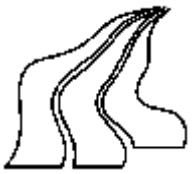
AALBORG UNIVERSITY

Department of Computer Science.

Fredrik Bajers Vej 7E, 9220 Aalborg Ø



Title:

Comparing Schemas for Spatiotemporal Data Warehouses

Project Period:
1st February 2003 to
6th June 2003

Semester:
DAT6

Author:
Luis Domínguez

Supervisor:
Torben Bach Pedersen

Copies: 6

Pages: 38

ABSTRACT

Analysis of traffic through tracking of moving objects, a location-based service, is an activity that requires storing and querying an amount of data which is typically too large to use the classical OLTP approach in a practical way.

A spatiotemporal data warehouse for moving objects is built, with different designs. Observations in position (X, Y, road segment) and time for cars, utilization of roads, and length and duration of trips are stored facts in the DW. The DW is populated from two data sources, one from real, GPS data from moving cars, and one with synthetic data from a data generator. Indexes and materialized views are built upon this DW to improve performance. A set of typical queries is issued to the DW to test this performance and compare the designs.

Luis Domínguez

Table of Contents

1 Introduction

Spatiotemporal applications are being increasingly used in various fields, from location based services in mobile communications to traffic analysis; databases are being built to deal with data that is based in space and time. The amount of this data is typically too large to use the classical OLTP (On-Line Transaction Processing) approach in a practical way. A Data Warehouse (DW) is a database used for OLAP (On-Line Analytical Processing), where in order to support analysis of data, the architecture is changed from the relational database philosophy. An example of location-based service would be traffic jam detection system in which the user is informed of the traffic conditions of the route he is taking. Tracking of moving objects is another location based service; it can be divided into two areas. One area is the past where discrete locations of the moving objects are stored, and can be interpolated in order to determine the location between the samples. Another area is the future where predictions about the positions of moving objects rely on knowledge about the objects' movements. Positioning of moving objects can be performed in one-, two-, or three-dimensional space. Positioning can additionally be constrained to subspaces where objects move along lines and curves in two or three dimensional space. An example of moving objects, in which this constraint would be a road network, is traffic.

The goal of this project is to build a spatiotemporal DW for moving objects, specifically cars, and test through a set of typical queries the effect that structures that support data warehousing and different designs have in the performance of these queries.

The design for the DW built comprehends dimensions as X, Y Position for the position of cars in the map, Time of Day and Date for time, a Road Network in which cars move, etc. A fact table keeps track of the moving objects, associating the position (X, Y, and Road Network), time, car, and other information with the respective dimensions. This fact table contains measures, speed and acceleration, to analyze the behavior of cars respecting these parameters. Other interesting information that is gathered is the utilization of each road. Having a fact table that stores the number of cars on a road or road segment in each moment would be useful to detect traffic jams. Measuring the duration and length of trips (traveling from one location to another) can provide information about travel patterns. If information about trips is maintained during several years, trends and changes in the travel patterns can be revealed. This could be information about the total distance and time traveled this year compared to last year. A fact table holds information about the length and duration of these trips, that can be used for analyzing such trends.

To populate this DW two data sources were available. The first comes from the INFATI project [INFATI, 2003], a experiment done in the Danish town of Aalborg in which 20 cars were equipped with a GPS receiver that was used to determine the position of the car every second the car was moving, for almost two months. The other source is a data generator, a program written in Java whose approach is to simulate an environment in which objects follow a given network. This data generator can be fed with the Aalborg road network to simulate as many cars as needed, for the desired time interval.

To test the performance of the spatiotemporal DW built, a set of queries was made, like “number of observations in a certain space window during the last hour” or “Most used road in the last x hours in the city center”. These queries are issued to the DBMS with four different configurations. First over the tables with no indices nor materalized views, then with indices and then with indices and materialized views. A materialized view is a pre-computed table comprising aggregated data.

This kind of tests over a spatiotemporal DW in other work is unknown to the author, as it is the comparison between designs.

The reminder of the thesis is structured as follows. Section 2 gives a brief overview of related work in data warehousing theory and spatiotemporal databases. Section 3 describes the different data sources that have been used to populate the DW. Section 4 contains a description of the DW design used. Section 5 details the different steps involved in the setup of the Data Warehouse. Section 6 describes the set of queries used to test the DW performance. Section 7 contains the results of the experiments done with the queries. In Section 8 the conclusion is presented together with suggestions for future work in the area.

2 Related Work

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 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.

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. 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. The cumulative snapshot fact is the cumulative value of a snapshot fact.

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. 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. 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 fact-type. 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 onto (all paths from the root to a leaf in the hierarchy have equal lengths), covering (only immediate parent and child values can be related), and strict (each child in a hierarchy has only one parent) [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. The next section deals with spatiotemporal databases.

2.1 Multidimensional Design

The multidimensional model [Kimball et al., 1998, p. 27] organizes each fact-type in a cube. A cube is constructed of the dimensions of the fact-type such that each combination of dimension values creates a cell where the measures of a fact can be stored. 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 a higher level of a dimension the number of cells in that dimension will be reduced. In the theory of multidimensional models viewing the data at a higher level 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.

2.2 Star Schema Design

The star schema is an alternative to the multidimensional design and is based on the relational database design. 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. 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 Oracle Analytic Workspace [Oracle, 2003], integrated in Oracle9i R2, or Microsoft Analysis Services [Microsoft, 2003].

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.

2.3 Spatiotemporal Databases

An common use for a Data Warehouse is storing spatiotemporal information. Real objects have position and shape in space, and these and other properties can change as time passes. This results in a data warehouse with two or three dimensions for space (x, y, z dimensions), and two time dimensions: a valid-time dimension that stores the time when the fact is true in the modeled reality, and a transaction-time dimension for the time when the fact is or was current in the database. A historical database stores data with respect to valid time, while a rollback database stores data with respect to transaction time. A bitemporal database stores data with respect to both valid time and transaction time.

Indexing spatiotemporal databases effectively has different solutions [Papadias et al., 2001]. Based in the fact that many of the real-world applications use summarized data more than individual facts, the same index stores pre-aggregated data, which could substitute the entire data cube if that is the only desired information. There are typical spatial indexing solutions (R-tree and R-tree variants) and temporal indexing solutions (aggregation tree) [Papadias et al., 2001].

An R-tree [Guttman, 1984] is an height-balanced tree in which internal nodes contain couples of pointers to the lower nodes in the tree and the corresponding bounding box, while index records in leaf nodes contain bounding boxes and pointers to the actual data object.

The aggregation tree [Kline and Snodgrass, 1995] indexes constant intervals, i.e. the maximum continuous intervals where the value of the aggregation function is constant. The nodes store the difference of the aggregation value between the current and the next level of the tree; therefore, the aggregation function is computed by accumulation all the values in the path from the root to the leaves.

Then [Papadias et al., 2001] describes a framework for supporting OLAP operations over spatio-temporal data and presents data structures based on the aggregation tree integrating spatio-temporal indexing with pre-aggregation: For indexing static spatial dimensions, the aggregate R-B-tree (aRB-tree) is proposed: the regions that constitute the spatial hierarchy are stored only once and indexed by an R-tree. For each entry on the R-tree (including intermediate level entries), there is a pointer to a B-tree which stores historical aggregated data about the entry. For indexing dynamic spatial dimensions, in which regions can appear, disappear or change their extent over time, the aggregate historical RB-tree (aHRB-tree) and the 3DRB-tree are proposed.

2.4 Design of a Spatiotemporal Data Warehouse

A Data Warehouse for spatiotemporal data has already been designed [Elliasen, Kjær and Urban, 2002],

and the design described in section 4 is based upon this work.

The design also has dimensions X, Y Position for the position of cars in the map, Time of Day and Date for time, and a Road Network in which cars move. In Road Network, the finest granularity is at Road Code. This implies that a road with more than one speed limit is simplified to having just one. In our design this is solved by storing information about road segments. A road is divided into segments when there is a change in the speed limit in a section, or a intersection with another road occurs. A fact table keeps track of the moving objects, associating the position in X, Y, and Road Network, time, car, and other information with the respective dimensions. This fact table contains measures, speed and acceleration, to analyze the behavior of cars respecting these parameters. Other fact table presented is Road Utilization. It stores the number of cars on a road in each moment. Trip Fact, another fact table, measures the duration and length of trips (traveling from one location to another).

3 Data Sources

The project documented in this report uses data gathered from two data sources which will be detailed below.

3.1 Data from the INFATI project

During the spring of 2000 the "Urban and Transport Planning Division" under "Department of Development and Planning" at Aalborg University started a project about cars and their speed called INFATI [INFATI, 2003]. The purpose of the project was to investigate car-drivers' response to alerts, when the speed of the car exceeded the speed limit on the road. The experiment involved 20 cars and 20 drivers, and the area covered by the experiment was the Aalborg Municipality, which is the town of Aalborg and its surrounding area. A description of the experiment and the results can be found in [Nielsen and Boroch, 2001] and [Madsen and Madsen, 2001].

For almost two months every movement of each of the 20 cars was registered using the Global Positioning System (GPS). The positions were stored in the Universal Transverse Mercator (UTM) format. The UTM system [USGS, 2001] applies the Transverse Mercator projection (cylindrical projection of the world) to map the world, and measure in meters east and north from two perpendicular reference baselines. Each car had a GPS receiver and a small computer. The GPS receiver was used to determine the position of the car every second the car was moving and the computer recorded these positions. No observations were made when the car was parked, since the computer was then turned off.

The GPS has an uncertainty of measurement and thus, the measured position of a car need not be on the road the car is driving on. However, the position on the road is the interesting observation, not a position five meters out in the field, and, since the car is assumed to be driving on a road, the measured position of the car is mapped onto the road that the car is assumed to be driving on. If the car is not driving on a road but on a field or a parking lot, the position of the car is still mapped onto a road.

3.1.1 Description of the Data

The observations recorded during the experiment were saved in four different files. These files, together with two other files, one describing the roads in the Aalborg Municipality and another giving the speed limit on the roads, will be described in the following two sections.

3.1.2 Road Information

The information about the roads in Aalborg is obtained from the administration in the Aalborg Municipality. All local roads in Denmark have a road code of seven digits attached. The first three are a code for the municipality to which the road belongs, and the last four are a unique number within the municipality for that road. The file describing the roads in Aalborg Municipality, `RoadNames.dat` contains data about all roads in the municipality. Each road in the file is associated with a road code and the name of the road. Furthermore, the postal code and postal district of the road is contained in the file, along with the starting and ending number of the houses on the road. An extract of the file is shown in Figure 4.1.

Road code	Road name	start	end	Postal code	Postal district
0068	Abelsvej	1	36	9000	Aalborg
0073	Abildgårdsvej	21	64	9400	Nørresundby
0078	Absalonsgade	1	34	9000	Aalborg
0080	A. C. Jacobsens Vej	5	34	9400	Nørresundby

Figure 3.1 Part of `RoadNames.dat`

There is another file, `Veje.dat`, which contains details about speed limit for each segment of a road. A road is divided into segments when there is a change in the speed limit in a section, or a intersection with another road occurs. Figure 4.2 shows part of the file.

X-coord	Y-coord	Vejkode	kmt	Unique
55430572	632455870	7486	50	23
55430979	632457914	7486	50	23
55431749	632458306	7486	50	23
55449649	632456885	-9	0	0
55419427	632454790	6607	50	23
55417961	632455407	6607	50	23
55416386	632455047	-9	0	0

Figure 3.2 Part of `Veje.dat`

Fields are X and Y UTM coordinates, road code, speed limit in km/h and "Unique", a not used field. When Each row is a starting point of a segment and can be an ending point for the following segment, unless road code is -9, which means only an ending point.

Observations

The actual observations from INFATI are contained in four different files. As described previously, the experiment lasted for almost two months, and during this period the position of each car was recorded each second it was driving. The four files contain data from different periods of the experiment. Figure 4.3 shows part of one of the files.

UNIQUE	BilNr	Driver	DATE	TIME	X	Y	MPX	MPY	SAT	HDOP	LIMIT	SPEED	VEJ	USERINPUT
5071200144347	5	0	071200	144347	553495	6299654	553495	6299654	0	0,0	0	65	-9	0
5071200154317	5	0	071200	154317	556384	6322126	556407	6322125	7	1,2	50	8	3640	0
5071200154318	5	0	071200	154318	556384	6322124	556407	6322122	7	1,2	50	8	3640	0
5071200154319	5	0	071200	154319	556383	6322122	556383	6322111	7	1,2	50	7	3640	0
5071200154408	5	0	071200	154408	556325	6322216	556323	6322205	7	1,2	70	27	9749	0
5071200154409	5	0	071200	154409	556319	6322219	556316	6322207	7	1,2	70	29	9749	0
5071200154410	5	0	071200	154410	556312	6322221	556309	6322209	8	0,8	70	28	9749	0

Figure 3.3 Part of an observation data file

The field unique is a concatenation of `BilNr`, `Date` and `Time`. `BilNR` is the car number, followed by `DRIVER` number, and the `DATE` and `TIME` of the observation. `X, Y` is the position of the car at the time of the observation; `MPX` and `MPY` is the position obtained by mapping the observed position onto a road. The next field, `SAT`, is the number of satellites the GPS is receiving information from. `HDOP` meaning is unknown to the author. `LIMIT` and `SPEED` are the speed limit in that segment of the road and the current speed, respectively. `VEJ` is the road code. `USERINPUT` is also unknown.

A road code of -9 is set when the car was within ten meters of an intersection the coordinates; these were not mapped onto a road because at this time it was not known on which road the car would proceed when leaving the intersection.

3.1.3 Integrity of the Data

The data for the INFATI project has been used in former projects [Elliasen, Kjær and Urban, 2002], where some filtering related to the integrity of the data was made.

The file containing information about roads and the corresponding postal code has some roads with more than one postal code. Intuitively, this should also be the case because roads can stretch through several postal districts. However, to achieve strictness, the roads with more than one postal code associated have been assigned only one of these, as the segment file doesn't contain information about postal codes. Road were assigned a town according to the postal district.

In the files containing all the observations of the experiment a more thorough integrity check is needed. Each observation in the files has been mapped onto a road and assigned the corresponding road code. However when the car is within 10 meters of an intersection, the uncertainty is too high to determine if the car is turning or continuing down the road. In these situations the observations were not mapped at the observation time, and thus added in the database. The information in the observation files are recordings of the cars' positions every second when the cars are moving. This implies that no car should have more than one observation at any second. Such duplicate information has been removed from the file. Furthermore, for some observations the value in the time-field did not describe a point in time. These observations were also deleted. A total of 669,252 observations were deleted out of 2,234,799 corresponding to 29.95%.

3.2 Data Generator

Due to the relatively small amount of data that INFATI provides, a data generator [Brinkhoff, 2002] was used, in order to provide sufficient data size.

The data generator is a program written in Java whose approach is to simulate an environment in which objects follow a given network, which is suitable for our needs. Other aspects are considered, such as maximum speeds, capacity of connections or the influence of the moving objects between each other.

3.2.1 Using the Data Generator

The program displays the network which is fed to it through a binary format file. In order to get the Aalborg network running in the generator, former work on this program [Elliasen, Kjær and Urban, 2002] included a script to convert the data from a text based format to a binary file format, which was used in this project. The script needs a collection of nodes and edges that define the network, therefore another script was made that generated this data from the “Aalborg Kommune” files, which is fed alongside the speed limit and the road codes.

The data generator has a set of parameters that can be established through the user interface. These parameters include the duration of the simulation, the initial number of moving objects, the rate of creation of new moving objects, etc. Other parameters can be set through a configuration file. These include the network data input files, visualization parameters, and maximum values for time and moving objects. A screenshot of the data generator is shown in Figure 3.4. The map appears upside down, that is, the Y coordinates are swapped. This is probably because of the java coordinate system, which places the origin at the upper left corner of the window, and growing direction is right for X but down for Y. UTM Y axis coordinates, though, grow upwards.

The configuration file provided was adjusted to the needs of our simulation. The displaying of moving objects on the screen were turned off as it slows considerably the application with a large number of moving objects. The simulator takes almost as much time as the simulation time, so this was adjusted for a maximum of 10000 seconds, and three simulations were run separately for a total time of 8 hours. To make the data more realistic, two rush hours were generated, with an initial amount of cars of 1000. One car was generated each second, which gave an average number of cars moving after the rush hour of 200. The remaining simulation had a smaller amount of initial traffic, 500 cars, but 2 new objects were generated per second, which gave an average number of cars of 400. All the simulations were run for 10000 seconds.

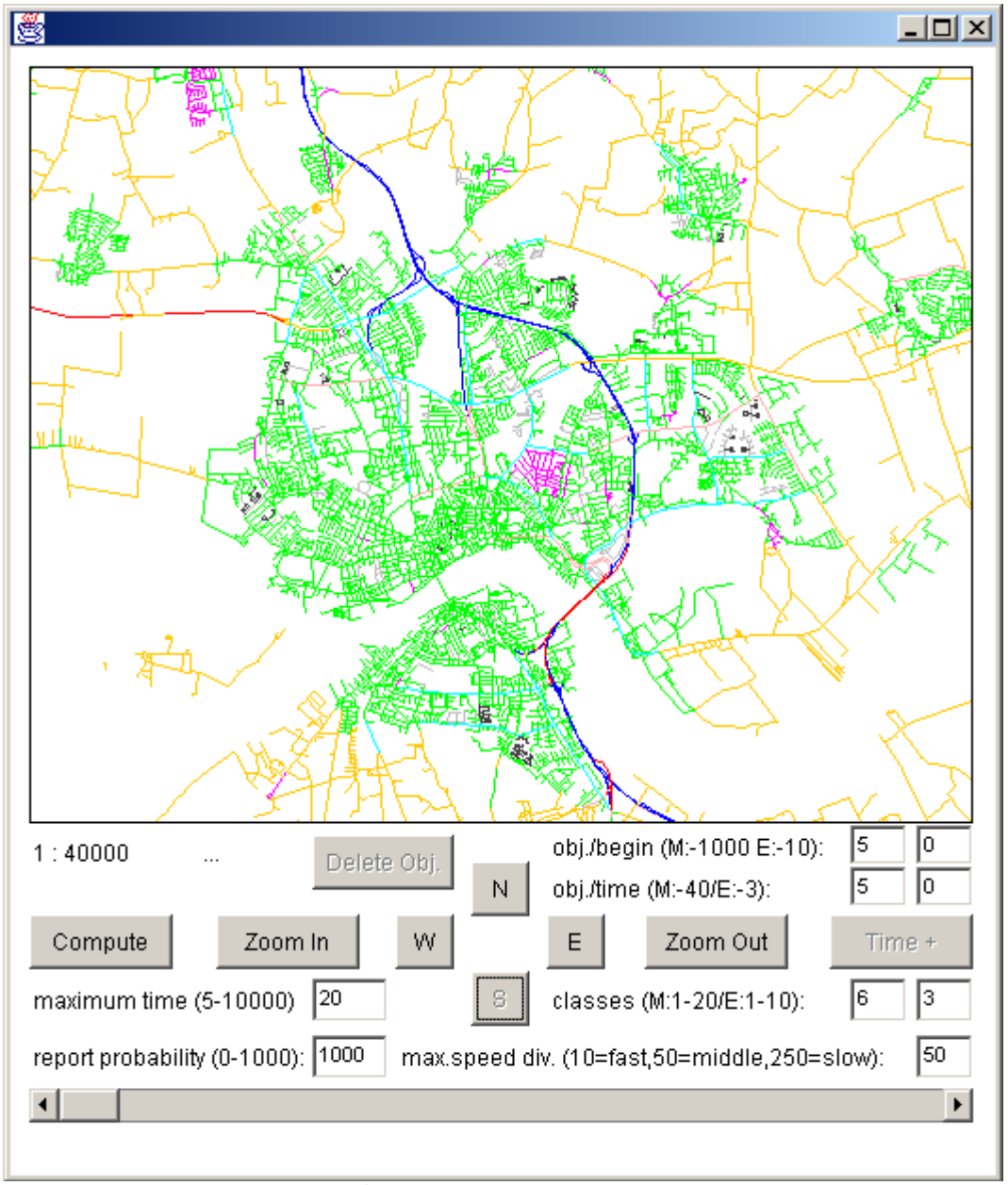


Figure 3.4 The Data Generator

3.2.2 Modifications to the Data Generator

The data generator is supplied with the source code of some of the classes which allows the modification of its behavior. The modifications were applied to the output generation, that now includes, for a given moving object:

- Identifier for the edge in which the object is
- Road code
- Calculated speed and acceleration

And the default output:

- X, Y position
- Car Identifier
- Time

Another modification was made: The default implementation assumes a set of speeds for each class of road, and then all the speed limits are adjusted by a divisor, which wouldn't match exactly Denmark's speed limits in different roads. Thus, the speed limit for each road had to be programmed and the divisor setting was ignored.

3.2.3 Drawbacks

The data generator has some minor drawbacks for this project.

Moving objects will never speed. They accelerate until their allowed maximum speed and unless an interaction with other moving object occurs, the speed will not vary for that road. This results in a slightly less realistic simulation.

There are limited options in object generation. The only way of generating new moving objects is a constant rate per timestamp, which leads to the need of more than one simulation to recreate varying traffic conditions.

4 Data Structure

This chapter describes the design of the data warehouse. It is designed as a star schema and it is based on the work of [Elliasen, Kjær and Urban, 2002]. The data warehouse is composed of three fact tables and their accompanying dimensions, which will be detailed below.

4.1 Observation

The first fact in the design of the data warehouse is the Observation fact. It is a measurement in a certain space ("X", "Y", "Road Network" dimensions) and time ("Time of Day", "Date of Year") of the speed and the acceleration of a car. The star schema for the Observation fact can be seen in Figure 4.1.

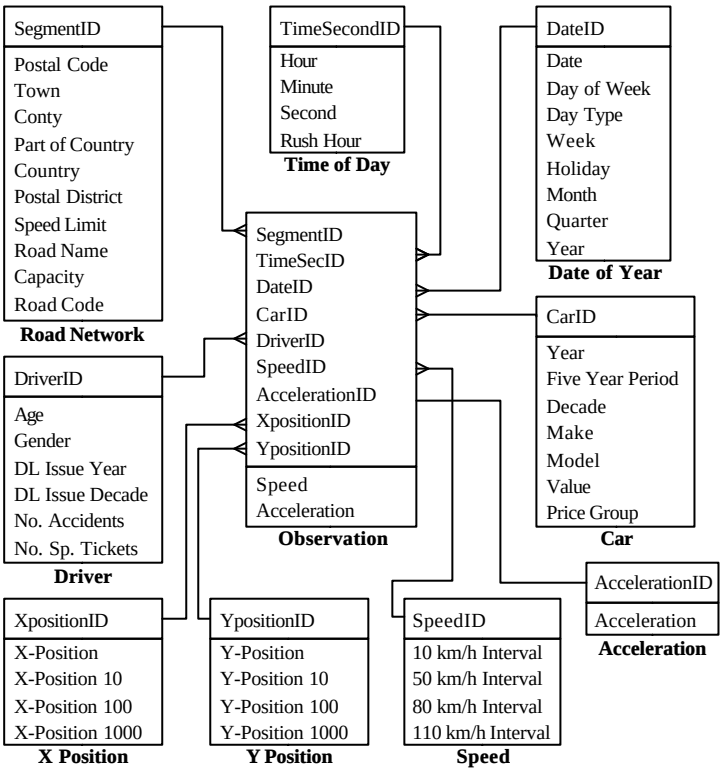


Figure 4.1 The star schema for Observation

The speed is measured in kilometers per hour, and it can be the result of a direct measurement, either from a car speedometer or a GPS, or calculated through the difference of position with the previous observation and the time between observations.

Acceleration is a purely calculated measure, analogous to speed, through previous observations. It is measured in meters per second per second.

Each observation is associated with more data, in the form of dimensions that will be detailed below.

The sum of speeds of different observations does not describe a real world phenomenon. Therefore, these two measures are physical measures that should not be visible to an end user. The "Average Speed" measure, on the other hand, is a logical measure. It is calculated by dividing the sum of the "Speed" by the sum of the "No. of Observations".

The "Average Speed" cannot be added but it can be calculated at any level by using the aggregated values of "Speed" and "No. of Observations" at the given level. The attribute "Acceleration" cannot be added over any dimension but the "Minimum" and "Maximum" values can be found at every level of the

hierarchies in the dimensions.

4.1.1 Road Network Dimension

The road network dimension is a geographical dimension that represents the road infrastructure. The infrastructure is divided into regions: postal district, city and country.

The lowest level of the “Road Network” dimension is “Segment”, which is identified by a number. A segment is a section of a road where the speed is constant, and is not intersected with other roads. The attributes that describe a segment are represented in in Figure 4.1.

The attributes can be divided into geographical, and not geographical. The latter are: “Road Name”, the name for the road that the segment belongs to; “Speed Limit” tells the legal speed limit in that segment of the road. “Capacity” describes the maximum number of cars that can pass through that road segment within a given time period.

The geographical attributes can be organized in the following hierarchy.

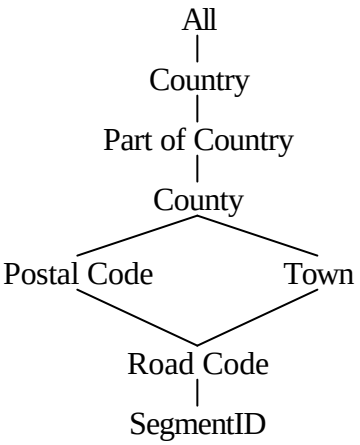


Figure 4.2 Road Network Dimension Hierarchy

The hierarchy of the “Road Network” dimension splits into a parallel hierarchy after the “Road Code” level. This split is caused by a non-uniform mapping between levels of the hierarchy, as each road has a postal code but a road does not belong to a “Town” if it is in the country side. Due to this structure the “Road Network” is non-covering and therefore the “Road Network” dimension is non-summarizable. This means that the values at one level cannot be used to aggregate the values at the next level of the hierarchy.

If the measures at the “Road Code” level were used to aggregate values at the “Town” level and only these were used for computing values at the “Country” level some values would be missing because some roads do not belong to a town. This problem can be solved by constructing an alternative value at the “Town” level to hold the aggregated values that do not belong to a town. In this way, the “Town” level holds all information needed for further aggregation at higher levels. If a dummy attribute is not added to the “Town” level, only the values aggregated at the “Postal Code” level should be used for further aggregation. The “Town” only holds aggregated values from part of the “Road Code” but “Postal Code” holds values from all measures in the “Road Code” level because each road has a postal code. The hierarchy is both strict and onto because no data have more than one parent and all levels contain values from the lowest level. The “All” level contains all the roads in the “Road Network” dimension. Each of the lower levels define a subset of this set by geographic region.

4.1.2 Time of Day Dimension

The “Time of Day” dimension represents a time of day, e.g. 5:22. Separating the time of day from the

date has two advantages: The amount of disk space required to store the information about time and date in separate dimensions is much smaller than storing all information in a single dimension. This is so because one dimension for storing both date and time would require the Cartesian product of the two dimensions. Faster execution of statistical purpose queries like “What is the average speed on Hadsundvej between 2 and 4 o'clock?” because there are fewer tuples between 2 and 4 o'clock in the “Time of Day” dimension than in a combined time and date dimension.

The disadvantage of storing time and date in separate dimensions is that the fact table referencing these dimensions requires two attributes instead of one. This, however, is only a small amount of extra storage compared to the savings. The table for the “Time of Day” dimension is illustrated in Figure 4.1.

The table contains one tuple for every second in a day, that is, 86.400 tuples. The hierarchy of the “Time of Day” dimension can be seen in Figure 4.3

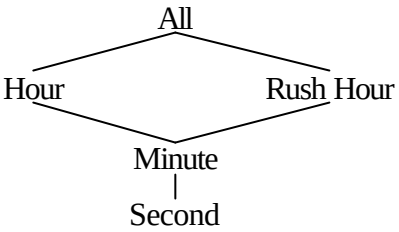


Figure 4.3 Time of Day Dimension Hierarchy

The lowest level is “Second” and the mapping between the levels is strict, onto and non-covering. The latter is due to the fact that not all minutes are contained in a rush hour period.

4.1.3 Date Dimension

The dimension “Date”, shown in Figure 4.1, represents days in a year. This dimension can only contain a subset of all possible days since there is an infinite number of days. Furthermore, for our needs only the experiment period is interesting. The “Date”, “Month”, “Quarter”, and “Year” attributes are the usual date descriptions. The “Day of Week” attribute is a textual description of the day, e.g. “Monday”. The “Day Type” is a categorization of a day into weekdays and weekends and the “Week” is a number. The attribute “Holiday” describes if the day is a holiday. It contains two parts, namely the industrial summer holiday and Christmas.

The hierarchy of the “Date” dimension, as seen in Figure 4.4, is both strict and onto, but it is not covering because some dates are not contained in a holiday. Therefore, the dimension is non-summarizable at the “Holiday” part of the hierarchy. The date is independent from the week and day of week, because a date can be uniquely specified without specifying the “Week” and “Day of Week”.

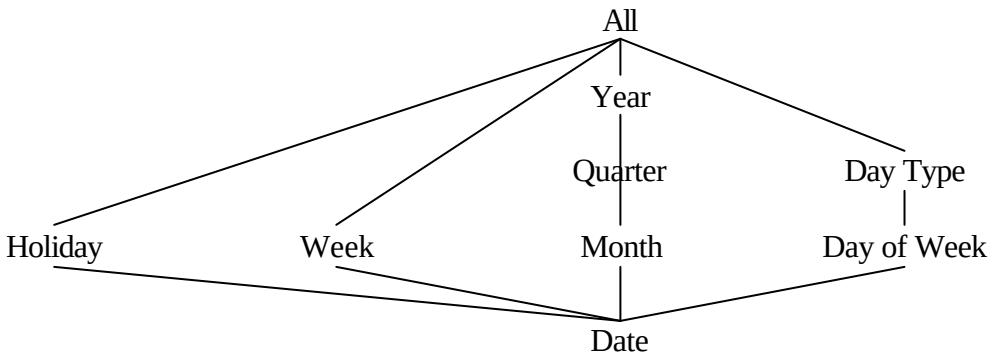


Figure 4.4 Date of Year Dimension Hierarchy

4.1.4 Driver Dimension

The “Driver” dimension, shown in Figure 4.1, represents the drivers of cars. This includes information about a driver's age, “Age” and gender, “Gender”. The dimension also contains information about the issue year of the driving license, “DL Issue Year”, the number of speeding tickets, “No. Sp. Tickets”, and the number of accidents, “No. Accidents”. The three attributes “Age”, “No. Sp. Tickets”, and “No. Accidents” can change over time and therefore the “Driver” dimension is a slowly changing dimension. In this design the change is handled by overwriting the old values with the new values.

Except from the driving license the attributes of the “Driver” dimension cannot be grouped by different granularities such as the “Road Network” and “Date” dimensions. However, the attributes are still related as shown in Figure 4.5.

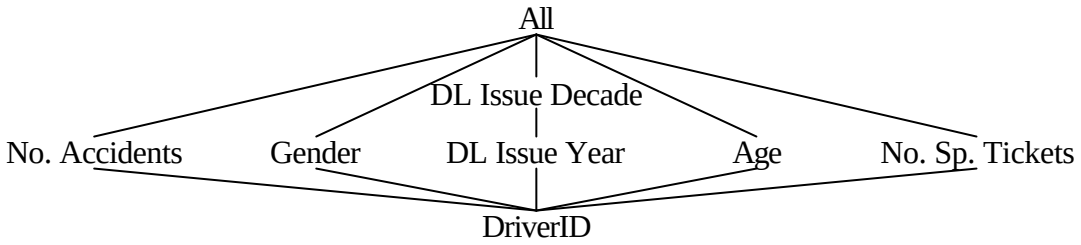


Figure 4.5 Driver Dimension Hierarchy

The hierarchy is strict, onto and covering.

4.1.5 Car Dimension

The “Car” dimension represents cars. The attributes of the “Car” dimension holds information describing the car e.g. make and model of the car. The dimension is a slowly changing dimension because three of the attributes may change over time. These are “License Number”, “Value”, and “Price Group”. As for the “Driver” dimension, the old values of these attributes are overwritten when new values are obtained. The dimension table of the “Car” is shown in Figure 4.1 (page 12).

The attributes of the “Car” dimension are grouped by different granularities. The maker can be divided into different models and the value can be gathered in price groups. The production year is gathered into a five-year grouping and a decade. The hierarchy with the different levels of the “Car” dimension is shown in Figure 4.6.

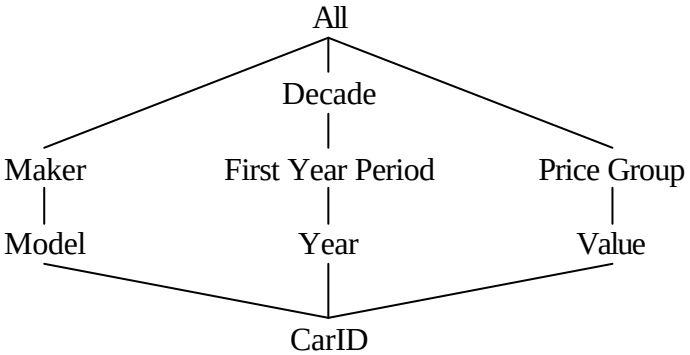


Figure 4.6 Car Dimension Hierarchy

The hierarchy is both strict, onto and covering.

4.1.6 Speed Dimension

The “Speed” dimension represents the speed of cars. It is divided into intervals and can be used to select all the observations that have a speed within a given interval. The attributes of the “Speed” can be seen in Figure 4.1.

The speed dimension hierarchy illustrated in Figure 4.7 is strict, onto and covering. The lowest level is the "10 km/h Interval" which divides the speed into intervals of 10 km/h. All other levels determine if a certain speed limit is exceeded. The three speed limits, 50 km/h, 80 km/h and 110 km/h, are the general limits in towns, main roads and motorways, respectively. For instance, the "50 km/h Interval" level determines if the speed limit for cities is exceeded. This eases the task of finding the cars that exceed the speed limit of a certain region.

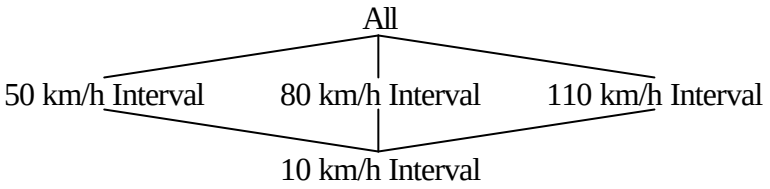


Figure 4.7 Speed Dimension Hierarchy

4.1.7 Acceleration Dimension

The "Acceleration" is useful for finding all the cars that brake sharply because of a dangerous situation. Apart from the primary key the acceleration dimension table only contains the attribute "Acceleration" as seen in Figure 4.1.

The acceleration is divided into intervals termed "Very High", "High", "Normal", "Low", and "Very Low". The "Low" and "Very Low" intervals are used to determine the dangerous situations. "Very High" acceleration is considered from $9 m/s^2$. The hierarchy of the dimension is illustrated in the Figure 4.8.

Figure 4.8 Acceleration Dimension Hierarchy

It only has the two levels "Acceleration" and "All". The hierarchy is summarizable since there are only two levels and all tuples at the "Acceleration" level map to the "All" level.

4.1.8 Position Dimensions

The "Position" dimensions represent positions in two-dimensional space. The position of a car is determined by a coordinate set "X" and "Y". However, the exact position is not important and due to the GPS the positions are only known with precision of five meters or less. Therefore, instead of storing the recorded positions of the cars, only the 5x5 rectangle that contains the observed position of the car is stored. To save storage the position is given by two dimensions instead of one. The tables are illustrated in Figure 4.1.

The "X-Position" dimension has four attributes. The value of the attribute "X-Position" is the starting point of a five meter interval. The same applies for the other three attributes "X-Position 10", "X-Position 100", and "X-Position 1000" except that these have the intervals 50, 500, and 5000 meters, respectively. The "Y-Position" dimension is constructed in the same way and by selection on both dimensions squares can be found for range queries on the observations. The "Position" dimensions have the hierarchies illustrated in Figure 4.9.

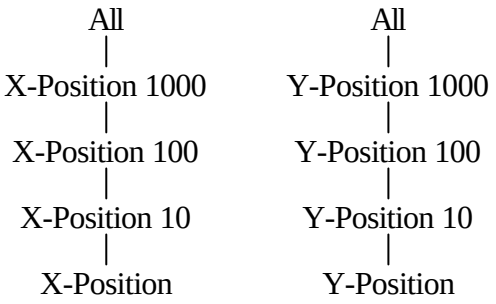


Figure 4.9 X Position and Y Position Dimension Hierarchy

4.2 Road Utilization

In this section it is described how the "Road Utilization" fact is designed in order to hold information about the number of cars.

4.2.1 Road Utilization Fact

The "Road Utilization" fact is constituted by three physical and one logical measure and provides information about several cars and their movement. The "Number of Cars" is a static count of the amount of cars over a spatial dimension, "Road Network". At the lowest level, this measure contains the number of different cars on a segment of a road at a given segment at a given moment. If the measure is added over time some cars could be counted more than once and thus, the result would be higher than the actual number of different cars on the road within the period. Therefore the measure is semi-additive. The "Road Utilization" fact also contains the measure "Traffic Flow" which is the number of cars passing through an area over time. It is calculated by counting the different cars on a segment of a road within a minute and subtracting the number of cars initially on the segment, that is, the value in "Number of Cars". This "Traffic Flow" measure is also semi-additive because if the value is added over space some cars could be counted on more than one segment within the same minute. The "Number of Cars" and "Traffic Flow" measures are necessary for finding the density of cars in space, for instance to detect a traffic jam. Another measure of the "Road Utilization" fact is "Capacity". This measure contains the number of cars that should at maximum pass through on the road segment within a period of time. The value is the same as the "Capacity" of the "Road Network" dimension. It is used to calculate the logical measure "Load". The load on a road is the traffic flow divided by the capacity. The measures of the "Road Utilization" fact is shown in Figure 4.10.

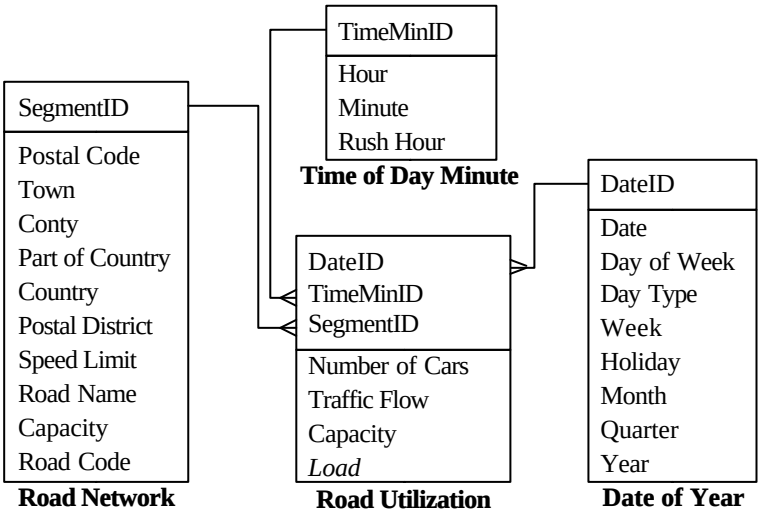


Figure 4.10 Road Utilization star schema

4.2.2 Shared Dimensions

The number of cars on a given segment on a are associated with the “Road Network” dimension and with “Date of Year” dimension, both formerly described.

4.2.3 Minute Dimension

To find information about the number of cars at a given point in time, or to determine when rush hours and congestion occur based on information about the traffic flow, the fact must be related to a dimension describing this particular point in time. For this purpose, the “Time of Day” dimension could be used. However, knowing the number of cars at each second in time might not be necessary. Maintaining this information every minute instead of every second would be sufficient to determine if the number of cars is increasing or decreasing within a given time interval. Besides, this would also require less computation and the storage needed would only be one sixtieth of the space required to store the measures every second. Therefore, the “Number of Cars” fact is related to a dimension “Minute”. In a pure multi-dimensional design the “Time of Day” dimension would be used to create a shrunken dimension [Kimball et al., 1998, p. 556]. The dimension-table of the “Minute” dimension is shown in Figure 4.10.

This dimension has the two attributes “Minute” and “Hour” that together describe a time of day. The hierarchy of the dimension is illustrated in Figure 4.11.

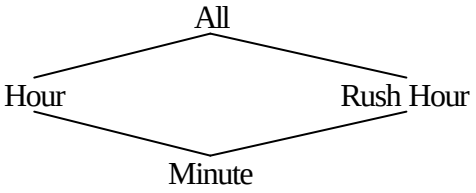


Figure 4.11 Minute Dimension Hierarchy

The level “Minute” adds up to the level “Hour” which is contained in the level “All”. However, not all minutes are part of a rush hour and therefore, the hierarchy is both strict and onto but it is non-covering and thus, it is not summarizable at the rush hour attribute.

4.2.4 Road Utilization Fact and Dimensions

The number of cars in a given area can be found by counting the cars on the roads in that area. This value is found by applying the “Sum” function. The “Number of Cars” measure is semi-additive since it cannot be added over the time-dimensions “Minute” and “Day”. By using the “Sum” aggregation function, the car flow on a road can be calculated over the different time intervals in the dimensions “Minute” and “Day”. The fact is a snapshot fact because it holds the number of cars at a given point in time and the number of cars per given time interval.

4.3 Trip

This section contains a description of a fact for obtaining information about trips.

4.3.1 Trip Fact

Measuring the duration and length of trips can provide information about travel patterns. If information about trips is maintained during several years trends and changes in the travel patterns can be revealed. This could be information about the total distance and time traveled this year compared to last year. A fact that holds information about the length and duration of trips can be used for analyzing such trends. Therefore, the “Trip” fact, is constructed. The star schema for the trip fact is shown in Figure 4.12 (page 19).

The “Trip” fact holds information about each individual trip and it has the three physical measures “Distance”, “Duration”, and “Number of Trips”. The first measure is the distance of the trip from the

starting point to the final point measured in kilometers. The second measure is the duration of the trip measured in minutes.

4.3.2 Shared Dimensions

A trip is done by a given driver on a car, so “Car” and “Driver” dimension are needed. The trip starts on a given moment so it's related the fact to the dimension “Day” and to the dimension “Minute”. The trip has an starting point and a final position, thus the fact is related to “X Position” and a “Y Position” by two entries, namely an initial position and a final position of the trip.

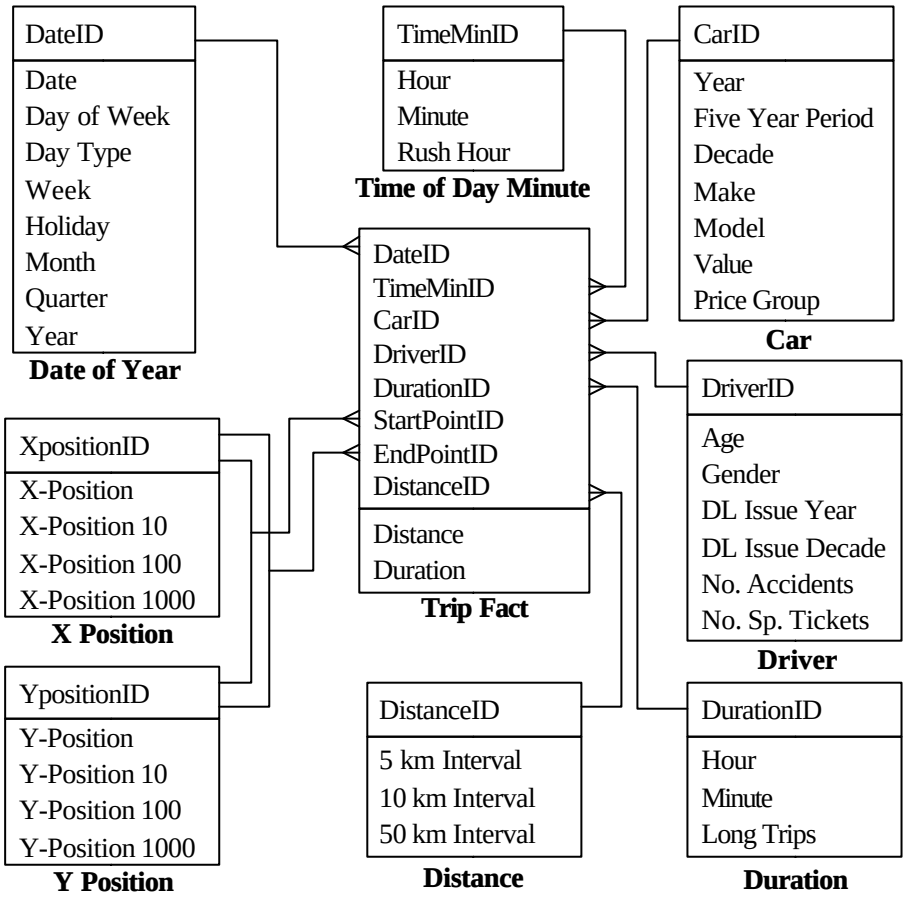


Figure 4.12 Trip Fact star schema

4.3.3 Duration Dimension

The “Duration” dimension represents the duration of a trip. This could be for determining the age of people who take trips lasting for less than 10 minutes. When selecting such trips, it is an advantage to have a dimension with intervals representing the number of minutes traveled in a trip. It has the attributes “Minute” and “Hour”. If being interested in the trips lasting for a certain period, this period is selected in the “Duration” table. The highest value of this dimension is 23 hours and 59 minutes. Theoretically, however, a trip could last for arbitrarily long time, even longer than 24 hours. This would very rarely be the case but the situation still needs to be handled. The problem is solved by the attribute “Long Trips” to which all trips lasting for 24 hours or more are associated. In the fact “Trip” the real duration of the trip can be recorded, and thus, all information is retained. The hierarchy of the dimension is shown in Figure 4.13. The lowest level is “Minute” which adds up to “Hour” and the level “Hour” adds up the super level “All”. The hierarchy is strict, onto and covering.

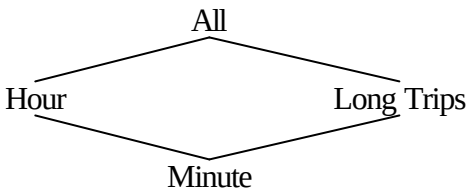


Figure 4.13 Duration Dimension Hierarchy

4.3.4 Distance Dimension

The “Distance” describes the length of a trip and enables the possibility of selecting trips of a certain length. The trip distances are measured in five, ten and fifty km intervals. The “Distance” hierarchy can be seen in Figure 4.14. The hierarchy is summarizable since all intervals at one level are contained in the level above.

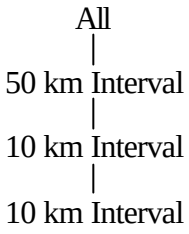


Figure 4.14 Distance Dimension Hierarchy

5 Setup Description

In previous sections we described the design of the data warehouse. In the following section, the actual implementation of the data warehouse will be detailed.

5.1 Configuration

The machine in which the database was hosted is a Pentium III – 1GHz, with 632MB of RAM, and 40GB of hard disk. The OS is Windows XP Professional Edition. The DBMS is Oracle 9i Enterprise Edition, version 9.0.1.

5.2 Creating the Database

The data from the INFATI project, the road network data and the output from the data generator were all loaded to the database with SQLLDR. The data was not processed at this stage, and every field on the file had its corresponding attribute on a database table.

The dimensions were created with PL/SQL scripts. Some of these scripts to create the database and populate the dimensions were inherited from former work [Elliasen, Kjær and Urban, 2002].

To create the Road Network dimension it was necessary a PL/SQL script and a Java program. The PL/SQL script uses the information from `RoadNames.dat` that is already loaded into the database. Information about speed limit and capacity of the road. The capacity attribute is assigned a random number between 50 and 400. If the postal code is 9000, 9210, 9220, or 9400 the town is set to "Aalborg". Otherwise the town is set to the same value as the postal district. The java program uses `veje.dat`, which provides the speed limit of each segment of a road, and generates an identifier for each of this segment, which is also used on the data generator, thus facilitating the loading of the data from that source.

Due to the lack of segment information in the observation data files, the loading of the observation fact table has to be done in several steps. The first step is a join between the raw data table and all the dimensions related to it: "Date of Year", "Road Network" , "X Position" , "Y Position", "Time of Day", "Driver", "Car" and "Speed". This eliminates some rows, as some observation data has incorrect values, such as values for minute greater than 60, for hour greater than 24, etc. The relationship with the "Road Network" is established through the containment of the observation point into the rectangle that encloses a segment. Here the GPS precision is considered by adding a 5 meters tolerance. That is, given X and Y are the observation position and startX, startY the starting point of the segment, and endX, endY the ending point of the segment, the condition for the relation is:

$$x \in (min(startX, endX) - 5, max(startX, endX) + 5) \wedge y \in (min(startY, endY) - 5, max(startY, endY) + 5)$$

This can yield more than one segment for each observation, so a further step is required. The redundancy is solved by calculating which is the nearest segment to the observation point; the distance between a segment and a point is calculated through the following algorithm [Sunday, 2001]:

```
distance(Point P, Segment P0:P1)
{
      v = P1 - P0
      w = P - P0
      if ( (c1 = w·v) <= 0 )
            return d(P, P0)
      if ( (c2 = v·v) <= c1 )
            return d(P, P1)
      b = c1 / c2
      Pb = P0 + bv
      return d(P, Pb)
}
```

Where P, P0, P1 and Pb are points in a 2 dimension space, **v,w** are vectors and b, c1 and c2 are numbers.

Thus, only the nearest segment is kept, and other segment-observation associations are just deleted. Then the values for Speed and Acceleration measures are calculated with another PL/SQL script. Data from the data generator doesn't need this process, as the generator was modified to have Segment, Acceleration and Speed as an output. A SQLLDR script is used for loading this data.

The Road Utilization Fact is created from the Observation Fact table, by counting the different cars in a minute, summing up with a group by clause.

The Trip Fact is created by processing all rows in Observation Fact, ordering them by car and then by time; then, for each car, distance and duration is accumulated until a 10 minutes gap is detected and a new trip is filed.

5.3 Technical Issues

Having amounts of data in the order of tens of millions of rows implies dealing with some problems.

In the process of creating some of the tables, it has been noticed that updating a table can be a lot more cumbersome than simply inserting the same data into another table, depending on the “undo_retention” Oracle parameter. The default value for this parameter, 3 hours, can produce an undo tablespace the size of the fact table, with the consequent overhead. The speed of the query can be several times slower with an update rather than an insert. As sometimes this is unavoidable, resetting this parameter to just a few seconds and restarting the database will free most of the space in the tablespace. Then, the space that the undo tablespace takes can be reduced with the “ALTER DATABASE DATAFILE <filename> RESIZE <new_size>” command.

6 Queries

To test the database capabilities in data warehousing, a set of queries is made. In order to have the widest range of types of work covered, we need a classification of the kind of queries that can be made.

From a multidimensional point of view, [Ho, Agrawal, Megiddo, Srikant, 1997] considers a taxonomy on the way of addressing a dimension. Frequently, queries are made that address attributes with natural semantics in ordering, such as age, time, etc. Thus, a ranged query is one that is applied over such an attribute, specifying a contiguous range in its domain. So, a classification over the way of addressing a dimension can be made:

Way of addressing a dimension

– Single value: a value on a dimension (contiguous or not)

– Ranged: continuous valued attribute on a dimension

  • Single window: a continuous range on an attribute

  • Multiple windows: more than a continuous range.

– Whole Dimension: all the values on a dimension (contiguous or not)

For selective queries, that is, the ones that address attributes without natural semantics in ordering, only single value or whole dimension have a practical application.

In this project, the most important dimensions that are centered around attributes with natural semantics in ordering are X, Y and time. This has been taken into account in the form of trying to cover the combination of addressing these dimensions and other types of dimension; this way, other dimensions are addressed selectively, independently of its potential continuous nature.

From another point of view, a classification [Tsotras, Jensen and Snodgrass, 1997] for spatiotemporal databases can be made. Generically, in a spatiotemporal database there are X, Y, Z, valid and transactional dimensions (described in section 3.3), which we will call implicit attributes, and a number of explicit attributes (keys). The most common spatiotemporal queries are selection-based. These are queries applied on a single data set (a relation), asking for all data objects (tuples) that satisfy the given query predicate. For this kind of queries a notation is proposed, in the form:

Key//X_dimension/Y_dimension/Z_dimension//Valid/Transaction

where each of the attributes can be qualified, through any of these qualifiers:

- *S* slice, single value
- *R* range, continuous time interval
- *E* element, set of intervals
- * any value
- - not applicable

Query Description

1. *The number of observations in a certain space window during the last hour*. This query is intended to be a generic query, as it provides a range for space dimensions and time dimension, getting a still useful comparative value. For this query, space has been addressed via the coordinates in the position dimensions or via the coordinates on the road network dimension. For each of these methods, different sizes of the query window will be tried. With the notation explained above, this query would be *//R/R/-//R/S, where * means that any kind of observation will be accepted, R/R/- that we only set a qualifier (a range) for X and Y dimension, and R/S that we are asking for a valid time of 1 hour (ranged), and a transaction time is now.

   i. Space window: a small part of the city center (~1% of the space)

ii. Space window: city center (~10% of the space)

iii. Space window: all the map (100% of the space)

2. *The number of cars in the city center during the last hour*. Counting the number of different cars is a similar measure as the number of observations, but it will have a different performance. Notation: *//R/R/-//R/S

3. *For a given id car, number of times it exceeded the speed limit when it was in the city center*. This query addresses a particular car, is not statistic, and will test the performance of looking for specific information and not statistic. The notation is S//R/R/-//*/S

4. *Compare high decelerations between city center and the rest of the city.* This query doesn't give a time condition, and tries to get a overview of an attribute for all observations. Notation: *//R/R/-//*/S

5. *The most used road in the last x hours in the city center.* This query addresses a smaller fact table, Road Utilization. It will also be tested over the Observation Fact table, to show the performance differences between the two designs. Notation: *//R/R/-//R/S

6. *The average length of trips in the midnight compared to the morning*. This query addresses the Trip Fact table. Unlike Query 5, making an equivalent query over Observation Fact would take as much time as building the Trip Fact table itself, therefore it will not be compared. Notation: *//*/*/-//R/S

7. *The average speed on a given road.* This query doesn't give a condition for time nor X, Y or Z dimensions. Notation: S//*/*/-//*/S

7 Experimental Results

The set of queries described in Section 6 were issued to the DBMS with four different configurations. First the tables had no indices nor materialized views, then indices were built and then materialized views. Tests were also done with materialized views and no indices. In this section we describe the experiment setup, the query results and for each of the configurations, the configuration setup process and experiment timing results.

7.1 Experiment Setup

7.1.1 Timing Setup

To measure the time each query takes, the "timing" variable was set to "on" in the SQL Plus environment. For each configuration, a command file with all the queries was run three times; therefore, timing results presented in Sections 7.3, 7.4 and 7.5 are the average time for these three experiments.

Oracle uses different caches to improve performance, and this influence is not desired in the experiments, as we want to analyze the performance gain of the different structures built. Allowing Oracle use caches could hide this information. Oracle stores the actual code it is executing along with the execution plan. It stores this in the Library Cache which is a component of the Shared Pool which is a component of the System Global Area (SGA). To delete this data from the cache, "ALTER SYSTEM FLUSH SHARED_POOL" can be issued. Secondly, Oracle stores the blocks that contain the rows of interest in a Buffer Cache that is also a component of the SGA. The buffers in the cache are organized in two lists: the dirty list and the least recently used (LRU) list. The dirty list holds dirty buffers, which contain data that has been modified but has not yet been written to disk. The LRU list holds free buffers (unmodified and available), pinned buffers (currently being accessed), and dirty buffers that have not yet been moved to the dirty list [Oracle, 2003]. There is no flush Buffer Cache command. The second option is shutting down and restarting the database before issuing each query which would flush all caches. The three options, along the "no action" option, were tested by repeating a query five times. The database was shutdown between each of the method "batch". The results can be seen in Figure 7.1.

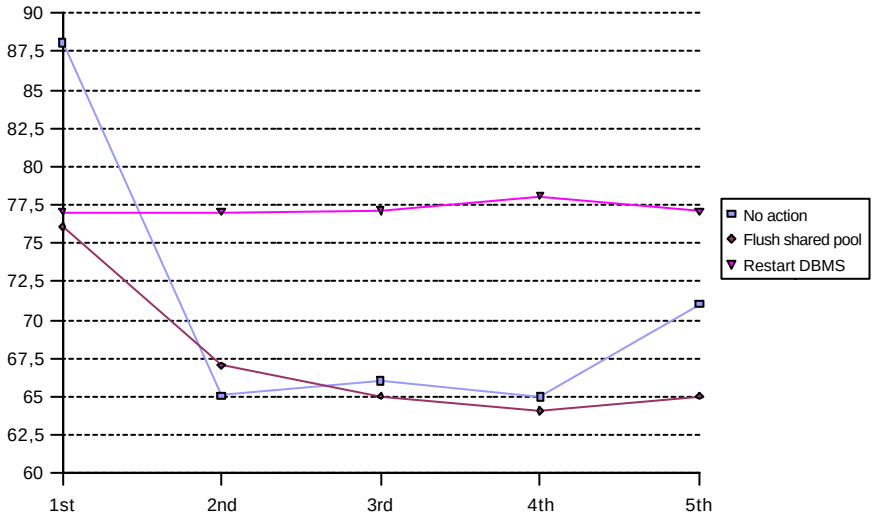


Figure 7.1 Caching Effects in Query Time

The best course of action is, then, restarting the DBMS before each query, as query time remains constant.

7.1.2 Table sizes, indices, materialized views

Timing results can't be properly analyzed without knowing the size of the tables involved. These sizes are displayed on Table 7.1. All dimension tables but Road Network and Time of Day Second are very small compared to the Observation Fact table, which takes more than one gigabyte, while the biggest dimension apart from these two holds 256KB. On the other hand, Road Utilization and Trip Fact, being aggregated derivatives from Observation Fact, are small compared to it.

Table	Rows	Disk Space (KB)
Acceleration	5	64
Car	23000	256
Date of Year	1096	64
Distance	101	64
Driver	21	64
Duration	1441	64
Road Network	27559	4096
Time of Day Minute	1440	64
Time of Day Second	86400	3072
X Position	3222	128
Y Position	7255	256
Observation Fact	12998093	1021952
Road Utilization	1102168	23552
Trip Fact	7748	512

Table 7.1 Table size

To improve the performance of the queries, indices were build over these tables. Oracle creates, by default an index for each table that has a primary key. Thus, over Observation Fact Table there is a unique index, “ind_obs_fact”, over attributes (*Time_of_day_second_ID, date_ID, X_Position_ID, Y_Position_ID, Car_ID*). This index takes almost half the size of the fact table itself. Other indices were built over this table; “ind_x_obs” over column *X_Position_ID*, “ind_y_obs” over column *Y_ Position_ID* and “ind_time_obs” over *Time_of_day_second_ID*. This indices take more than 200MB each, almost a quarter of the fact table. Bitmap indices were also built. The advantages of using bitmap indices are greatest for columns in which the ratio of the number of distinct values to the number of rows in the table is under 1% [Oracle, 2003]. This is the case for *Date_ID*, *Acceleration_ID* and *Speed_ID*, so bitmap indices (“ind_dat_obs”, “ind_acc_obs”, “ind_spd_obs”, respectively) were built for these. The size of these indices is proportional to the variety of the attribute which is built upon, and thus acceleration, which only has 4 different value options, takes only 8MB.

Two indices were built for Road Utilization Table, “ind_road_util” for columns (*Time_of_Day_Minute, Date_of_Year_ID, Segment_ID*) to ease querying by time and then by segment, and “ind_ruf_rev” with the same columns in different order: (*Segment_ID, Date_of_Year_ID, Time_of_Day_Minute_ID*), to ease querying by segment and then by time. These indices are the same size as the fact table they are indexing.

No index was built for Trip Fact, as query time is already 0,06 seconds without indices, as shown in Section 7.3. The space that these indices take is shown in Table 7.2.

Index	Disk Space (KB)
ind_acc_obs	8192
ind_dat_obs	14336
ind_spd_obs	28672
ind_time_obs	229376
ind_xpos100	128
ind_x_obs	234496
ind_ypos_100	192
ind_y_obs	251904
ind_obs_fact	431104
ind_road_util	26624
ind_ruf_rev	26624
Observation Fact	*1021952*
Road Utilization	*23552*
Trip Fact	*512*

Table 7.2 Index size

To further improve query performance we used materialized views. Materialized views are a pre-computed table comprising aggregated or joined data from fact and possibly dimension tables. It is also known as a summary or aggregate table [Oracle, 2003]. As this database is supposed to be used for statistical purposes, aggregation will be done to optimize getting statistical data.

Materialized View	Rows	Disk Space (KB)
mv_ruf_t_hour	85003	5120
mv_obs_general	47971	5120
mv_obs_gen_rn	202791	13312
sp_100_t_hour_d_day	7099	448
Observation Fact	*12998093*	*1021952*
Road Utilization	*1102168*	*23552*
Trip Fact	*7748*	*512*

Table 7.3 Materialized View size

Four materialized views were built. “mv_ruf_t_hour” only rolls up the time dimension over Road Utilization Fact from minutes to hours, keeping segment and date information; the aggregation is done with a sum function over the flow measure. “mv_obs_general”, over Observation Fact, rolls time up to

hour, and X and Y Position Dimension to XPos100 and YPos100, thus grouping objects on hundreds of meters, and keeping information for Date of Year, Acceleration Id and Speed Id. "mv_obs_gen_rn" also rolls time over Observation Fact up to hour, but keeps information for Segment ID, Date of Year, Acceleration Id and Speed. "sp_100_t_hour_d_day", over Observation Fact, also rolls up X and Y to hundreds of meters, and time to hour, but only keeps this information. The aggregation for these three materialized views is done through the count function, to have a measure of the number of observations.

The size of the materialized views is shown in table 7.3. At least one index is created for the primary key of the materialized view. This index corresponds to the primary key of the target master table (the table over which the view is built) and has the name I_SNAP$_*Materialized_View_Name;* these indices are added to the materialized view size. Anyway, the size of the materialized views is negligible compared to the size of the fact tables or the indices, less than 25MB.

7.2 Query Results

The result for the queries described on Section 6 are the following:

1. *The number of observations in a certain space window during the last hour:* **a.** Via position dimension **b.** Via road network dimension (one end of the segment in the area) **c.** via road network dimension (both ends of the segment in the area)
   i. *~1% space:*
      a. Result: 5095 observations
      b. Result: 5539 observations
      c. Result: 4308 observations
   ii. *~10% space*:
      a. Result: 68651 observations
      b. Result: 74866 observations
      c. Result: 63143 observations
   iii. *100% space:* Result: 1561456 observations
2. *The number of cars in the city center during the last hour.* Result: 741 cars
3. *For a given id car, number of times it exceeded the speed limit when it was in the city center.* Result: 0 times (id. of the car: 1)
4. *Compare high decelerations between city center and the rest of the city.* To compare the acceleration, four queries were made; for high and not high acceleration, in the city center and outside the city center. It yields a 1% less of high acceleration in the city center
   i. City center
      a. High acceleration: 749 observations
      b. Normal (not high) acceleration 565662 observations
   ii. Outside the center:
      a. High acceleration. Result:14606 observations
      b. Normal (not high) acceleration. Result: 8135580 observations
5. *Most used road segment in the last x hours in the city center.* Number of different cars in the most used road segment.
   i. Over Road Utilization Fact. Result: 23 different cars
   ii. Over Observation Fact. Result: 23 different cars

6. *The average length of trips in the midnight compared to the morning.*
   i. Midnight 21,8430016 km
   ii. Morning 13,7792208 km
7. *Average speed on a given road.* In road 317(Asgården): 63,75 km/h

7.3 Timing Results for the Bare Database

There was no modification to the setup described in Section 5. The queries were made over tables with no indices or materialized views. All tables were analyzed with the dbms_stats package, in "compute" mode, which will use all the data in the tables to compute statistics. Query numbering is the same as in Section 7.2.

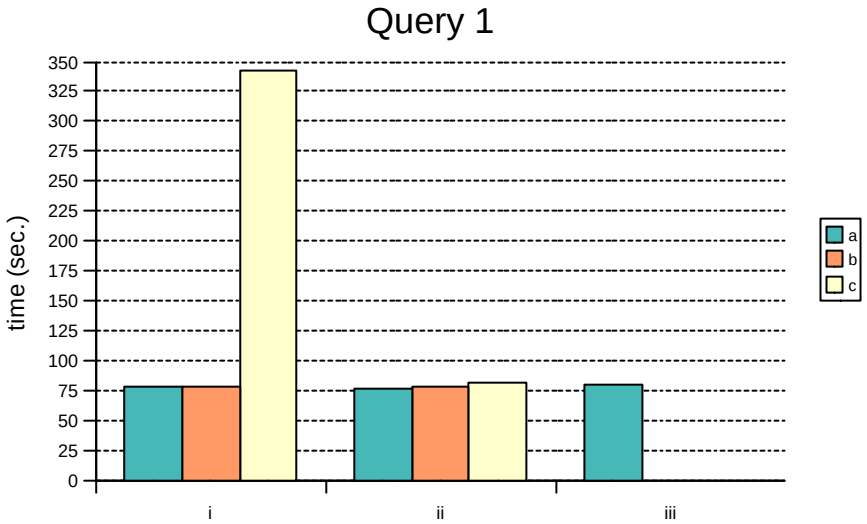


Figure 7.2 Time Results for Query 1, bare database

Figure 7.2 shows query timings for Query 1. It takes four times longer to search for the segments with both ends on the 1% area than other segments. The execution plan shows that the query optimizer does only hash joins for the other queries, and a merge join (Cartesian) and a hash join for Query 1.i.c. Execution plan for this query and Query 1.ii.c is shown in Figure 7.3.

Query 1.i.c

```
SELECT STATEMENT ()
 SORT (AGGREGATE)
  HASH JOIN ()
   HASH JOIN ()
    TABLE ACCESS (FULL)        ROAD_NETWORK
    TABLE ACCESS (FULL)        OBSERVATION_FACT
   TABLE ACCESS (FULL)         TIME_OF_DAY_SECOND
```

Query 1.ii.c

```
SELECT STATEMENT ()
 SORT (AGGREGATE)
  HASH JOIN ()
   MERGE JOIN (CARTESIAN)
    TABLE ACCESS (FULL)        ROAD_NETWORK
    BUFFER (SORT)
     TABLE ACCESS (FULL)       TIME_OF_DAY_SECOND
    TABLE ACCESS (FULL)        OBSERVATION_FACT
```

Figure 7.3 Execution Plan for Queries i and ii (1.c)

Figure 7.4 Timing Results for Queries 2, 3 and 7

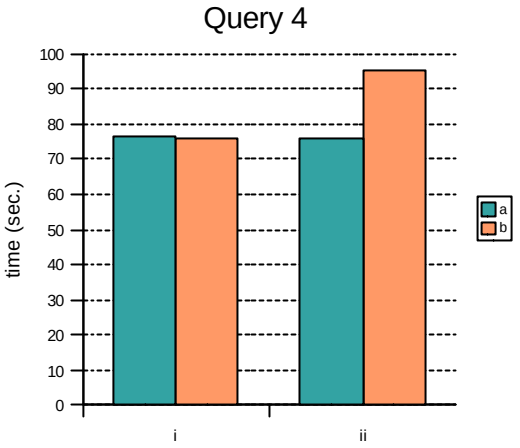


Figure 7.5 Timing Results for Query 4

Timing results for Query 2, 3 and 7 are shown in figure 7.4. Query 2 retreives the same data as Query 1.ii.a, but counting distinct cars instead of counting observations. Query 2 uses a SORT (GROUP BY) instead of a SORT (AGGREGATE) for Query 1.ii.a. The cost reflected in the execution plan is slightly lower for Query 2 (19793, 19907 for Query 1.ii.a) but setting "autotrace" to on gives a result of 115388 physical reads for Query 2 and 99518 for Query 1.ii.a.

Query 4.ii.b takes longer that Query 4.i.b, 4.i.a and 4.ii.a as it has to address traffic not in the city center and with normal acceleration, that is, execution time relates to the amount of data retrieved.

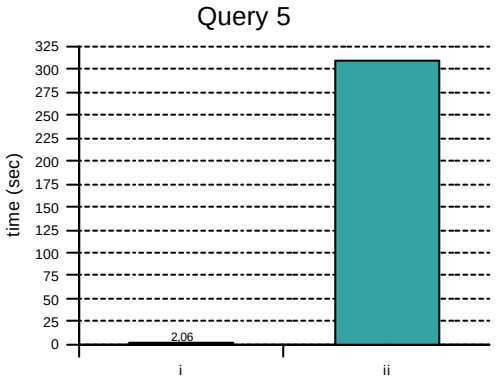


Figure 7.6 Timing Results for Query 5

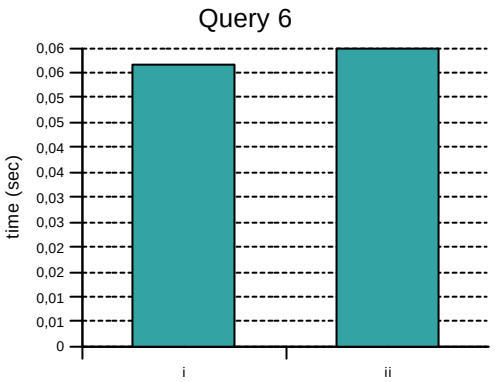


Figure 7.7 Timing Results for Query 6

In Figure 7.6 the difference between addressing Observation Fact table and Road Utilization table can be seen. As Trip Fact and Road Utilization are much smaller than Observation Fact, query time is greatly reduced. It's worth mentioning that Road Utilization is still a 1 million row table and query time is 2 seconds, without indices or materialized views. Figure 7.7 shows that Query 6 is almost instant, as it only has to retrieve 512KB of data.

7.4 Timing Results for the Database with Indexes

These are the results for the queries done over the database with the indexes described in Section 7.1.2.

Figure 7.8 shows timing results for Query1. Query 1.i.c still takes four times longer, but all queries take advantage of the indices. The execution plans show that only “ind_obs_fact” was used, however. Indexes defined for one column or bitmap indices were not used; dropping these indices would save more than 700MB

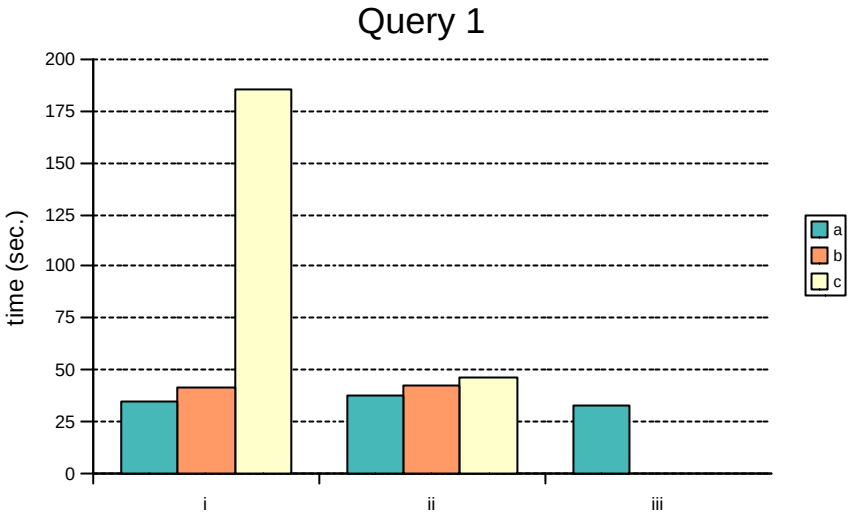


Figure 7.8 Timing Results for Query 1, database with index

Figure 7.10 shows timing results for queries 2, 3 and 7. Query 2 uses the index "ind_obs_fact", but 3, 7 and 4 still do a full scan of the Observation Fact Table. In queries 3 and 4 Time of Day, the first field of the index, is not addressed. Query 7 doesn't make a select over any field of the index. Building an index to cover these queries to improve their performance should depend on the frequency of use of this kind of queries and the variability of them, as building materialized views that cover these specific queries is much cheaper in terms of disk space and query time, as shown on Section 7.5.

Figure 7.10 Timing Results for Queries 2, 3 and 7

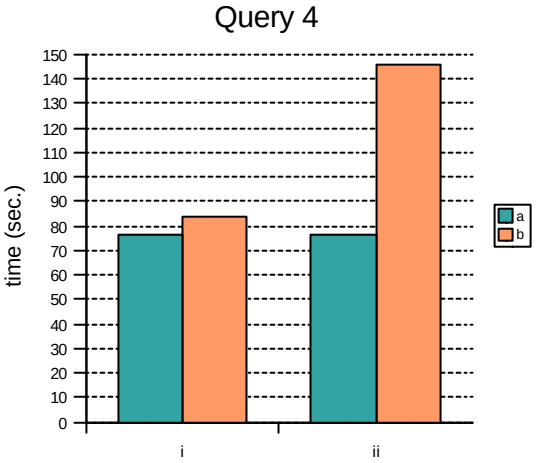


Figure 7.9 Timing Result for Query 4

Query 5.i (Figure 7.12) uses the index "ind_road_util", though the time remains the same due to the relatively small size of the table, and 5.ii uses "ind_obs_fact". Query 6 (Figure 7.11) results are unchanged, as there is no index built on that table.

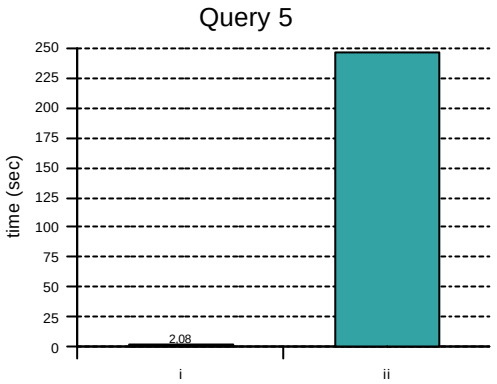


Figure 7.12 Timing Results for Query 5

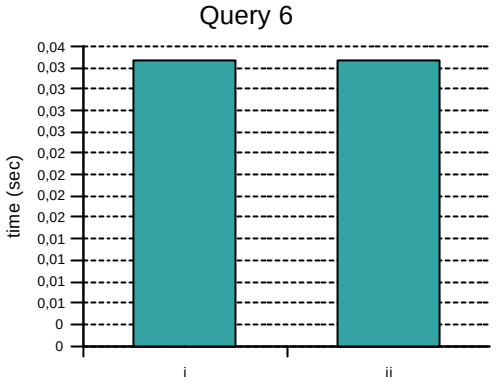


Figure 7.11 Timing Results for Query 6

7.5 Timing Results for the Database with Indexes and Materialized Views

These are the results for the queries done over the database with the materialized views described in Section 7.1.2.

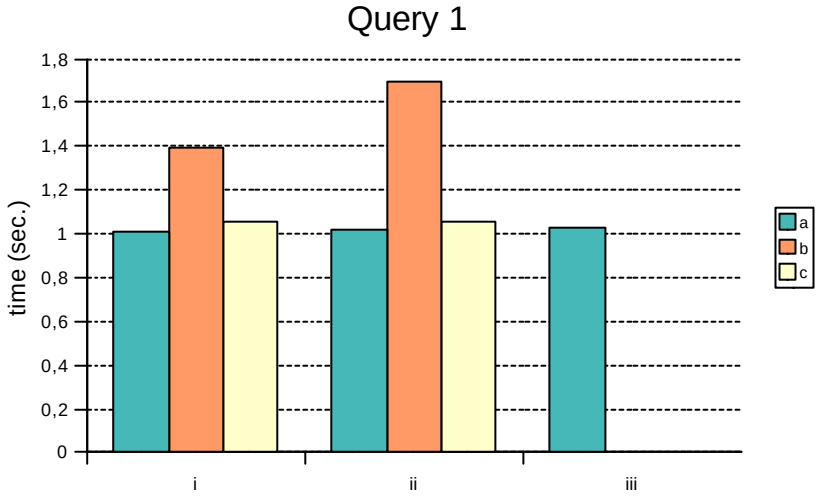


Figure 7.13 Timings for Query 1

Figure 7.13 shows timing results for Query1. Query 1.a uses the “mv_obs_general” materialized view, and 1.b uses “mv_obs_gen_rn”, though the latter takes a little bit longer, as “mv_obs_gen_rn” is double the size of “mv_obs_general”. Query 1.a had to be rewritten in order to use the view, as it expects a join between the fact tables and all dimension tables stated in the query that defines the materialized view, even if no condition is set for the attributes kept in the view. For this query “sp_100_t_hour_d_day” has enough information and is much smaller, but “mv_obs_general” is kept as it is more general.

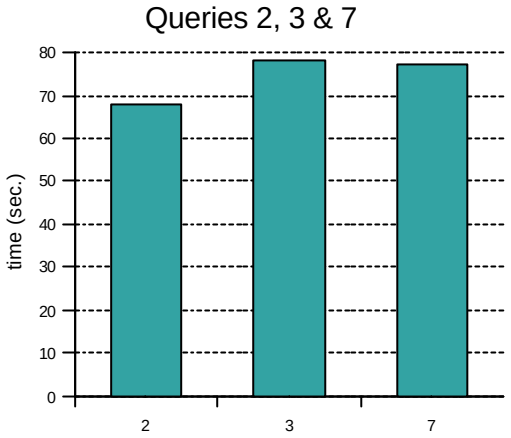


Figure 7.14 Timings for Queries 2, 3 and 7

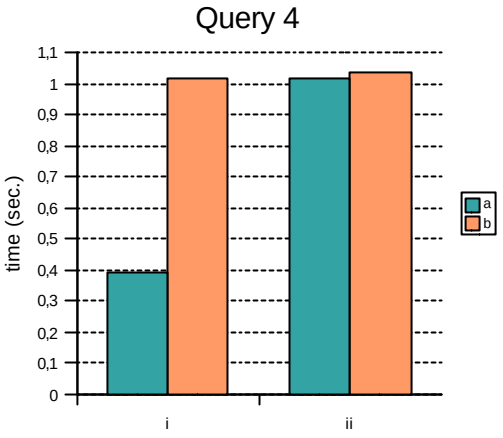


Figure 7.15 Timings for Query 4

Query 2 (Figure 7.14) can't take advantage of the materialized view, as it uses a “select count(distinct)” clause which was not taken into account in any of the materialized views built. However, it's still benefited by the index “ind_obs_fact”. The same happens with Query 3, that selects a car, which wasn't kept in any materialized view, and Query 7, that would need an aggregation over speed with the average function. Query 4 (Figure 7.15) takes advantage of “mv_obs_general”, and query time drops to a second or less.

Query 5.i (Figure 7.16), uses “mv_ruf_t_hour” over Road Utilization, but 5.ii has to do a “select count(distinct)” clause to count the distinct cars in a segment, which prevents it from using any of the materialized views built. Building a materialized view to speed up this query could be equivalent to build Road Utilization. Query 6, shown in Figure 7.17, remains unchanged as there is no materialized view built for it due to its small size.

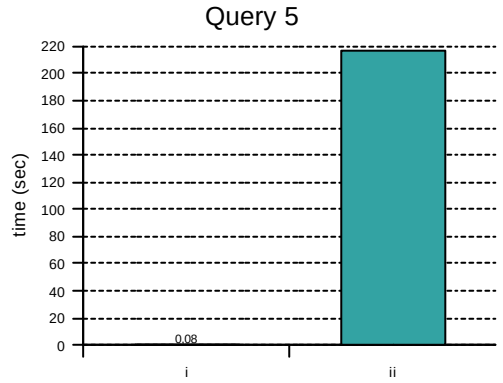


Figure 7.16 Timings for Query 5

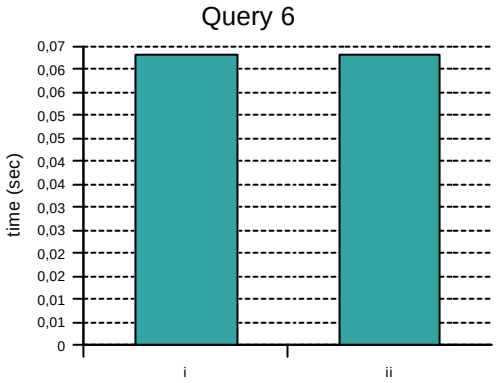


Figure 7.17 Timing for Query 6

7.6 Timing Results for the Database with Materialized Views

The queries were also made over the database maintaining the materialized views and dropping the indices. Graphs for these results will not be presented here, as the result present nothing new. Where materialized views are used, times are the same as for the database with materialized views and index, and when not used the results are the same as for the bare database (no indices to “fall-back” to). For comparison purposes, data for these timing results can be seen in Section 7.7.

7.7 Comparison

A comparison of the timings described above, along with the timings for the configuration in which index are dropped and materialized views enabled, is shown in Figures 7.18, 7.19 and 7.20. The numbering for the queries is the same as in Section 7.2.

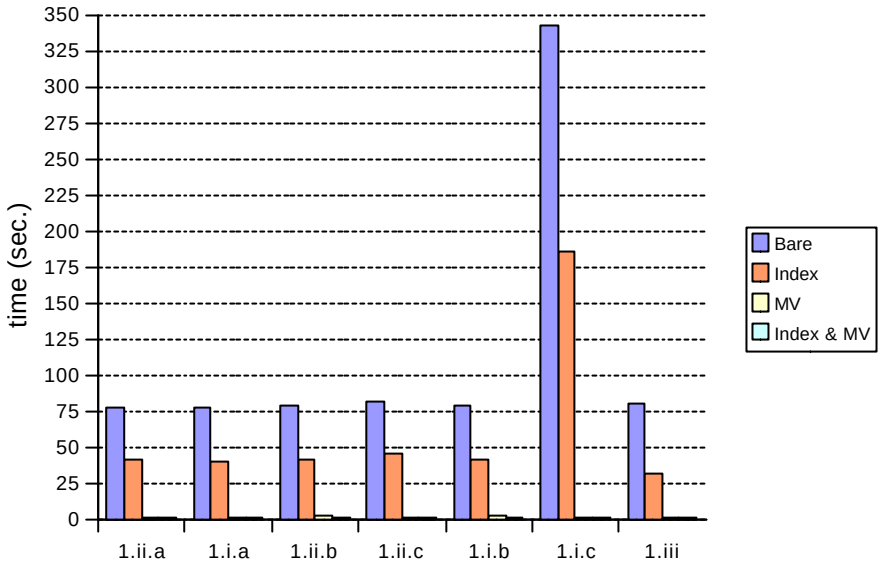


Figure 7.18 Timing for Query 1, All Configurations

In Figure 7.18 a comparison of the different configurations for Query 1 is shown. The average decrease in time for using the indices over the bare database is 50,15%, and for the materialized views over the indices, 96,93% (32 times faster). The main index for Observation Fact takes 421MB, and the materialized view takes 13MB. However, any query that doesn't take advantage of the materialized view (Query 2, Query 5.ii) can still use the index.

Query 4 is also benefited from the materialized view, as shown in Figure 7.19. Indexes decrease query time by an average 5,45%, and materialized view decreases time by 98,56% (69 times faster).

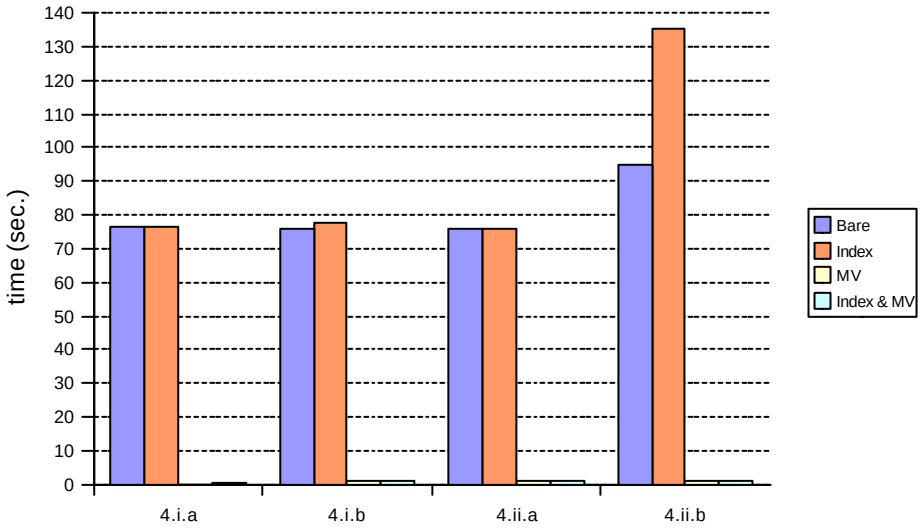


Figure 7.19 Timing for Query 4, All Configurations

Query 2 (Figure 7.20), as stated in section 8.5, doesn't use any materialized view, but uses an index for a 41,47% reduction in time; the same happens with query 5.ii, for an average 12%. Query 6 remains mostly unaffected by indices, as the Trip Fact table is too small for the influence to be noticed. Queries 3 and 7 gain nothing from these structures.

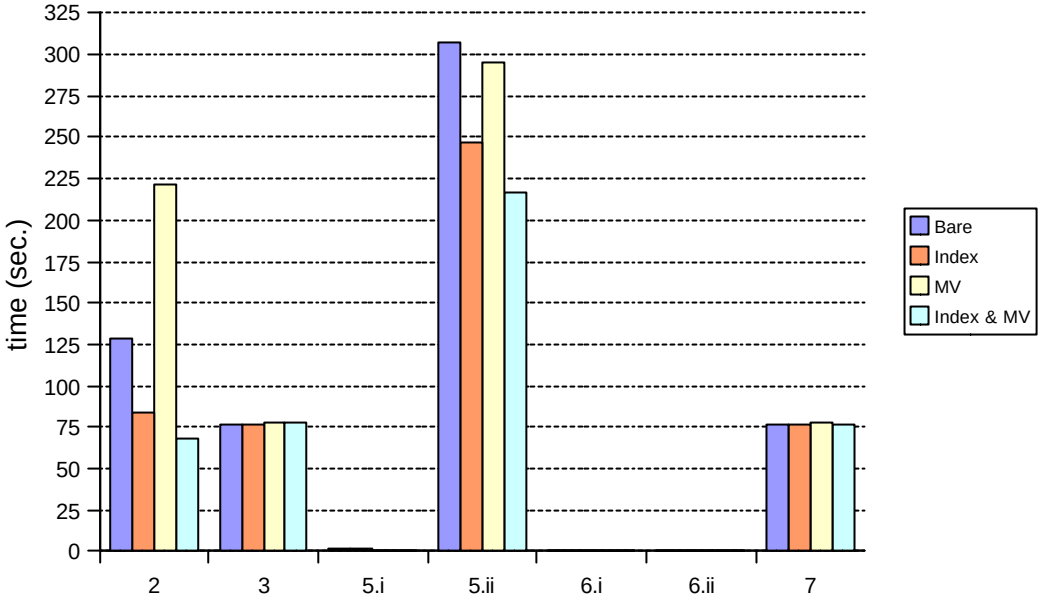


Figure 7.20 Timing Results for Queries 2, 3, 5, 6 and 7

While used indices took half the size or more of the fact tables, this is, a total of half a gigabyte, for a decrease in time around 50% in some cases, materialized views decreased query times around 95% with less than a total of 25 megabytes. Materialized views are a powerful tool to handle summarized data. They can boost performance up to the order of hundreds with a very little space requirement. However, new needs or kinds of queries that were not anticipated in the moment of the design will not take advantage of this improvement. Thus, an active maintenance and a thoughtful design are a must in this environment.

8 Conclusion and Future Work

8.1 Conclusion

Analysis of traffic through tracking of moving objects, a location-based service, is an activity that requires storing and querying an amount of this data which is typically too large to use the classical OLTP approach in a practical way.

A Data Warehouse design for spatiotemporal data was implemented in Oracle. It comprised dimensions as X, Y Position for the position of cars in the map, Time of Day and Date for time, a Road Network in which cars move, etc. The Observation Fact table keeps track of the moving objects, associating the position in X, Y, and Road Network, time, car, and other information with the respective dimensions. This fact table contains speed and acceleration measures, to analyze the behavior of cars respecting these parameters. Two other fact tables were built, Road Utilization, that stores the number of cars on a road segment in each moment, and Trip Fact, measuring the duration and length of trips.

This fact tables were populated with the two data sources available: real data from the INFATI project together with data from a data generator, that simulates an environment in which objects follow a given network, in this case the Aalborg road network; one day of traffic was simulated.

To improve the performance of the queries, indices were build over the database; to further improve query performance we used materialized views. A set of queries was tested over different configuration of the database, this is, without indices or materialized views, with indices and with materialized views. Timing results for these configurations were gathered and compared, showing the utility of the materialized views as a powerful tool to handle summarized data. While indices took half the size or more of the fact tables, for a decrease in time around 50% in some cases, materialized views decreased query times around 95% with a few megabytes. However, materialized views have to be built according to the set of queries that will be issued to the DBMS, as they gather specific data, and thus new needs not taken into account in the design phase will not take advantage of these structures.

It remains to be seen if it would be more effective to build Road Utilization fact, gathered from the observations, as a materialized view of Observation fact. As it is, queries related to this measurement are more than a hundred times faster over Road Utilization than over Observation. Oracle facilities to keep the materialized views updated automatically would ease maintenance of this information, while as an independent fact table it has to be separately updated.

Information about trips is computationally expensive to gather from the Observation fact. Observations have to be ordered by car and time, traveled distance accumulated for each observation and time gaps of more than 10 minutes detected, to establish a trip. This makes complex doing a materialized view to aggregate this data or even query the Observation Fact table for trip information. However, once built, the Trip Fact table is very small and fast to query to.

8.2 Future Work

It would be interesting to compare this approach to Data Warehousing support with the MOLAP (Multidimensional OLAP) model, in which the data is stored in a Multi-dimensional database. Oracle Analytic Workspace [Oracle, 2003], integrated in Oracle9i R2, or Microsoft Analysis Services [Microsoft, 2003] are commercial products with this philosophy.

Another interesting field is analyzing how these structures to support Data Warehousing behave regarding the loading or updating of data. Real-time applications that load data frequently or without the possibility of doing off-line loads are a possible target for analysis.

Specific spatiotemporal structures [Papadias et al., 2001] like R-trees, aggregation trees, aRB-tree, etc. can be used and their performance compared to the structures used in this thesis, indices and materialized views.

9 Bibliography

Thomas Brinkhoff
Generating Network-based Spatiotemporal Datasets
http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator.shtml current as of 4th Jun 2003

Jan Eliasen, Casper Kjær , Helen Urban
Data Warehouse Design for Spatio-Temporal Data.
DAT5 Report, January 2002. Aalborg University

Jan Eliasen, Casper Kjær , Helen Urban
Data Warehouse Based Traffic Jam Detection.
DAT6 Report, January 2002. Aalborg University

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

INFATI.
http://www.infati.dk, current as of 1st May 2003

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

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.

Nick Kline, Richard Snodgrass.
Computing Temporal Aggregates.
International Conference on Data Engineering, 1995

Microsoft Corporation
http://www.microsoft.com/sql/evaluation/bi/bianalysis.asp, current as of 6th Jun 2003

Oracle Corporation
Oracle9i Database Concepts, Memory Architecture
http://otn.oracle.com/documentation/, current as of 4th Jun 2003

Oracle Corporation
Oracle9i Data Warehousing Guide
http://otn.oracle.com/documentation/, current as of 4th Jun 2003

Oracle Corporation
Oracle9i Database Performance Tuning Guide and Reference
http://otn.oracle.com/documentation/, current as of 4th Jun 2003

Oracle Corporation
http://www.oracle.com/olap/, current as of 6th Jun 2003

Dimitris Papadias, Yufei Tao, Panos Kalnis, Jun Zhang
Indexing Spatio-Temporal Data Warehouses.
Proceedings of IEEE International Conference on Data Engineering (ICDE), San Jose, 26 Feb - 1 Mar, 2002.

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

Torben B. 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.

Dan Sunday.
About Lines and Distance of a Point to a Line.
http://geometryalgorithms.com current as of 1st Apr 2003

Vassilis J. Tsotras, Christian S. Jensen, Richard T. Snodgrass
A Notation for Spatiotemporal Queries
TR-10 a Time Center Technical Report, 1997
http://www.cs.auc.dk/TimeCenter/, current as of 4th Jun 2003

USGS Eastern Region Geography
The Universal Transverse Mercator (UTM) Grid
http://mac.usgs.gov/mac/isb/pubs/factsheets/fs07701.html current as of 4th Jun 2003