As future work, the remaining parts of the design could be implemented such that periodic traffic jams can also be detected. Also, a real version that takes runtime observations could be implemented.

Another interesting task is to acquire real data and test the methods against this data to find out how they perform and to set the quantifiers more accurately.

It could also be interesting to implement a function that could determine if a car can get to its destination faster if the driver is warned about the traffic jam. If the car can not get to the destination faster if it avoids the traffic jam then it is not necessary to warn the driver.

Recently, a new area within databases has approached. The so called DSMS's (Data Stream Management Systems) can take a possibly never ending stream of data and perform queries on these. Joins can be made between these and persistent relations. Such a DSMS could be a natural implementation of a real-life implementation of this design.

The design of the traffic jam methods contains additional attributes not used by the traffic jam detection. The purpose of having these additional attributes is to provide the ability for statistical analysis on the traffic information. The design has not been optimized for statistics but aggregations at different levels of the hierarchies is possible. This could for instance be used to determine the number of traffic jams on different roads and at different times.

An additional application of traffic information is analysis to predict traffic jams. This enables the possibility of warning drives before the get stuck, and it furthermore allows for informing many drivers such that the traffic jam could possibly be avoided.

A topic not considered in this project is the investigation of how to determine which drivers should be warned. Drivers that are heading in the direction of the traffic jam should be warned before they get there but they should not be warned if they are too far away because then it is too uncertain in which direction they will continue. If they turn in another direction long before reaching the traffic jam the warning will be irrelevant for them

As described previously, the reliability of the traffic jam information can be enhanced by reporting a traffic jam only when it has been detected for several consecutive intervals. The additional reliability obtained this way can be compared to the normal design and the advantages can be compared to disadvantage of delayed warnings. Another concern is the length of the intervals. It could for instance be investigated if more traffic jams could be detected if the intervals depend on the point in time.

There are many different topics for interesting future work and the results of the experiments indicate that the methods can actually be employed for traffic jam detection.

Bibliography

B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. 2002. To appear in Proceedings of the Twenty-First ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Madison, Wisconsin, June 2002.

Thomas Brinkhoff. Webpage for brinkhoffs datagenerator. Webpage.

Thomas Brinkhoff. Description of the format of the network files. March 2000. Unpublished.

Thomas Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 2002.

Ralf Hartmüt Guting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. 25:1–42, 2000. ISSN 0362-5915.

Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *In Proceedings of ACM-SIGMOID Conference on the Management of Data*, pages 47–57, 1984.

Christian S. Jensen and R. T. Snodgrass. Temporal data management. In *IEEE Transactions on Knowledge and Data Engineering*, volume 11, pages 36–44, January/February 1999.

Ralph Kimball, Laura Reeves, Margy Ross, and Warren Thornthwaute. *The Data Warehouse Lifecycle Toolkit*. John Wiley and Sons, Inc., 1998. ISBN 0-471-25547-5.

Ralph Kimball. *The Data Warehouse Toolkit*. John Wiley and Sons, Inc., 1996. ISBN 0-471-15337-0.

Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient olap operations in spatial data warehouses. In *SSTD*, volume 2121 of *Lecture Notes in Computer Science*, pages 443–459. Springer, 2001. ISBN 3-540-42301-X.

Dimitris Papadias, Yufei Tao, Panos Kalnis, and Jun Zhang. Indexing spatio-temporal data warehouses. 2001. To appear in the Proceedings of IEEE International Conference on Data Engineering (ICDE), 2002.

Torben B. Pedersen and Christian S. Jensen. Multidimensional database technology. *IEEE Computer*, 34:40–46, 2001. ISSN 0018-9162.

Torben Bach Pedersen, Christian S. Jensen, and Curtis E. Dyreson. Extending practical pre-aggregation in on-line analytical processing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*. Morgan Kaufmann, 1999. TR R-99-5004.

Torben B. Pedersen, Christian S. Jensen, and Curtis E. Dyreson. A foundation for capturing and quering complex multidimensional data. *Information Systems*, 26(5):383–423, 2001.

Dieter Pfoser and Christian S. Jensen. Capturing the uncertainty of moving-object representations. In *Proceedings of the Sixth International Symposium on Spatial Databases*, pages 111–132, 1999.

Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel approaches in query processing for moving object trajectories. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*. Morgan Kaufmann, 2000. ISBN 1-55860-715-3.

Evaggelia Pitoura and George Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):571–592, July/August 2001.

Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the positions of continously moving objects. In *TODS*, volume 29, pages 331–342. ACM Press, 2000. ISSN 0613-5808.

Erik Thomsen, George Spofford, and Dick Chase. *Micrisoft OLAP Solutions*. John Wiley & Sons, Inc., 1999. ISBN 0-471-33258-5.

Ouri Wolfson, Liqin Jiang, A. Prasad Sistla, Sam Chamberlain, Naphtali Rishe, and Minglin Deng. Databases for tracking mobile units in real time. In *Database Theory - ICDT'99*, volume 1540 of *Lecture Notes in Computer Science*, pages 169–186. Springer, 1999. ISBN 3-540-65452-6.

A Source Code

In this appendix, the source code for all written programs is given. First, the source code of the Java programs that read and write the binary network files of the data generator is listed, and then the SQL-statements for creating the fact table, the dimensions and loading data into the dimensions is given. Next, the source code for the ETL process is given, and finally, the method-calculation source code is given.

A.1 Reading and Writing Network

In this section, the source code for the Java programs that we programmed in order to read and write the binary network files for the data generator are given.

```
/* This class is a program that will read to textinput files
   describing a network for the data generator and produce
   the two binary files needed by the data generator and a
   textfile with modified edge-data for SQLLoader. */

import java.io.*;

class Hec {
    static int taeller = 0;
    private static void writeNode(DataOutputStream dos, String name,
                                  long id, int xc, int yc) {
        try {
            dos.writeByte((byte)(name.length()));
            dos.write(name.getBytes());
            dos.writeLong(id);
```

```
            dos.writeInt(xc);
            dos.writeInt(yc);
        } catch (IOException e) {
            System.out.println("Pis også! "+e);
        }
    }
    private static void writeNodes(String infilename, String outfilename){
        System.out.println("Nodes!");
        BufferedReader br = null;
        try {
            br = new BufferedReader(new InputStreamReader(
                                    new FileInputStream(infilename)));
        } catch (FileNotFoundException e) {
            System.out.println("File not found: "+e);
        }
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(outfilename);
        } catch (FileNotFoundException e) {
            System.out.println("File not found   "+e);
            System.exit(1);
        }
        DataOutputStream dos = new DataOutputStream(fos);
        String input = new String();
        while (input != null) {
            try {
                input = br.readLine();
            } catch(IOException e) {
                System.out.println("IOException: "+e);
                System.exit(1);
            }
            if (input != null) {
                int index = input.indexOf(",");
                Integer intId = new Integer(input.substring(0,index));
                int oldindex = index+1;
                index = input.indexOf(",",index+1);
                String name = input.substring(oldindex,index);
                oldindex = index+1;
                index = input.indexOf(",",index+1);
                Integer xc = new Integer(input.substring(oldindex,index));
                oldindex = index+1;
                Integer yc = new Integer(input.substring(oldindex,
                                         input.length()));
                writeNode(dos,name,intId.longValue(),xc.intValue(),
```

```
                                                    yc.intValue());
            }
        }
        try {
            dos.close();
            br.close();
        } catch (IOException e) {
            System.out.println("IOException! : "+e);
            System.exit(1);
        }
    }
    private static void writeBinEdge(DataOutputStream dos, String
                                     name, long id, int edgeclass,
                                     long n1id, long n2id) {
        try {
            dos.writeLong(n1id);
            dos.writeLong(n2id);
            dos.writeByte((byte)(name.length()));
            dos.write(name.getBytes());
            dos.writeLong(id);
            dos.writeInt(edgeclass);
        } catch (IOException e) {
            System.out.println("Pis også! "+e);
        }
    }
    private static void writeTxtEdge(DataOutputStream fos, String
                                     name, long id, int edgeclass,
                                     long n1id, long n2id) {
        String idString = String.valueOf(id);
        try {
            fos.writeBytes(idString);
            fos.writeBytes(",");
            fos.writeBytes(name);
            fos.writeBytes(",");
            fos.writeBytes(String.valueOf(edgeclass));
            fos.writeBytes(",");
            fos.writeBytes(String.valueOf(n1id));
            fos.writeBytes(",");
            fos.writeBytes(String.valueOf(n2id));
            fos.writeBytes(",");
            fos.writeBytes("0");
            fos.writeBytes(",");
            taeller++;
            fos.writeBytes(String.valueOf(taeller));
```

```
            fos.writeBytes("\n");
            fos.writeBytes(idString);
            fos.writeBytes(",");
            fos.writeBytes(name);
            fos.writeBytes(",");
            fos.writeBytes(String.valueOf(edgeclass));
            fos.writeBytes(",");
            fos.writeBytes(String.valueOf(n1id));
            fos.writeBytes(",");
            fos.writeBytes(String.valueOf(n2id));
            fos.writeBytes(",");
            fos.writeBytes("1");
            fos.writeBytes(",");
            taeller++;
            fos.writeBytes(String.valueOf(taeller));
            fos.writeBytes("\n");
        } catch (IOException e) {
            System.out.println("Pis! " + e);
        }
    }
    private static void writeEdges(String infilename,
                          String txtoutfilename, String binoutfilename){
        System.out.println("Edges!");
        BufferedReader br = null;
        try {
            br = new BufferedReader(new InputStreamReader(
                                    new FileInputStream(infilename)));
        } catch (FileNotFoundException e) {
            System.out.println("File not found: "+e);
        }
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(binoutfilename);
        } catch (FileNotFoundException e) {
            System.out.println("File not found   "+e);
            System.exit(1);
        }
        DataOutputStream dos = new DataOutputStream(fos);
        FileOutputStream tfos = null;
        try {
            tfos = new FileOutputStream(txtoutfilename);
        } catch (FileNotFoundException e) {
            System.out.println("File not found   "+e);
            System.exit(1);
```

```
        }
        DataOutputStream tdos = new DataOutputStream(tfos);
        String input = new String();
        while (input != null) {
            try {
                input = br.readLine();
            } catch(IOException e) {
                System.out.println("IOException: "+e);
                System.exit(1);
            }
            if (input != null) {
                int index = input.indexOf(",");
                Integer Id = new Integer(input.substring(0,index));
                int oldindex = index+1;
                index = input.indexOf(",",index+1);
                String name = input.substring(oldindex,index);
                oldindex = index+1;
                index = input.indexOf(",",index+1);
                Integer edgeclass = new Integer(input.substring
                                                (oldindex,index));
                oldindex = index+1;
                index = input.indexOf(",",index+1);
                Integer n1id = new Integer(input.substring(oldindex,index));
                oldindex = index+1;
                Integer n2id = new Integer(input.substring
                                           (oldindex,input.length()));
                writeBinEdge(dos,name,Id.longValue(),
                                 edgeclass.intValue(),
                                 n1id.longValue(),n2id.longValue());
                writeTxtEdge(tdos,name,Id.longValue(),
                                 edgeclass.intValue(),
                                 n1id.longValue(),n2id.longValue());
            }
        }
        try {
            dos.close();
            br.close();
        } catch (IOException e) {
            System.out.println("IOException! : "+e);
            System.exit(1);
        }
    }
    public static void main(String[] args) {
        String nodeoutfile = new String("hec.node");
```

```
        String edgebinoutfile = new String("hec.edge");
        String nodeinfile = new String("hec.node.in");
        String edgeinfile = new String("hec.edge.in");
        String edgetxtoutfile = new String("hec.edge.txt");
        writeNodes(nodeinfile, nodeoutfile);
        writeEdges(edgeinfile, edgetxtoutfile, edgebinoutfile);
    }
}


/* This class is a program that takes the two binary network files
   from the data generator as input and generates two textfiles
   suited for SQLLoader */

import java.io.*;
import java.lang.reflect.Array;

class Main {
    private static int taeller = 0;

    private static void writeNode(DataOutputStream dos, long id,
                                  byte[] name, int n1id, int n2id) {
        try {
            dos.writeBytes(String.valueOf(id));
            dos.writeBytes(",");
            dos.writeBytes(new String(name));
            dos.writeBytes(",");
            dos.writeBytes(String.valueOf(n1id));
            dos.writeBytes(",");
            dos.writeBytes(String.valueOf(n2id));
            dos.writeBytes("\n");
        } catch (IOException e) {
            System.out.println("Pis! " + e);
        }
    }

    private static void readNodes(String filename, String nodeoutfile) {
        try {
            DataInputStream dis = new DataInputStream(
                                  new FileInputStream(filename));
            DataOutputStream dos = new DataOutputStream(
                                   new FileOutputStream(nodeoutfile));
            try {
                while (true) {
```

```
                byte length = dis.readByte();
                byte[] name = new byte[length];
                dis.read(name);
                long id = dis.readLong();
                int xcoord = dis.readInt();
                int ycoord = dis.readInt();
                writeNode(dos, id, name, xcoord, ycoord);
            }
        } catch(EOFException e) {
            ;
        } catch(IOException e) {
            System.out.println("IOException "+e);
        }
        try {
            dis.close();
            dos.close();
        } catch(IOException e) {
            System.out.println("IOException "+e);
        }
    } catch(FileNotFoundException e) {
        System.out.println("FileNotFoundException "+e);
    }
}

private static void writeEdge(DataOutputStream dos, long id,
                  byte[] name, int edgeclass, long n1id, long n2id) {
    try {
        dos.writeBytes(String.valueOf(id));
        dos.writeBytes(",");
        dos.writeBytes(new String(name));
        dos.writeBytes(",");
        dos.writeBytes(String.valueOf(edgeclass));
        dos.writeBytes(",");
        dos.writeBytes(String.valueOf(n1id));
        dos.writeBytes(",");
        dos.writeBytes(String.valueOf(n2id));
        dos.writeBytes(",");
        dos.writeBytes("0");
        dos.writeBytes(",");
        taeller++;
        dos.writeBytes(String.valueOf(taeller));
        dos.writeBytes("\n");
        dos.writeBytes(String.valueOf(id));
        dos.writeBytes(",");
```

```
            dos.writeBytes(new String(name));
            dos.writeBytes(",");
            dos.writeBytes(String.valueOf(edgeclass));
            dos.writeBytes(",");
            dos.writeBytes(String.valueOf(n1id));
            dos.writeBytes(",");
            dos.writeBytes(String.valueOf(n2id));
            dos.writeBytes(",");
            dos.writeBytes("1");
            dos.writeBytes(",");
            taeller++;
            dos.writeBytes(String.valueOf(taeller));
            dos.writeBytes("\n");
        } catch (IOException e) {
            System.out.println("Pis! " + e);
        }
    }

    private static void readEdges(String filename, String outputfile) {
        try {
            DataInputStream dis = new DataInputStream(
                                  new FileInputStream(filename));
            DataOutputStream dos = new DataOutputStream(
                                   new FileOutputStream(outputfile));
            try {
                while (true) {
                    long n1id = dis.readLong();
                    long n2id = dis.readLong();
                    byte length = dis.readByte();
                    byte[] name = new byte[length];
                    dis.read(name);
                    long id = dis.readLong();
                    int edgeclass = dis.readInt();
                    writeEdge(dos, id, name, edgeclass, n1id, n2id);
                }
            } catch(EOFException e) {
                ;
            } catch(IOException e) {
                System.out.println("IOException "+e);
            }
            try {
                dis.close();
                dos.close();
            } catch(IOException e) {
```

```
                System.out.println("IOException "+e);
            }
        } catch(FileNotFoundException e) {
            System.out.println("FileNotFoundException "+e);
        }
    }

    public static void main(String[] args){
        if (args.length == 0) {
            System.out.println("Usage:    java Main
        <mapfilename>\n");
            System.out.println("mapfilename: name of the map WITHOUT
                        the  \".node\" or the
        \".edge\"-extension");
            System.exit(1);
        }
        String nodefile = new String(args[0]+".node");
        String nodeoutfile = new String(nodefile+".txt");
        String edgefile = new String(args[0]+".edge");
        String edgeoutfile = new String(edgefile+".txt");
        readNodes(nodefile, nodeoutfile);
        readEdges(edgefile, edgeoutfile);
    }
}
```

A.2 Fact Table and Dimensions

In this section, the SQL-statements for creating the fact table and dimensions is listed. After that, some functions for laoding data into the dimensions are given.

```
/*The Time of Day dimension */
create table time_of_day_dim (
pk numeric(2) primary key,
description varchar(12) not null,
hec_hour numeric(2) not null
);

/* The Day dimension */
create table day_dim (
pk numeric(4) primary key,
day_type varchar(7) not null,
day_of_week varchar(9) not null,
```

```
hec_date numeric(2) not null,
hec_month numeric(2) not null,
hec_year numeric(4) not null
);

/* The Road Segments dimension */
create table road_segments_dim (
pk numeric(5) primary key,
road_code numeric(11) not null,
road_name varchar(17) not null,
direction numeric(1) not null,
capacity numeric(2) not null,
kommune_code numeric(3) not null,
kommune_name varchar(7) not null,
country varchar(7) not null
);

/* The Fact table */
create table fact_table (
time_of_day_fk numeric(2) references time_of_day_dim(pk),
day_fk numeric(4) references day_dim(pk),
road_segments_fk numeric(5) references road_segments_dim(pk),
max_speed_sum numeric(10),
traffic_flow_sum numeric(10),
occupation_time_sum numeric(10),
no_of_cars_Sum numeric(10),
no_of_obs_sum numeric(10),
no_of_intervals numeric(10) not null
);


/* This function determines if a given year is a leap year. */
create or replace function leapyear(aar number) return number is
skudaar number;
begin
skudaar := 0;
if aar mod 4 = 0 then
        skudaar := 1;
        if aar mod 100 = 0 then
                skudaar := 0;
                if aar mod 400 = 0 then
                        skudaar := 1;
                end if;
        end if;
```

```
end if;
return skudaar;
end leapyear;

/* This function determines which weekday a given date is */
create or replace function DayofWeek(aar number, maaned number, dag
number) return number is
a number;
y number;
m number;
d number;
begin
a := floor((14-maaned)/12);
y := aar - a;
m := maaned + ( 12*a ) - 2;
d := (dag + y + floor(y/4) - floor(y/100) + floor(y/400) +
     floor((31*m)/12)) mod 7;
return d;
end DayofWeek;

/* This function loads data into the Road Segments Dimension.
   The data is taken from the table "Edges" which contains the raw
   Data from the network for the data generator */
declare --Load data into road_segments_dim
TYPE arrayseven IS VARRAY(7) OF NUMBER;
capacityA arrayseven;
cursor veje is select * from edges;
vV veje%rowtype;
vejnavn varchar2(20);
begin
capacityA := arrayseven(30,20,20,10,10,10,10);
open veje;
loop
        fetch veje into vV;
        exit when veje%notfound;
        vejnavn := 'Edge ' || to_char(vV.id);
        insert into road_segments_dim values
        (vV.pk,vV.id,vejnavn,vV.retning,capacityA(vV.class+1),851,
        'Aalborg','Danmark');
end loop;
end;

/* This function loads data into the Time of Day dimension
declare
```

```
begin
insert into time_of_day_dim values(1,'Night',0);
insert into time_of_day_dim values(2,'Night',1);
insert into time_of_day_dim values(3,'Night',2);
insert into time_of_day_dim values(4,'Night',3);
insert into time_of_day_dim values(5,'Night',4);
insert into time_of_day_dim values(6,'Night',5);
insert into time_of_day_dim values(7,'Night',6);
insert into time_of_day_dim values(8,'Morning',7);
insert into time_of_day_dim values(9,'Morning',8);
insert into time_of_day_dim values(10,'Daytime',9);
insert into time_of_day_dim values(11,'Daytime',10);
insert into time_of_day_dim values(12,'Daytime',11);
insert into time_of_day_dim values(13,'Daytime',12);
insert into time_of_day_dim values(14,'Daytime',13);
insert into time_of_day_dim values(15,'Daytime',14);
insert into time_of_day_dim values(16,'Afternoon',15);
insert into time_of_day_dim values(17,'Afternoon',16);
insert into time_of_day_dim values(18,'Evening',17);
insert into time_of_day_dim values(19,'Evening',18);
insert into time_of_day_dim values(20,'Evening',19);
insert into time_of_day_dim values(21,'Evening',20);
insert into time_of_day_dim values(22,'Night',21);
insert into time_of_day_dim values(23,'Night',22);
insert into time_of_day_dim values(24,'Night',23);
end;

/* This function loads data into the Day dimension
declare
TYPE arrayseven IS VARRAY(7) OF VARCHAR(9);
DayOfWeekA arrayseven;
DayTypeA arrayseven;
taeller number;
slutaar constant number := 2010;
startaar constant number := 2000;
maaned number;
aar number;
dag number;
dow number;
skudaar number;

begin
aar := startaar;
```

```
DayOfWeekA := arrayseven('Monday', 'Tuesday', 'Wednesday', 'Thursday',
                         'Friday', 'Saturday', 'Sunday');
DayTypeA := arrayseven('Weekday','Weekday','Weekday','Weekday',
                       'Weekday','Weekend','Weekend');
maaned := 1;
dag := 1;
dow := DayofWeek(aar,maaned,dag);
skudaar := leapyear(aar);
taeller := 0;

loop -- Kør denne loop indtil vi har nået alle de år igennem som vi skal.
exit when aar = slutaar+1;
        taeller := taeller + 1; -- Næste tupel skal have en unik taeller.
        insert into day_dim values(taeller,DayTypeA(dow),DayOfWeekA(dow),
                                   dag,maaned,aar);
        dag := dag + 1; -- Dagen skal altid tælles op.
        dow := dow + 1; -- Ugedagen skal også tælles en op.
        if dow = 8 then -- Blev den talt en for meget op?
                dow := 1; -- så nulstiller vi den da bare
        end if;

        -- Wrap the month, quarter and year??
        if (dag > 28) then -- Vi skal ikke wrappe hvis vi ikke er over
                           -- den 28. i måneden.
                if (dag = 29) and (maaned = 2) and (skudaar = 0) then
                           -- Er vi for langt?
                   dag := 1;
                   maaned := 3;
                elsif (dag = 30) and (maaned = 2) then
                           -- Er vi for langt i februar?
                   dag := 1;
                   maaned := 3;
                elsif (dag = 31) and ((maaned = 4) or (maaned = 6) or
                      (maaned = 9) or (maaned = 11)) then
                   dag := 1;
                           -- Er vi for langt i en måned med 30 dage?
                   maaned := maaned + 1;
                elsif (dag = 32) then
                     -- Eller er vi for langt i en måned med 31 dage?
                   dag := 1;
                   maaned := maaned + 1;
                   if maaned = 13 then -- Ups - nyt år!
                           maaned := 1;
                           aar := aar + 1;
```

```
                                skudaar := leapyear(aar);
                        end if;
                end if;
        end if;
end loop;
end;
```

A.3 ETL

```
drop table max_speed_med_tid;
create table max_speed_med_tid (maximum_speed number(20,10) NOT NULL,
                                road_fk NUMBER (5) NOT NULL,
                                time_of_day_fk NUMBER (2) NOT NULL,
                                day_fk NUMBER (4) NOT NULL,
                                minutes NUMBER(10));



drop table occupation_time_med_tid;
create table occupation_time_med_tid
(Occupation_TIME NUMBER (6),road_fk NUMBER (5) NOT NULL,
time_of_day_fk NUMBER (2) NOT NULL,
day_fk NUMBER (4) NOT NULL, minutes number(10));


create table complete_cycle_set (CYCLENO NUMBER (7),
                                 FK_TIL_EDGE NUMBER (2));
create table object_occupation
(starttid  NUMBER (7), sluttid NUMBER (7),
                FK_TIL_EDGE NUMBER (2));

drop table number_objects_med_tid;
create table number_objects_med_tid
(number_objects number(5), road_fk NUMBER (5),
time_of_day_fk NUMBER (2) , day_fk NUMBER (4), minutes NUMBER (10) );


drop table traffic_flow_med_tid; --Vi starter med at droppe tabellen.
create table traffic_flow_med_tid
(traffic_flow number(4), ROAD_FK NUMBER (5), TIME_OF_DAY_FK  NUMBER (2),
```

```
DAY_FK NUMBER (4), minutes NUMBER(10));


/* PROCEDURE:  calc_maximum_speed
*
*  Denne procedure gemmer beregner maximum speed værdierne
*   og gemmer dem sammen med de tilsvarende
*  fremmednøgler fra dimensionerne.
*
*  INPUT: minus1020
*         tabellen, som indeholder alle alle observationerne
*         og cykel nummer og edgeid for den kant
*         observationen er foretaget.
*
*         ARGUMENT: time_offset, hvis observationerne
*                   ikke skal mappes ind til time 0 i DW
*         ARGUMENT: day_ofset, hvis observationerne ikke
*                   skal mappes ind til dag 0 i DW
*
*         DE BLIVER ENDNU IKKE BRUGT!!!!
*
*  OUTPUT: max_speed_med_tid, som indeholder maximum speed per munit
*          med referencer til time, dag og segment i
*          dimensionstabellen.
*
*/;

create or replace procedure calc_maximum_speed
                   (time_offset IN number, day_ofset IN number)AS

eval_period number(4) := 60;
begin

--Vi starter med at droppe tabellen.
execute immediate 'drop table max_speed_med_tid';

execute immediate 'create table max_speed_med_tid
(maximum_speed number(20,10) NOT NULL,
 road_fk NUMBER (5)
 NOT NULL,time_of_day_fk
 NUMBER (2) NOT
 NULL, day_fk NUMBER (4) NOT NULL, minutes NUMBER(10))';

commit;
```

```
-- INDSÆT attributterne! Ingen index eller nøgler!
insert into max_speed_med_tid ( --Her indsætter vi tuplerne
select t1.max_speed, t1.fk_til_edge, tod.pk, dd.pk, minutes
from (select max(speed) as max_speed, floor(cycleno/eval_period)
       as minutes, fk_til_edge
      from minus1020

      group by floor(cycleno/eval_period),fk_til_edge) t1,
      time_of_day_dim tod,
      day_dim dd
where floor(t1.minutes/60)+7 -2*floor(t1.minutes/120) = tod.hec_hour
      and floor(t1.minutes/120)+1=dd.pk);
--where mod(floor(t1.minutes/60),24) = tod.hec_hour and
           floor(t1.minutes/1440)+1=dd.pk);
commit;
execute immediate 'create bitmap index vej_maxsp on
                       max_speed_med_tid(road_fk)';
execute immediate 'create bitmap index dagtid_maxsp on
                       max_speed_med_tid(time_of_day_fk)';
execute immediate 'create bitmap index dage_maxsp on
                       max_speed_med_tid(day_fk)';
commit;

insert into all_buffer_table (max_speed_sum, road_segments_fk,
                                  time_of_day_fk , day_fk )(
select sum(maximum_speed),  ROAD_FK, TIME_OF_DAY_FK,  DAY_FK
from max_speed_med_tid
group by   ROAD_FK, TIME_OF_DAY_FK,  DAY_FK );

end;

/*************************************************************
***                                                        ****
***       BEREGN OCCUPATION TIME                           ****
***                                                        ****
*************************************************************/




create or replace procedure calc_occupation_time (time_offset IN
number, day_ofset IN number)AS
eval_period number(4) := 60;
begin
```

```
execute immediate 'drop table complete_cycle_set';
execute immediate 'create table complete_cycle_set as
        (select cycleno, fk_til_edge from minus1020
         where mod(cycleno,60)= 59 group by cycleno, fk_til_edge )';

execute immediate 'alter table complete_cycle_set modify
                                    (cycleno NOT NULL)';
execute immediate 'alter table complete_cycle_set modify
                                (FK_TIL_EDGE NOT NULL)';
commit;

execute immediate 'create bitmap index cc_edge on
                            complete_cycle_set(FK_TIL_EDGE)';
execute immediate 'create index  cc_cycle on
                            complete_cycle_set(CYCLENO)';
commit;

/* OBJECT_TID: hvor lang tid hvert object har brugt på hver vej
                         den har kørt i løbet af simuleringen.
*/
execute immediate 'drop table object_occupation';
commit;
execute immediate 'create table object_occupation as (
        SELECT min(cycleno) as starttid, max(cycleno) as sluttid,
                            fk_til_edge
        FROM minus1020

        GROUP BY fk_til_edge, objectid, edgenumber,
                            floor(cycleno/7200))';

execute immediate 'alter table object_occupation modify
                             (fk_til_edge not null)';
execute immediate 'create bitmap index object_occupation on
                            object_occupation(FK_TIL_EDGE)';
commit;

execute immediate 'drop table occupation_time_med_tid';
execute immediate 'create table occupation_time_med_tid as (
        select t1.occupation_time, t1.fk_til_edge as road_fk,
               tod.pk as time_of_day_fk, dd.pk as day_fk,
               floor(minutes/60)+1 as minutes
        from ( SELECT sum (cycleno-starttid) as occupation_time,
                      o1.fk_til_edge, cycleno as minutes
```

```
               from object_occupation o1, complete_cycle_set o2
               WHERE (o2.cycleno between o1.starttid AND o1.sluttid)
                     AND (o1.fk_til_edge = o2.fk_til_edge)
               GROUP BY o2.cycleno, o1.fk_til_edge) t1,
               time_of_day_dim tod,
               day_dim dd
        where floor(t1.minutes/3600)+7 -2*floor(t1.minutes/7200) =
              tod.hec_hour and floor(t1.minutes/7200)+1=dd.

/* læg mærke at der divideres med er 3600.
** Det er fordi at selvom der kun er en cykel per
** minut, har cyklen stadig sekundværdien af den
** sidste cykel i det minut. Se compleate_cycle_set
**/

execute immediate 'alter table occupation_time_med_tid modify
                                        (road_fk NOT NULL)';
execute immediate 'alter table occupation_time_med_tid modify
                                   (time_of_day_fk NOT NULL)';
execute immediate 'alter table occupation_time_med_tid modify
                                         (day_fk NOT NULL)';
commit;

execute immediate 'create bitmap index vej_otime on
                           occupation_time_med_tid (road_fk)';
execute immediate 'create bitmap index
                   dagtid_otime on occupation_time_med_tid
                  (time_of_day_fk)';
execute immediate 'create bitmap index dage_otime on
                             occupation_time_med_tid (day_fk)';

--Indsæt i all_buffer_table. ;
/*næsten klar til at smide i datawarehouset ;)
*/
insert into all_buffer_table (occupation_time_sum, road_segments_fk ,
                              time_of_day_fk , day_fk )(
                              select sum(occupation_time),
                              ROAD_FK, TIME_OF_DAY_FK,  DAY_FK
                              from occupation_time_med_tid
                              group by   ROAD_FK, TIME_OF_DAY_FK,
                              DAY_FK );


end;
```

```
/*****************************************************
***                                                ****
***                                                ****
***            BEREGN NUMBER OF CARS               ****
***                                                ****
*****************************************************/


create or replace procedure calc_number_objects
         (time_offset IN number, day_ofset IN number)AS
eval_period number(4) := 60;
begin


/*
* beregn antal biler pr vej og tid
* KUN BEREGNE ANTALLET FOR HVER 60 sekund
* Vi registrerer alt på minut niveau også altal objekter selvom
* vi kan registrere det hvert cycle. Grunden er at vi
* vi vil gemme det i samme tablel som de andre "measures",
* og derfor skal det være på samme granularitet.
* Man kunne også have valgt at lave et gennemsnit over alle
* antal for et helt minut men det gør vi ikke da det er
* begrænset hvor mange variationer der kan være i løbet af et minut.
*/


execute immediate 'drop table number_objects_med_tid';
commit;
execute immediate 'create table number_objects_med_tid
                  (number_objects number(5), road_fk NUMBER (5),
                   time_of_day_fk NUMBER (2) , day_fk NUMBER (4),
                   minutes NUMBER (10) )';
commit;

insert into number_objects_med_tid ( --Her indsætter vi tuplerne
select t1.no_cars, t1.fk_til_edge, tod.pk, dd.pk, minutes
from (select count(objectid) as no_cars,  cycleno/eval_period as
                                          minutes, fk_til_edge
      from minus1020
      where mod(cycleno,eval_period)=59
      group by cycleno, fk_til_edge) t1,
      time_of_day_dim tod,
```

```
      day_dim dd
where floor(t1.minutes/60)+7 -2*floor(t1.minutes/120) = tod.hec_hour
and floor(t1.minutes/120)+1=dd.pk);

commit;
execute immediate 'create bitmap index vej_nocars on
                              number_objects_med_tid (road_fk)';
execute immediate
'create bitmap index dagtid_nocars on
                        number_objects_med_tid(time_of_day_fk)';
execute immediate 'create bitmap index dage_nocars on
                        number_objects_med_tid (day_fk)';
commit;

insert into all_buffer_table
(no_of_cars_sum, road_segments_fk , time_of_day_fk , day_fk )(
select sum(number_objects),  ROAD_FK, TIME_OF_DAY_FK,  DAY_FK
from number_objects_med_tid
group by   ROAD_FK, TIME_OF_DAY_FK,  DAY_FK );

end;

/*************************************************
***                                            *****
***                                            *****
***         BEREGRN TRAFFIC FLOW               *****
***                                            *****
*************************************************/


create or replace procedure calc_traffic_flow (time_offset IN number,
                                            day_ofset IN number) AS
eval_period number(4) := 60;
begin
 --Vi starter med at droppe tabellen.
execute immediate 'drop table traffic_flow_med_tid';
execute immediate 'create table traffic_flow_med_tid
(traffic_flow number(4), ROAD_FK NUMBER (5), TIME_OF_DAY_FK  NUMBER (2),
                        DAY_FK NUMBER (4), minutes NUMBER(10))';

commit;

/*
* Her finder vi ud af hvad trafficflow er for hvert segment i løbet af
```

```
* 1 minut. Forespørgslen består af to sub-forespørgsler. Den ene
* beregner flowet i løbet af et minut. Den anden beregner antallet af
* biler på et segment ved periodens udløb. Traffik-flowet beregnes så
* ved at trække antallet af biler ved periodens udgang fra antallet af
* biler der har befundet sig på segmentet i løbet af perioden.
*
* Vi trækker 1 fra cycleno når end_flow skal bestemmes,
* det er fordi at end_flow for det første minut ville blive
* associeret med det 60 cykel som hører til det andet minut når
* interval_flow beregnes. Ved at trække 1 fra cycelno
* vil end_flow passe med det rigtige interval_flow d.v.s
*
*           Interval_flow : 0-59 cycleno => end_flow : 59 cycleno
*/

              -- INDSÆT attributterne! Ingen index eller nøgler!
insert into traffic_flow_med_tid ( --Her indsætter vi tuplerne
select t1.traffic_flow, t1.fk_til_edge, tod.pk, dd.pk, minutes
from (select (interval_flow.interval_flow - end_flow.end_flow) as
               traffic_flow, end_flow.minutes, end_flow.fk_til_edge

      from (    select count(distinct objectid) as interval_flow,
                floor(cycleno/eval_period) as minutes, fk_til_edge
                from minus1020 group by floor(cycleno/eval_period),
                                        fk_til_edge) interval_flow,

                (select count(objectid) as end_flow, fk_til_edge,
                 floor(cycleno/eval_period) as minutes --alle object
                 id'er inden for et cykel er distinct

                from minus1020
                where mod(cycleno-1,eval_period)=0 and
                cycleno <> 1-- -1 fordi det skal være fra samme interval
                group by  fk_til_edge, cycleno) end_flow
      where end_flow.fk_til_edge=interval_flow.fk_til_edge and
            end_flow.minutes=interval_flow.minutes) t1,
      time_of_day_dim tod,
      day_dim dd
where floor(t1.minutes/60)+7 -2*floor(t1.minutes/120) = tod.hec_hour
                and floor(t1.minutes/120)+1=dd.pk);
commit;
execute immediate 'create bitmap index vej_tflow on
                                traffic_flow_med_tid (road_fk)';
execute immediate 'create bitmap index dagtid_tflow on
```

```
                        traffic_flow_med_tid (time_of_day_fk)';
execute immediate 'create bitmap index dage_tflow on
                                traffic_flow_med_tid (day_fk)';

--Indsæt i all_buffer_table. ;

/*næsten klar til at smide i datawarehouset ;)
*/
insert into all_buffer_table
(traffic_flow_sum, road_segments_fk , time_of_day_fk , day_fk )(
select sum(traffic_flow),  ROAD_FK, TIME_OF_DAY_FK,  DAY_FK
from traffic_flow_med_tid
group by   ROAD_FK, TIME_OF_DAY_FK,  DAY_FK );

end;


/*****************************************
***                                    ****
***                                    ****
***     BEREGN ANTAL OBSERVATIONER     ****
***                                    ****
*****************************************/

create or replace procedure calc_number_observations
                   (time_offset IN number, day_ofset IN number)AS
begin

execute immediate 'drop table number_observations_med_tid';
execute immediate 'create table number_observations_med_tid
(number_observations number(7) NOT NULL, road_fk NUMBER (5) NOT NULL,
 time_of_day_fk NUMBER (2) NOT NULL, day_fk NUMBER (4) NOT NULL)';

--Så opretter vi den igen. Det er hurtigere end at slette alle tupler.
commit;

-- INDSÆT attributterne! Ingen index eller nøgler!
insert into number_observations_med_tid ( --Her indsætter vi tuplerne
select t1.no_obs, t1.fk_til_edge, tod.pk, dd.pk
from (select count(*) as no_obs, cycleno, fk_til_edge
      from observations
      group by cycleno,fk_til_edge) t1,
      time_of_day_dim tod,
      day_dim dd
```

```
where floor(t1.cycleno/3600)+7 -2*floor(t1.cycleno/7200) =
tod.hec_hour and floor(t1.cycleno/7200)+1=dd.pk);
commit;

execute immediate 'create bitmap index vej_nobs on
                            number_observations_med_tid(road_fk)';
execute immediate 'create bitmap index dagtid_nobs on
                       number_observations_med_tid(time_of_day_fk)';
execute immediate 'create bitmap index dage_nobs on
                             number_observations_med_tid(day_fk)';


/* her læses skidtet ind i all_buffer_table
*  som vil indeholde alle measures for metoderne og
*  null værdier for de metoder det ikke relerterer.
*/

insert into all_buffer_table (no_of_obs_sum, road_segments_fk,
                                      time_of_day_fk , day_fk )(
select sum(number_observations),  ROAD_FK, TIME_OF_DAY_FK,  DAY_FK
from number_observations_med_tid noo
group by   ROAD_FK, TIME_OF_DAY_FK,  DAY_FK );

end;

/*********************************************
***                                        ****
***   INDLÆS I FACT TABELLEN               ****
***                                        ****
*********************************************/

declare
begin
calc_maximum_speed (0,0);
calc_number_objects(0,0);
calc_traffic_flow(0,0);
calc_occupation_time(0,0);
calc_number_observations(0,0);
commit;

/* Læg mærke til at alle aggregeringsfunktioner ud over count
 * ignorerer nulls,
*/
```

```
insert into fact_table (TIME_OF_DAY_FK, DAY_FK, ROAD_SEGMENTS_FK,
                 MAX_SPEED_SUM, TRAFFIC_FLOW_SUM, NO_OF_CARS_SUM,
                          NO_OF_INTERVALS, occupation_time_sum)
(select TIME_OF_DAY_FK, DAY_FK, ROAD_SEGMENTS_FK, sum(MAX_SPEED_SUM),
sum(TRAFFIC_FLOW_SUM), sum(NO_OF_CARS_SUM), 60, sum(occupation_time_sum)
from all_buffer_table group by TIME_OF_DAY_FK, DAY_FK, ROAD_SEGMENTS_FK);

 --Vi starter med at droppe tabellen.
execute immediate 'drop table all_buffer_table';
execute immediate 'create table ALL_BUFFER_table
(time_of_day_fk numeric(2), day_fk numeric(4),
road_segments_fk numeric(5), max_speed_sum numeric(20),
traffic_flow_sum numeric(20), occupation_time_sum numeric(20),
no_of_cars_Sum numeric(20), no_of_obs_sum numeric(20),
no_of_intervals numeric(20))';

end;
```

A.4 Source Code

```
create table results(method1_detections number(10), method2_detections
number(10), method3_detections number(10), method4_detections
number(10), overlap_12 number(10), overlap_23 number(10), overlap_31
number(10), overlap_14 number(10), overlap_24 number(10), overlap_34
number(10), overlap_123 number(10), overlap_124 number(10),
overlap_234 number(10), overlap_1234 number(10));


drop table number_objects_log;
create table number_objects_log
(TIME_OF_DAY_FK NUMBER (2), DAY_FK NUMBER (4),
ROAD_SEGMENTS_FK NUMBER (5), number_objects_value Number(20,10));


drop table traffic_flow_log;
create table traffic_flow_log
(TIME_OF_DAY_FK NUMBER (2), DAY_FK NUMBER (4), ROAD_SEGMENTS_FK
NUMBER(5), traffic_flow_value Number(20,10), treshold number(10,8),
```

```
         result varchar(5));


drop table max_speed_log;
create table max_speed_log
(TIME_OF_DAY_FK NUMBER (2), DAY_FK NUMBER (4), ROAD_SEGMENTS_FK
NUMBER(5), max_speed_value Number(20,10), treshold number(10,8),
                                          result varchar(5));


drop table occupation_time_log;
create table occupation_time_log
(TIME_OF_DAY_FK NUMBER (2), DAY_FK NUMBER (4), ROAD_SEGMENTS_FK
NUMBER(5), occupation_time_value Number(20,10), treshold number(10,8),
                                           result varchar(5));


declare
begin
calc_all_realtime_values ();
collect_result ();
end;


create or replace procedure calc_all_realtime_values as
declare
begin
calc_maximum_speed(0,0);
calc_number_objects(0,0);
calc_traffic_flow(0,0);
calc_occupation_time(0,0);

execute immediate 'drop table combined_log';
execute immediate

'create table combined_log(TIME_OF_DAY_FK NUMBER (2), DAY_FK NUMBER (4),
          ROAD_SEGMENTS_FK NUMBER (5), traffic_flow_value number(20,10),
          traffic_flow_result number(1), max_speed_value number(20,10),
          max_speed_result number(1),
          occupation_time_value number(20,10),
          occupation_time_result number(1),  no_of_cars number(10),
          capacity number(2), minutes NUMBER(10))';
```

```
log_traffic_flow_method();
log_max_speed_method ();
log_occupation_time_method ();
end;

create or replace procedure collect_result as
declare
cursor combined_calculator is select  TIME_OF_DAY_FK, DAY_FK,
                ROAD_SEGMENTS_FK,
                max(traffic_flow_value) as traffic_flow_value,
                max(traffic_flow_result) as traffic_flow_result,
                max(max_speed_value) as max_speed_value,
                max(max_speed_result) as max_speed_result,
                max(occupation_time_value) as occupation_time_value,
                max(occupation_time_result) as occupation_time_result,
                max(no_of_cars) as no_of_cars,
                max(capacity) as capacity
        from combined_log
        group by TIME_OF_DAY_FK, DAY_FK, ROAD_SEGMENTS_FK, minutes;

combined_row combined_calculator%rowtype;

method1_detections number(10) :=0;
method2_detections number(10) :=0;
method3_detections number(10) :=0;
method4_detections number(10) :=0;
overlap_12 number(10) :=0;
overlap_23 number(10) :=0;
overlap_31 number(10) :=0;
overlap_14 number(10) :=0;
overlap_24 number(10) :=0;
overlap_34 number(10) :=0;
overlap_123 number(10) :=0;
overlap_124 number(10) :=0;
overlap_234 number(10) :=0;
overlap_1234 number(10) :=0;

combined_value number;
combined_result boolean;

th1 number := 5/4;
th2 number := 4/3;
th3 number := 3/2;
combined_treshold number := th1 + th2 + th3;
```

```
true_val number(1) :=1;
false_val number(1) :=0;

begin
open combined_calculator;
loop

combined_result := false;
fetch combined_calculator into combined_row;
exit when combined_calculator%NOTFOUND;
combined_value := combined_row.traffic_flow_value +
combined_row.max_speed_value + combined_row.occupation_time_value;

if combined_value > combined_treshold and
combined_row.no_of_cars*0.2 > combined_row.capacity then

method4_detections := method4_detections +1;
combined_result := true;
end if;

if combined_row.max_speed_result = true_val then
method1_detections := method1_detections +1;
end if;

if combined_row.traffic_flow_result = true_val then
method2_detections := method2_detections +1;
end if;

if combined_row.occupation_time_result = true_val then
method3_detections := method3_detections +1;
end if;

if combined_row.max_speed_result = true_val and
combined_row.traffic_flow_result = true_val then
overlap_12 := overlap_12 +1;
end if;

if combined_row.traffic_flow_result = true_val and
combined_row.occupation_time_result = true_val then
overlap_23 := overlap_23 +1;
end if;

if combined_row.max_speed_result = true_val and
```

```
combined_row.occupation_time_result = true_val then
overlap_31 := overlap_31 +1;
end if;

if combined_row.max_speed_result = true_val and
combined_result = true then
overlap_14 :=overlap_14 +1;
end if;

if combined_row.traffic_flow_result = true_val and
combined_result = true then
overlap_24 :=overlap_24 +1;
end if;

if combined_row.occupation_time_result = true_val and
combined_result = true then
overlap_34 :=overlap_34 +1;
end if;

if combined_row.max_speed_result = true_val and
combined_row.traffic_flow_result = true_val and
combined_row.occupation_time_result = true_val then
overlap_123 := overlap_123 +1;
end if;

if combined_row.max_speed_result = true_val and
combined_row.traffic_flow_result = true_val and combined_result = true
then overlap_124 :=overlap_124 +1;
end if;

if combined_row.traffic_flow_result = true_val and
combined_row.occupation_time_result = true_val and combined_result =
true then overlap_234 := overlap_234 +1;
end if;

if combined_row.max_speed_result = true_val and
combined_row.traffic_flow_result = true_val and
combined_row.occupation_time_result = true_val and combined_result =
true then overlap_1234 := overlap_1234 +1;
end if;

end loop;
close combined_calculator;
```

```
insert into results values
(method1_detections, method2_detections, method3_detections,
method4_detections, overlap_12, overlap_23, overlap_31, overlap_14,
overlap_24, overlap_34, overlap_123, overlap_124, overlap_234,
overlap_1234, 'minus10');
commit;
end;

/*****************************************************************
**                                                              ***
**              NUMBER OF CARS METODEN                          ***
**                                                              ***
*****************************************************************/

/*
* Denne procedure har til formål at beregne om der har været
* trafikprop på en vej på et bestemt tidspunkt.
*
* Metoden er TRAFFIC FLOW metoden.
*
* INPUT: proceduren bruger to tabeller
*        Traffic_flow_med_tid: indeholder værdierne der skal testes,
*        d.v.s. traffic flow for en vej over 1 minut
*        fact_table: datawarehouse tabellen som indeholder
*        de hitoriske aggregerede værdier på time niveau
*
* output: Resultatet skrives til en log-tabel som hedder
*         number_objects_log. For hver tupel i number_objects_med_tid
*         skrives en
*         tupel i number_objects_log som indeholder fremmednøgler til
*         dimensionen som beskriver hvor/hvornår det er og
*         numerisk væri som angiver sværhedsgraden af trafik proppen.
*/

create or replace procedure log_number_objects_method as
--declare

cursor number_objects_values is

select realtime.road_fk, realtime.day_fk, realtime.time_of_day_fk,
realtime.number_objects as realtime_number_objects,
history.number_objects_sum/history.NO_OF_INTERVALS as
history_number_objects
```

```
from number_objects_med_tid realtime, (
        select sum(no_of_cars_sum) as number_objects_sum,
        sum(NO_OF_INTERVALS) as NO_OF_INTERVALS, road_segments_fk,
        time_of_day_fk
        from fact_table
        group by road_segments_fk, time_of_day_fk) history
where realtime.road_FK = history.road_segments_fk and
realtime.time_OF_DAY_FK = history.time_of_day_fk;

number_objects_row number_objects_values%rowtype;

tf_value number(20,10);
result number(1);
begin

 --Vi starter med at droppe tabellen.
execute immediate 'drop table number_objects_log';
commit;
execute immediate
        'create table number_objects_log
        (TIME_OF_DAY_FK NUMBER (2), DAY_FK NUMBER (4),
         ROAD_SEGMENTS_FK NUMBER (5), number_objects_value Number(20,10))';
commit;
        open number_objects_values;
        loop
                fetch number_objects_values into number_objects_row;
                exit when number_objects_values%notfound;
                tf_value := number_objects_row.realtime_number_objects/
                            number_objects_row.history_number_objects;
                insert into number_objects_log values
                            (number_objects_row.time_of_day_fk,
                             number_objects_row.day_fk,
                             number_objects_row.road_fk, tf_value);
        end loop;
        close number_objects_values;
end;
/* Læg mærke til at alle aggregeringsfunktioner ud over count
* ignorerer nulls,
*/

/*************************************************************
**                                                         ***
**             TRAFFIC FLOW METODEN                        ***
**                                                         ***
```

```
************************************************************/

/*
* Denne procedure har til formål at beregne om der har været
* trafikprop på en vej på et bestemt tidspunkt.
*
* Metoden er TRAFFIC FLOW metoden.
*
* INPUT: proceduren bruger to tabeller
*        Traffic_flow_med_tid: indeholder værdierne der skal
*        testes, d.v.s. traffic flow for en vej over 1 minut
*        fact_table: datawarehouse tabellen som indeholder de
*        hitoriske aggregerede værdier på time niveau
*
* output: Resultatet skrives til en log-tabel som hedder
*         traffic_flow_log. For hver tupel i traffic_flow_med_tid
*         skrives en
*         tupel i traffic_flow_log som indeholder fremmednøgler til
*         dimensionen som beskriver hvor/hvornår det er og
*         numerisk væri som angiver sværhedsgraden af trafik proppen.
*/

create or replace procedure log_traffic_flow_method as
--declare

cursor traffic_flow_values is

select realtime.road_fk, realtime.day_fk, realtime.time_of_day_fk,
       realtime.traffic_flow as realtime_traffic_flow,
       history_traffic_flow, number_OBJECTS as no_of_cars,
       capacity, avg_no_of_cars, realtime.minutes

from   traffic_flow_med_tid realtime, number_objects_med_tid,
              ROAD_SEGMENTS_DIM, (
              select sum(traffic_flow_sum)/sum(NO_OF_INTERVALS) as
                     history_traffic_flow,
                     sum(NO_OF_CARS_SUM)/sum(NO_OF_INTERVALS) as
                     avg_no_of_cars, road_segments_fk, time_of_day_fk
              from fact_table
              group by road_segments_fk, time_of_day_fk) history

where realtime.road_FK = history.road_segments_fk and
realtime.time_OF_DAY_FK = history.time_of_day_fk and
realtime.road_FK = number_objects_med_tid.road_fk and
```

```
realtime.time_OF_DAY_FK = number_objects_med_tid.time_of_day_fk
and realtime.day_fk = number_objects_med_tid.day_fk and
realtime.minutes = number_objects_med_tid.minutes and
ROAD_SEGMENTS_DIM.pk = realtime.road_FK;

traffic_flow_row traffic_flow_values%rowtype;

treshold number(10,8) := 4/3;
tf_value number(20,10);
result number(1);

true_val number(1) :=1;
false_val number(1) :=0;
begin

 --Vi starter med at droppe tabellen.
execute immediate 'drop table traffic_flow_log';
commit;
execute immediate
        'create table traffic_flow_log
                (TIME_OF_DAY_FK NUMBER (2), DAY_FK NUMBER (4),
                 ROAD_SEGMENTS_FK NUMBER (5), traffic_flow_value
                 Number(20,10),
                 treshold number(10,8), result varchar(5))';

open traffic_flow_values;
    loop
        fetch traffic_flow_values into traffic_flow_row;
        exit when traffic_flow_values%notfound;
        if traffic_flow_row.realtime_traffic_flow = 0 then
                tf_value := 0;
        else
                tf_value :=
                      traffic_flow_row.history_traffic_flow/
                      traffic_flow_row.realtime_traffic_flow;
        end if;
        if  tf_value >= treshold
            and traffic_flow_row.capacity*0.2 >
                traffic_flow_row.no_of_cars
            and traffic_flow_row.no_of_cars /
                traffic_flow_row.avg_no_of_cars
                  >0.9 then
                        result := true_val;
                else
```

```
                        result := false_val;
                end if;
                insert into combined_log
                      (TIME_OF_DAY_FK, DAY_FK, ROAD_SEGMENTS_FK,
                       traffic_flow_value, traffic_flow_result,
                minutes, capacity)
                values(traffic_flow_row.time_of_day_fk,
                       traffic_flow_row.day_fk,
                       traffic_flow_row.road_fk, tf_value, result,
                       traffic_flow_row.minutes,
                       traffic_flow_row.capacity);
        end loop;
        close traffic_flow_values;
end;

/*****************************************
**                                      ***
**   MAXIMUM SPEED METODEN              ***
**                                      ***
*****************************************/

/*
* Denne procedure har til formål at beregne om der har været
* trafikprop på en vej på et bestemt tidspunkt.
*
* Metoden er TRAFFIC FLOW metoden.
*
* INPUT: proceduren bruger to tabeller
*        Traffic_flow_med_tid: indeholder værdierne der skal
*        testes, d.v.s. traffic flow for en vej over 1 minut
*        fact_table: datawarehouse tabellen som indeholder de
*        hitoriske aggregerede værdier på time niveau
*
* output: Resultatet skrives til en log-tabel som hedder max_speed_log.
*         For hver tupel i max_speed_med_tid skrives en
*         tupel i max_speed_log som indeholder fremmednøgler til
*         dimensionen som beskriver hvor/hvornår det er og
*         numerisk væri som angiver sværhedsgraden af trafik proppen.
*/

create or replace procedure log_max_speed_method as
--declare

cursor max_speed_values is
```

```
select realtime.road_fk, realtime.day_fk,
       realtime.time_of_day_fk,realtime.maximum_speed as
       realtime_max_speed, history_max_speed,
       number_OBJECTS, capacity, realtime.minutes
from max_speed_med_tid realtime, number_objects_med_tid,
             ROAD_SEGMENTS_DIM, (
        select sum(max_speed_sum)/sum(NO_OF_INTERVALS) as
               history_max_speed, road_segments_fk, time_of_day_fk
        from fact_table
        group by road_segments_fk, time_of_day_fk) history

where realtime.road_FK = history.road_segments_fk and
realtime.time_OF_DAY_FK = history.time_of_day_fk and
realtime.road_FK = number_objects_med_tid.road_fk and
realtime.time_OF_DAY_FK = number_objects_med_tid.time_of_day_fk
and realtime.day_fk = number_objects_med_tid.day_fk and
realtime.minutes = number_objects_med_tid.minutes and
ROAD_SEGMENTS_DIM.pk = realtime.road_FK;

max_speed_row max_speed_values%rowtype;

treshold number(10,8) := 5/4;
tf_value number(20,10);
result number(1);

true_val number(1) :=1;
false_val number(1) :=0;
begin

 --Vi starter med at droppe tabellen.
execute immediate 'drop table max_speed_log';
execute immediate
        'create table max_speed_log
        (TIME_OF_DAY_FK NUMBER (2), DAY_FK NUMBER (4),
         ROAD_SEGMENTS_FK NUMBER (5), max_speed_value Number(20,10),
         treshold number(10,8), result varchar(5))';

        open max_speed_values;
        loop
                fetch max_speed_values into max_speed_row;
                exit when max_speed_values%notfound;
                if max_speed_row.realtime_max_speed = 0 then
                        tf_value := 0;
```

```
                else
                        tf_value := max_speed_row.history_max_speed/
                                    max_speed_row.realtime_max_speed;
                end if;
                if tf_value >= treshold and
                   max_speed_row.capacity*0.2 >
                   max_speed_row.number_objects then
                        result := true_val;
                else
                        result := false_val;
                end if;
                insert into combined_log
                      (TIME_OF_DAY_FK, DAY_FK, ROAD_SEGMENTS_FK,
                      max_speed_value, max_speed_result,
                      no_of_cars, minutes)
                values(max_speed_row.time_of_day_fk,peed_row.day_fk,
                    max_speed_row.road_fk, tf_value, result,
                    max_speed_row.number_objects, max_speed_row.minutes);
        end loop;
        close max_speed_values;
end;

/*****************************************************************
**                                                              ***
**                 OCCUPATION TIME METODEN                      ***
**                                                              ***
*****************************************************************/

/*
* Denne procedure har til formål at beregne om der har været
* trafikprop på en vej på et bestemt tidspunkt.
*
* Metoden er TRAFFIC FLOW metoden.
*
* INPUT: proceduren bruger to tabeller
*        Traffic_flow_med_tid: indeholder værdierne der skal testes,
*        d.v.s.
*        traffic flow for en vej over 1 minut
*        fact_table: datawarehouse tabellen som
*        indeholder de hitoriske aggregerede værdier på time niveau
*
* output: Resultatet skrives til en log-tabel som hedder
*         occupation_time_log. For hver tupel i
*         occupation_time_med_tid skrives en
```

```
*         tupel i occupation_time_log som indeholder fremmednøgler til
Ø         dimensionen som beskriver hvor/hvornår det er og
*         numerisk væri som angiver sværhedsgraden af trafik proppen.
*/

create or replace procedure log_occupation_time_method as

cursor occupation_time_values is

select realtime.road_fk, realtime.day_fk, realtime.time_of_day_fk,
       realtime.occupation_time/number_objects as
       realtime_occupation_time,
       history.occupation_time_sum/history.NO_OF_cars_sum as
       history_occupation_time, number_OBJECTS, capacity,
       realtime.minutes

from occupation_time_med_tid realtime, number_objects_med_tid,
        ROAD_SEGMENTS_DIM, (
        select sum(occupation_time_sum) as occupation_time_sum,
        sum(NO_OF_cars_sum) as NO_OF_cars_sum, road_segments_fk,
        time_of_day_fk
        from fact_table
        group by road_segments_fk, time_of_day_fk) history

where realtime.road_FK = history.road_segments_fk and
realtime.time_OF_DAY_FK = history.time_of_day_fk and
realtime.road_FK = number_objects_med_tid.road_fk and
realtime.time_OF_DAY_FK = number_objects_med_tid.time_of_day_fk
and realtime.day_fk = number_objects_med_tid.day_fk and
realtime.minutes = number_objects_med_tid.minutes and
 ROAD_SEGMENTS_DIM.pk = realtime.road_FK;

occupation_time_row occupation_time_values%rowtype;

treshold number(10,8) := 3/2;
tf_value number(20,10);
result number(1);
true_val number(1) :=1;
false_val number(1) :=0;
begin


 --Vi starter med at droppe tabellen.
execute immediate 'drop table occupation_time_log';
```

```
execute immediate 'create table occupation_time_log(TIME_OF_DAY_FK
                  NUMBER (2), DAY_FK NUMBER (4),
                  ROAD_SEGMENTS_FK NUMBER (5),
                  occupation_time_value Number(20,10),
                  treshold number(10,8),
                  result varchar(5))';
commit;
        open occupation_time_values;
        loop
            fetch occupation_time_values into occupation_time_row;
            exit when occupation_time_values%notfound;
            tf_value := occupation_time_row.realtime_occupation_time/
                        occupation_time_row.history_occupation_time;
            if tf_value >= treshold and occupation_time_row.capacity
                           *0.2 > occupation_time_row.number_objects
                    then result := true_val;
            else
                    result := false_val;
            end if;
            insert into combined_log (TIME_OF_DAY_FK, DAY_FK,
                                      ROAD_SEGMENTS_FK,
                                       occupation_time_value,
                                       occupation_time_result, minutes)
             values(occupation_time_row.time_of_day_fk,
                        occupation_time_row.day_fk,
                        occupation_time_row.road_fk,
                        tf_value, result, occupation_time_row.minutes);
    end loop;
    close occupation_time_values;
end;
```

B Global Positioning System

The Global Positioning System (GPS) is a positioning system, that lets a GPS receiver determine its position on the earth within a few meters. At least 24 satellites are orbiting the world at all times and they continuously send out a signal to GPS receivers. The orbits of the GPS satellites are chosen such that a GPS receiver can receive signals from five to eight satellites if there are no obstacles between the receiver and the satellites. These obstacles could be mountains and buildings. A GPS receiver needs the signal from at least four satellites to determine three-dimensional coordinates and a time offset. Three satellites provide enough information to determine two-dimensional coordinates but it is insufficient for determining the altitude of the position.

The GPS system was funded by and controlled by the U. S. Department of Defense (DOD) and intended to be for the U.S. army only. Since the GPS system was launched, the DOD has opened up for private use, and now GPS receivers can determine their position down to within 5 meters.