Declarative approach for long-term sensor data storage

Manel Charfi

N°d'ordre NNT : 2017LYSEI081

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
INSA de Lyon

École Doctorale N° EDA 512
École Doctorale d'Informatique et Mathématiques de Lyon

Spécialité de Doctorat : Informatique

Soutenue publiquement le 21/09/2017, par :
Manel CHARFI

Declarative Approach for Long-Term Sensor Data Storage

Devant le jury composé de :

BOUZEGHOUB, Amel	Pr	Telecom SudParis	Présidente
BIDOIT, Nicole	Pr	Université Paris Sud 11	Rapporteure
WIJSEN, Jef	Pr	Université de Mons	Rapporteur
LAKHAL, Lotfi	Pr	Aix-Marseille Université	Examinateur
PETIT, Jean-Marc	Pr	INSA de Lyon	Directeur de thèse
GRIPAY, Yann	MdC	INSA de Lyon	Co-directeur
FOURTY, Nicolas	MdC	Université Grenoble Alpes	Co-encadrant
BEN YAHIA, Sadok	Pr	Faculté des Sciences de Tunis	Invité

Département FEDORA – INSA Lyon - Ecoles Doctorales – Quinquennal 2016-2020

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **http://www.edchimie-lyon.fr** Sec : Renée EL MELHEM Bat Blaise Pascal 3e etage secretariat@edchimie-lyon.fr Insa : R. GOURDON	M. Stéphane DANIELE Institut de Recherches sur la Catalyse et l'Environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 avenue Albert Einstein 69626 Villeurbanne cedex directeur@edchimie-lyon.fr
E.E.A.	**ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE** **http://edeea.ec-lyon.fr** Sec : M.C. HAVGOUDOUKIAN Ecole-Doctorale.eea@ec-lyon.fr	M. Gérard SCORLETTI Ecole Centrale de Lyon 36 avenue Guy de Collongue 69134 ECULLY Tél : 04.72.18 60.97 Fax : 04 78 43 37 17 Gerard.scorletti@ec-lyon.fr
E2M2	**EVOLUTION, ECOSYSTEME, MICROBIOLOGIE, MODELISATION** **http://e2m2.universite-lyon.fr** Sec : Sylvie ROBERJOT Bât Atrium - UCB Lyon 1 04.72.44.83.62 Insa : H. CHARLES secretariat.e2m2@univ-lyon1.fr	M. Fabrice CORDEY CNRS UMR 5276 Lab. de géologie de Lyon Université Claude Bernard Lyon 1 Bât Géode 2 rue Raphaël Dubois 69622 VILLEURBANNE Cédex Tél : 06.07.53.89.13 cordey@univ-lyon1.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTE** **http://www.ediss-lyon.fr** Sec : Sylvie ROBERJOT Bât Atrium - UCB Lyon 1 04.72.44.83.62 Insa : M. LAGARDE secretariat.ediss@univ-lyon1.fr	Mme Emmanuelle CANET-SOULAS INSERM U1060, CarMeN lab, Univ. Lyon 1 Bâtiment IMBL 11 avenue Jean Capelle INSA de Lyon 696621 Villeurbanne Tél : 04.72.68.49.09 Fax :04 72 68 49 16 Emmanuelle.canet@univ-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHEMATIQUES** **http://infomaths.univ-lyon1.fr** Sec :Renée EL MELHEM Bat Blaise Pascal, 3e étage Tél : 04.72. 43. 80. 46 Fax : 04.72.43.16.87 infomaths@univ-lyon1.fr	M. Luca ZAMBONI Bâtiment Braconnier 43 Boulevard du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél :04 26 23 45 52 zamboni@maths.univ-lyon1.fr
Matériaux	**MATERIAUX DE LYON** **http://ed34.universite-lyon.fr** Sec : Marion COMBE Tél:04-72-43-71-70 –Fax : 87.12 Bat. Direction ed.materiaux@insa-lyon.fr	M. Jean-Yves BUFFIERE INSA de Lyon MATEIS Bâtiment Saint Exupéry 7 avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72.43 71.70 Fax 04 72 43 85 28 Ed.materiaux@insa-lyon.fr
MEGA	**MECANIQUE, ENERGETIQUE, GENIE CIVIL, ACOUSTIQUE** **http://mega.universite-lyon.fr** Sec : Marion COMBE Tél:04-72-43-71-70 –Fax : 87.12 Bat. Direction mega@insa-lyon.fr	M. Philippe BOISSE INSA de Lyon Laboratoire LAMCOS Bâtiment Jacquard 25 bis avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72 .43.71.70 Fax : 04 72 43 72 37 Philippe.boisse@insa-lyon.fr
ScSo	**ScSo*** **http://recherche.univ-lyon2.fr/scso/** Sec : Viviane POLSINELLI Brigitte DUBOIS Insa : J.Y. TOUSSAINT Tél : 04 78 69 72 76 viviane.polsinelli@univ-lyon2.fr	M. Christian MONTES Université Lyon 2 86 rue Pasteur 69365 LYON Cedex 07 Christian.montes@univ-lyon2.fr

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Dedication

For my son and my husband,
For my mother and my father,
For my sister and my brother,
For my family-in-law,
For all my friends,
For all those who ever supported me.

Manel

Aknowledgements

First of all I would like to express my sincerest thanks and my deepest gratitude to my thesis directors ***Mr. Jean-Marc PETIT*** *and* ***Mr. Yann GRIPAY*** *for their benevolence and willingness to supervise my research project. Thank you Jean-Marc for all your precious advices all along these 4 years whether for work or for everyday life. And thank you Yann for the positive vibes you were and still sharing since the first day I came to Lyon. It was a huge pleasure to know both of you and to work together.*

I would like to thank ***Mrs. Nicole BIDOIT*** *and* ***Mr. Jef WIJSEN*** *for agreeing to be referees of my thesis as well as* ***Mrs. Amel BOUZEGHOUB***, ***Mr. Lotfi LAKHAL***, *and* ***Mr. Nicolas FOURTY*** *for accepting to be part of the jury.*

Thanks to the Database team, LIRIS laboratory for their support and advices. Thanks to any other LIRIS member that somehow helped me during my thesis.

This PhD thesis was financed by the ARC6 program of the Rhône-Alpes region, France.

I would like to address my thanks to anyone who directly or indirectly contributed to the success of this work.

Abstract

Nowadays, sensors are cheap, easy to deploy and immediate to integrate into applications. These thousands of sensors are increasingly invasive and are constantly generating enormous amounts of data that must be stored and managed for the proper functioning of the applications depending on them. Sensor data, in addition of being of major interest in real-time applications, e.g. building control, health supervision..., are also important for long-term reporting applications, e.g. reporting, statistics, research data...

Whenever a sensor produces data, two dimensions are of particular interest: the *temporal* dimension to stamp the produced value at a particular time and the *spatial* dimension to identify the location of the sensor. Both dimensions have different granularities that can be organized into hierarchies specific to the concerned context application. For instance, the spatial dimension for intelligent building applications can hold the following granularities: house, room, sensor.

In this PhD thesis, we focus on applications that require long-term storage of sensor data issued from sensor data streams. Since huge amount of sensor data can be generated, our main goal is to select only relevant data to be saved for further usage, in particular long-term query facilities. More precisely, our aim is to develop an approach that controls the storage of sensor data by keeping only the data considered as "relevant" according to the spatial and temporal granularities representative of the application requirements. For instance, an application asking for the temperature of a given house per day along a year may have to consider every temperature value sent from different sensors in every room every minute. In such cases, approximating data in order to reduce the quantity of stored values enhances the efficiency of those queries. Our key idea is to borrow the declarative approach developed in the seventies for database design from constraints and to extend functional dependencies with spatial and temporal components in order to revisit the classical database schema normalization process.

Given sensor data streams, we consider both spatio-temporal granularity hierarchies and Spatio-Temporal Functional Dependencies (STFDs) as first class-citizens for designing sensor databases on top of any RDBMS. Since application requirements relate to data over long periods (e.g. a few months, several years), sensor data can be approximated using STFDs to reduce storage space and increase the efficiency of ap-

plication queries. We propose a specific axiomatisation of STFDs and the associated attribute closure algorithm, leading to a new normalization algorithm. We thus define a granularity-aware spatio-temporal database model based on these formal notions along with an algorithm to design a database schema from STFD constraints.

Then, we propose a granularity-aware sensor database architecture supporting, through a declarative approach, both the database schema design and the specification of the data loading from sensor data streams. We define an annotation of schema attributes with aggregate functions to specify which summarized values are of interest in the raw data. Thus, from sensor data streams schemas, domain-specific spatio-temporal granularity hierarchies, and application-specific STFDs and attribute annotations, we can automatically both produce a concrete SQL DB on a RDBMS and configure the data loading process through a middleware.

We have implemented a prototype of this architecture to deal with both database design and data loading. We conducted experiments with synthetic and real-life data streams from intelligent buildings. We compared our solution with the baseline solution and we obtained promising results in terms of query performance and memory usage. We have also studied the trade-off to be found between data reduction and data approximation.

Résumé

De nos jours nous avons de plus en plus de capteurs qui ont tendance à apporter confort et facilité dans notre vie quotidienne. Ces capteurs sont faciles à déployer et à intégrer dans une variété d'applications (monitoring de bâtiments intelligents, aide à la personne, ...). Ces milliers (voire des millions) de capteurs sont de plus en plus envahissants et génèrent sans arrêt des masses énormes de données que nous devons stocker et gérer pour le bon fonctionnement des applications qui en dépendent. En plus d'avoir un intérêt majeur pour les applications en temps réel, par exemple pour le contrôle des bâtiments, le suivi médical, ..., les données capteurs sont également importantes pour les applications de reporting à long terme, par exemple les applications de statistiques et les données de recherche.

A chaque fois qu'un capteur génère une donnée, deux dimensions sont d'un intérêt particulier : la dimension temporelle et la dimension spatiale. Ces deux dimensions permettent d'identifier l'instant de réception et la source émettrice de chaque donnée. Chaque dimension peut se voir associée à une hiérarchie de granularités qui peut varier selon le contexte d'application. Par exemple, la dimension spatiale pour les applications de bâtiments intelligents peut contenir les granularités suivantes : maison, pièce, capteur.

Dans cette thèse, nous nous concentrons sur les applications nécessitant une conservation à long terme des données issues des flux de données capteurs. Notre objectif est de mettre en œuvre une approche qui vise à contrôler le stockage des données capteurs en ne gardant que les données jugées pertinentes selon la spécification des granularités spatiotemporelles représentatives des besoins applicatifs. Par exemple, une application demandant la température d'une maison donnée par jour le long d'une année peut avoir à considérer toutes les valeurs de température envoyée par différents capteurs dans chaque pièce à chaque minute. Dans une situation pareille, le fait d'approximer les données reçues réduit la quantité de valeurs stockées et peut améliorer l'efficacité de certaines requêtes. Notre idée clé consiste à emprunter l'approche déclarative développée dans les années soixante-dix pour la conception de bases de données à partir de contraintes et d'étendre les dépendances fonctionnelles avec des composantes spatiales et temporelles afin de revoir le processus classique de normalisation de schéma de base de données.

Étant donné des flux de données capteurs, nous considérons à la fois les hiérarchies de granularités spatio-temporelles et les Dépendances Fonctionnelles SpatioTemporelles (DFSTs) comme objets de premier ordre pour concevoir des bases de données de capteurs compatibles avec n'importe quel SGBDR. Comme les besoins applicatifs concernent les données sur une longue période de temps (par exemple quelques mois, plusieurs années), les flux de données capteurs peuvent être approximés grâce aux DFSTs pour réduire l'espace de stockage et augmenter l'efficacité des requêtes applicatives.

Nous proposons une axiomatisation spécifique aux DFSTs ainsi que l'algorithme de fermeture d'attributs associé, conduisant à un nouvel algorithme de normalisation. Nous nous basons sur ces notions formelles afin de définir un modèle de base de données spatiotemporelle prenant en compte les granularités, ainsi qu'un algorithme pour concevoir un schéma de base de données à partir des contraintes DFSTs.

Ensuite, nous proposons une architecture de base de données capteurs prenant en compte les granularités, suivant une approche déclarative, qui prend en charge à la fois la conception du schéma de la base de données et la spécification du chargement des données à partir des flux de données capteurs. Nous définissons aussi une annotation d'attributs de schéma avec des fonctions d'agrégation pour spécifier les valeurs récapitulatives qui seront retenues à partir des données brutes selon les besoins applicatifs. Ainsi, à partir des schémas des flux de données capteurs, des hiérarchies spatiotemporelles spécifiques au domaine et des STFDs spécifiques à l'application suivies des annotations d'attributs, nous pouvons produire automatiquement à la fois une base de données SQL concrète sur un SGBDR et configurer le processus de chargement de données via un middleware.

Nous avons implémenté un prototype de cette architecture qui traite à la fois la conception de la base de données ainsi que le chargement des données. Nous avons mené des expériences avec des flux de donnés synthétiques et réels provenant de bâtiments intelligents. Nous avons comparé notre solution avec la solution de base et nous avons obtenu des résultats prometteurs en termes de performance de requêtes et d'utilisation de la mémoire. Nous avons également étudié le compromis entre la réduction des données et l'approximation des données.

Contents

Page

Part I Introduction 1

Chapter 1 Introduction 3

1.1 Context . . . 4

1.1.1 Intelligent Buildings . . . 4

1.1.2 Sensor Data & Data Streams . . . 5

1.1.3 Long-Term Reporting Applications . . . 7

1.2 Thesis Objective . . . 8

1.2.1 Thesis Context . . . 9

1.2.2 Declarative Approach for Long-Term Sensor Data Storage . . . 10

1.2.3 Problem Statement . . . 11

1.3 Contributions . . . 11

1.4 Document Organization . . . 13

Chapter 2 Preliminaries 15

2.1 General Notions . . . 16

2.2 Spatio-Temporal Granularity Hierarchies . . . 17

2.2.1 Granularity . . . 17

2.2.2 Granularity Hierarchy . . . 18

2.2.3 Application to Spatio-Temporal Dimensions . . . 18

2.3 Temporal Functional Dependency . . . 19

2.3.1 Functional Dependency . . . 20

2.3.2 Temporal Module 20
2.3.3 Temporal Functional Dependency 21

Part II Contributions **23**

Chapter 3 Granularity-Aware Spatio-Temporal Database Model **25**
3.1 Granularity-Aware Spatio-Temporal Database 26
3.2 Spatio-Temporal Functional Dependency 28
3.3 Logical Implication with STFDs 30
3.3.1 The Implication Problem 31
3.3.2 Inference Axioms for STFDs 31
3.3.3 Soundness and Completeness of the Proposed Axioms 33
3.4 Normalization of Granularity-Aware Spatio-Temporal Database 37
3.4.1 Closure of Attributes 37
3.4.2 Minimal Cover of STFD Set 40
3.4.3 Normalization of Granularity-Aware Spatio-Temporal Database Schema 41

Chapter 4 Granularity-Aware Sensor Database Architecture **43**
4.1 Sensor Database Schema Design 44
4.1.1 Abstract Schema 45
4.1.2 Semantic Value Assumption 45
4.1.3 Concrete Schema 47
4.1.4 Synthesis Example 47
4.2 Data Loading Specification 48
4.3 Architecture 49

Chapter 5 Prototype & Experiments **51**
5.1 Prototype Architecture 52
5.2 Implementation 54
5.2.1 Prototype Core 54
5.2.2 Web GUI 54
5.2.3 Underlying RDBMS 55
5.2.4 Technologies and Tools 55

5.2.5 Experimentation Platform 57
5.3 Experiments 57
5.3.1 Experimental Data Sets 58
5.3.2 Data Storage Evolution w.r.t. Granularities 59
5.3.3 Data Accuracy w.r.t. Data Reduction 59
5.3.4 Sensor Data Loading Efficiency 62
5.3.5 Query Workload 63
5.4 Demonstration 69
5.4.1 Purpose of the Demonstration 69
5.4.2 Database Schema Design 70
5.4.3 Observing On-The-Fly Data Loading 74

Part III Related Work & Conclusion **77**

Chapter 6 Related Work **79**
6.1 Database Design 80
6.2 Temporal Database 80
6.2.1 Temporal Functional Dependency 81
6.2.2 Semantic Assumptions 81
6.3 Sensor Data Management 82
6.3.1 Storage 82
6.3.2 Load Shedding 82
6.4 Data Warehouse 83

Chapter 7 Conclusion **85**
7.1 Research Summary 86
7.2 Future Work & Perspectives 87

Part IV References **89**

Part V Appendixes **99**

Chapter A Other algorithms **A1**
A.1 Algorithm Member A1

A.2 Algorithm NonRedundant A1
A.3 Algorithm LeftReduce A2
A.4 Algorithm Union A3

Chapter B Extended Version of the Paper for IDEAS 2016 A5

List of Figures

1.1 Intelligent building among research and application domains 5
1.2 An example of temporal and spatial hierarchies 6
1.3 An estimation of sensor data amount in a sensor data example 8
1.4 An overview of a declarative system for sensor data 12

2.1 Example of granularities 18

4.1 An example of SVAs 46
4.2 An example of the database schema generation 48
4.3 An overview of the proposed architecture 50

5.1 A detailed view of the proposed architecture 52
5.2 Spatio-temporal granularity hierarchies 58
5.3 Number of tuples when varying temporal and spatial granularities . . . 60
5.4 Data accuracy w.r.t. data reduction 61
5.5 Difference between the results of Q_0 and Q_1 62
5.6 Total execution time w.r.t. the number of sensors, duration 1 hour . . . 63
5.7 Total execution time w.r.t. the duration, for 100 sensors 63
5.8 The query performances of the first set of queries w.r.t. the data acquisition period 68
5.9 The elapsed time of the second set of queries w.r.t. to the data acquisition period 69
5.10 The elapsed time of the third set of queries w.r.t. to the data acquisition duration 69
5.11 Screenshot of the creation of the spatial hierarchy 70
5.12 Screenshot of the creation of the temporal hierarchy 71

5.13 Screenshot of the input of the concerned attributes 71
5.14 Screenshot of the first example of the input of STFD 72
5.15 Screenshot of the second example of the input of STFD 72
5.16 Screenshot of the minimal cover generation 73
5.17 Screenshot of the outputted SQL queries 73
5.18 Screenshot of the number of tuples in *raw DB* and *sensor DB* at the beginning 74
5.19 Screenshot of the number of tuples in *raw DB* and *sensor DB* at the middle 75
5.20 Screenshots of the number of tuples in *raw DB* and *sensor DB* at the end 75

List of Tables

1.1 Example of temperature sensor data stream 7

2.1 Example of temperature sensor data stream 19
2.2 A relation r 20
2.3 A relation r' 21

3.1 Example of temperature sensor data stream 27
3.2 A counterexample relation $r_0 = \varphi(g_0, h_0)$ 36

5.1 Data reduction w.r.t. STFDs 61
5.2 Database relation 64

Part I

Introduction

CHAPTER 1

Introduction

Introduction

Nowadays, sensors are cheap, easy to deploy and immediate to integrate into applications. Our environment is more and more invaded with these increasingly tiny devices. They are capable of measuring with high precision a large variety of physical phenomena such as temperature, humidity, presence, radiation, electromagnetism, luminosity, noise, chemicals... Due to their contribution in different application domains all over the last decades, they are tending to pervade our digital environment quite rapidly. Currently, we can find sensors everywhere, they may be present in houses, offices, hospitals, schools, cars, wearable devices... We are more and more speaking about intelligent buildings, smart cars, smart cities, to mention a few.

Thousands and even millions of sensors can be deployed easily, generating data streams that produce cumulatively huge volumes of data. Those data can be useful for real-time applications (e.g. supervision) as well as for long-term applications (e.g. reporting, statistics). One of the main problems in the case of long-term applications is ensuring an efficient storage of data issued from numerous sensors.

In this thesis, we propose a declarative approach that aims to guarantee an efficient storage and querying of the "relevant" sensor data for long-term reporting applications. We focus on a context application based on intelligent buildings, however many other application contexts could benefit from this work.

In this chapter we present the context of our work. Then, we define the thesis objectives and highlight our problem statement. After that, we detail our contributions and present the organisation of this document.

1.1 Context

In this section, we present the context of intelligent buildings, discuss about the specificities of sensor data and data streams, and focus on the requirements of long-term reporting applications.

1.1.1 Intelligent Buildings

The major goal of intelligent buildings is to simplify the daily lives of users and make them comfortable by ensuring the adaptation of the environment according to their needs. These buildings become increasingly of major importance in the supervision and personal assistance fields (elderly, disabled, sick). For example, by supervising the state of health of patients or by tracking particular events that may detect eventual problematic states. Intelligent buildings are also useful for ensuring the security of the building and of its occupants.

All these possible functions generally come with an effort to save and reduce the building energy consumption. This reduction may be the result of a good management of any considered functionality of the building. For instance, the management of lighting, by turning off the lamps depending on the external brightness and the occupation of spaces, or the management of heating, by changing the heating system power with respect to the external and the internal temperatures of the building.

Therefore, intelligent buildings are instrumented with several sensors (temperature, brightness, presence...) along with several actuators (alarms, light switches, heating control...). Thanks to their increasing presence in every day life, more and more research fields are interested in the different application domains of sensors: some research and application domains are given in Figure 1.1.

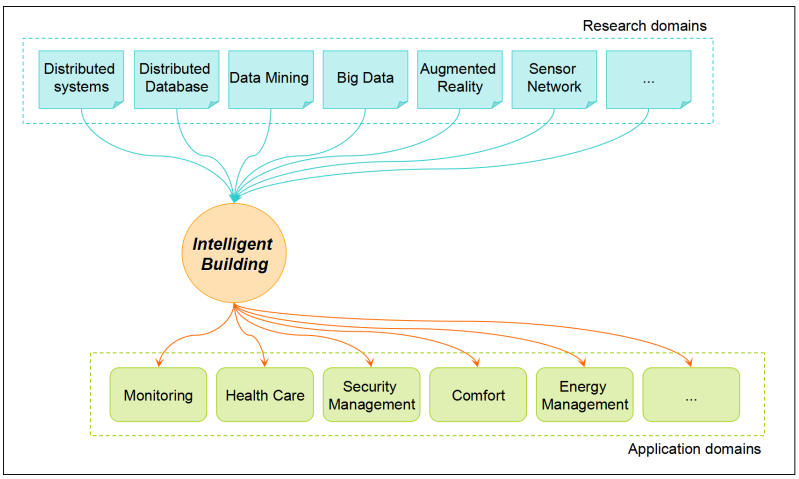


Figure 1.1 — Intelligent building among research and application domains

To be concrete and without loss of generality, we consider intelligent building of medium to big size hosting, typically several hundreds of inhabitants over a hundred apartments. Each intelligent building is equipped with different sensors and actuators dedicated to the control of the main functions of the building. In our setting, the intelligent building is managed by a central entity, called the manager hereinafter. The management of an intelligent building may require the interoperability of different applications managing different issues related to the building using the same sensor data.

1.1.2 Sensor Data & Data Streams

Sensor data, i.e. data produced by a sensor, come as a continuous stream of data, or data streams. Data streams are an infinite ordered sequence of data records which usually are associated with a timestamp. Each sensor can be set up to a given frequency which defines the period between the sending of two successive values.

We are interested in heterogeneous data streams where each stream concerns a physical phenomena (e.g. temperature, humidity...) and whose data acquisition frequencies may be different and not synchronized. Queries over such data are complicated to express as they intend to synchronise such heterogeneous data.

Whenever a sensor produces data, two dimensions are of particular interest: the *temporal* dimension to stamp the produced value at a particular time and the *spatial* dimension to identify the location of the sensor. Both dimensions can be represented with different granularities organized into hierarchies that may vary according to the specific needs of different applications. For instance, Figure 1.2 depicts a hierarchy of temporal granularities (a) and a hierarchy of spatial granularities (b). Each granularity, e.g. *day*, gathers a set of granules which represent the same temporal (resp. spatial) unit, e.g. 2017/02/11, 2017/09/21... Intuitively a granule is a concrete instance at a given granularity.

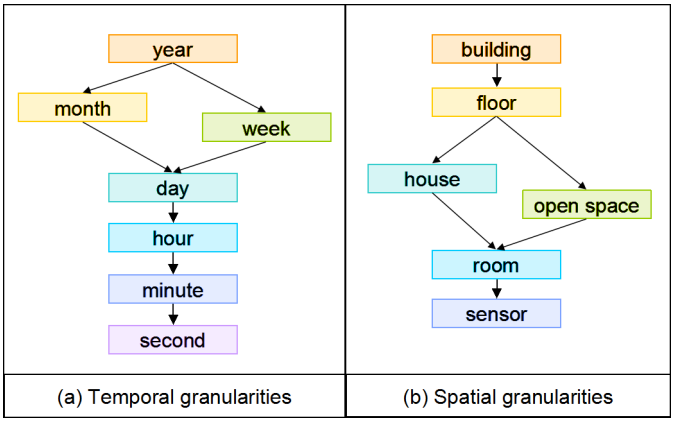


Figure 1.2 — An example of temporal and spatial hierarchies

Example 1 *We consider a running example of sensor data streams from intelligent buildings. In each building different sensors (temperature, luminosity, humidity, CO_2...) are deployed. Each building contains several floors; each floor has over ten houses (apartments) and each house has several rooms equipped with sensors. Therefore, at the scale of several buildings, a huge number of sensors exists, each one sending values at its own rate. Let us consider a snapshot of a temperature sensor data stream as given in Table 1.1, where sensor and timestamp contain data at the finest granularities belonging to the granularity hierarchies presented in Figure 1.2: timestamp is indicated at the granularity second and sensor at the granularity sensor.*

This stream is representative of a wide variety of sensor data streams in the context of intelligent buildings or in other contexts. Moreover, in our context, we consider static sensors, i.e. sensors set to a particular place to sense the environment and whose locations never change.

value	*sensor*	*timestamp*
21	s11	2016/03/02 11:59:00
20	s21	2016/03/02 11:59:30
20	s12	2016/03/02 12:01:00
23	s22	2016/03/02 12:01:00
20	s11	2016/03/02 12:01:30
24	s31	2016/03/02 12:02:00
20	s12	2016/03/02 12:02:30
23	s21	2016/03/02 12:15:00

Table 1.1 — Example of temperature sensor data stream

1.1.3 Long-Term Reporting Applications

Sensor data can be useful for monitoring applications which have to react in quasi real-time (e.g. supervision) as well as for applications requiring long-term storage of data streams (e.g. reporting, statistics). Therefore, sensor data management can be seen from two points of view: real-time, i.e. continuous queries [ACÇ+03, ABB+03, ABW06, TATV11] where data are analysed on-the-fly and are not stored versus historical data management [LH05, LCXA06, DGMS07, PNJ09] where stream data are kept for later use.

For long-term reporting applications many types of data management systems can be used from classical file systems (in json, flat text...) to Relational Database Management Systems (RDBMS). In the former, a file is created for a given period of time (e.g. week, month). In the latter, data streams are stored as classical tables with specific attributes to cope with the spatio-temporal aspects of the stream. In both cases, the burden is let to application designers who have to pose complex queries (e.g. full-text search or SQL-like queries) to deal with temporal and spatial dimensions.

Example 2 *Building managers need to have long-term storage of sensor data streams to be able to produce reports for inhabitants (on a contractual basis) such as:*

- *temperature averages of each room during a given year,*
- *the maximum CO_2 values per floor per month,*
- *the minimum temperature values per house per day.*

When we intend to store all received sensor data for a post treatment, we can be faced with huge amounts of data. Therefore, in these application domains, an important storage space as well as important query execution delays have to be considered. Whenever the data volume is high, the query processing time can also be prohibitive.

Selecting only relevant data from sensor data streams to be saved for long-term query facilities, is still an issue since huge amount of sensor data can be produced for a specific application, as shown in Figure 1.3.

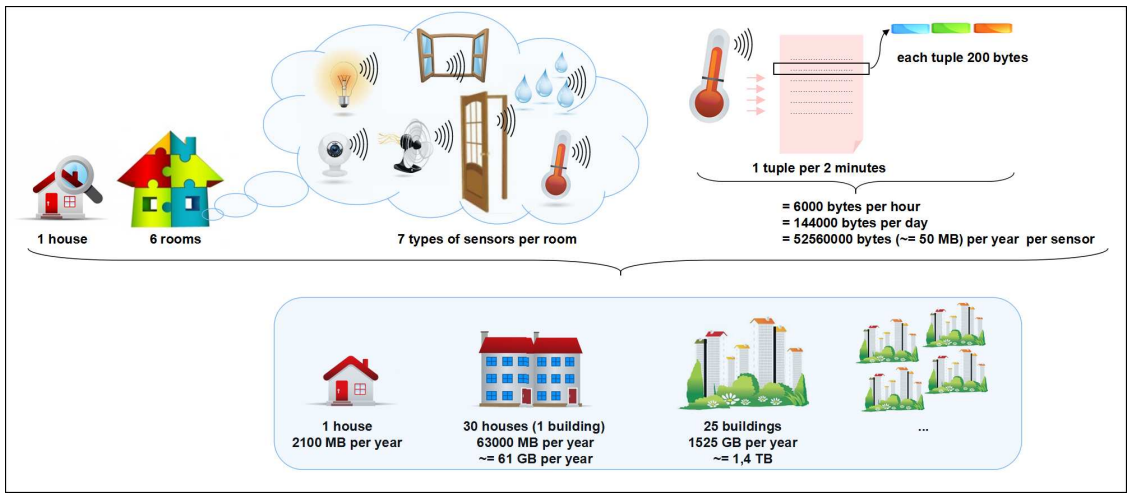


Figure 1.3 — An estimation of sensor data amount in a sensor data example

As presented before, sensor data can be produced from different locations with different frequencies. In our context, we want to adjust data storage to the user needs with respect to specific representations of time and space. Therefore, we point out the need to model sensor data according to application-specific spatio-temporal representation. We propose to highlight the presence of the spatio-temporal hierarchies that can be very significant for data organization and may vary with respect to each application domain. For instance, we consider the two building applications below:

- *Application A* that computes temperature averages in order to compare thermal insulation of different buildings all over the year,
- *Application B* that aims to study the variation in the use of electric light with respect to the external light.

Application A can be satisfied with average temperature values per building per day as it aims to compare two different buildings and as an average temperature all over the day can be representative of the general temperature of each building. However, *Application B* needs more precise values as the outside light varies throughout the day: we may consider luminosity values per hour per room.

1.2 Thesis Objective

In this section, we first present the academic context of this thesis. We then describe the main objectives of this work and highlight the general problem statement.

1.2.1 Thesis Context

This work is financed by the ARC6 program of the Rhône-Alpes region, France. The challenge was to work on decreasing the energy consumption of an intelligent building via data-based perspective by taking into account user usage. Experiments were meant to be performed locally on the experimental platform LIRIS SoCQ4Home (an instrumented building at our campus at INSA de Lyon) and then on a larger scale in a park of about a hundred individual housing renovated thanks to a partnership of the LCIS[1] a partner laboratory with the DREAL[2] (local administration agency).

Our initial idea was to propose a declarative architecture for an interactive monitoring system that provides a logical and abstract view of the building resources, in particular sensors [CGF+14]. In such case the required optimization of data management is supported by the system instead of the application developers. Such systems make it possible to represent the environment with conventional data and data streams. Therefore, the building applications can be expressed in the form of complex queries (like SQL: SELECT ... FROM ... WHERE ...) that the system will perform optimally from the energy point of view using an accurate cost-model. The main task of the system is to manage sensor data: monitoring, storing and querying data from sensor data streams.

The work was intended to take part in the Building Information Modeling (BIM) process. Actually, BIM is a digital representation of physical and functional characteristics of the concerned buildings [2]. A BIM is supposed to contain all building information that is necessary to plan, design, construct, operate and maintain its diverse physical infrastructures, such as water, electricity, gas, communication utilities... from its earliest conception to demolition. Thus the monitoring system can dynamically extract the necessary information about the sensors from the BIM data of each building.

At the beginning we focused on query optimization over data streams [Cha14]. Then we worked on the semantic optimization of temporal queries [CGP16b] (an extended version of the proposed article is given in Appendix B) using Temporal Functional Dependencies (TFD) [BJW00]. The main difficulty that we encountered at this stage was the complexity and the length of the logical queries (e.g. queries with eight variables) needed by our application context.

Therefore, we decided to change our viewing angle: instead of trying to resolve very complex queries we focused on putting in place a system that holds data accessible via simpler queries. So, instead of using complex queries over huge amounts of data, our main idea is to keep only the "relevant" data with accordance to the application requirements including user's preferences. Then, as the complexity and the amount of

[1] http://lcis.grenoble-inp.fr/le-laboratoire/

[2] Direction Régionale de l'Environnement de l'Aménagement et du Logement, http://www.auvergne-rhone-alpes.developpement-durable.gouv.fr/

data decrease, application queries become simpler and more efficient [CGP17, CGP16a].

Thus, in this document, we present our work on the last issue and we focus on the problematic of designing the appropriate sensor database according to application-specific constraints.

1.2.2 Declarative Approach for Long-Term Sensor Data Storage

We are in between two domains: real-time management and long-term data storage of sensor data. We are interested in establishing a data management tool that constructs a so-called granularity-aware sensor database, which is a historical sensor database, from real-time sensor data streams. Our aim is to decrease the storage space and increase query efficiency of long-term reporting applications.

In order to achieve this goal, we have two main issues:

1. The design of a database for sensor data at application-specific spatio-temporal granularities,
2. The specification of sensor data loading from sensor data streams into such database.

Our key idea is to extend functional dependencies with spatial and temporal components in order to revisit the classical database schema normalisation process. Actually, through these dependencies, our approach would allow the user to define the configuration of the database design and then to declare how data will be loaded. We think that a declarative approach is appropriate in our case: the user will just have to define data constraints according to the spatial and temporal granularity hierarchies that correspond to her requirements in order to build a relevant sensor database.

Indeed, the main goal of a declarative approach is to allow the user (application developer, building manager...) to express what she wants to obtain without having to care about the way the system will have to realize it. The optimization is supported by the system rather than the user, like with SQL, and contrary to application development with imperative programming (Java, C/C ++). This allows the user to present her needs to the system in the form of queries and to obtain the results without worrying about the way they have been built.

Such an approach provides a logical and abstract view of the resources and allows to represent the environment with classic data and data streams. In fact, an intelligent building can be seen as a dynamic distributed environment: a set of sensors and actuators, gateways and computers which are connected within the building network using wired or wireless connections. The building sensors represent data sources that

produce data in the form of data streams. Those data are analysed in real-time and then stored in the database for further use, i.e. queries that require an overview over a long period of time.

With this declarative approach for long-term data storage, the building manager defines application requirements in term of spatial and temporal data constraints: the manager defines for each application the considered spatio-temporal hierarchies as well as the relevant granularities for each data attribute. According to these initial application requirements, "raw" data from sensors may be approximated with respect to specific spatio-temporal granularities, and then only relevant data are stored.

The definition of these constraints should reflect the actual application needs. Queries over this approximated data will be efficient if they respect the specified constraints. We make the hypothesis that this specification is done seriously by a person who is aware of the application context and its evolution. Otherwise, queries may not be as efficient.

1.2.3 Problem Statement

The challenges of applying traditional database management systems to extract business value from sensor data are still there. In summary, our objective is to establish a declarative system (on top of any RDBMS) allowing to handle the massive incoming data from intelligent building sensors and to keep only relevant entries according to application requirements, as depicted in Figure 1.4.

In this setting, the problem we are interested in is the following:

> Given a set of sensor data streams, how to declaratively design and feed a relevant sensor database for long-term reporting applications?

1.3 Contributions

In this thesis, we propose a declarative approach that aims to carry out the storage of the "relevant" sensor data at application-specific granularities. The main objective of this approach is to simplify and homogenize sensor data according to the application requirements, in order to efficiently support long-term reporting applications.

Instead of relying on query workload or data storage budget to define an optimized database [ZRL+04], we borrow the declarative approach developed in the seventies for database design using functional dependencies as constraints. We apply a similar approach in order to transform real-time sensor data streams into a sensor database. The sensor database satisfies the specified constraints while approximating raw data

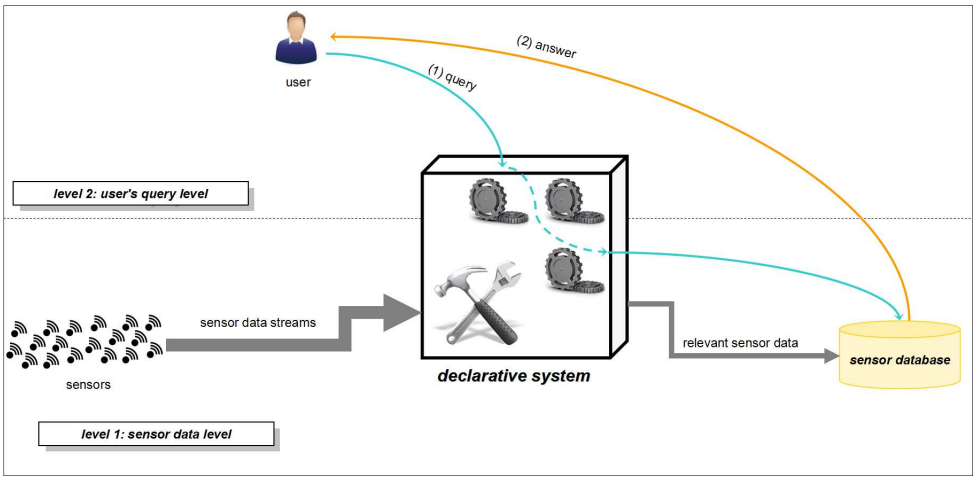


Figure 1.4 — An overview of a declarative system for sensor data

stream values. Thus, the approximation is "controlled" by the set of constraints. As the required results concern data over long periods of times (e.g. couple of months to several years), approximating sensor data streams should decrease storage space while allowing to express "relevant queries" and increasing query efficiency.

To do so, we introduce two levels of approximation:

1. Data constraints augmented with spatial and temporal dimensions allow to define the right granularities according to application requirements. We represent these constraints as Spatio-Temporal Functional Dependencies (STFDs) which extend Functional Dependencies (FDs). Such constraints are straightforward to understand and convey the semantics allowing to decide what are the "relevant" data granularities to be considered in the database. Classical database normalization techniques have been revisited in this setting to obtain a granularity-aware sensor database schema.

2. Database schema annotations based on semantic assumptions of temporal databases [BJW00], so called "Semantic Value Assumptions" (SVA). These annotations allow to specify the manner to choose "relevant" data from sensor data streams and therefore to transform sensor data streams at the right database schema spatio-temporal granularities.

Therefore, given a set of sensor data streams, we propose a declarative approach to build a relevant sensor database on top of any RDBMS with the following key features:

- Both spatio-temporal granularity hierarchies and STFDs are considered as first class-citizens.

- A specific axiomatisation of STFDs and an associated attribute closure algorithm are introduced, leading to an efficient normalization algorithm.

- Relevant data are automatically loaded into the sensor database thanks to SVAs.

We have implemented a prototype to deal with both database design and data loading. We propose a middleware to load on-the-fly relevant data from sensor streams into the granularity-aware sensor database. We have conducted experiments with synthetic and real-life sensor data streams coming from Intelligent Building. We have compared our solution with the baseline solution, with results order of magnitude better in terms of memory usage and performance.

Our contribution can be thought as a "declarative load shedding process" since we allow to prune data streams using declarative constraints, instead of using sampling techniques [KNV03].

1.4 Document Organization

This document contains seven chapters. In the current chapter, Chapter 1, we presented the general context of our work, our problematic as well as our contributions. The next chapter, Chapter 2, gathers the preliminary notions needed to support our proposition.

In Chapter 3, we present our main contribution: a model of granularity-aware spatio-temporal database. We define the formalism of our spatio-temporal structure and constraints, and we propose an axiomatisation of STFDs and the associated attribute closure algorithm, leading to a new normalization algorithm. In Chapter 4 we detail the implementation of the proposed declarative approach to build a granularity-aware sensor database, from the sensor database design to the data loading process. We present the architecture of the prototype of our system in Chapter 5, along with some experiments we conducted in order to compare our solution with the baseline solution. We also illustrate a demonstration of our prototype.

In Chapter 6, we discuss the corresponding related work. Finally, in Chapter 7, we conclude this thesis by summarizing our main contributions, and we outline possible future works.

CHAPTER 2

Preliminaries

Introduction

In this chapter, we present the preliminary notions that will be used throughout this document, in particular in Chapter 3. Those notions are required to introduce our contributions, namely a model for granularity-aware spatio-temporal database.

Many researches propose various temporal perspectives for databases. For instance, in [Sno95] the authors use *transaction time* and *valid time*; in [NA89] the authors propose a temporal relational data model with a *Time-Start* and a *Time-End*. In our work, we chose to extend the temporal model presented in [BJW00]: in our case with sensor data, we are more concerned by the data acquisition time than the validity time.

We start by defining general DB notions like *relation*, *stream*, *functional dependency*... Then, we define the spatio-temporal granularity hierarchies. Finally, we present some temporal notions.

2.1 General Notions

Here, we recall some fundamental notions for the modelling of relational data [LL12].

The universe $\mathcal{U}$ represents an infinite countable set of attributes or names of attributes. The domain $\mathcal{D}$ is an infinite countable set of constant values. For a given attribute $A \in \mathcal{U}$, $dom(A)$ denotes the domain of A such that $dom(A) \subseteq \mathcal{D}$.
We consider relation symbols $\mathcal{R}$ disjoint from $\mathcal{U} \cup \mathcal{D}$. The set of attributes of a relation symbol $R \in \mathcal{R}$ is defined by $schema(R)$: $schema(R) = \{A_1, \ldots, A_m\} \subseteq \mathcal{U}$. When clear from context, we use a shorter notation $R = \{A_1, \ldots, A_m\} \subseteq \mathcal{U}$.

Let R be a *relation symbol*. A tuple over R is an element of the Cartesian product $dom(A_1) \times \ldots \times dom(A_m)$. $Tup(R)$ is the set of all possible tuples defined over R. A relation r over R is a finite set of tuples over R.

A data stream can also be defined over a relation schema R, e.g. a stream s over R, as an infinite set of tuples over R.

Example 3 *Table 2.1 on page 7 consists in a subset of tuples of a temperature data stream over the following schema:*
$schema(temperatureSensors) =< value, sensor, timestamp >$.

$t_1 =< 21, s11, 2016/03/02\ 11{:}59{:}00 >$ *is a tuple from the temperature data stream* $temperatureSensors$.

2.2 Spatio-Temporal Granularity Hierarchies

In this section we intend to define the spatio-temporal notions that we need to define our data constraints. We first define a general notion of granularity and granularity hierarchy, borrowed from time granularity definition [BJW00]. We then apply these notions to the spatial and temporal dimensions.

2.2.1 Granularity

Let $\mathcal{I}$ be a countably infinite set of dimensional items (e.g. instants for the time dimension) and $\leq$ a total order on $\mathcal{I}$.

A granularity over $\mathcal{I}$ is a mapping G from $\mathbb{N}$ to $\mathcal{P}(\mathcal{I})$ where $\mathcal{P}(\mathcal{I})$ is the powerset of $\mathcal{I}$. A non-empty subset $G(i) \subseteq \mathcal{I}$, $i \in \mathbb{N}$, of a granularity G is called a granule. The granules in a granularity do not overlap.

More formally, for all integers i and $j \in \mathbb{N}$ with $i < j$ the following two conditions are satisfied:

1. $G(i) \neq \emptyset$ and $G(j) \neq \emptyset$ imply that each element of $G(i)$ is less than all elements of $G(j)$, and
2. for a given integer k such that $i < k < j$, if $G(i) \neq \emptyset$ and $G(j) \neq \emptyset$ then $G(k) \neq \emptyset$.

A granularity G is *finer than* a granularity H, denoted $G \preceq H$, if for each integer i, there exists an integer j such that $G(i) \subseteq H(j)$. Intuitively this means that each granule of H holds a set of granules of G. If G is finer than H, then H is coarser than G.

Example 4 *Figure 2.1 contains some examples of different granularities. The first group is a general granularity where $G_1 \preceq G_2$ and $G_2 \preceq G_3$. The first five granules of G_1 belong to the first granule of G_2, as $G_1(0) \cup G_1(1) \cup G_1(2) \cup G_1(3) \cup G_1(4) \subseteq G_2(1)$. The temporal granularities (e.g. second, minute and hour) and the spatial granularities (e.g. sensor, room, house) can also be represented with integers through a linear representation. For instance, minute(61) belongs to hour(2) and sensor(15) is in room(3).*

G is *collectively finer* than $\{G_1, \ldots, G_m\}$, denoted by $G \preceq_c \{G_1, \ldots, G_m\}$, if for each positive integer i, there exists $j, k \in \mathbb{N}$ such that $G(i) \subseteq G_k(j), 1 \leq k \leq m$. Intuitively this means that for each granule $G(i)$ taken independently there exists in the set of granularities $\{G_1, \ldots, G_m\}$ at least one granularity that is coarser than G. As a particular case, $G \preceq G'$ implies $G \preceq_c \{G'\}$.

When clear from context, we use a shorter notation for granules: $g \in G$ means that $\exists i \in \mathbb{N}$ such that $g = G(i)$.

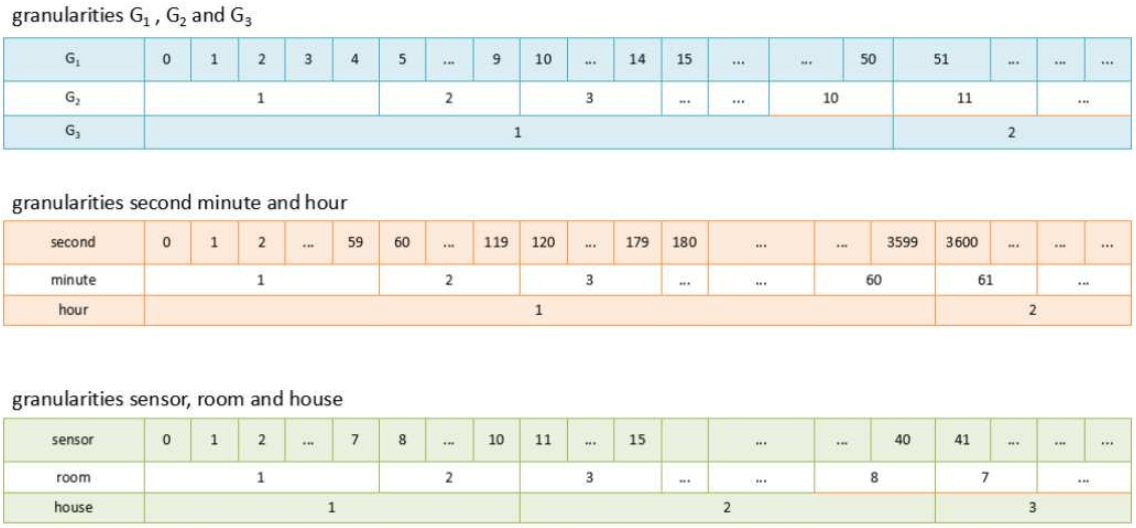


Figure 2.1 — Example of granularities

2.2.2 Granularity Hierarchy

A granularity hierarchy is a set of granularities with a partial order for which we require a lattice structure.

Let $\mathcal{G}$ be a set of granularities and $\preceq$ a partial order on $\mathcal{G}$. We assume that the poset $(\mathcal{G}, \preceq)$ is a lattice, meaning that each two-element subset $\{G_1, G_2\} \subseteq \mathcal{G}$ has a join (i.e. least upper bound) and a meet (i.e. greatest lower bound). For $X \subseteq \mathcal{G}$, $glb(X)$ (resp. $lub(X)$) denotes the greatest lower (resp. lowest upper) bound of X in $\mathcal{G}$. The greatest and least elements of $\mathcal{G}$ with respect to $\preceq$, or the top and bottom elements, are denoted by Top and $Bottom$, respectively. Top is the coarsest granularity of the hierarchy (it contains only one granule that includes all granules of finer granularities), whereas $Bottom$ is the finest granularity of the hierarchy.

2.2.3 Application to Spatio-Temporal Dimensions

For the spatial and the temporal dimensions, a hierarchy of granularities is defined accordingly. For the sake of clearness, we assume without loss of generality that a total order exists for time instants and locations, meaning that two time instants (resp. two locations) can always be compared.

In this setting, we define two granularity hierarchies $(\mathcal{T}, \preceq_t)$ and $(\mathcal{S}, \preceq_s)$. $\mathcal{T}$ is a set of temporal granularities and $\mathcal{S}$ a set of sensor location granularities.

Example 5 *An example of a spatial and a temporal granularity hierarchies is given in Figure 1.2 (page 6) where the arrows connect the coarser to the finer granularities.*

Example 6 *We consider our running example, Example 1 (page 6) of sensor data streams from intelligent buildings where data source and acquisition time are respectively indicated in attributes location and time. These attributes respectively follow the spatial and the temporal granularity hierarchies given in Figure 1.2 (page 6). A snapshot of such temperature sensor data stream is given in Table 2.1.*

value	*location*	*time*
21	oxygen:f1:h1:livingRoom:s11	2016/03/02 11:59:00
20	oxygen:f1:h1:kitchen:s21	2016/03/02 11:59:30
20	oxygen:f1:h1:livingRoom:s12	2016/03/02 12:01:00
23	oxygen:f1:h1:kitchen:s22	2016/03/02 12:01:00
20	oxygen:f1:h1:livingRoom:s11	2016/03/02 12:01:30
24	oxygen:f1:h1:bathroom:s31	2016/03/02 12:02:00
20	oxygen:f1:h1:livingRoom:s12	2016/03/02 12:02:30
23	oxygen:f1:h1:kitchen:s21	2016/03/02 12:15:00

Table 2.1 — Example of temperature sensor data stream

The temporal dimension has the advantage that each timestamp contains information relative to all the considered temporal granularities, e.g. through the date 2016/03/02 11:59:00 we know that the concerned year is 2016, month is March... Such information is needed in the location variable too. Without loss of generality, we shall assume hereinafter that the location attribute contains a string value containing the concatenation of all its granules. For instance, the location oxygen:$f1$*:*$h1$*:livingRoom:*$s11$ *indicates that data are coming from the sensor* $s11$ *which is installed in livingRoom of the house* $h1$ *on the first floor of the building oxygen. Ensuring such a representation makes it possible to compare the different spatial granules.*

2.3 Temporal Functional Dependency

Functional dependency in classical database theory serves as a data constraint between two sets of attributes. The use of functional dependencies in relational database is important for the database design and normalization. In this section, we present an extension of functional dependencies to the temporal dimension [WBBJ97, BJW00].

Let $(\mathcal{T}, \preceq_t)$ be a temporal granularity hierarchy and R a relation symbol. We consider the following two sets of attributes $X, Y \subseteq schema(R)$.

2.3.1 Functional Dependency

A Functional Dependency (FD) [LL12] specifies that attributes X *functionally determines* attributes Y in a relation r over R, if for each pair of tuples $t_1, t_2 \in r$, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$, i.e. the X-values determine the Y-values in the relation r. Such FD is denoted by $X \rightarrow Y$.

We say that the relation r over R satisfies a set F of dependencies over R, denoted $r \models F$, if for all dependencies $f \in F$ the *logical implication* $r \models f$ holds, i.e. each $f \in F$ is satisfied by r. The fact that the relation r satisfies the FD $X \rightarrow Y$, is denoted by the logical implication $r \models X \rightarrow Y$.

Example 7 *Consider the relation r over R, with schema$(R) = \{A, B, C\}$ as follows:*

Table 2.2 — A relation r

A	B	C
1	2	3
1	2	4
1	2	3

We can observe that: $r \models A \rightarrow B$ and $r \models A, C \rightarrow B$, but $r \not\models A \rightarrow C$ (two different values of C for $< A >=< 1 >$) and $r \not\models A, B \rightarrow C$ (two different values of C for $< A, B >=< 1, 2 >$).

2.3.2 Temporal Module

A *temporal module schema* [BJW00] is a pair $\mathbf{M} = (R, G)$, where $R \in \mathcal{R}$ is a relation schema and $G \in (\mathcal{T}, \preceq_t)$ is a time granularity.

A *temporal module* [BJW00] is a triple $\mathcal{M} = (R, G, \varphi)$, where (R, G) is a temporal module schema and φ is a mapping, called a *time windowing* function, from $\mathbb{N}$ to $\mathcal{P}(Tup(R))$. The function φ in a temporal module (R, G, φ) gives the tuples $\varphi(i) \subseteq Tup(R)$ that hold at a time granule $G(i)$ of granularity G.

Example 8 *Consider the following relation r' over R, with schema$(R) = \{A, B, C, time\}$:*
This relation can be presented using the module $\mathcal{M}_{r'} = (< A, B, C >, second, \varphi)$ where the mapping function φ is:
φ(2016/06/12 12:30:11) $= \{< 1, 2, 3 >\}$
φ(2016/06/12 12:30:55) $= \{< 1, 2, 3 >\}$
φ(2016/06/12 12:40:11) $= \{< 1, 2, 4 >, < 2, 3, 4 >\}$.

Table 2.3 — A relation r'

A	B	C	*time*
1	2	3	2016/06/12 12:30:11
1	2	3	2016/06/12 12:30:55
1	2	4	2016/06/12 12:40:11
2	3	4	2016/06/12 12:40:11

2.3.3 Temporal Functional Dependency

Temporal functional dependency (TFD) [BJW00] extends classical FD with a granularity, allowing to specify that the X-values determine the Y-values *at a given granularity* H in a temporal module (R, G, φ), i.e. the FD holds within each granules of H, but may not hold across two different granules of H or with another granularity.

More formally, a TFD over a temporal module schema $\mathbf{M} = (R, G)$ is denoted by $X \rightarrow_H Y$, where $X, Y \subseteq schema(R)$ and H is a time granularity.

A TFD $X \rightarrow_H Y$ is satisfied by a temporal module $\mathcal{M} = (R, G, \varphi)$, denoted $(R, G, \varphi) \models X \rightarrow_H Y$, if for all tuples t_1 and t_2 and positive integers i_1 and i_2, the following three conditions imply $t_1[Y] = t_2[Y]$:

1. $t_1[X] = t_2[X]$, and
2. $t_1 \in \varphi(i_1)$ and $t_2 \in \varphi(i_2)$, and
3. $\exists j$ such that $G(i_1) \cup G(i_2) \subseteq H(j)$.

The third condition means that both granules $G(i_1)$ and $G(i_2)$ of the time granularity G belong to the same granule of the time granularity H, i.e. $H(j)$. Thus, such TFD means that, during one granule of the time granularity H, the values for the attributes in X determine unique values for the attributes in Y.

Example 9 *We consider the temporal module $\mathcal{M}_{r'}$ given in Example 8 (page 21). We can observe that $\mathcal{M}_{r'} \models A \rightarrow_{hour} B$ and $\mathcal{M}_{r'} \models A \rightarrow_{minute} C$ (for minute 30 of hour 12 we have the same values of C for $< A >=< 1 >$) but $\mathcal{M}_{r'} \not\models A, B \rightarrow_{hour} C$ (for hour 12 we have different values of C for $< A, B >=< 1, 2 >$).*

Axiomatization

An axiomatization for TFDs has been proposed in [BJW00]. It allows to define the closure of TFDs and the closure of attribute at a given temporal granularity. We recall the axioms below:

- *Restricted reflexivity*: if $Y \subseteq X$, then $X \rightarrow_{Top} Y$

- *Augmentation*: if $X \rightarrow_G Y$, then $XZ \rightarrow_G YZ$
- *Extended transitivity*: if $X \rightarrow_{G_1} Y$ *and* $Y \rightarrow_{G_2} Z$, then $X \rightarrow_{glb(G_1,G_2)} Z$
- *Descendability*:

$$\text{if } \begin{cases} F \vdash X \rightarrow_{G_1} Y \\ \dots \\ F \vdash X \rightarrow_{G_n} Y \\ \textit{with } n \geq 1 \end{cases} \quad \text{then } F \vdash X \rightarrow_G Y \text{ such that } G \preceq_C \{G_1, \dots, G_n\}$$

Closure

The notation $F \vdash X \rightarrow_G Y$ means that the TFD $X \rightarrow_G Y$ can be derived from a set of TFDs F using the inference axioms.
The closure $\overline{F}^+$ of a set F of TFDs represents the set of all the TFDs derivable from F using those axioms: $\overline{F}^+ = \{X \rightarrow_G Y \mid F \vdash X \rightarrow_G Y\}$.
The closure of a set X of attributes with respect to a set F of TFDs, is defined as: $\overline{X}^+_F = \{(B, G) \mid X \rightarrow_G B \in \overline{F}^+$ and there is no $X \rightarrow_H B$ *in* $\overline{F}^+$*such that* $G \preceq H$ *and* $G \neq H\}$ where B is an attribute.

The implication problem, i.e. determining whether a TFD $X \rightarrow_G Y$ can be derived from a set F of TFDs, is polynomial. The implication of a TFD $X \rightarrow_G B$ can be determined from the closure of X with respect to F:

Property 1 $F \models X \rightarrow_G B$ ***iff*** *there exists* $\{(B, G_1), \dots, (B, G_m)\} \subseteq \overline{X}^+_F$ *such that* $G \preceq_C \{G_1, \dots, G_m\}$.

Conclusion

In this chapter, we gave the definitions that we require to present our contributions in the next chapter. We defined the spatio-temporal hierarchies that will be of major interest in order to elaborate our declarative approach. The temporal notions defined in this chapter, e.g. *temporal module schema*, *temporal module* and *TFD*, will be extended in the next chapter to this spatio-temporal hierarchy system.

Part II

Contributions

CHAPTER 3

Granularity-Aware Spatio-Temporal Database Model

Introduction

Dedicated Functional Dependencies (FDs) for sensor data streams have to take into account the temporal and the spatial dimensions. Many extensions of FDs to temporal DB have been proposed [Via87, Wij95, JSS96, WBBJ97, Wij99, CS14], but none of them extends FDs to both temporal and spatial dimensions.

In this chapter, we extend the temporal database definitions presented in Chapter 2 to take into account the spatial dimension. We start by extending the notion of temporal database to the notion of spatio-temporal database with granularity hierarchies. Next, we define Spatio-Temporal Functional Dependencies (STFDs) which represent the foundation of our approach. We propose a specific axiomatisation of STFDs and the associated logical implication that leads to a new normalization algorithm for spatio-temporal database schemas.

STFDs may remind us Roll-Up Dependencies (RUDs) [WN99] defined for OLAP DB that generalize FD to spatial and other dimensions. Their aim is to generalize temporal functional dependency to non-temporal dependencies in OLAP and data mining applications. However, we focus in this thesis on the logical reasoning with STFDs to define the closure of a set of attributes and a minimal cover of a set of STFDs and then to propose a new normalization algorithm.

3.1 Granularity-Aware Spatio-Temporal Database

In this section, we extend the temporal model presented in the preliminaries chapter with the spatial dimension. Thus, we define the notions of spatio-temporal module schema and spatio-temporal module.

As stated in the preliminaries (Chapter 2), let $\mathcal{U}$ be a universe (set of attributes) and $\mathcal{D}$ be a countably infinite set of constant values. Hereinafter, we shall use $G \in (\mathcal{S}, \preceq_s)$ as a spatial granularity and $H \in (\mathcal{T}, \preceq_t)$ as a temporal granularity. And for the sake of simplicity, we will denote $schema(R)$ by R, e.g. attribute $A \in R$, $R \subseteq \mathcal{U}$.

Definition 1 *A spatio-temporal module schema over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$ is a triplet $M = (R, G, H)$, where $R \subseteq \mathcal{U}$ is a relation schema, $G \in (\mathcal{S}, \preceq_s)$ is a spatial granularity and $H \in (\mathcal{T}, \preceq_t)$ is a temporal granularity.*

Definition 2 *A spatio-temporal module is a quadruple $\mathcal{M} = (R, G, H, \varphi)$, where (R, G, H) is a spatio-temporal module schema and φ is a mapping from $\mathbb{N} \times \mathbb{N}$ to $\mathcal{P}(Tup(R))$.*

Actually, the mapping function $\varphi(i, j)$ gives the tuples over R that hold at each couple of granules $(G(i), H(j))$. When clear from context, we shall use the time instants

and sensor locations instead of integers to describe a particular mapping function, e.g. $\varphi(building_x{:}house_y{:}room_z{:}sensor_i, 2016/03/02\ 11{:}59{:}00)$.

Definition 3 *A granularity-aware spatio-temporal database schema* $\mathbf{R}$ *over* $(\mathcal{U}, \mathcal{S}, \mathcal{T})$ *is a finite set of spatio-temporal module schemas over* $(\mathcal{U}, \mathcal{S}, \mathcal{T})$.

Definition 4 *A granularity-aware spatio-temporal database* $\mathbf{d}$ *over* $\mathbf{R}$ *is a finite set of spatio-temporal modules defined over* $\mathbf{R}$.

Example 10 *We reconsider the "raw" data stream, i.e. with data at the finest spatio-temporal granularities, given in Table 3.1 (same as Table 2.1 page 19).*

value	*location*	*time*
21	oxygen:f1:h1:livingRoom:s11	2016/03/02 11:59:00
20	oxygen:f1:h1:kitchen:s21	2016/03/02 11:59:30
20	oxygen:f1:h1:livingRoom:s12	2016/03/02 12:01:00
23	oxygen:f1:h1:kitchen:s22	2016/03/02 12:01:00
20	oxygen:f1:h1:livingRoom:s11	2016/03/02 12:01:30
24	oxygen:f1:h1:bathroom:s31	2016/03/02 12:02:00
20	oxygen:f1:h1:livingRoom:s12	2016/03/02 12:02:30
23	oxygen:f1:h1:kitchen:s21	2016/03/02 12:15:00

Table 3.1 — Example of temperature sensor data stream

Clearly, it can be represented at different granularities. For instance, we consider the spatio-temporal module which corresponds to the finest granularities $\mathcal{M}_1 = (R, G_1, H_1, \varphi_1)$ *with* $R = \{temperature\}$, $G_1 = sensor$, $H_1 = second$, *and with the windowing function* φ_1 *defined as follows:*

$\varphi_1(\texttt{oxygen:f1:h1:livingRoom:s11 , 2016/03/02 11:59:00}) = \{< 21 >\}$
$\varphi_1(\texttt{oxygen:f1:h1:kitchen:s21 , 2016/03/02 11:59:30}) = \{< 20 >\}$
$\varphi_1(\texttt{oxygen:f1:h1:livingRoom:s12 , 2016/03/02 12:01:00}) = \{< 20 >\}$
$\varphi_1(\texttt{oxygen:f1:h1:kitchen:s22 , 2016/03/02 12:01:00}) = \{< 23 >\}$
$\varphi_1(\texttt{oxygen:f1:h1:livingRoom:s11 , 2016/03/02 12:01:30}) = \{< 20 >\}$
...

The windowing function associates a set of temperature values (here, only one) for each couple composed by a spatial and a temporal granule. Here, we are positioned at the spatial granularity sensor and the temporal granularity second. For instance, the line: $\varphi_1(\texttt{oxygen:f1:h1:livingRoom:s12 , 2016/03/02 12:01:30}) = \{< 20 >\}$ *means that at the* 30^{th} *second of the minute* 1 *of the hour 12 of* 2^{nd} *of March 2016, the temperature of the sensor* $s12$ *installed in the living room in house* h_1, *that is at the first floor of the building oxygen, is 20.*

Example 11 *We consider the "raw" data stream given in Table 3.1. Those data can be represented through the module* $\mathcal{M}_2 = (R, G_2, H_2, \varphi_2)$ *with* $R = \{temperature\}$, $G_2 = room$, $H_2 = hour$ *and with the windowing function* φ_2 *defined as follows:*

$\varphi_2(\texttt{oxygen:f1:h1:livingRoom , 2016/03/02 11}) = \{< 21 >\}$
$\varphi_2(\texttt{oxygen:f1:h1:kitchen , 2016/03/02 11}) = \{< 20 >\}$
$\varphi_2(\texttt{oxygen:f1:h1:livingRoom , 2016/03/02 12}) = \{< 20 >\}$
$\varphi_2(\texttt{oxygen:f1:h1:kitchen , 2016/03/02 12}) = \{< 23 >\}$
$\varphi_2(\texttt{oxygen:f1:h1:bathroom , 2016/03/02 12}) = \{< 24 >\}$.

Here, we are positioned at the spatial granularity room and the temporal granularity hour. The line: $\varphi_2(\texttt{oxygen:f1:h1:livingRoom , 2016/03/02 11}) = \{< 21 >\}$ *means that at hour 11 of* 2^{nd} *of March 2016 the temperature of the living room in house* h_1*, that is at the first floor of the building oxygen, is 21. Some temperature values (i.e. temperature values that are equals all over the same hour for the same room) are not duplicated in the presentation of* φ_2 *since the mapping function output is a set of tuples. Actually, in the case of this example, each line of* φ_2 *contains just one temperature value. However it is possible to have more than one temperature value in other cases. For instance, if in table 3.1 the third line was:*

value	*location*	*time*
21	oxygen:f1:h1:livingRoom:s12	2016/03/02 12:01:00

instead of the current third line, we would have two temperature values for the living room in house $h1$ *during hour* 12*:*
$\varphi_2(\texttt{oxygen:f1:h1:livingRoom , 2016/03/02 12}) = \{< 21 >, < 20 >\}$.

3.2 Spatio-Temporal Functional Dependency

We have recalled the classical FD and the TFD in Chapter 2 and defined the notions of granularity-aware spatio-temporal database with spatio-temporal module schemas and spatio-temporal modules in the previous section. In this section, we define the notion of Spatio-Temporal Functional Dependency (STFD) and the associated concept of satisfaction. In the sequel we will distinguish two types of attributes:

- *data attributes (non spatio-temporal attributes)*: attributes from $\mathcal{U}$ that contain "sensor" values (or any values) from $\mathcal{D}$, and
- *spatio-temporal attributes*: attributes disjoint from $\mathcal{U}$ that allow to take into consideration granularity hierarchies and contain granules; they are associated to a given spatial and temporal granularity and are denoted with this granularity in superscript.

Let *location* and *time* be the special spatial and temporal attributes respectively. Intuitively, a STFD means that the X-values determine the Y-values within each granule of the given spatio-temporal granularities.

Let $G \in (\mathcal{S}, \preceq_s)$ be a spatial granularity, $H \in (\mathcal{T}, \preceq_t)$ a temporal granularity and $R \subseteq \mathcal{U}$ a relation schema.

Definition 5 *A Spatio-Temporal Functional Dependency (STFD) over* $(\mathcal{U}, \mathcal{S}, \mathcal{T})$ *is an expression of the form:* $X, location^G, time^H \rightarrow Y$ *where* $X, Y \subseteq \mathcal{U}$, $G \in (\mathcal{S}, \preceq_s)$ *and* $H \in (\mathcal{T}, \preceq_t)$.

We shall see that the case $X = \emptyset$ is meaningful for STFDs whereas the classical FD counterpart is almost useless (i.e. $\emptyset \rightarrow A$ means that in every possible relation, only one value for A is allowed).

When clear from context, the attributes *location* and *time* will be abbreviated by L and T respectively.

Example 12 *We want to express the fact that the temperature does not change upon some given spatial and temporal granularities, e.g. one may consider that the temperature of the same room does not change all along the same hour. This can be represented as follows:*
$\emptyset, L^{room}, T^{hour} \rightarrow temperature$ *or simply:* $L^{room}, T^{hour} \rightarrow temperature$.
Such data constraints represent an approximation over raw sensor data, that can be useful to express a required (or tolerated) simplification of data.

Satisfaction of a STFD

We first reformulate the satisfaction of classical FDs. A dependency f: $X \rightarrow Y$ is satisfied by a relation r over the relation schema R if for all tuples t_1, t_2, the following two conditions imply $t_1[Y] = t_2[Y]$:
(1) $t_1[X] = t_2[X]$
(2) $t_1, t_2 \in r$.

In order to take into account granularity hierarchies, the satisfaction of a STFD with respect to a spatio-temporal module is defined as follows:

Definition 6 *Let* $\mathcal{M} = (R, G, H, \varphi)$ *be a spatio-temporal module,* $X, Y \subseteq R$ *and a STFD* f: $X, L^{G'}, T^{H'} \rightarrow Y$.
f *is satisfied by* $\mathcal{M}$, *denoted* $\mathcal{M} \models f$, *if for all tuples* t_1 *and* t_2 *and positive integers* $i_1, i_2, j_1, j_2 \in \mathbb{N}$, *the following three conditions imply* $t_1[Y] = t_2[Y]$:
(1) $t_1[X] = t_2[X]$,
(2) $t_1 \in \varphi(i_1, j_1)$ *and* $t_2 \in \varphi(i_2, j_2)$, *and*
(3) $\exists\ i', j' \in \mathbb{N}$ *such that* $G(i_1) \cup G(i_2) \subseteq G'(i')$ *and* $H(j_1) \cup H(j_2) \subseteq H'(j')$.

This definition extends the satisfaction of classical FDs as follows:

(1) is the classical condition for FDs, i.e. the left-hand sides have to be equal on X for the two considered tuples;

(2) bounds tuples t_1, t_2 to hold at two spatio-temporal granules of $\mathcal{M}$ (equivalent to $t_1, t_2 \in r$);

(3) restricts eligible tuples t_1 and t_2 so that the union of their spatial (resp. temporal) granules of G (resp. H) has to be included in some granule of G' (resp. H'), i.e. the two spatial (resp. temporal) granules belong to a common granule in G' (resp. H').

Example 13 *If we take a look at data presented in Table 3.1 (page 27) and we consider the module $\mathcal{M}_1$ from Example 10 (page 27) we can observe that: $\mathcal{M}_1 \models L^{room}, T^{hour} \rightarrow$ temperature. In fact, it is easy to check that over the different spatio-temporal granules of the granularities room and hour the temperature values are the same, e.g. the temperature values of the different sensors belonging to the living room for the hour* 12 *are all equals to* 20 *degrees. Example 11 gives an insight of these values.*

Example 14 *We consider again data presented in Table 3.1 (page 27) and module $\mathcal{M}_1$ from Example 10 (page 27). We have: $\mathcal{M}_1 \not\models L^{house}, T^{hour} \rightarrow$ temperature.*

As for FDs, the non-satisfaction is easier to explain since we just need to exhibit a counterexample. For instance, the two first tuples given in Table 3.1 (page 27) form a counterexample: temperature values are different ($20 \neq 21$), but both sensors belong to the house $h1$ *(although to two different rooms) and values have both been produced at the hour* 11.

3.3 Logical Implication with STFDs

In this section, we are interested in the reasoning about STFDs. We first recall the implication problem. Then, we define inference axioms for STFDs, and check the soundness and the completeness of the proposed axiom system.

Let us consider a set F of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$, a STFD f: $X, L^G, T^H \rightarrow Y$ over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$ and a spatio-temporal module $\mathcal{M} = (R, G', H', \varphi)$ over the module schema $\mathbf{M} = (R, G', H')$, where $R \subseteq \mathcal{U}$ is a relation schema, $G, G' \in (\mathcal{S}, \preceq_s)$ are spatial granularities and $H, H' \in (\mathcal{T}, \preceq_t)$ are temporal granularities.

3.3.1 The Implication Problem

The implication problem is a fundamental notion in the reasoning about the classical functional dependencies. The aim is to check if a dependency can be proved or deduced from a given set of dependencies. To do so, there exist two approaches: the model theory and the proof theory [LL12]. In order to define and discuss the proposed axiom system for STFDs, we need to recall these approaches and to adapt them to the spatio-temporal hierarchies:

1. *The model theory*
   In order to resolve an implication problem we have to scan all the databases that verify the set F and check if they also verify f. More formally:

   $$F \models f$$

   holds whenever for all spatio-temporal modules $\mathcal{M}$ over $\mathbf{M}$, the following condition is true:

   $$\text{if } \mathcal{M} \models F \text{ then } \mathcal{M} \models f \text{ also holds.}$$

2. *The proof theory*
   Given an axiom system $\mathcal{A}$, for a class of integrity constraints, a *proof* (in $\mathcal{A}$) of an integrity constraint f from a given set of integrity constraints F is a finite sequence of integrity constraints, whose last element is f, such that each constraint in the said sequence is either in F or can be derived from a finite number of previous constraints in the sequence by using one of the inference rules of the axiom system $\mathcal{A}$.
   In order to resolve the implication problem, we need to check that there exists *a proof* in the axiom system of an integrity constraint f from a set of integrity constraints F. Therefore:

   $$F \vdash f$$

   denotes the fact that we can derive f from F using the axiom system (i.e. there exists a proof).

3.3.2 Inference Axioms for STFDs

In order to derive all the possible STFDs logically implied by a set of STFDs, we need to define the inference axioms corresponding to STFDs. The three following axioms are extended from the Armstrong's axioms [DC73] with the spatio-temporal granularities:

(A1) Reflexivity:
if $Y \subseteq X$ then $F \vdash X, L^{Top}, T^{Top} \rightarrow Y$
where Top refers to the granularity that is coarser than any granularity of the granularity hierarchy, i.e. any granule at any granularity belongs to the Top unique granule. In fact, the use of Top reflects the fact that the STFD $X, L^{Top}, T^{Top} \rightarrow Y$ means that the X-values determine the Y-values regardless of the granularity hierarchies, which is equivalent to the classical FD $X \rightarrow Y$.

(A2) Augmentation:
if $F \vdash X, L^G, T^H \rightarrow Y$ then $F \vdash X, Z, L^G, T^H \rightarrow Y, Z$
where X, Z is an abbreviation of $X \cup Z$.

(A3) Transitivity:
$$\text{if } \begin{cases} F \vdash X, L^G, T^H \rightarrow Y \\ F \vdash Y, L^G, T^H \rightarrow Z \end{cases} \quad \text{then } F \vdash X, L^G, T^H \rightarrow Z$$

Due to the presence of spatio-temporal granularities, a fourth axiom is required to allow variations between granularities (as it was the case in [BJW00] for TFDs).
The descendability axiom ensures the fact that whenever there exists a STFD over some spatio-temporal granularities G and H, this dependency holds at all couple of granularities *finer than* G and H. For instance, if some STFD holds at the granularities $(room, hour)$, it also holds at the granularities $(sensor, minute)$.

Before defining the descendability axiom, we need to define the collectively-finer relationship between couples of spatio-temporal granularities. We define that a couple of spatio-temporal granularities (G, H) is *collectively finer than* a set of spatio-temporal granularities $\{(G_1, H_1), \ldots, (G_n, H_n)\}$, denoted $(G, H) \preceq_C \{(G_1, H_1), \ldots, (G_n, H_n)\}$ if for each couple of granules (g, h), $g \in G$ and $h \in H$, there exists a couple of granules (g', h') where $g' \in G_i$ and $h' \in H_i$ and $(G_i, H_i) \in \{(G_1, H_1), \ldots, (G_n, H_n)\}$, such that $g \subseteq g'$ and $h \subseteq h'$. As a particular case, we can redefine the finer-than relation for a couple of spatio-temporal granularities. A granularity couple (G, H) is *finer than* another granularity couple (G', H'), denoted $(G, H) \preceq (G', H')$, if $G \preceq_s G'$ and $H \preceq_t H'$, e.g. the couple $(sensor, minute)$ is finer than the couple $(room, hour)$.

Therefore, the descendability axiom is defined as follows:

(A4) Descendability:
$$\text{if } \begin{cases} F \vdash X, L^{G_1}, T^{H_1} \rightarrow Y \\ \ldots \\ F \vdash X, L^{G_n}, T^{H_n} \rightarrow Y \\ with\ n \geq 1 \end{cases} \quad \text{then } F \vdash X, L^G, T^H \rightarrow Y$$
such that $(G, H) \preceq_C \{(G_1, H_1), \ldots, (G_n, H_n)\}$.

As a particular case, if $F \vdash X, L^{G'}, T^{H'} \rightarrow Y$ and $(G, H) \preceq (G', H')$ then $F \vdash X, L^{G}, T^{H} \rightarrow Y$.

From these four axioms we can derive the following two axioms. As presented in the preliminaries (Section 2.2 page 17), $G_3 = glb(G_1, G_2)$ is the greatest lower bound of G_1 and G_2 so we have $G_3 \preceq G_1$ and $G_3 \preceq G_2$.

(B1) Extended transitivity:
This axiom is an extension of the transitivity axiom **(A3)** that can be derived thanks to the descendability axiom **(A4)**. It tells that if the X-values determine the Y-values at the granularities G_1 and H_1 and if the Y-values determine the Z-values at the granularities G_2 and H_2 then we can say that the X-values determine the Z-values at the granularities $glb(G_1, G_2)$ and $glb(H_1, H_2)$. It is defined as follows:
$$\underline{\text{if}} \begin{cases} F \vdash X, L^{G_1}, T^{H_1} \rightarrow Y \\ F \vdash Y, L^{G_2}, T^{H_2} \rightarrow Z \end{cases} \quad \underline{\text{then}}\ F \vdash X, L^{G_3}, T^{H_3} \rightarrow Z$$
where $G_3 = glb(G_1, G_2)$ and $H_3 = glb(H_1, H_2)$.

(B2) Union:
We can extend the union axiom for classical dependencies with spatio-temporal granularities as follows:
$$\underline{\text{if}} \begin{cases} F \vdash X, L^{G}, T^{H} \rightarrow Y \\ F \vdash X, L^{G}, T^{H} \rightarrow Z \end{cases} \quad \underline{\text{then}}\ F \vdash X, L^{G}, T^{H} \rightarrow Y, Z.$$
This axiom can be derived from the axioms previously presented as follows:

- Using the augmentation axiom **(A2)**, the STFD $X, L^{G}, T^{H} \rightarrow Y$ gives $X, L^{G}, T^{H} \rightarrow X, Y$.
- Using the augmentation axiom **(A2)**, the STFD $X, L^{G}, T^{H} \rightarrow Z$ gives $X, Y, L^{G}, T^{H} \rightarrow Y, Z$.
- Using the transitivity axiom **(A3)**, the STFDs $X, L^{G}, T^{H} \rightarrow X, Y$ and $X, Y, L^{G}, T^{H} \rightarrow Y, Z$ give $X, L^{G}, T^{H} \rightarrow Y, Z$.

This axiom can be generalized thanks to the descendability axiom **(A4)** as follows:
$$\underline{\text{if}} \begin{cases} F \vdash X, L^{G_1}, T^{H_1} \rightarrow Y \\ F \vdash X, L^{G_2}, T^{H_2} \rightarrow Z \end{cases} \quad \underline{\text{then}}\ F \vdash X, L^{G_3}, T^{H_3} \rightarrow Y, Z$$
where $G_3 = glb(G_1, G_2)$ and $H_3 = glb(H_1, H_2)$.

3.3.3 Soundness and Completeness of the Proposed Axioms

We check now the soundness and the completeness of the stated axioms in order to investigate the inference structure for different sets of our constraints, i.e. STFDs. In order to check the soundness and the completeness of this axiom system, we aim to check if the conditions below are *true* for the set F of STFDs and for the STFD f:

(C1) Soundness:
if $F \vdash f$ holds then $F \models f$ also holds.

(C2) Completeness:
if $F \models f$ holds then $F \vdash f$ also holds.

When clear from context, we use a shorter notation for tuples from a spatio-temporal module (R, G', H', φ) that hold at coarser granules (g, h), with $g \in G$, $G' \preceq_s G$ and $h \in H$, $H' \preceq_t H$: $\varphi(g, h)$ means $\{t \mid t \in \varphi(i', j')$ with $G'(i') \subseteq g$ and $H'(j') \subseteq h\}$, the set of tuples from the module (R, G', H', φ) that hold at granules $g' \in G'$ and $h' \in H'$ that belong to granules $g \in G$ and $h \in H$, i.e. $g' \subseteq g$ and $h' \subseteq h$.

Lemma 1 *Soundness (for each axiom A1, A2, A3 and A4)*

(A1) Reflexivity
Let $\mathcal{M} = (R, G', H', \varphi)$ be a spatio-temporal module.
Let t_1, t_2 be tuples respectively from $\varphi(i_1, j_1)$ and $\varphi(i_2, j_2)$ where $i_1, i_2, j_1, j_2 \in \mathbb{N}$ and $t_1[X] = t_2[X]$.
Let $Y \subseteq X$.

We have $t_1[X] = t_2[X]$ then $\forall A \in X$, $t_1[A] = t_2[A]$.
As $Y \subseteq X$ then $\forall B \in Y$, $B \in X$ and then $t_1[B] = t_2[B]$.
So, whenever t_1 and t_2 agree on all attributes of X they agree on all attributes of Y independently of their respective granule (i_1, j_1) and (i_2, j_2), which gives the functional dependency $X \rightarrow Y$ that can be also written using the STFD formalism as $X, L^{Top}, T^{Top} \rightarrow Y$.

(A2) Augmentation
Let $\mathcal{M} = (R, G', H', \varphi)$ be a spatio-temporal module such that $\mathcal{M} \models X, L^G, T^H \rightarrow Y$.
Let t_1, t_2 be tuples respectively from $\varphi(i_1, j_1)$ and $\varphi(i_2, j_2)$ where $i_1, i_2, j_1, j_2 \in \mathbb{N}$ such that $\exists i' \in \mathbb{N}, G'(i_1) \cup G'(i_2) \subseteq G(i')$ and $\exists j' \in \mathbb{N}, H'(j_1) \cup H'(j_2) \subseteq H(j')$ and $t_1[X] = t_2[X]$.

Consider $Z \subseteq R$.
We have two cases:

1. $\exists A \in Z$ such that $t_1[A] \neq t_2[A]$, then the property holds as $t_1[X, Z] \neq t_2[X, Z]$.
2. $t_1[Z] = t_2[Z]$, then since $t_1[X] = t_2[X]$, $t_1[X, Z] = t_2[X, Z]$.
   Since $\mathcal{M} \models X, L^G, T^H \rightarrow Y$, we have $t_1[Y] = t_2[Y]$ and then $t_1[Y, Z] = t_2[Y, Z]$.

The STFD $X, Z, L^G, T^H \rightarrow Y, Z$ holds over $\mathcal{M}$.

(A3) Transitivity
Let $\mathcal{M} = (R, G', H', \varphi)$ be a spatio-temporal module such that $\mathcal{M} \models X, L^G, T^H \rightarrow Y$ and $\mathcal{M} \models Y, L^G, T^H \rightarrow Z$.
Let t_1, t_2 be tuples respectively from $\varphi(i_1, j_1)$ and $\varphi(i_2, j_2)$ where $i_1, i_2, j_1, j_2 \in \mathbb{N}$ such that $\exists i' \in \mathbb{N}, G'(i_1) \cup G'(i_2) \subseteq G(i')$ and $\exists j' \in \mathbb{N}, H'(j_1) \cup H'(j_2) \subseteq H(j')$ and $t_1[X] = t_2[X]$.

We have $\mathcal{M} \models X, L^G, T^H \rightarrow Y$ and $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$ at the granules $G(i')$ and $H(j')$.
Then according to the second STFD, whenever two tuples agree on Y they agree on Z. Thus we also have $t_1[Z] = t_2[Z]$ at the granules $G(i')$ and $H(j')$.

Therefore, at the granules $G(i')$ and $H(j')$ of the spatio-temporal granularities G and H, whenever two tuples agree on X they also agree on Z. The STFD $X, L^G, T^H \rightarrow Z$ holds over $\mathcal{M}$.

(A4) Descendability
Let $\mathcal{M} = (R, G', H', \varphi)$ be a spatio-temporal module such that

$$\mathcal{M} \models \begin{cases} X, L^{G_1}, T^{H_1} \rightarrow Y \\ \dots \\ X, L^{G_n}, T^{H_n} \rightarrow Y \\ with\ n \geq 1. \end{cases}$$

According to the set of STFDs, for each couple of spatio-temporal granularities $(G_i, H_i) \in \{(G_1, H_1), \dots, (G_n, H_n)\}$, whenever two tuples that hold at the same spatio-temporal granule agree on the X-values they agree on the Y-values, which means that if $t_1[X] = t_2[X]$ and $t_1, t_2 \in \varphi(g_i, h_i)$ then $t_1[Y] = t_2[Y]$ for all couple of granules (g_i, h_i), $g_i \in G_i$ and $h_i \in H_i$.
Moreover, according to the definition of the collectively-finer relationship for a couple of granularities, since we have $(G, H) \preceq_C \{(G_1, H_1), \dots, (G_n, H_n)\}$ then for any granules (g, h) of (G, H) there exists at least one couple (G_i, H_i) with a couple of granule (g_i, h_i) such that we have $g \subseteq g_i$ and $h \subseteq h_i$.

Then, the tuples at the granules (g, h) represent a subset of the tuples at the granules (g_i, h_i), which means that if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$ for all couple of granules (g, h) of granularities (G, H).
The STFD $X, L^G, T^H \rightarrow Y$ holds over $\mathcal{M}$.

Lemma 2 *Completeness*
Completeness requires the condition (C2), that is:
$F \models X, L^G, T^H \rightarrow Y \Rightarrow F \vdash X, L^G, T^H \rightarrow Y$.

Equivalently, we have to prove the contraposition:
$F \not\vdash X, L^G, T^H \rightarrow Y \Rightarrow F \not\models X, L^G, T^H \rightarrow Y$ which means that if we cannot derive $X, L^G, T^H \rightarrow Y$ from F (i.e. a proof using the inference axioms does not exist) then any spatio-temporal module satisfying F does not satisfy $X, L^G, T^H \rightarrow Y$. Therefore, $F \not\models X, L^G, T^H \rightarrow Y$ means that there exists a counterexample spatio-temporal module $\mathcal{M} = (R, G, H, \varphi)$ which satisfies F and does not satisfy $X, L^G, T^H \rightarrow Y$.
Then, there exists at least one couple of spatio-temporal granules g_0 and h_0, from the spatial granularity G and the temporal granularity H, where the set of tuples belonging to $\varphi(g_0, h_0)$ satisfies F but does not satisfy $X, L^G, T^H \rightarrow Y$. We build a possible instance of this set of tuples and we will prove that it consists in a counterexample. In order to make the reasoning simpler we consider f_0: $X, L^G, T^H \rightarrow Y$ and $r_0 = \varphi(g_0, h_0)$. Thus, $r_0 \models F$ but $r_0 \not\models f_0$.
The considered relation is presented in Table 3.2 where $X^\urcorner$ is defined as follows:

$$X^\urcorner = \{A_i \mid \exists f\text{: } X, L^G, T^H \rightarrow A_i \text{ such that } F \vdash f\}$$

i.e. the set of attributes A_i such that we can derive the STFD $X, L^G, T^H \rightarrow A_i$ from F. In particular, $X \subseteq X^\urcorner$. The relation contains only two tuples that agree on $X^\urcorner$ attributes and do not agree on any other attribute.

Table 3.2 — A counterexample relation $r_0 = \varphi(g_0, h_0)$

$X^\urcorner$	$schema(R) \setminus X^\urcorner$
1 ... 1	1 ... 1
1 ... 1	0 ... 0

In order to prove that $r_0 \models F$, we suppose the contrary: we suppose that $r_0 \not\models F$. Thus $\exists f'$: $V, L^G, T^H \rightarrow W$, $f' \in F$ such that $r_0 \not\models f'$. Then, by the construction of r_0, it follows that $V \subseteq X^\urcorner$ because $r_0 \not\models f'$ means that $\exists t_1, t_2 \in r_0$ such that $t_1[V] = t_2[V]$ and $t_1[W] \neq t_2[W]$, which means that V is in the $X^\urcorner$ side. It also follows that $\exists B \in W$ such that $B \in schema(R) \setminus X^\urcorner$. Since $V \subseteq X^\urcorner$, $\forall A_i \in V$ we have $F \vdash X, L^G, T^H \rightarrow A_i$. Then using the union axiom **(B2)**, we have $F \vdash X, L^G, T^H \rightarrow V$. And since $V, L^G, T^H \rightarrow W$ and $B \in W$, by the axioms of reflexivity **(A1)** and extended transitivity **(B1)**, we have $V, L^G, T^H \rightarrow B$. Therefore, $F \vdash X, L^G, T^H \rightarrow B$ by the axiom of transitivity **(A3)**. It follows that $B \in X^\urcorner$.
This leads to a contradiction.
Thus $r_0 \models F$.

Second, we want to show that $r_0 \not\models f_0$, with f_0: $X, L^G, T^H \rightarrow Y$.
We suppose the contrary: $r_0 \models f_0$.
By the construction of r_0, we have $\forall t_1, t_2 \in r_0\ t_1[X] = t_2[X]$, which then means that $t_1[Y] = t_2[Y]$. Thus, it follows that $Y \subseteq X^{\urcorner}$. This means that for all $A \in Y$ $F \vdash X, L^G, T^H \rightarrow A$, therefore $F \vdash X, L^G, T^H \rightarrow Y$ using the union axiom **(B2)**, which is exactly $F \vdash f_0$.
We have a contradiction as $F \nvdash f_0$ (from the initial contraposition).
Thus, f_0 is not satisfied by r_0.
Even though each dependency in F is satisfied by $r_0 = \varphi(g_0, h_0)$, it is not the case of $X, L^G, T^H \rightarrow Y$. So, we can conclude that whenever $F \nvdash X, L^G, T^H \rightarrow Y$ using the inference axioms, $F \not\models X, L^G, T^H \rightarrow Y$.
This means that the axioms are complete.

Theorem 1 *The inference axioms **(A1) (A2) (A3)** and **(A4)** for STFDs are sound and complete.*

The fact that these axioms are sound and complete ensures that from an initial set of STFDs we derive only logically implied STFDs (i.e. soundness) and we can obtain all logically implied STFDs (i.e. completeness). It is important to be able to derive all possible logically implied STFDs from an initial set in order to compute the minimal set of the representative constraints of a given database schema.

3.4 Normalization of Granularity-Aware Spatio-Temporal Database

Our aim is to extend the well known *synthesis algorithm* for database design [LL12] from a set of classical FDs in our setting.

With STFDs, we propose a normalization technique based on two main steps:

1. first, computing a minimal cover of a set of STFDs, and
2. then producing spatio-temporal modules.

3.4.1 Closure of Attributes

The closure of attributes plays a crucial role for reasoning on classical FDs. In fact, the closure of attributes with respect to a set F of FDs is an efficient tool to decide whether or not a given FD is implied by F. We now generalize the closure of attributes to STFDs on the steps of [BJW00] where they extended FD with temporal granularities.

Let $\mathcal{M} = (R, G, H, \varphi)$ be a spatio-temporal module over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$, F a set of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$ and $X \subseteq \mathcal{U}$.

The closure of X with respect to F is denoted by X_F^+. It gathers the attributes B_i that can be derived from X at granularities (G, H). Thus, X_F^+ contains elements of the form (B, G, H) over $\mathcal{U} \times \mathcal{S} \times \mathcal{T}$ and is defined as follows:

Definition 7 $X_F^+ = \{(B, G, H) \mid F \vdash X, L^G, T^H \rightarrow B$ *and* $\nexists f : X, L^{G'}, T^{H'} \rightarrow B$ *with* $G \preceq_s G', H \preceq_t H'$ *and* $(G \neq G'$ *or* $H \neq H')$ *such that* $F \vdash f\}$.

The closure will be useful in the reasoning about STFDs in order to compute a minimal set of STFDs from which we can generate a normalized sensor database schema.

Algorithm 1 computes the finite closure of a set of attributes X with respect to F. This algorithm is a generalization of the classical closure algorithm for FDs [LL12] taking into account the granularities. Its basic idea is to compute progressively the set X_F^+. Line 1 encodes the first axiom (A1). The following procedure is repeated over all STFDs until a fix point is reached (line 13). For each STFD of line 4, if $A_1, .., A_k$ appears in the current closure $X_{previous}$, then $B_1, .., B_m$ are added to X_F^+ with the corresponding spatial and temporal granularities. Line 14 ensures that the closure is composed of elements with incomparable granularities for the same attribute.

Example 15 *We consider* $\mathcal{U} = \{temperature, humidity, luminosity, CO2\}$*, four sensor types sending temperature, humidity, luminosity and* CO_2 *values, the granularity dimensions given in Figure 1.2 (page 6). Let us consider the following set* F *of STFDs:*
$F = \{location^{room}, time^{hour} \rightarrow temperature;$
$location^{house}, time^{day} \rightarrow humidity;$
$location^{room}, time^{day} \rightarrow luminosity;$
$location^{room}, time^{minute} \rightarrow CO2;$
$location^{room}, time^{hour} \rightarrow humidity;$
$location^{room}, time^{minute} \rightarrow temperature;$
$location^{house}, time^{hour} \rightarrow humidity;$
$location^{sensor}, time^{hour} \rightarrow luminosity;$
$location^{room}, time^{hour} \rightarrow CO2 \ \}$
The closure of $\{temperature\}$ *with respect to* F *is:*
$\{temperature\}_F^+ = \{(temperature, Top, Top), (humidity, house, day), (luminosity, room, day), (CO2, room, hour)\}$.

As we already mentioned, the closure of attributes allows us to know whether or not a given STFD is implied by a set F of STFDs, as shown in the following property.

Property 2 $F \vdash X, L^G, T^H \rightarrow B \Leftrightarrow \exists Y \subseteq X_F^+$ *such that* $Y = \{(B, G_i, H_i) \mid i \in 1..n\}$ *with* $(G, H) \preceq_c \{(G_1, H_1), \ldots, (G_n, H_n)\}$.

Algorithm 1 ClosureAttribute

Require:

F: a set of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$

$X \subseteq \mathcal{U}$

Ensure:

X_F^+: the finite closure of X with respect to F

1: $X_F^+ := \{(A, Top, Top) \mid a \in X\} \cup \{(\emptyset, Top, Top)\}$
2: **repeat**
3: $X_{previous} := X_F^+$
4: **for each** $A_1, .., A_k, L^G, T^H \rightarrow B_1, .., B_m \in F$ **do**
5: **for each** $\{(A_1, G_1, H_1), .., (A_k, G_k, H_k)\} \subseteq X_{previous}$ **do**
6: $G' := glb(G_1, .., G_k, G)$
7: $H' := glb(H_1, .., H_k, H)$
8: **for each** $B \in \{B_1, .., B_m\}$ **do**
9: $X_F^+ := X_F^+ \cup \{(B, G', H')\}$
10: **end for**
11: **end for**
12: **end for**
13: **until** $X_F^+ = X_{previous}$
14: Minimize X_F^+ such that whenever we have two elements (A, G, H) and (A, G', H') with $G \preceq_s G', H \preceq_t H'$ and ($G \neq G'$ or $H \neq H'$) the element (A, G, H) is removed

To prove Property 2 we discuss both senses of this equivalence.

1^{st} implication: $\Rightarrow$

We suppose that $F \vdash X, L^G, T^H \rightarrow B$.

Therefore, either $(B, G, H) \in X_F^+$ or $(B, G, H) \notin X_F^+$.

If $(B, G, H) \in X_F^+$, $Y = \{(B, G, H)\} \subseteq X_F^+$ and $(G, H) \preceq (G, H)$ which is equivalent to $(G, H) \preceq_c \{(G, H)\}$.

In the case that $(B, G, H) \notin X_F^+$, as $F \vdash X, L^G, T^H \rightarrow B$, we know that (through the definition of X_F^+) $\exists (B, G', H') \in X_F^+$ such that $(G, H) \preceq (G', H')$ that implies $(G, H) \preceq_c \{(G', H')\}$.

Thus, $F \vdash X, L^G, T^H \rightarrow Y \Rightarrow \exists Y \subseteq X_F^+$ such that $Y = \{(B, G_i, H_i) \mid i \in 1..n\}$, $(G, H) \preceq_c \{(G_1, H_1), \ldots, (G_n, H_n)\}$.

2^{nd} implication: $\Leftarrow$

We suppose that $\exists Y \subseteq X_F^+$ such that $Y = \{(B, G_i, H_i) \mid i \in 1..n\}$ with $(G, H) \preceq_c \{(G_1, H_1), \ldots, (G_n, H_n)\}$.

This means, by the definition of X_F^+, that we have:

$$\begin{cases} F \vdash X, L^{G_1}, T^{H_1} \rightarrow B \\ \dots \\ F \vdash X, L^{G_n}, T^{H_n} \rightarrow B \end{cases}$$

Therefore, using the descendability axiom ***(A4)*** and the fact that $(G, H) \preceq_c \{(G_1, H_1), \dots, (G_n, H_n)\}$, we can derive $F \vdash X, L^G, T^H \rightarrow B$.
Thus, $\exists Y \subseteq X_F^+$ such that $Y = \{(B, G_i, H_i) \mid i \in 1..n\}$ with $(G, H) \preceq_c \{(G_1, H_1), \dots, (G_n, H_n)\} \Rightarrow F \vdash X, L^G, T^H \rightarrow B$.

Example 16 *Let us consider the set F of STFDs given in Example 15. We consider the following STFDs:*
f_1: $temperature, location^{room}, time^{hour} \rightarrow luminosity$; and
f_2: $temperature, location^{openspace}, time^{day} \rightarrow humidity$;

As $(luminosity, room, day) \in \{temperature\}_F^+$ and $hour \preceq_t day$ we have $F \vdash f_1$. However, $F \nvdash f_2$ because $\{temperature\}_F^+$ only contains $(humidity, house, day)$ and $openspace \not\preceq_s house$.

3.4.2 Minimal Cover of STFD Set

Our initial aim is to design database schema using spatio-temporal data constraints, i.e. STFDs. To do so, it is interesting to have a set of STFDs with a small size. Thus, having an initial set F of STFDs over R we want to build a set F' of STFDs over R that is equivalent to F but having a smaller size.

In order to build F', we need the notion of minimal cover. In the previous sections, we defined inference axioms for the data constraints which allowed us to define the closure of attributes with respect to a set of STFDs. The closure of attributes allows us to compute the closure of a set of STFDs.

Definition 8 *The closure of a set F of STFDs, denoted F^+, is the set of all the STFDs that can be obtained from F using the inference axioms, i.e. $F^+ = \{X, L^G, T^H \rightarrow Y \mid F \vdash X, L^G, T^H \rightarrow Y\}$.*

We can now define a cover of a set of STFDs and then a minimal cover.

Definition 9 *A set F' of STFDs is a* cover *of a set F of STFDs if $F^+ = F'^+$.*

The size of a set of dependencies F, denoted $|F|$, is the number of STFDs belonging to F.A minimal cover of a set of attributes is then defined as follows:

Definition 10 *A cover F' of a set F of STFDs is* minimal *if there does not exist a cover F'' of F such that $|F''| < |F'|$.*

In Algorithm 2, we sketch the main steps to compute a minimal cover. First, we saturate the initial set of STFDs with STFDs induced by the closure of each set of attributes (lines 2-6). Then, we apply the classical minimization procedure to our spatio-temporal model (line 7). This procedure consists on using the algorithms $LeftReduce$ and $NonRedundant$ as for classical FDs. This step consists in removing redundant STFDs and extra attributes (left-hand sided attributes and right-hand sided attributes) and then merging STFDs whose left-hand attributes are the same by taking the union of their right-hand attributes. These algorithms are inspired from [Mai83] and are presented in Appendix A.

Algorithm 2 MinimalCover

Require:
F: a set of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$

Ensure:
F': a minimal cover of F

1: $F' := \emptyset$
2: **for each** $X, L^G, T^H \rightarrow Y \in F$ **do**
3: **for each** $(A, G', H') \in X_F^+$ **do**
4: $F' := F' \cup \{X, L^{G'}, T^{H'} \rightarrow A\}$
5: **end for**
6: **end for**
7: $F' := Reduce(F')$.

Example 17 *We consider the set F of STFDs in Example 15. Using Algorithm 2 we get the following minimal cover F':*
$F' = \{location^{room}, time^{hour} \rightarrow temperature, CO2;$
$location^{house}, time^{day} \rightarrow humidity;$
$location^{room}, time^{day} \rightarrow luminosity\}$

3.4.3 Normalization of Granularity-Aware Spatio-Temporal Database Schema

At this point, the granularity-aware spatio-temporal database schema can be deduced from the obtained minimal cover: for each STFD, a module is generated. Algorithm 3 presents the main steps allowing to get a granularity-aware spatio-temporal database schema from a set of STFDs.

Algorithm 3 Normalization

Require:

F: a set of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$

Ensure:

$\mathbf{R}$: a granularity-aware sensor database schema

$\mathbf{R} := \emptyset$

1: $F' := MinimalCover(F)$
2: **for each** $A_1, .., A_k, L^G, T^H \rightarrow B_1, .., B_m \in F'$ **do**
3: $R := \{A_1, .., A_k, B_1, .., B_m\}$
4: $M := (R, G, H)$
5: $\mathbf{R} := \mathbf{R} \cup M$
6: **end for**

Example 18 *Continuing the previous example, we obtain a granularity-aware sensor database schema* $\mathbf{R} = \{M_1, M_2, M_3\}$ *with:*
$M_1 = (< temperature, CO2 >, room, hour)$,
$M_2 = (< humidity >, house, day)$, *and*
$M_3 = (< luminosity >, room, day)$.

Conclusion

This chapter contains our major contributions namely the definition of STFDs, an axiomatization and their related normalization algorithm. We did not discuss the decomposition and keys, as in the case of classical FDs, because our approach primarily aims to approximate spatio-temporal data. This issue is a perspective for future work.

We argue that STFDs are quite natural to express *declarative constraints* over sensor data streams and provide a powerful abstraction mechanism towards granularity-aware spatio-temporal sensor database design. In fact, as both the spatio-temporal hierarchies and stream schemas are easy to define, this approach can be successfully applied for different contexts with application designer and/or domain expert defining their input data as well as their own constraints. It is worth noting that every module of such a database schema is easily implementable on top of any RDBMS.

CHAPTER 4

Granularity-Aware Sensor Database Architecture

Introduction

In the context of long-term reporting applications, considering all incoming data from sensor data streams leads us to face huge amounts of data that becomes difficult to manage and to query. Our idea is to define some application specific data constraints allowing to restrain sensor data storage to relevant data and to prune data that are evaluated as not relevant. This may remind us the load shedding process [TÇZ+03]: in order to lighten the charge of the system in stream environments, different heuristics are proposed to define which data one can get rid of.

In this thesis, the choice of the relevant data is done thanks to user-defined semantic information which we define in this chapter as a form of database schema annotations, namely Semantic Value Assumption (SVA). These annotations are similar to a data exchange mechanism [FKP05]. They are inspired from interval-based semantic assumptions designed for temporal databases [BJW00] where point-based and interval-based semantic assumptions can be used to derive or compress temporal data across granularities. The idea behind the use of SVAs is to annotate the database schema with some semantic information in order to declaratively configure the loading of the relevant data into the sensor database.

In Chapter 3, we defined data constraints through spatio-temporal granularity hierarchies, i.e. STFDs. STFDs can be used to express application-specific approximations on data according to the spatio-temporal granularities that match the application requirements. Then, we proposed a normalization algorithm allowing to define database schema using STFDs. These formal notions allowed us to define a declarative approach to model a granularity-aware spatio-temporal database. Therefore, in this chapter, we propose a granularity-aware sensor database architecture dealing with, in a declarative manner, both the database schema design and the specification of the data loading from sensor data streams.

4.1 Sensor Database Schema Design

Our objective is to establish a sensor database design that eases the storage of spatio-temporal data streams and subsequently enhances the execution of queries. The underlying data storage system prunes incoming sensor data to keep only representative data, i.e. data that the user judges representative of her (application) requirements.
We claim that storing relevant data according to user/application requirements at specific spatio-temporal granularities enables the use of simpler queries: simple SELECT queries with simple WHERE clauses instead of queries with nested SELECT and JOINS. Actually, the stored relevant data can be considered as prior-computed query results like materialized views.

We aim to capture these application requirements thanks to spatio-temporal constraints, namely STFDs, which allow to express the dependencies between sets of attributes over spatio-temporal granularities. For example, thanks to STFDs, we can express the fact that room temperature values do not vary within each hour and/or within each room, although several measures arrive at different moments and from different temperature sensors in the sensor data stream. This approximation may correspond to relevant granularities of measures for some application requirements.

4.1.1 Abstract Schema

In order to design the granularity-aware sensor database schema, the schema of the sensor data streams should be known and the spatio-temporal granularity hierarchies should be defined by the designer. The designer can then define the STFDs that express her (application) requirements, i.e. define dependencies between stream attributes at given spatio-temporal granularities.

We can then apply the normalization algorithm on the set of STFDs which leads to the initial database schema, the so-called *abstract schema*, composed of spatio-temporal module schemas. The keys of the proposed relations are the attributes of the left-hand part of each STFD including the location and time attributes. In order to take into account the application requirements concerning the choice of the relevant data, the abstract schema has now to be augmented with semantic information.

4.1.2 Semantic Value Assumption

Given a set of sensor data streams and granularity hierarchies, constraints for long-term storage can be defined as a set of STFDs, and a sensor database schema can be obtained from these constraints. At this stage the objective is to load stream data into the corresponding spatio-temporal modules at the right granularities. In fact, the idea behind using STFDs to approximate data is to store only the values corresponding to each couple of spatio-temporal granules of the defined granularities. In this section we explain how we aggregate the representative data.

At given spatio-temporal granularities, we have different alternatives to choose relevant data within each spatio-temporal granule: each one could be seen as an aggregation of values with some aggregate functions. These functions can be simple aggregations (e.g. first, max...) or more elaborated ones (e.g. average of the 3 first values corresponding to a given valid domain) depending on the application context.

To do so, we introduce the so-called *Semantic Value Assumptions* (SVA) allowing to declaratively define the values to be selected or computed. Actually, a SVA is an assignment of an aggregation function to an attribute in a spatio-temporal module

schema. Therefore, we intend through our annotations to specify the spatio-temporal module schema, the concerned attribute and the desired aggregation function.

The definition of SVA is as follows:

Definition 11 *Let $M = (R, G, H)$ be a spatio-temporal module schema. Let $A \in R$ be an attribute and f an aggregation function (first,avg...) over the spatio-temporal granularities G and H. A SVA is a triplet (A, f, M).*

Example 19 *Consider $M_2 = (< humidity >, house, day)$ in Example 18 (page 42). The SVA $(humidity, first, M_2)$ means that the* first *humidity value per house per day has to be kept in M_2. However, the SVA $(humidity, avg, M_2)$ means that the average of the humidity values per house per day has to be kept in M_2.*

Such SVA has a similar effect to the following GROUP BY query in classical relational database:

```
SELECT AVG(humidity), trunc_S(location,'house'), trunc_T(time,'day')
FROM R_humidity_sensors
GROUP BY trunc_T(time,'day'),
         trunc_S(location,'house');
```

where the relation R_humidity_sensors contains all the humidity values coming from the humiditySensors data stream and $trunc\_T(time,'day')$ (resp. $trunc\_S(location,'house')$) truncates the value of the attribute time (resp. location) up to the granules of the temporal granularity day (resp. the spatial granularity house).

Example 20 *Figure 4.1 gives an example of some SVAs and the obtained results upon the Table 2.1 (page 19) if we are interested in the spatio-temporal granularities (house,hour). For instance, the SVA* `first` *when applied within the granules (h1,12), i.e. in house h1 at hour 12, gives the value 20. However, the SVA* `max` *when applied within the same granules gives the value 24.*

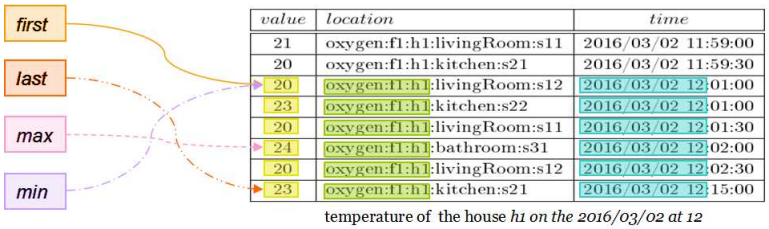


Figure 4.1 — An example of SVAs

These specific annotations allow, in a declarative manner, to annotate the database schema with the semantic information required to indicate which value is representative

in each granule. Depending on application requirements, SVAs can be very complex functions that allow to define complex aggregations and even to avoid noisy and incomplete data issues.

4.1.3 Concrete Schema

SVAs allow to indicate, in a declarative manner, which value is representative within each granule. Annotating an abstract schema with user-defined SVAs leads to a *concrete schema*, which can be implemented on top of classical RDBMS. Each non-key attribute from each module schema has to be associated with at least one SVA.

Example 21 *We consider M_2 and the SVA $(humidity, first, M_2)$.*
A corresponding relational table could be:
HumiditySensor(humidity_first, location, time)[1]
where humidity_first holds the first humidity value at the given spatio-temporal granularity, i.e. house and day.

Whenever multiple SVAs exist for a given couple $(attribute, module\ schema)$, new attributes could be created in the *concrete schema* of the underlying RDBMS.

Example 22 *Let us consider again M_2 and two SVAs: $(humidity, first, M_2)$ and $(humidity, avg, M_2)$.*
A corresponding relational table could be:
HumiditySensor(humidity_first, humidity_avg, location, time)
where humidity_first holds the first humidity value and humidity_avg the average value at the given spatio-temporal granularity.

4.1.4 Synthesis Example

Example 23 *Figure 4.2 sketches an example of the schema generation process. The first step consists in computing the minimal cover of the initial set of STFDs with respect to the user-defined spatio-temporal hierarchies. In this step, the second STFD (i.e. $A, L^{room}, T^{day} \rightarrow B$) is removed as it is logically implied by the first one.*
The spatio-temporal module schema is then generated based on the obtained minimal cover which leads to the abstract database schema.
Here, for the first module schema, the requirements specify to keep two values of the attribute C: the first value and the average value per house and day. Therefore, we have two defined SVAs for C in M_1: $(C, first, M_1)$ and (C, avg, M_1).
The abstract schema is then fully annotated with SVAs, leading to the concrete database schema.

[1] The underlined attributes are keys of the concerned relations.

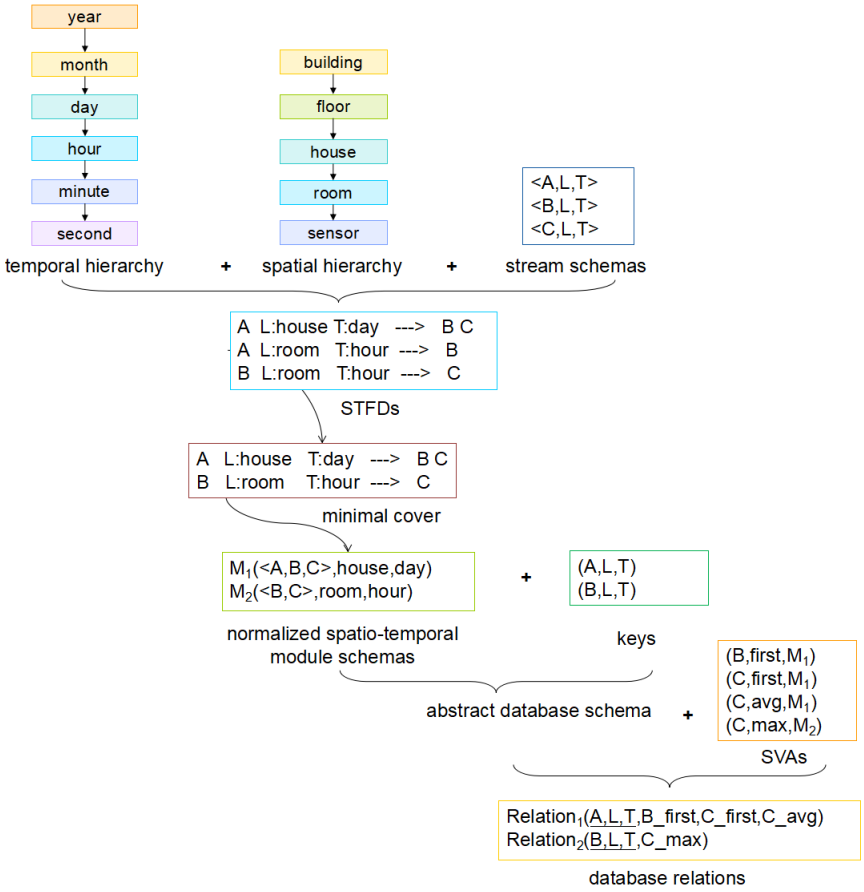


Figure 4.2 — An example of the database schema generation

4.2 Data Loading Specification

SVAs represent an important semantic information in the database design level as well as in the data loading procedure from data streams. Actually, the specification of the relevant tuples is defined thanks to SVAs.
For instance, consider the SVA (C, max, M_2) from Example 23 (page 47) where $\mathbf{M_2} = (< B, C >, room, hour)$. Through this SVA, we define that the relevant values for attribute C are the maximal ones among all the received values corresponding to each granules of the couple of granularities $(room, hour)$. This means that for each hour we keep just one C value per each room: this value corresponds to the maximum value for each hour among the different values sent by the different sensors belonging to the same room.

The aim at this point is to ensure data selection and storage according to the SVAs. The data loading process should enforce the real-time application of SVAs. For each SVA, a specific data wrapper is configured accordingly. These data wrappers will

be in charge of monitoring incoming data from pertinent data streams. If necessary, according to the aggregation function, a data wrapper uses data buffers in order to compute the final value. Once the data wrapper obtains the final value, it stores it in the appropriate database relation. It is the time dimension that mainly synchronises the storage process, i.e. once the current temporal granule finishes, the aggregated value is stored in the database relation.

As the user-defined constraints are the basis of this approach, we suppose that their definition is due to a serious analysis of the application requirements. Thus, the loaded data will match the defined constraints and be approximated according to the actual requirements.

4.3 Architecture

In this section, we sketch an architecture for the declarative approach that we have defined in the previous sections in order to implement a granularity-aware sensor database. This architecture gathers both database design and data stream loading.

The purpose behind the choice of a declarative approach is to let the user define simple inputs while all the complex reasoning is left to the system. Therefore, the user inputs are: spatio-temporal granularity hierarchies, stream schema, STFDs and SVAs. Then, the system proposes a database schema according to the user-defined inputs. Once the proposed schema is validated by the user, the database is created and the data stream monitoring is started in order to load incoming sensor data on-the-fly. Then, the stored data can be consulted by simple SQL queries.

An overview of the proposed architecture is presented in Figure 4.3 which highlights the following two levels:

1. ***Sensor database*** **design**
   Given a spatial and a temporal granularity hierarchies, this module determines a *granularity-aware sensor database* schema from a set of data streams, a set of STFDs and a set of SVAs using the normalization algorithm presented in Chapter 3. It is possible to implement SVA in different manners namely using triggers or a dedicated middleware. As triggers do not scale to important data stream loads [ACÇ+03, GÖ10], in the following, we chose to implement a middleware implementing a set of predefined SVAs as configurable data wrappers.
   Once the granularity-aware sensor database schema is defined, this module allows to create the corresponding database relations in SQL, i.e. the data description language implemented on top of any RDBMS.

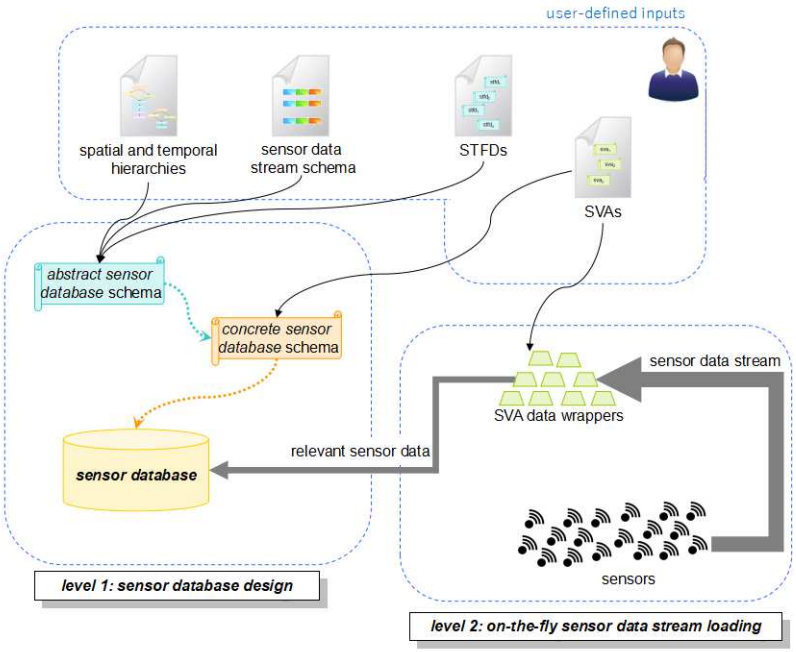


Figure 4.3 — An overview of the proposed architecture

2. **On-the-fly data loading**
   Once the database is created, this level ensures the selection of the relevant data from the received sensor data streams.
   The storage synchronisation is based on the temporal hierarchy. Actually, the system uses data buffers to store transitional values and the final value will be stored when the concerned temporal granule is over. Thanks to SVA data wrappers, this middleware monitors sensor data, chooses or computes data to be stored at specific granularities, and prunes the rest.

Conclusion

In this chapter, we presented the architecture for a granularity-aware sensor database based on the spatio-temporal database model defined in Chapter 3. This architecture contains two main parts. The first one concerns the design of the sensor database schema. It first generates an abstract schema from the application-specific STFDs. Then, this schema is annotated with SVAs leading to a concrete schema that will be used for the creation of the database relations. The second level concerns the specification of the on-the-fly data loading from sensor data streams into the database relations. At this level, each defined SVA will be associated to a SVA data wrapper that enforces the real-time application of its aggregation function in order to select or compute the relevant data value of each spatio-temporal granule.

CHAPTER 5

Prototype & Experiments

Introduction

In this chapter, we detail the implementation of a prototype for long-term storage of sensor data streams following the architecture proposed in the previous chapter. We then present the experiments we have conducted on synthetic and real data streams from the context of intelligent buildings. We compare our system to a baseline solution in terms of memory usage and query efficiency. Finally, we present some use cases of our prototype with a concrete schema design example in order to highlight the simplicity of this process.

5.1 Prototype Architecture

In order to check the validity of our proposition, we implemented a prototype according to the architecture proposed in Chapter 4 (Section 4.3 page 49).
A detailed view of the prototype architecture is presented in Figure 5.1.

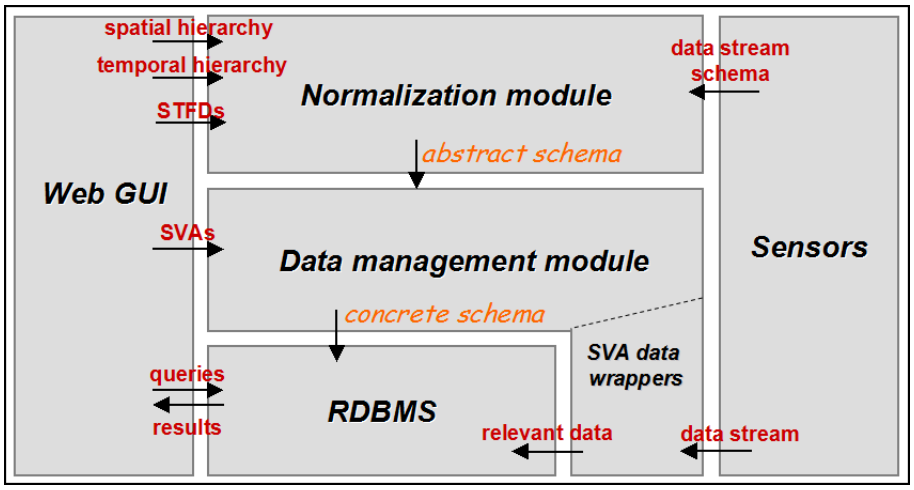


Figure 5.1 — A detailed view of the proposed architecture

The prototype contains the following main modules:

1. **Normalization module:**

   This module is in charge of the logical reasoning about STFDs. It takes as inputs the user-defined entries, i.e. spatio-temporal hierarchies, stream schemas and STFDs.
   This module implements the algorithms presented in Chapter 3, i.e. ClosureAttribute, MinimalCover and Normalization. Thereby, it computes the minimal cover of the given set of STFDs. The minimal cover is then transformed to the corresponding module schema. This process leads to the database *abstract schema*.

2. **Data management module:**

   In this module, the user defines the SVAs corresponding to the proposed abstract schema. Therefore, this module updates the abstract schema with the corresponding semantic annotations which leads to the *concrete schema*. Once the user validates the obtained sensor database schema, the database relations are created on the underlying RDBMS.
   This module also ensures the selection of the relevant data from sensor data streams, corresponding to the specification given by the user-defined set of SVAs. Each SVA indicates which value should be kept in the database over the different spatio-temporal granules. For each SVA the system instantiates a specific *SVA data wrapper*. Each SVA data wrapper monitors the sensor data stream concerning its attribute. According to its spatio-temporal granularities and to its aggregation function it computes the accurate values. A data wrapper mainly uses buffers in order to store the transitional data. For instance, computing the average of temperature, when scaling from *second* to *hour*, requires a buffer that keeps the current computed average while the received temperature values belong to the same hour. When the *hour* changes, the obtained average is stored in the database.

3. **Sensor module:**

   This module receives sensor data streams. It is the interface between the sensors and the system as it gathers sensor data and links it to the user-defined spatio-temporal hierarchies. Actually, as real-life sensors usually are identified by physical locations or unique identifiers, their locations are usually not based on any spatial hierarchy. The translation of such identifiers to ones that respect the user-defined spatial hierarchy is done by this module which, in the initialisation phase, reads a file containing the correspondence between the installed sensor identifiers and their locations.

4. **Web GUI:**

   This module is a web user interface allowing the designer to:

   (a) design the temporal and spatial granularity hierarchies,

   (b) define the stream schemas (i.e. gives the different attributes and their types),

   (c) declare relevant STFDs based on (a) and (b), and generate an abstract schema,

   (d) define a set of SVAs on the abstract schema and generate the concrete schema,

   (e) validate the concrete schema and launch the on-the-fly data loading (i.e. launching SVA data wrappers),

   (f) query data with simple SQL queries and view query results.

5.2 Implementation

5.2.1 Prototype Core

We implemented the prototype ensuring the sensor database design and the data loading in Java and Prolog programming languages. Thus, the reasoning about the spatio-temporal granularity hierarchies is proceeded through a *Prolog* environment with first order logic reasoning.

A Prolog file, containing the *finer than* relationships between the different granularities, is generated from the user-defined spatio-temporal granularity hierarchies. This file is checked whenever an algorithm needs to compare two spatial or two temporal granularities. For instance, the code defining the temporal granularity hierarchy presented in Figure 1.2 (page 6) corresponds in our case to the logical facts given below:

```
finer('year','topT').
finer('bottomT','second').
finer('second','minute').
finer('minute','hour').
finer('day','month').
finer('week','year').
finer('month','year').
```

And the definition of the rule *finerthan* that allows to compare two granularities is:

```
finerthan(Granularity1,Granularity2) :- finer(Granularity1,Granularity2).
finerthan(Granularity1,Granularity2) :- finer(Granularity1,Granularity3),
                                        finerthan(Granularity3,Granularity2).
```

So each time the new spatio-temporal hierarchies are defined, the facts representing each hierarchy are added to an initial Prolog file containing the predefined rules, such as *finerthan* and *glb*, which allow the reasoning about granularities.

5.2.2 Web GUI

On top of this prototype, we implemented a Java EE web application in order to demonstrate both levels of our architecture especially the granularity-aware database design. We used *Apache Tomcat* as an application server. The granularity hierarchy graphs were implemented thanks to the *JavaScript* library *D3.js*. Finally, the data loading figures were implemented with *Google Charts*.

5.2.3 Underlying RDBMS

We used *Oracle 11G* as DBMS for the implementation and the experiments. The connection to Oracle via the prototype was done using a JDBC connection.

The temporal function $trunc\_T$, seen in Example 19 (page 46), that truncates the value of the attribute time up to the corresponding granules of the given temporal granularity, is a predefined function under Oracle, i.e. `trunc()`. This function returns a date with the time portion of the day truncated to the unit specified by the format model. For instance, `trunc(to_date('2017/02/11 08:35:57','yyyy/mm/dd hh24:mi:ss'),'dd')` truncates the date 2017/02/11 08:35:57 to 2017/02/11.

In the case of spatial conversions, usual RDBMS do not include predefined functions to truncate sensor locations. We need to use more complex functions. In our work, we defined the following function which extracts from a given string the spatial granules up to the desired granularity using Oracle predefined function `REGEXP_SUBSTR` by detecting the separation symbol `:`. For instance, if we consider the spatial granularity hierarchy given in Figure 1.2 (page 6), the instruction $trunc\_S(location,'house')$ consists in considering the concatenation of the first three sub-strings of location, i.e. the sub-strings are build with respect to the separation symbol. Here, we consider the first three sub-strings because in the spatial hierarchy we have *building*, *floor* and then *house* which is the granularity to which we aim to truncate the location. This can be done as follows:

```
trunc_S(location,'house')=REGEXP_SUBSTR(location, '[^:"]+', 1, 1)
                    ||':'||REGEXP_SUBSTR(location, '[^:"]+', 1, 2)
                    ||':'||REGEXP_SUBSTR(location, '[^:"]+', 1, 3)
```

5.2.4 Technologies and Tools

Java

Java [6] is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2016, Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers.

Prolog

Prolog [9] is a logic programming language. Prolog has its roots in first-order logic, a formal logic. It is a declarative language. Thus, the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations.

SWI Prolog[1]

SWI-Prolog [10] is a free versatile implementation of the Prolog language. It has a rich set of features, libraries for constraint logic programming, multithreading, unit testing, GUI, interfacing to Java, ODBC and others, literate programming, a web server, SGML, RDF, RDFS, developer tools (including an IDE with a GUI debugger and GUI profiler), and extensive documentation.

Java EE

Java Platform, Enterprise Edition (Java EE) [7] is the standard in community-driven enterprise software. Java EE is developed using the Java Community Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals.

Apache Tomcat[2]

Apache Tomcat [1], often referred to as Tomcat Server, is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF). Tomcat implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment in which Java code can run. Tomcat is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation, released under the Apache License 2.0 license, and is open-source software.

D3.js[3]

D3.js [3], for Data-Driven Documents, is a JavaScript library for producing dynamic, interactive data visualizations in web browsers. It makes use of the widely implemented SVG, HTML5, and CSS standards.

[1] http://www.swi-prolog.org/
[2] http://tomcat.apache.org/
[3] https://d3js.org/

Google Charts API[4]

The Google Chart API [5] is an interactive Web service that creates graphical charts from user-supplied data. Google servers create a PNG image of a chart from data and formatting parameters specified by a user's HTTP request. The service supports a wide variety of chart information and formatting.

Oracle[5]

Oracle Database [8] is an object-relational database management system produced and marketed by Oracle Corporation.

Eclipse IDE[6]

Eclipse IDE [4] is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including C, C++, JavaScript, Perl, PHP, Prolog, Python...

5.2.5 Experimentation Platform

We chose to do our experiments in a user-like technical environment. Thus, the experiments were undertaken on a personal laptop with the following hardware and software:

- **CPU**: Intel ® Core^{TM} i7-4600U,
- **RAM**: 8 GB 1600MHz DDR3L 1DM,
- **HD**: 256 GB SATA-3,
- **OS**: Windows 7 Pro x64.

5.3 Experiments

In this section, we focus on experimenting the data loading middleware. In these experiments, we are interested in the following main concepts: *data storage*, *data accuracy*, *data reduction*, *sensor data loading overhead* and *query workload*.

[4]https://developers.google.com/chart/
[5]https://www.oracle.com/fr/database/index.html
[6]https://www.eclipse.org/

We start by checking the impact of the considered granularities on the number of preserved tuples. After that we study the trade off between the data reduction and the data accuracy. Then, we check the sensor data loading process. We compare data loading without SVA in *raw data* relations, i.e. relations directly storing all tuples from sensor data streams at the finest granularities from streams, and with SVAs in granularity-aware *sensor database* relations, i.e. relations holding only the relevant data. Finally, we compare the efficiency of long-term queries on these two types of relations.

5.3.1 Experimental Data Sets

In the sequel, we consider the spatial and the temporal granularity hierarchies presented in Figure 5.2.

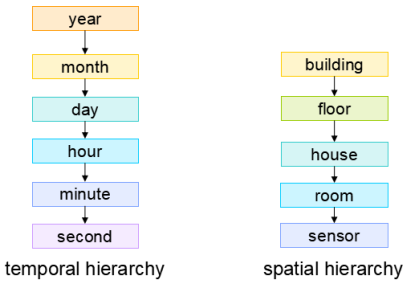


Figure 5.2 — Spatio-temporal granularity hierarchies

In the following, we used two different data sets:

1. **Data Set A**:
   In this data set, we simulate the sensors of an intelligent building containing 10 houses, each house contains 5 rooms. In each room we consider at least 2 sensors of each type (e.g. temperature, humidity...). Thus we simulate the operation of several hundreds of sensors, each sending data at a frequency of one value per minute.

2. **Data Set B**:
   For this data set, we consider real-life sensor data. We have conducted real-life experiments in two buildings in our university[7]. A total of around 400 heterogeneous physical sensors are deployed to measure temperature, humidity, CO2/VOC, presence, contact (for doors/windows), electricity consumption, weather conditions...

[7] http://liris.cnrs.fr/socq4home/site/

5.3.2 Data Storage Evolution w.r.t. Granularities

We evaluate data loading from a stream into a sensor database created with a simple STFD and a corresponding SVA. We take a sensor data stream, e.g. temperature, from *Data Set A* and we store temperature values in different relations with different temporal and spatial granularities. Each time, we focus on a spatial (or temporal) granularity and we compute the number of tuples in each relation. The obtained results are given in Figure 5.3 which contains two graphics:

1. In the first graphic, we vary the spatial granularities and we keep the same temporal granularity: with respect to the temporal granularity *minute*, we consider three different spatial granularities, i.e. *sensor*, *room* and *house*. For instance, for the spatial granularity *room*, the concerned relation contains one temperature value per minute per room.

2. In the second graphic, we follow the same logic. We consider different temporal granularities, i.e. *minute*, *hour* and *day*, with respect to the same spatial granularity *sensor*. For instance, for the temporal granularity *hour*, the concerned relation contains one temperature value per hour per sensor.

As expected, it is clear from Figure 5.3 that with finer granularities we have more tuples. We can note that the temporal granularities are more discriminating than the spatial granularities in our settings.

So, as we have seen, the use of STFDs with coarser granularities may reduce the number of tuples. This leads to more efficient queries as they have less tuples to scan. We also mention that storing relevant data according to application requirements at specific spatio-temporal granularities enables the use of simpler queries. The stored relevant data can be considered as prior-computed query results, like materialized views.

5.3.3 Data Accuracy w.r.t. Data Reduction

In this section, we are mainly interested in studying the trade-off between data reduction and data accuracy with respect to some STFDs. To do so, we use *Data Set B*. As mentioned before, this set is build from real sensors installed in our campus. In these experiments, we consider 19 temperature sensors belonging to 7 different rooms. Each sensor sends a new value per minute. We are interested in data coming from these sensors all along one day (June, 1^{st} 2016).

The idea is to compare the data reduction with respect to the considered set of STFDs. In these experiments, as we focus on STFDs, we took the same SVA for the different cases. We considered the SVA *first* which means that we store the first temperature value received at the concerned spatio-temporal granules.

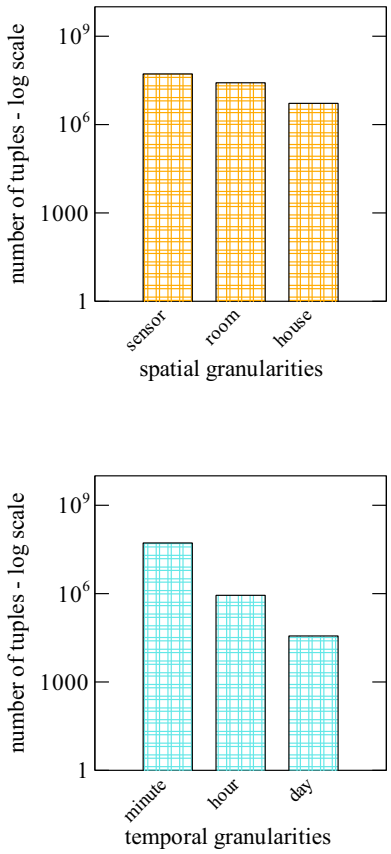


Figure 5.3 — Number of tuples when varying temporal and spatial granularities

Then, by comparing the stored data, i.e. relevant data, with the initial sensor data, we observe the impact of this reduction upon the accuracy of the results.

In our case, the initial sensor data stream spatio-temporal granularities are *sensor* and *minute*. All received sensor data are stored in a unique relation called r_0 (i.e. raw data). We consider the following three sensor relations and their corresponding STFDs:

1. sensor DB r_1: contains the first temperature value per sensor per hour with respect to the STFD $location^{sensor}, time^{hour} \rightarrow temperature$,

2. sensor DB r_2: contains the first temperature value per room per hour with respect to the STFD $location^{room}, time^{hour} \rightarrow temperature$, and

3. sensor DB r_3: contains the first temperature value per room per day with respect to the STFD $location^{room}, time^{day} \rightarrow temperature$.

Table 5.1 shows the important data reduction that may be done thanks to STFDs. As we can see, considering coarser granularities increases data reduction.

relation	number of tuples	ratio: $\mid r_i \mid / \mid r_0 \mid$
r_0	27379	100%
r_1	456	1,67%
r_2	162	0,59%
r_3	7	0,03%

Table 5.1 — Data reduction w.r.t. STFDs

Next, we are interested in data accuracy. Thus we aim to check how data approximation with STFDs impacts data. To do so, we compute the average of temperature per each room during one day from the different relations. These averages are given in Figure 5.4. Not surprisingly, approximating data with coarser granularities increases the error rate and decreases data accuracy.

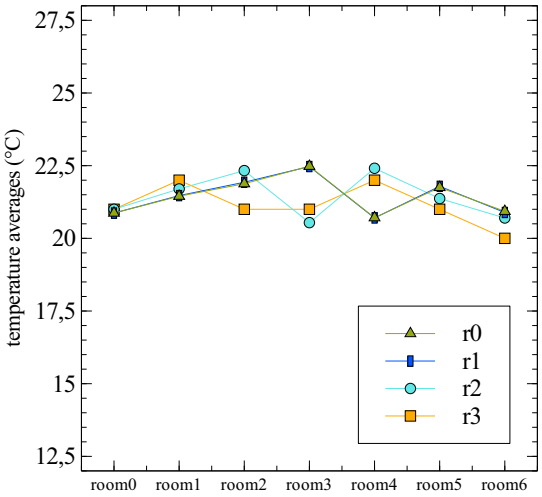


Figure 5.4 — Data accuracy w.r.t. data reduction

In order to evaluate the error rate and to check if the approximation deteriorates the evaluation of the temperature averages, we are now interested in the difference between the results over *raw data* and a *sensor database*. We compute the average of the temperature during each day of a given month for a given sensor on r_0 (raw data, with query Q_0) and r_1 (sensor database at the spatio-temporal granularities sensor and hour, with query Q_1) The obtained difference values are given in Figure 5.5. We can see that the difference is maintained under an error rate of 5% (most values being randomly disseminated on the $\pm 2\%$ rate stripe) which could be considered acceptable in a real life settings. It also evidences the fact that even though query Q_1 considers only one value per hour for the assessment of the average temperature per day, the output of the calculation sticks to the value obtained with a finer granularity for this data set based on real-life sensors.

Actually, considering coarser granularities may not have a major effect on the final results, due to the fact that some sensor values may change slowly over some spatio-temporal granules.

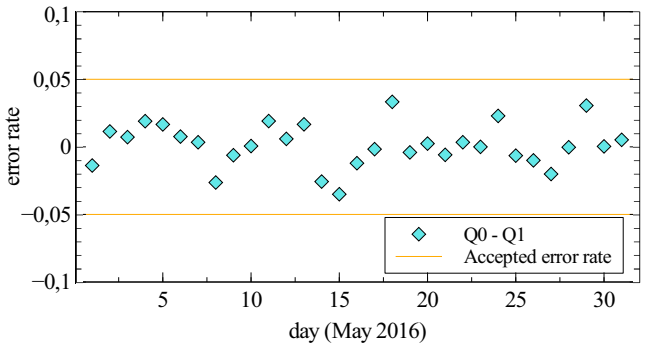


Figure 5.5 — Difference between the results of Q_0 and Q_1

5.3.4 Sensor Data Loading Efficiency

We are now interested in the total execution time of the data management module that receives data from streams, executes SVA data wrappers and/or inserts data into the RDBMS. This time represents the sum of the different time intervals during which data are received, treated and stored by each SVA data wrapper. For instance, in the case of the SVA *avg*, we consider the time that the data wrapper takes for putting the current average (the average of the new value and the previous ones) into the data buffer plus the time for storing the final value when the considered temporal granule is over. We compare the SVAs we have implemented (i.e. *first*, *last*, *maximum*, *average*) with the baseline solution, i.e. storing all received sensor data.

Each sensor sends one value per minute. We however simulate real time for data streams 60× faster, i.e. 1 second for 1 simulated minute. The stored data in the following sets of experiments is approximated upto the spatio-temporal granularities *room* and *hour*.

First, we focus on the variation of the total execution time with respect to the sensor number on a fixed duration of 1 hour. The obtained results are given in Figure 5.6 with a log scale for execution time. Secondly, we vary the duration of our experiments for a fixed number of sensors (100 sensors). Figure 5.7 contains the results of these experiments (log scale).

As we can see, using SVAs in order to select on-the-fly the relevant data is more efficient than the baseline solution: the total execution time in the case of SVAs is lower by 3 orders of magnitude in Figure 5.6 and by 2 orders of magnitude in Figure 5.7.

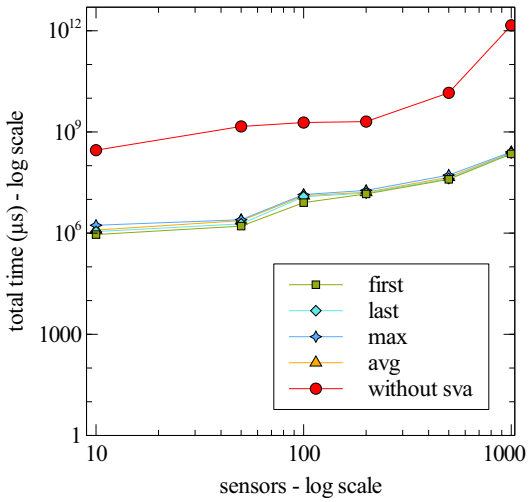


Figure 5.6 — Total execution time w.r.t. the number of sensors, duration 1 hour

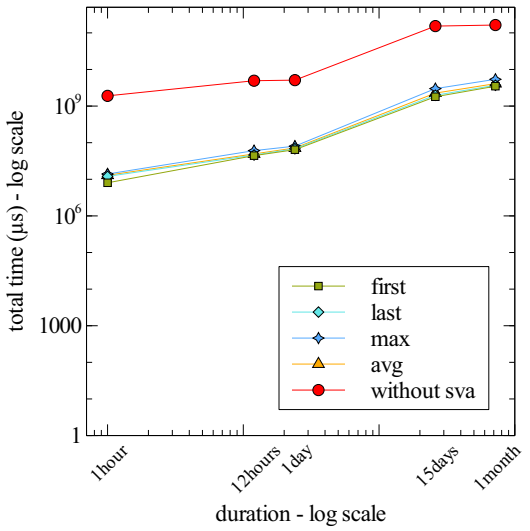


Figure 5.7 — Total execution time w.r.t. the duration, for 100 sensors

5.3.5 Query Workload

In this experiments, we are interested in *Data Set A* (simulated data). Here, we vary the periods of collecting data (i.e. data stored over 1 month, 6 months...) in order to observe the performances of some long-term queries with different loads.

The used relations are given in Table 5.2. The column *Type* indicates if it is a relation containing all incoming sensor data, i.e. raw database, or a relation containing data following our approach, i.e. granularity-aware sensor database, with data stored by a SVA.

Relation	Attributes	Spatial granularity	Temporal Granularity	Type
r_{31}	temperature	sensor	second	raw database
r_{32}	humidity	sensor	second	raw database
r_{33}	temperature,humidity	room	hour	sensor database
r_{34}	temperature	room	hour	sensor database

Table 5.2 — Database relation

We are interested in three different sets of queries:

1. Queries asking for the first temperature values per hour per room during a given day
   - query Q_{11}: over *raw data* relations, without any knowledge on which minute (resp. sensor) is the first of each hour (resp. room); using the keyword NOT EXISTS in the WHERE clause. Thus, the query, for each tuple, should scan all the relation tuples in order to check if there exits a tuple preceding it, belonging to the same hour and coming from the same room,
   - query Q_{12}: over *raw data* relations, to which we specify the minute and the sensor where to find the first tuples, e.g. $minute = 0$,
   - query Q_{13}: over *sensor database* relations which already contain the first value per hour per room (i.e. through SVA).
2. Queries asking for the averages per day of the temperature values during a given month
   - query Q_{21}: over *raw data* relations,
   - query Q_{22}: over *sensor database* relation with one value per room per hour.
3. Queries asking for both the averages of temperature and humidity per day per room during a given month (e.g. May 2014)
   - query Q_{31}: over *raw data* relations,
   - query Q_{32}: over *sensor database* relations (one value per room per hour).

We now write those queries in SQL in order to execute them on a RDBMS containing relations r_{31}, r_{32}, r_{33} and r_{34}. The query Q_{13} over *sensor database* relations can be written as follows:

```
SELECT temperature, time, location
FROM r_34
WHERE trunc_T(time,'hour') = to_date('25/05/2014','dd/mm/yyyy')
ORDER BY time, location;
```

where the function *trunc_T* (resp. *trunc_S*) truncates the time (resp. location) field to the given temporal (resp. spatial) granularity and the function *to_date* converts data to a date. For example, in Oracle we use the function *trunc* in order to get the time portion of a date truncated to the unit specified by the given format model.

When we use generated data, we make the hypothesis that both temperature and humidity sensors send data each minute. This allows us to make a join over the attribute *time* as we know that at each minute we have a value from each sensor. This hypothesis also allows us to ask for the minute 0 if we want to have the first value of the hour. Following the same logic, we can ask for tuples coming from the first sensors per each room (e.g. $t1.location\ like\ '\%temperaturesens1\%'$).

However, using real-life data, a query asking for the first value has to be written using *NOT EXISTS* in order to take into account all the existing tuples and not just the ones stored at minute 00 or coming from a particular sensor. This makes the query more complicated to write as it checks for each tuple if it is the first one of the concerned granules. The difference between these two types of queries can be observed with queries Q_{11} and Q_{12} which are asking for the first temperature value of each hour per each room of a given day.
The query Q_{12} can be written as follows:

```
SELECT temperature, trunc_T(time,'hour') as time,
       trunc_S(location,'room') as location
FROM r_34
WHERE trunc_T(time,'hour') = to_date('25/05/2014','dd/mm/yyyy')
  AND to_number(t1.time,'minute')=0
  AND location LIKE '%temperaturesens1%'
ORDER BY time, location;
```

And the query Q_{11} can be written as:

```
SELECT t1.temperature, trunc_T(t1.time,'hour') AS time,
       trunc_S(location,'room') AS location
FROM r_31 t1
WHERE  trunc_T(t1.time,'day') = to_date('25/05/2014','dd/mm/yyyy')
AND NOT EXISTS (
   SELECT *
   FROM r_31 t2
   WHERE t2.location = t1.location
     AND trunc_T(t1.time,'hour') = trunc_T(t2.time,'hour')
     AND t1.time>t2.time
  )
 AND NOT EXISTS (
```

```
   SELECT *
   FROM r_31 t3
   WHERE trunc_S(t1.location,'room') = trunc_S(t3.location,'room')
   AND trunc_T(t1.time,'hour') = trunc_T(t3.time,'hour')
   AND t1.time>=t3.time
   AND t1.location>t3.location
  )
ORDER BY time, location;
```

Now we present the third set of queries. We do not present the second set of queries, i.e. Q_{21} and Q_{22}, as their are similar to the third one and simpler. The query Q_{31} can be written as:

```
SELECT AVG(t1.temperature), AVG(t2.humidity),
       trunc_T(t1.time,'day') AS time,
       trunc_S(t1.location,'room') AS location
FROM r_31 t1,
     r_32 t2
WHERE trunc_T(t1.time,'month') = to_date('05/2014','mm/yyyy')
  AND trunc_S(t1.location,'room') = trunc_S(t2.location,'room')
  AND t1.time=t2.time
GROUP BY trunc_T(t1.time,'day'),
         trunc_S(t1.location,'room')
ORDER BY trunc_T(t1.time,'day'),
         trunc_S(t1.location,'room');
```

And the query Q_{32} can be written as:

```
SELECT AVG(temperature), AVG(humidity),
       trunc_T(t1.time,'day') as time, location
FROM r_33 t1
WHERE trunc_T(time,'month') = to_date('05/2014','mm/yyyy')
GROUP BY trunc_T(t1.time,'day'), location
ORDER BY trunc_T(t1.time,'day'), location;
```

Query Simplicity

Regarding the simplicity of the queries it is obvious that queries over *sensor database* relations are much more simpler to write. In fact, the queries over *sensor database* relations, do not need to search for the first values of each granules as this was already done when storing data, e.g. the first set of queries. And for the case of the second and

the third sets of queries, the queries over *sensor database* are simpler to write as they do not need to truncate the *location* and the *time* attributes because data are already at the right spatio-temporal granularities. Clearly, Q_{13} looks simpler than Q_{12} which its self is simpler than Q_{11}.

In the case that a query does not match the predefined constraints (i.e. asks for data at other granularities), the query may be less easy to write. For instance, when a query asks for finer granularities, it has to search each time for the granularity including them: the query result is then less accurate. And when a query asks for coarser granularities, there will be the same issues as with raw data but with less data to check.

Query Efficiency

We intend to compare the efficiency of the queries given in this section. We compare queries over raw data relations, Q_{11}, Q_{12}, Q_{21} and Q_{31}, to queries over sensor database relations, Q_{13}, Q_{22} and Q_{32}. The corresponding sensor database relations contain relevant data at the spatio-temporal granularities *room* and *hour*. We are mainly interested in the "elapsed" time of each query[8].

1. The results of the first set of queries, Q_{11}, Q_{12} and Q_{13}, are given in Figure 5.8. These result show that the *elapsed time* of the query Q_{13} is shorter than the other queries by several orders of magnitude. The same applies for the *buffer gets* and the *disk reads* which are lower for Q_{13} (disk reads for Q_{13} have a zero value). Actually, this is due to the fact that the query on *sensor database* relations has less tuples to check than the queries on *raw data* relations. Moreover, in this specific case, the query Q_{13} does not have to compute the first value with functions over the temporal and spatial granularities.

2. The results of the second set of queries, Q_{21} and Q_{22}, are given in Figure 5.9. As we can see, the query Q_{21} over *raw data* is less efficient than the query Q_{22} over *sensor database* relations.

3. The results of the third set of queries, Q_{31} and Q_{32}, are given in Figure 5.10. As for the second set of queries, the query Q_{32} over *sensor database* relations is faster than the query Q_{31} over *raw data*. In fact, computing averages over *raw data* considers all tuples present at each day. Whereas, when we compute averages over *sensor database* relations, we consider a smaller number of tuples as we just have one value per *hour*.

[8]The *elapsed time*, *buffer gets* and *disk reads* are obtained from the Oracle view *V$SQLAREA*. This view lists statistics on shared SQL area and contains one row per SQL string [11]. The *elapsed time* (in microseconds) is used by this cursor for parsing, executing, and fetching. The *buffer gets* represents the sum of buffer gets over all child cursors. The *disk reads* represents the sum of the number of disk reads over all child cursors.

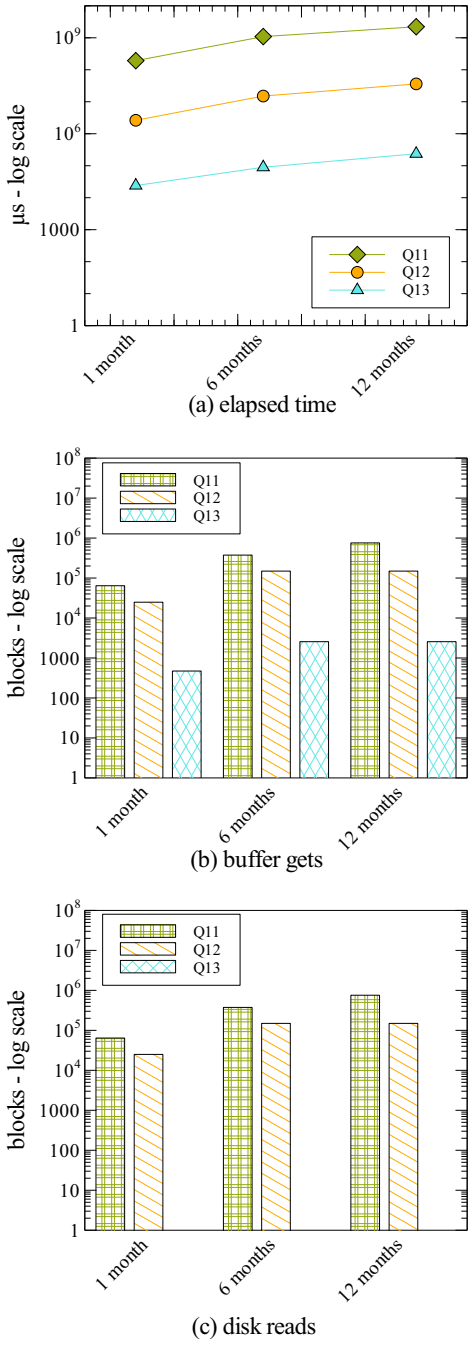


Figure 5.8 — The query performances of the first set of queries w.r.t. the data acquisition period

As we can see, for the three sets of queries, the queries over *sensor database* are more efficient because they have less tuples to take into account. This is due to the fact that the number of tuples decreases while scaling in granularities (cf. Section 5.3.2).

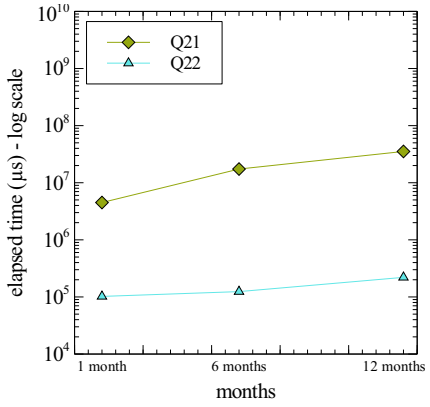


Figure 5.9 — The elapsed time of the second set of queries w.r.t. to the data acquisition period

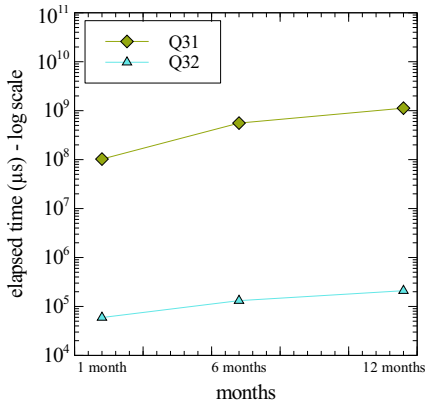


Figure 5.10 — The elapsed time of the third set of queries w.r.t. to the data acquisition duration

5.4 Demonstration

In order to demonstrate both levels of our architecture, especially the granularity-aware database design, we designed a demonstration of our prototype. This demonstration highlights the easiness of gathering application related spatio-temporal constraints and the benefits brought by these constraints in sensor data storage.

5.4.1 Purpose of the Demonstration

This demonstration allows the user to check and try our sensor database design prototype. It contains mainly two parts.

- In the first one, the control is left to the user so she defines her own spatio-temporal dimensions, stream schemas, STFDs and SVAs. Then the system generates the corresponding database schema.

- In the second part, the user observes the impact of transforming sensor data according to application-defined spatio-temporal dimensions over sensor data loading. Actually, the user can choose from predefined spatio-temporal hierarchies the granularities to which she wants to approximate data. Then she gets a view of data storage progression according to the chosen granularities.

5.4.2 Database Schema Design

The user has the ability to define her own dimensions, streams and approximations (STFDs and SVAs). This tool allows the user to handle spatio-temporal dimensions and STFDs in order to obtain a sensor database design in accordance to her inputs.

She can define any spatial and temporal dimensions through a graphical interface where she creates a graph structure respecting the form of a lattice as shown in Figure 5.11 and Figure 5.12. After that, the user defines the attributes, as depicted in Figure 5.13. Then, as shown in Figure 5.14 and Figure 5.15, the user defines the dependencies between the attributes with the corresponding spatio-temporal granularities. Therefore, as shown in Figure 5.16, the system generates the minimal cover and asks the user to provide meaningful relation names as well as SVAs for the concerned attributes. Finally the system shows the DDL queries corresponding to the generated schema, as shown in Figure 5.17.

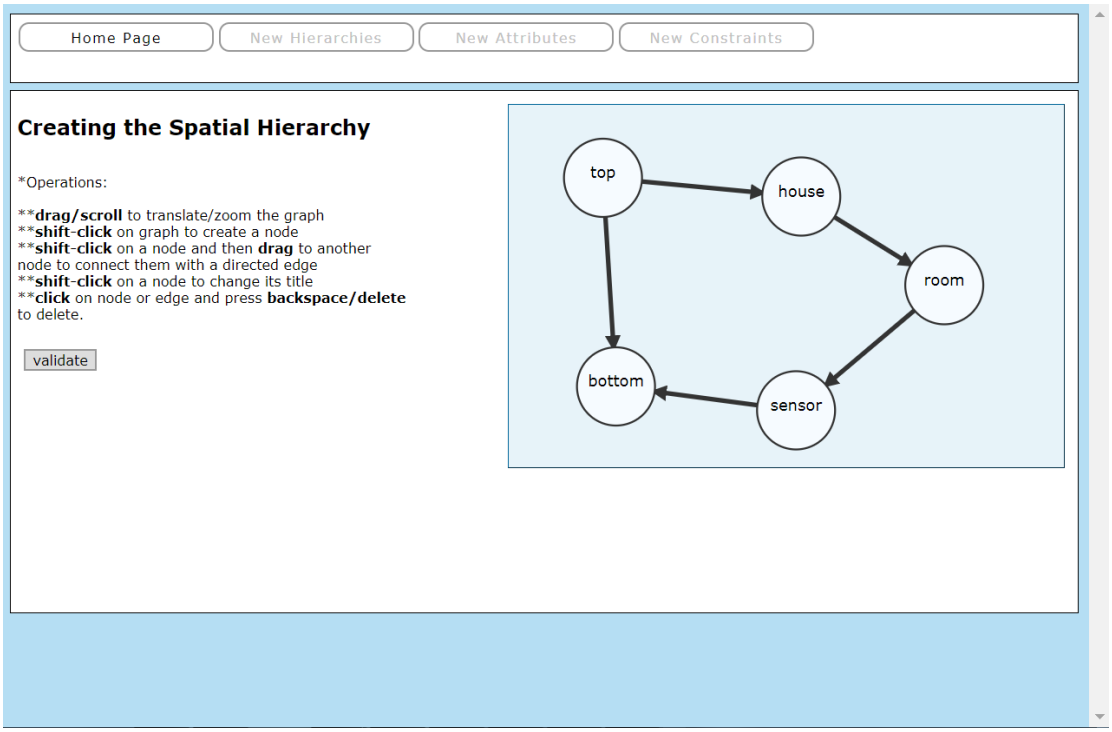


Figure 5.11 — Screenshot of the creation of the spatial hierarchy

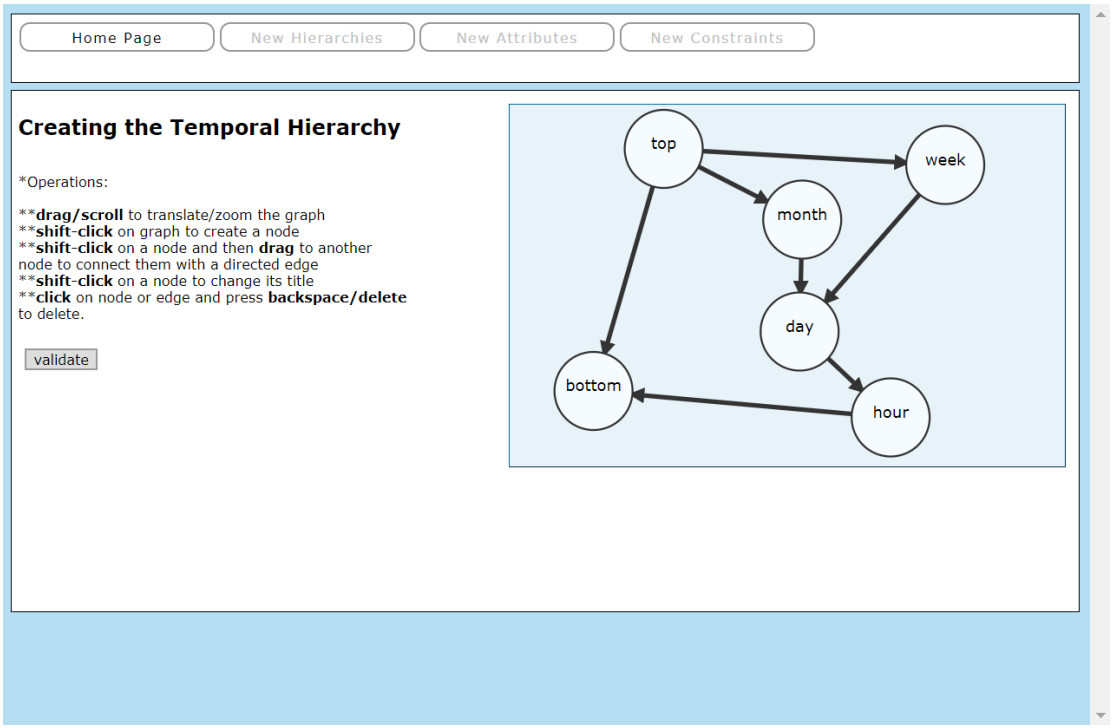


Figure 5.12 — Screenshot of the creation of the temporal hierarchy

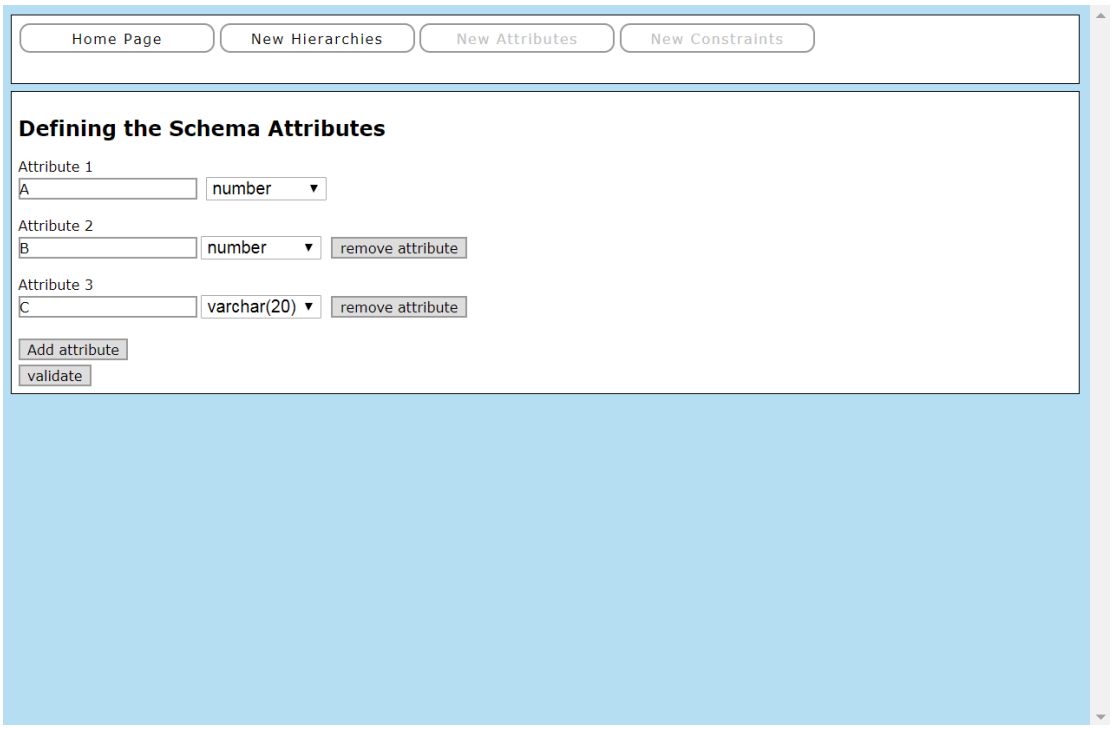


Figure 5.13 — Screenshot of the input of the concerned attributes

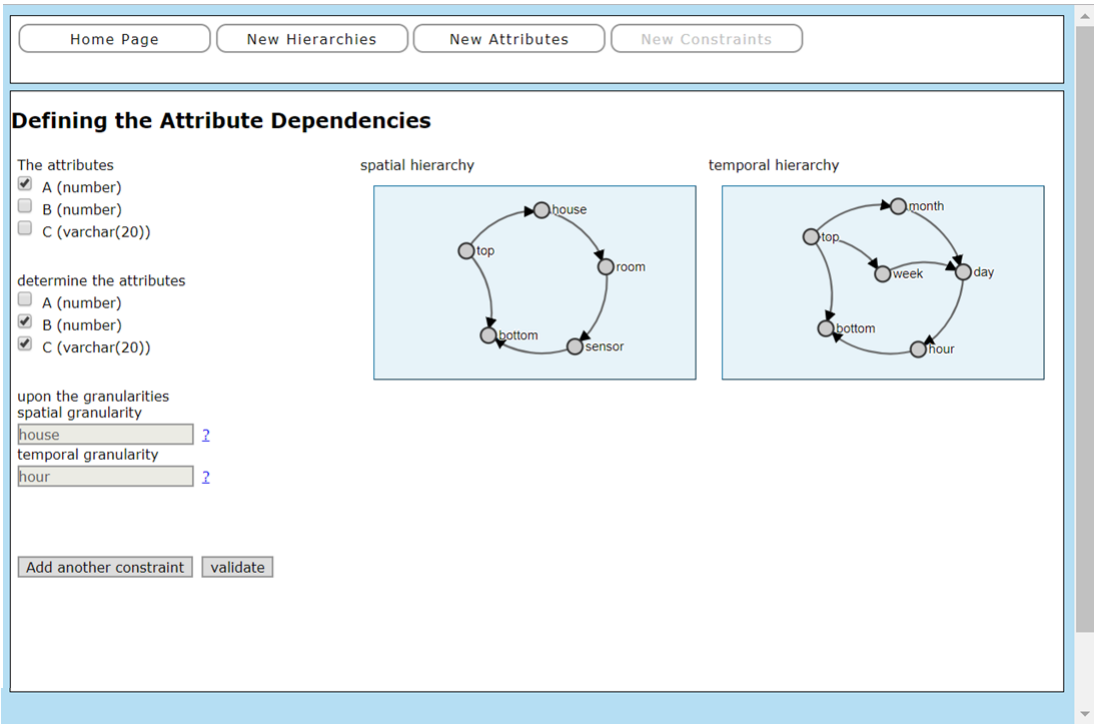


Figure 5.14 — Screenshot of the first example of the input of STFD

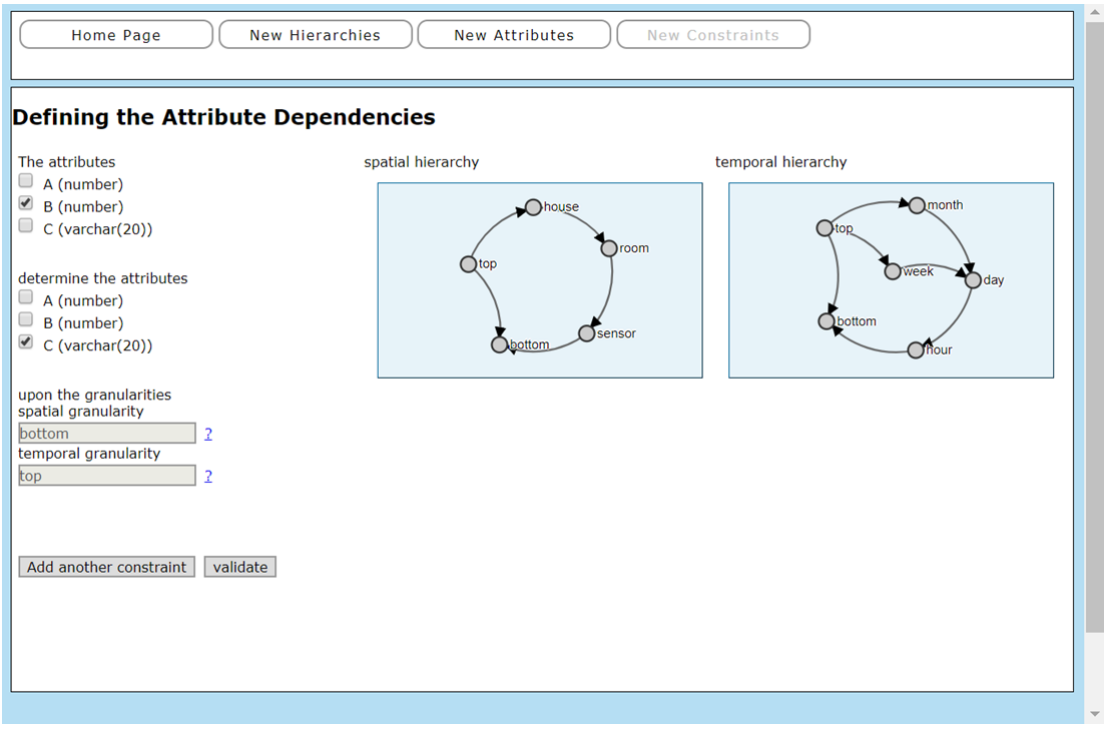


Figure 5.15 — Screenshot of the second example of the input of STFD

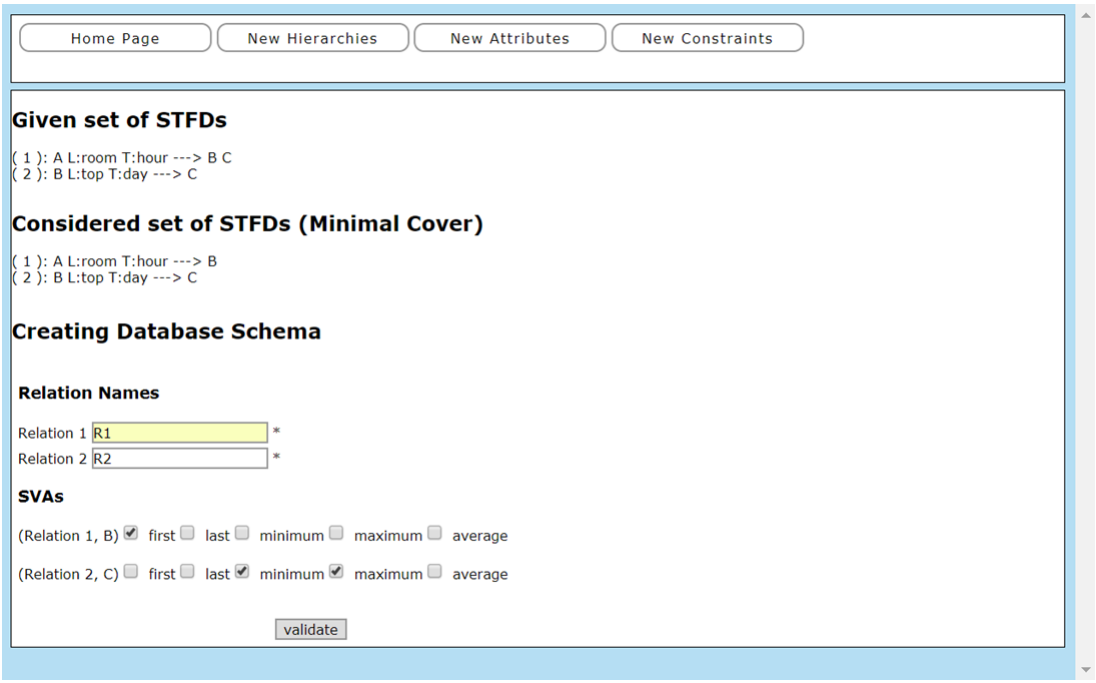


Figure 5.16 — Screenshot of the minimal cover generation

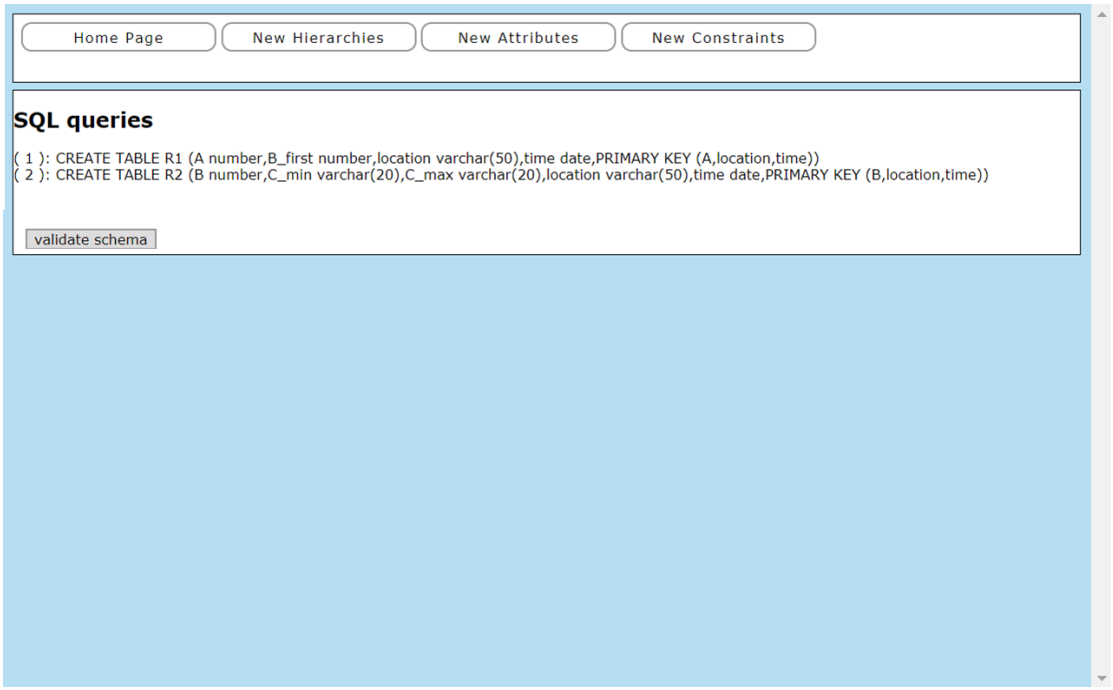


Figure 5.17 — Screenshot of the outputted SQL queries

5.4.3 Observing On-The-Fly Data Loading

This section contains screenshots of the second part of our demonstration. At this stage, we compare the content of two different databases:

1. *raw DB*: a database that holds all received data from sensor streams, and
2. *sensor DB*: a database created following our approach thanks to the user defined approximations.

In this part of the demonstration, a dynamic data chart shows the evolution of the number of tuples in *raw DB* and *sensor DB* while incoming sensor data are processed. It highlights the difference between the amount of the stored data in a *raw DB* and *sensor DB*. Figures 5.18, 5.19 and 5.20, show the evolution in time of the number of tuples in both databases.

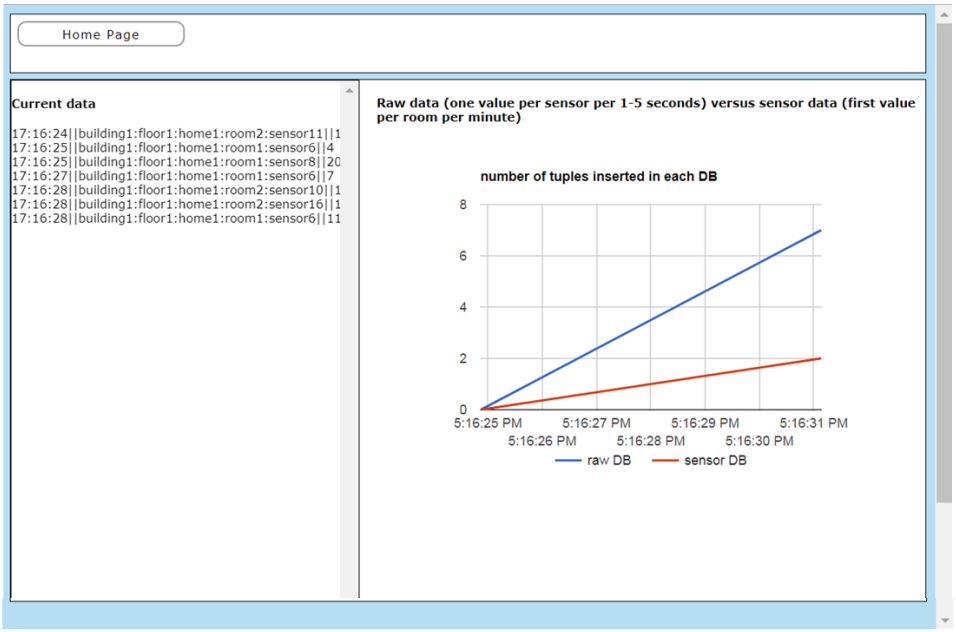


Figure 5.18 — Screenshot of the number of tuples in *raw DB* and *sensor DB* at the beginning

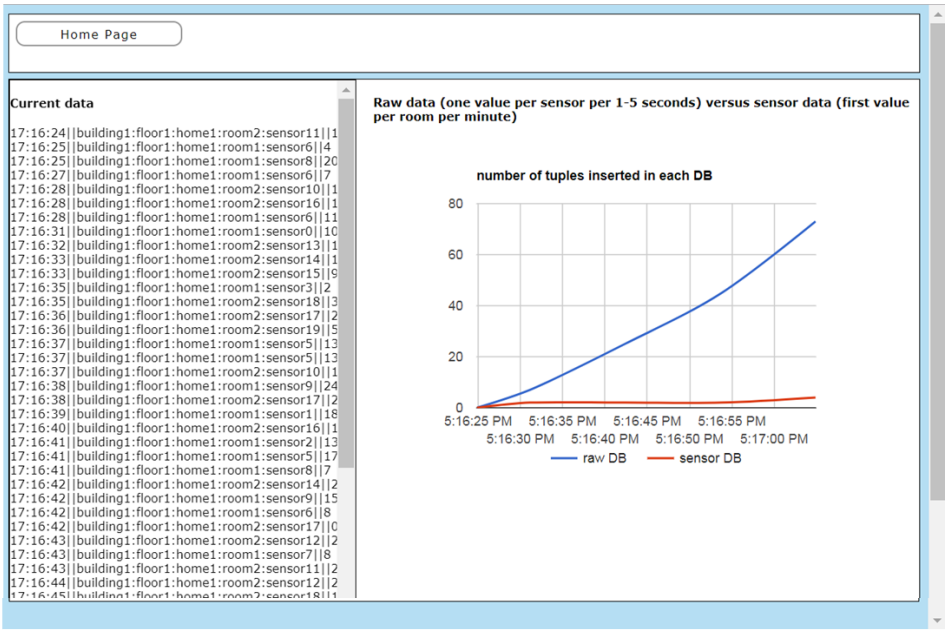


Figure 5.19 — Screenshot of the number of tuples in *raw DB* and *sensor DB* at the middle

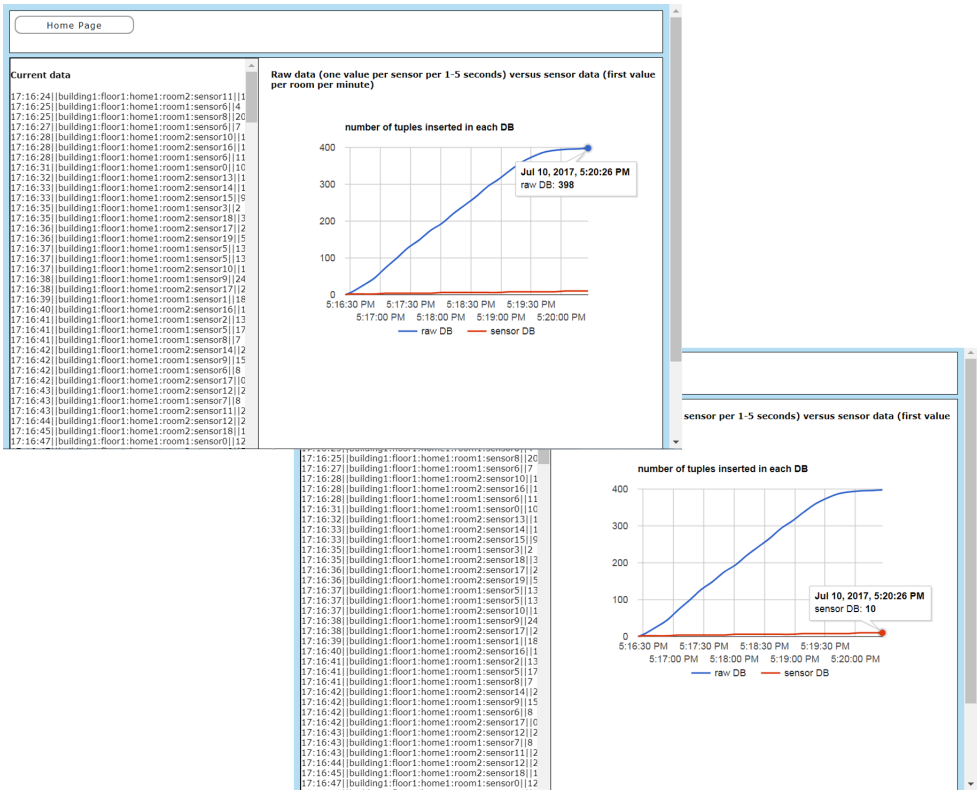


Figure 5.20 — Screenshots of the number of tuples in *raw DB* and *sensor DB* at the end

Conclusion

We implemented a prototype according to the proposed architecture for granularity-aware sensor database which deals with both database design and data loading.

We have conducted experiments with synthetic and real-life sensor data streams coming from Intelligent Buildings. We have compared our solution with the baseline solution, showing results orders of magnitude better in terms of memory usage and performance with our predefined scenarios.

We also devise a demonstration with our prototype: a user can design through it a granularity-aware sensor database schema from her own spatio-temporal granularity hierarchies, STFDs and SVAs.

Part III

Related Work & Conclusion

CHAPTER 6

Related Work

Introduction

All along the previous chapters we gave an insight of the related work. In this chapter we sketch an overview of the main research domains related to our context. We start by giving a short history of the relational model and Functional Dependencies. Then we discuss about some temporal database issues mainly Temporal Functional Dependencies. And finally, we talk about sensor data management, and links with OLAP concepts for data warehouses.

6.1 Database Design

Researches on relational data model started with [Cod70] aiming at proposing a user interface hiding the physical organization of data. The proposed model presents a relation as a table where columns correspond to attributes and row to records. Codd's model has been extensively studied since the seventies which offers us a huge heritage around the theory of relational databases.

One of the major studied fields in this theory is the design of the relational database schema which is mainly based on the study of dependencies and constraints. For instance, functional dependencies allow to incorporate data semantics into data models. Dependencies between attributes were first introduced in order to link attributes to keys [Cod71, Hea71]. An inference system was proposed in [DC73]. These axioms were studied in [Arm74] and were proved to be sound and complete. We can find in [BB79] a discussion over functional dependency issues and algorithms especially the computational complexity of the implication problem. We can mention other interesting papers in the theory of dependencies as [Mai80] which contains a discussion of the minimization of FD representation and in [ADS83] where we have a graph-theoretic approach for the manipulation and representation of FDs.

6.2 Temporal Database

Although many researches focused on temporal query languages namely [NA89, Sar90, Sno95], a general temporal SQL standardization is still missing. The current SQL standard *SQL*:2011 includes parts of TSQL2 [Sno95] which was a research work intending to present a temporal extension to SQL. Nowadays, each DBMS provides specific temporal features.

6.2.1 Temporal Functional Dependency

TFDs have been mainly introduced in order to constrain the temporal data in temporal databases. There have been an important number of articles aiming at defining and characterizing TFDs, e.g. [Via87, Wij95, JSS96, WBBJ97, Wij99].

The three first approaches [Via87, Wij95, JSS96] handle TFD without time granularity. In [Via87], the author defined a temporal relation as a temporal sequence of database states and extended each tuple with its updated version. He defined two classes of dependencies: static dependencies (i.e. standard dependencies that must be satisfied in each state) and dynamic dependencies (i.e. dependencies that relate every state to its successor). The data model in [Wij95] was extended with a valid time which represents a set of time points. The author presented the suitable definition of FD in the presence of valid time and defines two classes of temporal dependencies: Temporal Functional Dependencies (TFDs) and Dynamic Functional Dependencies (DFDs). In [JSS96], data contains two time dimensions: valid time and transaction time. The authors handle the problem of expressing the functional dependencies with such data. These works do not consider granularity as a central notion as we do in this work.

Both [WBBJ97, Wij99] handled multiple time granularities. The authors in [WBBJ97] (and in [BJW00]) defined the time granularities and the different relationships between them. They defined the *temporal module schema* and the *temporal module* as well as TFD. In [Wij99], the author extended the dependencies presented in [Wij95] using time granularity and object identity which is a time-invariant identity that relates the different versions of the same object.

In this thesis, we extend the concept of temporal module to build a more general concept of granularity-aware spatio-temporal database with Spatio-Temporal Functional Dependencies (STFDs).

6.2.2 Semantic Assumptions

In this thesis, along with Spatio-Temporal Functional Dependencies, we defined a mechanism to select on-the-fly relevant data thanks to Semantic Value Assumptions (SVAs). SVAs represent a sort of data exchange mechanism [FKP05] inspired from semantic assumptions designed for temporal databases [BJW00].

The authors in [BJW00] defined two classes of semantic assumptions for temporal databases, i.e. *point-based assumptions* and *interval-based assumptions*, which allow a compact representation of potentially infinite temporal data. These assumptions may be used by the DBMS to compute values not explicitly given and to convert data from one granularity to another. The main *point-based assumptions* discussed in [BJW00] is the *persistence* assumption which allows to say that the values of certain attributes

persist in time unless they are explicitly changed. The second type of assumptions are the *interval-based assumptions* which are needed whenever we want to derive information valid between different granularities. Different types of *interval-based assumptions* are given in [BJW00] namely the *first* and *liquidity* assumptions.

6.3 Sensor Data Management

In this thesis, we focus on data management issues related to a specific type of data: sensor data. In particular, we tackle sensor data storage issues and load shedding principles for data streams.

6.3.1 Storage

Nowadays, sensor data management can be seen from two points of view: real-time, i.e. continuous queries [ACÇ+03, ABB+03, ABW06, TATV11] where data streams are often considered as unbounded append-only databases and queries are producing answers in a continuous and timely fashion, versus historical data management [LH05, LCXA06, DGMS07]. These historical data management approaches were interested in resolving storage problems that can result from the important amount of data generated by sensors. In [LH05], the authors proposed to store sensor data according to the provenance meta-data. The authors in [LCXA06] proposed a semantic approach using ontologies. In [DGMS07], the authors focus on in-network data management, i.e. query processing is performed when data are transmitted from the source to the target node, so they propose a sensor database architecture that emphasizes local data archival and query processing at embedded sensors.

As far as we know, there is no formal approach for dealing with spatio-temporal stream querying considering different granularities. In this thesis, we were in between both domains: we were interested in establishing a data management tool that constructs so-called granularity-aware sensor database which is a historical sensor database from real-time sensor data streams.

6.3.2 Load Shedding

The data loading mechanism with SVAs proposed in this thesis is similar to a load shedding mechanism. In fact, the load shedding process [TÇZ+03] intends to reduce the workload of the data stream management system by dropping tuples from the system. Several approaches proposed different tuples dropping strategies, e.g. random [KNV03], with a priority order [CcC+02] or a semantic strategy [DGR03].

However, the load shedding usually interferes in the physical plan of the query while our approach aims to interfere sooner with the database design and to take into account predefined approximations. Our contribution can be thought as a "declarative load shedding process" since we allow to prune data stream from declarative constraints (STFDs, SVAs), instead of sampling techniques.

6.4 Data Warehouse

The context of this thesis is somehow related to OLAP database design for data warehouse [WK07]. In fact, our work focuses on selecting and storing "relevant" sensor data streams into a target database using constraints based on spatio-temporal dimensions. We propose a formal and complete approach that is independent of implementation issues so that the target database schema can be transformed into any logical schema depending on the application requirements which can be an OLAP cube. This means that our approach is not comparable to OLAP since our main goal is to organize sensor data for long-term storage and not to analyse multidimensional data from multiple perspectives.

Roll-Up Dependencies (RUDs) [WN99, CNW02] define dependencies with a higher abstraction level for OLAP DB and data mining. They extend FDs to non temporal dimensions by generalizing them for domains related by a partial order, i.e. levels. RUDs allow each attribute to roll up through the different levels of its associated hierarchy. Here, a generalization schema is obtained from a schema by duplicating attributes, by omitting attributes, or by replacing a level with a higher level. Then, a roll-up dependency is an implication between two generalization schemas of the same underlying relation schema. A RUD can be written as: $P \rightarrow Q$, where P and Q are generalization schemas of a relation schema R. Such dependency is satisfied by a relation r over R if and only if for all tuples $t_1, t_2 \in r$, if t_1 and t_2 are P-equivalent, i.e. t_1 and t_2 are equal after rolling up their attribute values to the levels specified by P, then t_1 and t_2 are also Q-equivalent, i.e. t_1 and t_2 are equal after rolling up their attribute values to the levels specified by Q. Therefore, RUDs allow to check if some attribute values are equal at different abstraction levels. In our case, we just need two dimensions, i.e. temporal and spatial. We distinguish two particular attributes, for spatial and temporal dimensions, and we combine them with classical attributes (i.e. attributes without associated hierarchies). The authors in [WN99] consider dependencies between different generalization schemas, i.e. they consider dependencies at the schema level, however, we consider data dependencies between different attributes, i.e. we consider dependencies at the attribute level. Algorithmic aspects such as attribute closure, as we focus on in our work, are not studied for RUDs.

Conclusion

Since huge amount of sensor data can be generated, we believe that selecting only relevant data to be saved for long-term query facilities is an important issue. We borrowed the declarative approach developed in the seventies for database design and we extended them with the spatio-temporal dimensions which are actually of particular relevance in our context. As far as we know there is no many contributions aiming at using the spatial and the temporal dimension in order to retrieve the relevant data from sensor data streams.

CHAPTER 7 Conclusion

7.1 Research Summary

Due to their massive emergence in everyday life, sensors become of major interest in various research domains. In our work, we were interested in sensor data storage for long-term reporting applications. Actually, thousands and even millions of sensors can be deployed easily, generating data streams that produce cumulatively huge volumes of data. These amounts of data have to be stored in an efficient way to enable long-term reporting application to easily retrieve it.

In this thesis, our aim was to propose a declarative approach that ensures the storage of the relevant data of sensor data streams at the application specific spatio-temporal granularities. The core idea is to take into account the spatial and temporal aspects of sensor data streams in order to approximate them.

We have presented a model for granularity-aware spatio-temporal database with the formal notions allowing to revisit the classical database schema normalization process, namely Spatio-Temporal Functional Dependencies (STFDs). In fact, STFDs are attribute constraints where we distinguish two particular attributes, for spatial and temporal dimensions, that we combine with classical attributes. These dependencies allow to express the fact that some attributes may depend on other attributes all along specific spatio-temporal granules. For instance, thanks to STFDs, we can express the idea that temperature values remain the same in each room all along each hour. We have then defined a dedicated normalization algorithm based on a novel closure algorithm for STFDs. The closure of attributes plays a crucial role in the generation of a minimal cover of a set of STFDs and thus in the production of normalized sensor database schemas.

According to this model, we proposed a granularity-aware sensor database architecture ensuring, in a declarative manner, both the database schema design and the specification of the data loading from sensor data streams. Once the granularity-aware sensor database schema is defined, the proposed architecture allows to create the corresponding database relations in SQL, data description language implemented on top of any RDBMS. The specification of relevant data was done through database schema annotations, i.e. Semantic Value Assumptions (SVAs). These annotations allow to configure the mechanism to load, automatically and on-the-fly, only the relevant data from sensor data streams into the sensor database.

We implemented a prototype following the proposed architecture, with a web user interface.We tested both sensor database design from STFDs and data loading techniques with SVAs.

We have conducted many experiments on synthetic and real-life data streams from Intelligent Building context. We have compared our system to a baseline solution and found significant improvement in terms of memory usage and query performance.

7.2 Future Work & Perspectives

We have highlighted our proposition in the context of intelligent buildings for domestic sensors. Nevertheless, our proposition relies upon clear theoretical foundations that enable to take both spatial and temporal dimensions into account for sensor data streams. The approach is quite versatile and could be adopted in a wide range of application contexts. Many extensions could be done, for instance towards *mobile sensors*.

In our application context there was no need to study the decomposition and discuss the identification of the primary key when generating the sensor database schema. However for other application domains we believe that a deep study of these important notions must be done.

Once a schema has been proposed, it may be interesting to allow the change of the user-defined inputs (i.e. hierarchy, STFDs and SVAs) for a further update of these entries with respect to the evolution of the user requirements. In this thesis, we did not focus on the update of any entry: we think that this issue may be interesting as future work.

In particular, the SVAs in our prototype are limited to predefined SVAs, however, as we mentioned when defining these annotations, SVAs may be more complex. It may even be interesting to allow the user to enter her own definition of each SVA when needed.

In order to limit the scope of certain STFDs to predefined granules it may be possible to extend the STFD formalism with conditions following the tracks of Conditional Functional Dependencies [BFG$^+$07]. For instance, it may be interesting to express the fact that temperature values of the second floor remain the same along each hour.

In our case study, we identified that we need two hierarchies for sensors in intelligent building, i.e. spatial and temporal. Therefore, we limited our work to these two hierarchies. We think that our reasoning may easily be applied to more hierarchies in a manner reminiscent to Roll-Up Dependencies (RUDs) [WN99]. In fact, the theoretical foundations of STFDs highlight the complex interactions between the two granularity hierarchies, w.r.t. the case with only the temporal dimension, and can then be more easily extended to three or more dimensions.

Part IV

References

Bibliography

[ABB+03] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. Stream: the stanford stream data manager (demonstration description). In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 665–665. ACM, 2003.

[ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal - The International Journal on Very Large Data Bases*, 15(2):121–142, 2006.

[ACÇ+03] Daniel J Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal-The International Journal on Very Large Data Bases*, 12(2):120–139, 2003.

[ADS83] Giorgio Ausiello, Alessandro D'Atri, and Domenico Saccà. Graph algorithms for functional dependency manipulation. *Journal of the ACM (JACM)*, 30(4):752–766, 1983.

[Arm74] William Ward Armstrong. Dependency structures of data base relationships. In *IFIP Congress*, pages 580–583, 1974.

[BB79] Catriel Beeri and Philip A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Trans. Database Syst.*, 4(1):30–59, March 1979.

[BFG+07] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 746–755. IEEE, 2007.

[BJW00] Claudio Bettini, Sushil Jajodia, and Sean Wang. *Time granularities in databases, data mining, and temporal reasoning*. Springer, 2000.

[CcC+02] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring streams: A new class of data management applications. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 215–226. VLDB Endowment, 2002.

[CGF+14] Manel Charfi, Yann Gripay, Nicolas Fourty, Denis Genon-Catalot, and Jean-Marc Petit. Approche déclarative pour le monitoring des bâtiments intelligents. *JNCT 2014*, page 123, 2014.

[CGP16a] Manel Charfi, Yann Gripay, and Jean-Marc Petit. Conception d'une base de données capteurs à l'aide de contraintes spatio-temporelles. In *BDA 2016*, 2016.

[CGP16b] Manel Charfi, Yann Gripay, and Jean-Marc Petit. Optimization of a class of temporal queries. In *Proceedings of the 20th International Database Engineering & Applications Symposium*, pages 346–351. ACM, 2016.

[CGP17] Manel Charfi, Yann Gripay, and Jean-Marc Petit. Spatio-temporal functional dependencies for sensor data streams. In *International Symposium on Spatial and Temporal Databases*, pages 182–199. Springer, 2017.

[Cha14] Manel Charfi. Optimisation sémantique des requêtes continues application aux bâtiments intelligents. In *BDA 2014: Gestion de données-principes, technologies et applications*, pages 26–27, 2014.

[CNW02] Toon Calders, Raymond T. Ng, and Jef Wijsen. Searching for dependencies at multiple abstraction levels. *ACM Trans. Database Syst.*, 27(3):229–260, September 2002.

[Cod70] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Cod71] E. F. Codd. Normalized data base structure: A brief tutorial. In *Proceedings of the 1971 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '71, pages 1–17, New York, NY, USA, 1971. ACM.

[CS14] Carlo Combi and Pietro Sala. Interval-based temporal functional dependencies: specification and verification. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):85–130, 2014.

[DC73] Claude Delobel and Richard G. Casey. Decomposition of a data base and the theory of boolean switching functions. *IBM Journal of Research and Development*, 17(5):374–386, 1973.

[DGMS07] Yanlei Diao, Deepak Ganesan, Gaurav Mathur, and Prashant J Shenoy. Rethinking data management for storage-centric sensor networks. In *CIDR*, volume 7, pages 22–31, 2007.

[DGR03] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. Approximate join processing over data streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 40–51, New York, NY, USA, 2003. ACM.

[FKP05] Ronald Fagin, Phokion G Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Transactions on Database Systems (TODS)*, 30(1):174–210, 2005.

[GÖ10] Lukasz Golab and M Tamer Özsu. Data stream management. *Synthesis Lectures on Data Management*, 2(1):1–73, 2010.

[Hea71] I. J. Heath. Unacceptable file operations in a relational data base. In *Proceedings of the 1971 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '71, pages 19–33, New York, NY, USA, 1971. ACM.

[JSS96] Christian S Jensen, Richard T Snodgrass, and Michael D Soo. Extending existing dependency theory to temporal databases. *Knowledge and Data Engineering, IEEE Transactions on*, 8(4):563–582, 1996.

[KNV03] Jaewoo Kang, Jeffery F Naughton, and Stratis D Viglas. Evaluating window joins over unbounded streams. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 341–352. IEEE, 2003.

[LCXA06] Micah Lewis, Delroy Cameron, Shaohua Xie, and Budak Arpinar. Es3n: A semantic approach to data management in sensor networks. In *Semantic Sensor Networks Workshop*, 2006.

[LH05] Jonathan Ledlie and DA Holland. Provenance-aware sensor data storage. In *Data Engineering Workshops, 2005. 21st International Conference on*, pages 1189–1189. IEEE, 2005.

[LL12] Mark Levene and George Loizou. *A guided tour of relational databases and beyond*. Springer Science & Business Media, 2012.

[Mai80] David Maier. Minimum covers in relational database model. *J. ACM*, 27(4):664–674, October 1980.

[Mai83] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[NA89] Shamkant B Navathe and Rafi Ahmed. A temporal relational model and a query language. *Information Sciences*, 49(1):147–175, 1989.

[PNJ09] Loïc Petit, Abdelhamid Nafaa, and Raja Jurdak. Historical data storage for large scale sensor networks. In *Proceedings of the 5th French-Speaking Conference on Mobility and Ubiquity Computing*, pages 45–52. ACM, 2009.

[Sar90] Nandlal L Sarda. Extensions to sql for historical databases. *Knowledge and Data Engineering, IEEE Transactions on*, 2(2):220–230, 1990.

[Sno95] Richard T Snodgrass. *The TSQL2 temporal query language*, volume 330. Springer Science & Business Media, 1995.

[TATV11] Jordi Creus Tomàs, Bernd Amann, Nicolas Travers, and Dan Vodislav. Roses: A continuous content-based query engine for rss feeds. In *Database and Expert Systems Applications*, pages 203–218. Springer, 2011.

[TÇZ+03] Nesime Tatbul, Uğur Çetintemel, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 309–320. VLDB Endowment, 2003.

[Via87] Victor Vianu. Dynamic functional dependencies and database aging. *Journal of the ACM (JACM)*, 34(1):28–59, 1987.

[WBBJ97] X Sean Wang, Claudio Bettini, Alexander Brodsky, and Sushil Jajodia. Logical design for temporal databases with multiple granularities. *ACM Transactions on Database Systems (TODS)*, 22(2):115–170, 1997.

[Wij95] Jozef Wijsen. Design of temporal relational databases based on dynamic and temporal functional dependencies. In *Recent Advances in Temporal Databases*, pages 61–76. Springer, 1995.

[Wij99] Jef Wijsen. Temporal fds on complex objects. *ACM Transactions on Database Systems (TODS)*, 24(1):127–176, 1999.

[WK07] Robert Wrembel and Christian Koncilia. *Data warehouses and OLAP: concepts, architectures, and solutions*. Igi Global, 2007.

[WN99] Jef Wijsen and Raymond T Ng. Temporal dependencies generalized for spatial and other dimensions. In *Spatio-Temporal Database Management*, pages 189–203. Springer, 1999.

[ZRL+04] Daniel C Zilio, Jun Rao, Sam Lightstone, Guy Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. Db2 design advisor: integrated automatic physical database design. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1087–1097. VLDB Endowment, 2004.

Webography

[1] Apache Tomcat. https://en.wikipedia.org/wiki/Apache_Tomcat. Accessed: 2017-05-19.

[2] BIM. https://www.nationalbimstandard.org/faqs. Accessed: 2017-06-19.

[3] D3.js. https://fr.wikipedia.org/wiki/D3.js. Accessed: 2017-05-18.

[4] Eclipse. https://en.wikipedia.org/wiki/Eclipse_(software). Accessed: 2017-05-19.

[5] Google Charts API. https://en.wikipedia.org/wiki/Google_Chart_API. Accessed: 2017-05-18.

[6] Java. https://en.wikipedia.org/wiki/Java_(programming_language). Accessed: 2017-05-18.

[7] Java EE. http://www.oracle.com/technetwork/java/javaee/overview/index.html. Accessed: 2017-05-18.

[8] Oracle RDBMS. https://en.wikipedia.org/wiki/Oracle_Database. Accessed: 2017-05-19.

[9] Prolog. https://en.wikipedia.org/wiki/Prolog. Accessed: 2017-05-18.

[10] SWI Prolog. https://en.wikipedia.org/wiki/SWI-Prolog. Accessed: 2017-05-18.

[11] V$SQLAREA. https://docs.oracle.com/cd/B19306_01/server.102/b14237/dynviews_2129.htm#REFRN30259. Accessed: 2017-05-19.

Part V

Appendixes

APPENDIX A

Other algorithms

In this appendix we extend the algorithms *RightReduce*, *LeftReduce* and *NonRedundant* presented in [Mai83] for FDs to STFDs. These algorithms are useful to compute a minimal cover according to the Algorithm 2 given in Section 3.4 (Chapter 3, page 41). They allow the removal of the redundant STFDs and extra attributes from the final set of STFDs.

At the end of the minimal cover algorithm (i.e. Algorithm 2, line 7) the idea is to clean up the obtained set F' of STFDs as follows:

$$F' := Union(LeftReduce(NonRedundant(F'))).$$

In our case, we do not consider the right-hand sided attributes in the minimization procedure as the considered STFDs always have just one attribute on their right side.

A.1 Algorithm Member

This algorithm (Algorithm 4) given a set F of STFDs and a STFD $f : X, L^G, T^H \rightarrow Y$, checks if f can be logically implied by F or not. It verifies that all the right-hand attributes of f (i.e. $A_i \in Y$) are present in X_F^+ with at least one granularity couple that is coarser than both G and H.

A.2 Algorithm NonRedundant

Algorithm 5 scans the set F of STFDs and eliminates the ones that already exist or that can be derived from the set F. The principle idea is to remove each time from the set F' the concerned STFD f (i.e. $F' \backslash \{f\}$) and then check if it is still derivable using the Member algorithm (Algorithm 4). If Member returns *false*, i.e. f can not be derived from $F' \backslash \{f\}$, the algorithm moves to the next STFD, otherwise the STFD f is removed from the final set of STFDs.

Algorithm 4 Member

Require:
F: a set of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$
f: a STFD $X, L^G, T^H \rightarrow Y$

Ensure:
B contains *true if* $F \vdash f$, *false otherwise*

1: $B := true$
2: **for each** $A \in Y$ **do**
3: **if** $\nexists \{(A, G_1, H_1), \ldots, (A, G_m, H_m)\} \subseteq X_F^+$ such that $(G, H) \preceq_C \{(G_1, H_1), \ldots, (G_n, H_n)\}$ **then**
4: $B := false$
5: **end if**
6: **end for**

Algorithm 5 NonRedundant

Require:
F: a set of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$

Ensure:
G: a non redundant set of dependencies equivalent to F

1: $F' := F$
2: **for each** $f : X, L^G, T^H \rightarrow Y \in F$ **do**
3: **if** $Member(F' \backslash \{f\}, f)$ **then**
4: $F' := F' \backslash \{f\}$
5: **end if**
6: **end for**

A.3 Algorithm LeftReduce

The aim of this algorithm, Algorithm 6, is to remove the not necessary left-hand sided attributes of each STFD f from the set F of STFDs. It follows the same idea as the previous algorithm: we remove the concerned attribute from f (which gives f') and then we check if f can be derived from the STFDs set without the removed attribute (i.e. $F' \backslash \{f\}$ to which we add f'). If f can be derived from the new set of STFDs (i.e. Member returns true) f will be replaced by f'.

Algorithm 6 LeftReduce

Require:

F: a set of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$

Ensure:

F': a left-reduced set of dependencies equivalent to F

1: $F' := F$
2: **for each** $f : X, L^G, T^H \rightarrow Y \in F$ **do**
3: **for each** attribute $A \in X$ **do**
4: $X' := X \backslash \{A\}$
5: $f' := X', L^G, T^H \rightarrow Y$
6: $f' := (X - A), L^G, T^H \rightarrow Y$
7: **if** $Member((F' - f) \cup f', f)$ **then**
8: $F' := (F' \backslash \{f\}) \cup f'$
9: **end if**
10: **end for**
11: **end for**

A.4 Algorithm Union

This algorithm (Algorithm 7) aims to reduce, if possible, the number of the STFDs of the final set. It scans the different STFDs and verifies if there are some other STFDs that share the same left-hand sided attributes and the same spatio-temporal granularities. In that case, the concerned STFDs will be in a single STFD together with right-hand sided attributes being the union of right-handed sided attribute from each STFD.

Algorithm 7 Union

Require:

F: a set of STFDs over $(\mathcal{U}, \mathcal{S}, \mathcal{T})$

Ensure:

F': the union of F

$F' := F$
repeat
  $F'' := F'$
  for each $f : X, L^G, T^H \rightarrow Y \in F'$ **do**
    $W := Y$
    for each $f' : X, L^G, T^H \rightarrow Z \in F' \backslash \{f\}$ **do**
      $W := W \cup Z$
    end for
    $F' := (F' \backslash \{f\}) \cup \{X, L^G, T^H \rightarrow W\}$
  end for
until $F' = F''$

APPENDIX B

Extended Version of the Paper for IDEAS 2016

Optimization of a Class of Temporal Queries: A Case Study on Intelligent Buildings

Abstract

Semantic Query Optimization (SQO) dates back to the eighties and basically relies on integrity constraints (mainly functional dependencies) to achieve logical query optimization. Applied to temporal databases with integrity constraints given as temporal functional dependencies (TFDs), SQO turns out to be very difficult in the general case due to the time dimension. Motivated by an application on Intelligent buildings, we have defined a simple class of temporal queries for which SQO techniques apply. Given a temporal database, a set of TFD and a logical temporal query belonging to the class, we propose a new rewriting technique allowing to produce a more efficient query, equivalent to the initial query. Interestingly, the time dimension opens new opportunities for SQO.

We apply our proposition to a scenario of Intelligent building (IBs). We point out that temporal databases and TFDs are well-suited for modeling sensor data in IB. For instance, we show how IB inhabitants' preferences on their life condition can be naturally represented with TFDs.

In this setting, we have implemented and experimented our approach on synthetic sensor data. We have defined several use cases and tested our semantic temporal query optimization technique.

1 Introduction

Temporal databases are required whenever large volumes of timestamped data need to be stored and queried with high-level query languages. Nowadays, for instance, many applications dealing with sensor data could benefit from temporal DB techniques [1]. For that reason, after decades of research, temporal DB are gaining popularity. As an evidence, SQL:2011 incorporates some temporal features (e.g., valid time) – although there is still no proper standardization of temporal SQL. Also, to meet the increasing needs of tools for handling temporal DB, major DBMS companies have developed their own temporal extensions. Together with temporal DB, temporal constraints, mainly Temporal Functional Dependencies (TFDs), have been defined in [17, 19, 8, 18, 20, 1, 3]. They extend classical Functional Dependencies (FDs) with time granularity.

In another context, Semantic Query Optimization (SQO) dates back to the eighties and basically relies on integrity constraints, mainly FDs, to achieve logical query optimization.

Several techniques have been proposed [6] in the past. One of the main problems of SQO was the lack of constraints from the application domain. Moreover, even though FDs are definitely useful at DB design time, the availability of a large set of FDs for SQO is questionable. Moreover, SQO turns out to be very difficult in temporal databases with TFDs in the general case due to the time dimension.

Intelligent Buildings. To be concrete and without loss of generality, we consider IB of medium to big size hosting, typically several hundreds of inhabitants over a hundred apartments. An IB is equipped with sensors and actuators dedicated to the control of the main functions of the building, like light, temperature, etc. In our setting, the IB is managed by a central entity, called the manager hereafter. The grand objective is to enhance both the comfort of the inhabitants and the ecological impact of the building's use at the same time. To do so, the inhabitants and the manager are tied together by a yearly contract, renewable for several years, in which the inhabitants define their individual preferences in terms of comfort, i.e., by stating the average temperature desired, etc. Such contracts allow to assess, at the end of each year, whether those preferences have been respected or not, with a personalized level of accuracy. A lower level of accuracy could be rewarded with a lower contract cost for the inhabitants, as it will ease the task for the manager. For example, regarding the temperature preferences, Bob may consider that he does not mind about temperature fluctuations at the minute scale but only at the hour scale, and therefore decide that, in the yearly report, when looking back, the temperature of his flat at any minute can be considered equal to the temperature at the first minute of the hour. With this scenario, we show how the inhabitants' preferences on their life condition (when they think that it is more interesting to check the temperature, ...) can be naturally represented and approximated with TFDs.
Then, we identify a particular class of simple queries that turn out to be useful in the context of IBs. This class gathers together queries asking for one particular value (first, last, min ...) from a given relation for each time interval. For instance, a query q may ask for the first value of the temperature for each hour of a given day.

Paper contribution. Motivated by an application on Intelligent buildings, we have defined a simple class of temporal queries for which SQO techniques apply. Given a temporal database, a set of TFD and a logical temporal query belonging to the class, we propose a new rewriting technique allowing to produce a more efficient query, equivalent to the initial query.

We apply our proposition to a scenario of Intelligent building (IBs). We point out that temporal databases and TFDs are well-suited for modeling sensor data in IB. For instance, we show how IB inhabitants' preferences on their life condition can be naturally represented with TFDs. In this setting, we have implemented and experimented our approach on synthetic IB sensor data. We have defined several use cases and tested our semantic temporal query optimization technique. To the best of our knowledge, this is the first attempt to use TFDs for semantic query optimization.

Paper organization. In the next section we review the temporal notions upon which our contribution is defined. In Section 3 we detail the steps to follow in order to rewrite the identified class of temporal queries into equivalent queries with respect to a set of TFD. Then, in Section 4, we explain how we intend to use our approach into intelligent building. We present the experiments we have performed in Section 5. Section 6 contains an overview of SQO approaches and temporal databases. In Section 7, we summarize our work and conclude by pointing to future work.

2 Preliminaries

We assume that the reader is familiar with classical data-bases terminology. In the following, we borrow the definitions of temporal databases with time granularities defined by Bettini et al. in [1]. In this section, we summarize their definitions that we use to build our proposition, and illustrate them using our IB scenario.

2.1 Temporal databases

A temporal database contains time-varying data [14], i.e., a set of facts associated with one or more temporal contexts [1] (e.g., validity time, transaction time, or other time dimensions).

Example 1.

sid	*owner*	*temperature*	*time*
s1	Bob	20	2015/03/02 11:59:00
s2	Clara	23	2015/03/02 11:59:00
s1	Bob	20	2015/03/02 12:01:00
s2	Clara	23	2015/03/02 12:01:00
s1	Bob	20	2015/03/02 12:01:30
s2	Clara	24	2015/03/02 12:02:00
s1	Bob	21	2015/03/02 12:03:00
s1	Bob	21	2015/03/02 12:05:00
s2	Clara	24	2015/03/02 13:02:00

Figure 1: An instance of *sensor*

Let us consider the data from two temperature sensors in an IB as given in Figure 1, where *sid* is the sensor identifier and *time* is the associated time value which corresponds to the data acquisition time instant. The attribute *time* holds the date (year/month/day) as well as the time of the day up to seconds. The owner of the sensor is also indicated (here, Bob or Clara). This example will be reused throughout the paper.

Our basic assumption is to consider discrete time domain, quite common in database and temporal reasoning. Let $T = (\mathbb{N}, \leq)$ be a discrete time domain with a total order $\leq$, a time instant is any value from $\mathbb{N}$. Thus, the time instant or the instant denote single points in time.

A chronon (or time unit) denotes the indivisible time period between 2 consecutive instants. This means that given T, no event may occur between 2 time instants. A chronon can be as small as possible, e.g., from $1s$ to $10^{-9}s$.

2.1.1 Time granularities

The representation of time granularities is essential. For instance, we would like to define the time granularity *hour* as follows, assuming chronons are seconds: $hour(1)$: is the set of the seconds from 1 to 3600, $hour(2)$: is the set of the seconds from 3601 to 7200 ...

A time granularity is a mapping G from $\mathbb{N}$ to subsets of T. A non-empty subset $G(i)$, $i \in \mathbb{N}$, of the time granularity G is called granule. The granules in a granularity do not overlap. More formally, for all integers i and $j \in \mathbb{N}$ with $i < j$ the following two conditions are satisfied: **(1)** $G(i) \neq \emptyset$ and $G(j) \neq \emptyset$ imply that each element of $G(i)$ is less than all elements of $G(j)$, and **(2)** for a given integer k such that $i < k < j$, if $G(i) \neq \emptyset$ and $G(j) \neq \emptyset$ then $G(k) \neq \emptyset$.

In our IB scenario, we may need different granularities, namely *second*, *minute*, *hour*, *day*, *month* and *year*. We say that a granularity G is *finer than* a granularity H, denoted $G \preceq H$, if for each integer i, there exists an integer j such that $G(i) \subseteq H(j)$. If $G \preceq H$, then H is *coarser than* G. For instance, *second* $\preceq$ *minute* and *hour* $\preceq$ *year*.

2.1.2 Temporal databases with multiple granularities

A *temporal module schema* is a pair $\mathbf{M} = (R, G)$, where R is a relation schema and G is a time granularity. Let $Tup(R)$ be the set of all possible tuples defined over R. A *temporal module* is a triple $\mathcal{M} = (R, G, \varphi)$, where (R, G) is a temporal module schema and φ is a mapping, called a *time windowing* function, from $\mathbb{N}$ to $\mathcal{P}(Tup(R))$. The function φ in a temporal module (R, G, φ) gives the tuples that hold at a time granule $G(i)$ of granularity G. When clear from context, we shall use the time instants instead of integer to describe a particular time windowing function, e.g., φ(2015/03/02 11:59:00) instead of $\varphi(i)$ for some integer value i.

Example 2. Let us consider the following temporal module schema $\mathbf{M}_1 = (sensor, second)$ where: **(1)** *sensor=(sid,owner,temperature)* is the relation schema, **(2)** *second* is the time granularity. Then, data from Table 1 corresponds to the function φ_0 given in Figure 2.

φ_0(2015/03/02 11:59:00) = $\{< s1, Bob, 20 >, < s2, Clara, 23 >\}$,
φ_0(2015/03/02 12:01:00) = $\{< s1, Bob, 20 >, < s2, Clara, 23 >\}$,
φ_0(2015/03/02 12:01:30) = $\{< s1, Bob, 20 >\}$,
φ_0(2015/03/02 12:02:00) = $\{< s2, Clara, 24 >\}$,
φ_0(2015/03/02 12:03:00) = $\{< s1, Bob, 21 >\}$,
φ_0(2015/03/02 12:05:00) = $\{< s1, Bob, 21 >\}$,
φ_0(2015/03/02 13:02:00) = $\{< s2, Clara, 24 >\}$.

Figure 2: The mapping function φ_0

φ_1(2015/03/02 10:00:00) = $\emptyset$,
φ_1(2015/03/02 11:00:00) = $\{< s1, Bob, 20 >, < s2, Clara, 23 >\}$,
φ_1(2015/03/02 12:00:00) = $\{< s1, Bob, 20 >, < s2, Clara, 23 >, < s2, Clara, 24 >, < s1, Bob, 21 >\}$,
φ_1(2015/03/02 13:00:00) = $\{< s2, Clara, 24 >\}$.

Figure 3: The mapping function φ_1

Clearly, the same data can be represented at a different granularity, let us say *hour*, with

another mapping function φ_1 as given in Figure 3.

A *temporal database schema* R is a fixed set of temporal module schemas. A *temporal database* d is a finite set of temporal modules defined over R, i.e., an instantiation of a windowing function for each module schema.

2.2 A temporal logical query language

As far as temporal data are concerned, we have to find some mechanisms to represent how the data is interpreted through time. Therefore, in this section we borrow the formalization of the temporal semantics introduced by Bettini et al. in [1] namely, MQL^F, point-based assumptions and interval-based assumptions. MQL^F is a query language, similar to domain relational calculus, on temporal databases that allows to specify semantic assumptions. Semantic assumptions for temporal databases allow a compact representation of potentially infinite temporal data. These assumptions may be used by the DBMS to compute values not explicitly given and to convert data from one granularity to another.

2.2.1 MQL^F

MQL^F (Multisorted Query Language with Functions) is a query language on temporal modules. An MQL^F query is of the form: $\{\mathbf{x_1} \dots \mathbf{x_k}, \mathbf{h} : \mathbf{H} \mid \psi(\mathbf{x_1}, \dots, \mathbf{x_k}, \mathbf{h})\}$ where $x_1 \dots x_k$ are variable names or constants of the nontemporal sort, h is a temporal variable of granularity H, and $\psi(x_1, \dots, x_k, h)$ is an MQL^F formula whose only free variables are among $x_1, \dots, x_k, h$.

Example 3. The following MQL^F query asks for the seconds in which a sensor gave a value under 7 degrees in Bob's house: $\{x, s : second \mid \exists y, z\ (sensor(x, y, z, s) \wedge (y = "Bob") \wedge (z < 7))\}$. In this query the variables x, y, z and s respectively bind the attributes *sid*, *owner*, *temperature* and *time*.

To handle temporal variables belonging to different granularities in the same query, the binary predicate **IntSec** is defined as follows: $\mathbf{IntSec}_{G,H}(h, g)$ is **TRUE** if and only if the intersection of the corresponding absolute time sets of granule g of G and granule h of H is not empty. For instance, if the granularity of data is *second* and the user wants a result with the granularity *minute* we have to ask for the tuples that occurred during the seconds of each minute. A more elaborated example follows.

Example 4. We can imagine another query that asks for the months in which a sensor gave a value over 48 degrees for at least one second:
$\{x, t : month \mid \exists y, z\ \exists s : second\ \ (sensor(x, y, z, s) \wedge (z \geq 48)\ \wedge\ \mathbf{IntSec}_{month,second}(t, s))\}$.

Given a temporal database *db*, the answer of a MQL^F query $Q = \{x_1 \dots x_k, g : G \mid \psi\}$, denoted $answer(Q, db)$, is a temporal module (R, G, φ), where R is a relation schema of arity k and φ is given as follows: $\langle a_1, \dots, a_k \rangle$ is in $\varphi(m)$ **iff** ψ is **TRUE** when the free variables $x_1, \dots, x_k$, and g are substituted with the constants $a_1, \dots, a_k$ and m. A complete description of MQL^F is out of the scope of this paper, the interested reader may refer to [1].

2.2.2 Semantic assumptions

Point-based assumptions are usually used on temporal databases with a single granularity in order to derive information at certain granules of time based on the information explicitly given at different granules of the same time granularity. In our IB scenario, *persistence assumption* are particularly relevant as it allows to say that the values of certain attributes persist in time unless they are explicitly changed. Hence, the assumption $P_X(Y^{persis})$ indicates that having explicit values of the attributes X and the attributes Y at a certain time, these values will persist in time until we find explicit values that are the same for X but different for Y.

Example 5. Considering the previous examples, we would like to express that the temperature of a given sensor persists in time until a different temperature, coming from the same sensor, is found. Such an assumption is written as: $P_{sid}((temperature)^{persis})$. For instance, for sensor s_1, the temperature is considered to be equal to $21°C$ for each second after 2015/03/02 12:05:00 until the arrival of a new value from the sensor having $sid = s_1$.

We need also to derive information valid between different granularities. This is known as *Interval-based assumptions*. Consider the following example.

Example 6. In order to derive from $\mathcal{M} = (sensor, second, \varphi)$ another temporal module $\mathcal{M}' = (sensor, hour, \varphi')$, we have to explicit how to proceed to pick up the desired value at the hour granularity.

Two conversion methods, namely *first* and *last*, are useful to derive a temporal module $\mathcal{M} = (R, H, \varphi')$ from a temporal module $\mathcal{M} = (R, G, \varphi)$ where $G \preceq H$. Thereby, for each granule $H(i) \in H$ there is a sequence of granules $G(j_1), \ldots, G(j_k)$ of granularity G, all contained in the granule $H(i)$. Hence, both conversion methods define which granule from the sequence $G(j_1), \ldots, G(j_k)$ of granularity G have to be associated to the granule $H(i)$, i.e., the value at the granule $G(j_1)$ if it is the conversion method $first$ and the value at the granule $G(j_k)$ if it is the conversion method $last$.

Example 7. Consider the assumption $first$ applied to the previous example. The mapping φ' contains for each hour the value of the temperature appearing at the first existing second belonging to the given hour. For instance, using $first$, φ' will be defined as given in Figure 4.

$$\varphi'(2015/03/02\ 11{:}00) = \{< s1, Bob, 20 >, < s2, Clara, 23 >\},$$
$$\varphi'(2015/03/02\ 12{:}00) = \{< s1, Bob, 20 >, < s2, Clara, 23 >\}$$
$$\varphi'(2015/03/02\ 13{:}00) = \{< s2, Clara, 24 >\}.$$

Figure 4: The mapping function φ'

Example 8. From previous examples, the MQL^F query with the assumption $first$ can be represented as follows: $Q = \{x, y, z, t : hour \mid \exists s : second(sensor(x, y, z, s) \wedge IntSec_{hour,second}(t, s) \wedge (\forall s' : second, y', z'(sensor(x, y', z', s') \wedge (s' < s)) \Rightarrow \neg IntSec_{hour,second}(t, s'))) \}$. This query selects a tuple at granule $hour(t)$ of granularity $hour$ if it appears at a granule $second(s)$ that

is covered by $hour(t)$ and there is no other tuple appearing at $second(s')$ that also belongs to $hour(t)$ and such that $s' < s$.

We can find conversion methods like *downward heredity*, *upward heredity*, *liquidity* in [1].

2.3 Temporal Functional Dependencies (TFDs)

Well-known Functional Dependencies (FDs) allow to say that X *functionally determines* Y in r, denoted $r \models X \rightarrow Y$ if for each pair of tuples $t_1, t_2 \in r,\ t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$, i.e., the X-values determine the Y-values in r. *Temporal functional dependency* (TFD) [18] extends classical FD with a granularity, allowing to specify that the X-values determine the Y-values *at a given granularity* in r. More formally, a TFD over a temporal module schema $\mathbf{M} = (R, G)$ is denoted by $X \rightarrow_H Y$, where $X, Y \subseteq R$ and H is a time granularity.

A TFD $X \rightarrow_H Y$ is satisfied by a temporal module $\mathcal{M} = (R, G, \varphi)$, denoted $(R, G, \varphi) \models X \rightarrow_H Y$, if for all tuples t_1 and t_2 and positive integers i_1 and i_2, the following three conditions imply $t_1[Y] = t_2[Y]$: **(1)** $t_1[X] = t_2[X]$, **(2)** $t_1 \in \varphi(i_1)$ and $t_2 \in \varphi(i_2)$, and **(3)** there exists j such that $G(i_1) \cup G(i_2) \subseteq H(j)$. The third condition means that both granules $G(i_1)$ and $G(i_2)$ of the time granularity G belong to the same granule of the time granularity H, i.e., $H(j)$. Thus, such TFD means that, during one granule of the time granularity H, the values for the attributes in X determine unique values for the attributes in Y.

Example 9. Consider the example of Table 1 and the TFDs $sid \rightarrow_{hour} temperature$ and $sid \rightarrow_{minute} temperature$. We have:

- $(sensor, second, \varphi) \not\models sid \rightarrow_{hour} temperature$:
  since a counter-example exists, when $hour = 12$ and $sid = s1$, the value of the attribute $temperature$ has two different values (20, 21).

- $(sensor, second, \varphi) \models sid \rightarrow_{minute} temperature$:
  since for every value of sid, there is a unique value of $temperature$ all along the granules of $minute$.

An axiomatization for TFDs has been proposed in [18]. It allows to define the closure of TFDs and the closure of attribute at a given granularity. We recall the axioms below:

- *restricted reflexivity* ($if\ Y \subseteq X,\ then\ X \rightarrow_{Top} Y$)[1],
- *augmentation* ($if\ X \rightarrow_G Y,\ then\ XZ \rightarrow_G YZ$), and
- *extended transitivity* ($if\ X \rightarrow_{G_1} Y\ and\ Y \rightarrow_{G_2} Z$, then $X \rightarrow_{glb(G_1,G_2)} Z$)[2].

The notation $F \vdash X \rightarrow_G Y$ means that the TFD $X \rightarrow_G Y$ is derived from F. $\overline{F}^+$ represents the set of all the TFDs derivable from F using the three finite axioms: $\overline{F}^+ =$

[1] Top denotes the temporal granularity that allows to express dependencies that always hold independently of time, i.e., classical functional dependencies.

[2] $glb(G_1, G_2)$ represents the greatest lower bound of the granularities G_1 and G_2 with respect to $\preceq$.

Table 1: Class $\mathcal{Q}$ of temporal queries

$\mathcal{Q}^{X,Y}_{H,G} = \{\mathcal{X}, \mathcal{Y}, h : H \mid \exists \mathcal{Z}, g : G\ (R(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, g) \wedge Sym\_R^X_{H,G}(\mathcal{X}, h, g) \wedge IntSec_{H,G}(h, g))\}$

$\{X \rightarrow_G Y \mid F \vdash X \rightarrow_G Y\}$. The finite closure of a finite set X of attributes with respect to a finite set F of TFDs, is defined as: $\overline{X}^+_F = \{(B, G) \mid X \rightarrow_G B \in \overline{F}^+$ and there is no $X \rightarrow_H B$ *in* $\overline{F}^+$*such that* $G \prec H\}$ where B is an attribute.

The implication problem is polynomial and the following property holds:
$F \models X \rightarrow_G B$ **iff** there exists $\{(B, G_1), \ldots, (B, G_m)\} \subseteq \overline{X}^+_F$ such that $G \preceq_C \{G_1, \ldots, G_m\}$[3].

3 Semantic Temporal Query Optimization (STQO) for a class of queries

Semantic Query Optimization (SQO) methods take advantage of the integrity constraints in order to detect the queries' anomalies [5] and to develop query's rewriting techniques, such as adding or deleting joins or predicates [22, 7, 4, 13, 9]. Given a query, the basic idea of SQO is to find an equivalent query that is less expensive (with respect to CPU and memory usage).

With the time dimension, the problem becomes harder and as far as we know, SQO has not been studied in the context of temporal databases with TFDs. From our application to intelligent buildings, we have identified the class $\mathcal{Q}$ of queries given in Table 1: We consider a single module $\mathcal{M} = (R, G)$ with $R = \{A_1 \ldots A_n\}$ and $R = X \cup Y \cup Z$, X, Y, Z pairwise disjoints. To alleviate the notations, for each set of attributes $X \subseteq R$, we associate a set of variables $\mathcal{X}$ and we shall denote by $R(X, Y, Z)$ the formula $R(A_1, \ldots, A_n)$. Each query has four parameters, X, Y, H and G. The symbol $Sym\_R^X_{H,G}(\mathcal{X}, h, g)$ refers to either $First\_R^X_{H,G}(\mathcal{X}, h, g)$ or $Last\_R^X_{H,G}(\mathcal{X}, h, g)$. For instance,
$First\_R^X_{H,G}(\mathcal{X}, h, g) = \forall g' : G, \mathcal{Y}, \mathcal{Z}(R(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) \wedge (g' < g) \Rightarrow \neg IntSec_{H,G}(h, g'))$.

The problem statement can be sketched as follows:

> Given a temporal database's schema R with some semantic hypothesis, a set of TFDs and a logical query $Q \in \mathcal{Q}$, rewrite Q into Q' such that Q is equivalent to Q' in every database d over R and Q' is more efficient than Q.

Let us consider the following query that computes the first value of temperature per each hour by using the assumption $first$:
$Q = \{x, z, t : hour \mid \exists y, s : second(sensor(x, y, z, s) \wedge IntSec_{hour,second}(t, s)$
$\wedge (\forall s' : second, y', z'(sensor(x, y', z', s') \wedge (s' < s)) \Rightarrow \neg IntSec_{hour,second}(t, s')))\ \}$.

[3] $G \preceq_C \{G_1, \ldots, G_m\}$ denotes the fact that G is *collectively finer* than the set $\{G_1, \ldots, G_m\}$ which means that for each positive integer i there exist $1 \leq k \leq m$ and a positive integer j such that $G(i) \subseteq G_k(j)$. As a particular case, $G \leq H$ implies $G \leq_C \{H\}$.

We first consider a shorthand for the assumption $first$ applied to $Sensor$.
To do so, we define a new symbol $First\_sensor$ as follows: $First\_sensor^{sid}_{hour,second}(x,t,s) = \forall s' : second, y', z'\ (sensor(x,y',z',s') \wedge (s' < s)) \Rightarrow \neg IntSec_{hour,second}(t,s')$.
This predicate is $TRUE$ if the $sid\ x$ appears first at $second\ s$ within $hour\ t$ (i.e., a tuple with sid x and $second$ s' such that $s' < s$ and s' in $hour\ t$ doesn't exist). Then, the previous query can be rewritten as: $Q = \{x, z, t : hour \mid \exists s : second, y\ (sensor(x,y,z,s) \wedge IntSec_{hour,second}(t,s) \wedge First\_sensor^{sid}_{hour,second}(x,t,s))\ \}$.

Assume now the following TFD has been defined: $f_0 = sid \rightarrow_{hour} temperature$. Clearly, the intuition is to remove from Q the predicate $First\_sensor$. Since the temperature values doesn't change all along the same hour. Indeed, the TFD f_0 implies that the first value of each hour is the same as all the hour's values for a given sensor. Hence, the query Q can be rewritten into Q_{rw} thanks to f_0 as follows:
$Q_{rw} = \{x, z, t : hour \mid \exists s : second, y(sensor(x,y,z,s) \wedge IntSec_{hour,second}(t,s)\}$.
We notice that $sid^+_{\{f_0\}} \supseteq \{(temperature, hour), (sid, Top)\}$. The mechanism presented in this example can be generalized to all logical queries using interval-based assumptions like $first$, $last$ or aggregate functions (e.g., min, max, ...), that are very common in temporal databases managing sensors data. TFDs allow us to remove some "costly" predicates. From a practical point of view, temporal extensions of SQL allow more flexibilities since such conditions are implemented with a `Group By` clause [1].

Given such a query in $\mathcal{Q}$, the rewriting technique is described in Algorithm 1. The idea is to identify the conditions under which the shorthand for the assumption Sym can be safely removed from the initial query. The conditions can be derived from a closure computation of the set of attributes occurring in Sym_R (line 1). It means that if all attributes in the result of Q can be inferred with the set of TFD F (line 2), the condition turns out to be useless and is removed (line 3).

Algorithm 1 rewriteQuery

Require:
$Q^{X,Y}_{H,G} \in \mathcal{Q}$
F: a set of TFDs
Ensure:
Q_{rw}: a query

1: Compute $\overline{X}^+_F$
2: **if** (**for all** $A_j \in Y$, $(A_j, I) \in \overline{X}^+_F$ such that $H \leq I$) **then**
3: $\quad Q_{rw} = \{\mathcal{X}, \mathcal{Y}, h : H \mid \exists \mathcal{Z}, g : G\ (\ R(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, g) \wedge IntSec_{H,G}(h,g))\}$
4: **else**
5: $\quad Q_{rw} = Q^{X,Y}_{H,G}$ /* *no rewriting* */
6: **end if**

Theorem 1 *The query Q_{rw}, resulting from Algorithm 1, is equivalent to the initial query*

$Q^{X,Y}_{H,G}$ *with respect to the set of TFDs.*

Proof.
($\mathbf{Q \subseteq Q_{rw}}$)
The result is obvious since Q is more selective than Q_{rw}.
($\mathbf{Q_{rw} \subseteq Q}$)
Assume the contrary, i.e. ($Q \subset Q_{rw}$).
Let d_0 be a database over R. Then, there exists a tuple $t_0 \in answer(Q_{rw}, d_0)$ and $t_0 \notin answer(Q, d_0)$. Suppose $t_0 =< X_0, Y_0, h_0 >$ such that h_0 is a granule of the granularity H. Assume without loss of generality that Sym_R is equal to $First\_R$.
Since $t_0 \in answer(Q_{rw}, d_0)$ there exists a tuple $t'_0 =< X_0, Y_0, Z_0, g_0 >$ over R such that g_0 is a granule of the granularity G, where g_0 is contained in h_0.
Since $t_0 \notin answer(Q, d_0)$, t'_0 is not the first tuple at the granularity G in the granule h_0. Then, there exists another tuple $t'_1 =< X_0, Y_1, Z_1, g_1 >$ over R with g_1 a granule of G and g_1 is contained in h_0 such that t'_1 is the first tuple at the granularity G in the granule h_0 with $t'_0[X] = t'_1[X] = X_0$. The tuple t'_1 produces a corresponding tuple $t_1 =< X_0, Y_1, h_0 >$ that belongs to both $answer(Q, d_0)$ and $answer(Q_{rw}, d_0)$.
Since $t_1 \in answer(Q, d_0)$ and $t_0 \notin answer(Q, d_0)$, then $t_0 \neq t_1$ and $Y_0 \neq Y_1$, hence $t_0[Y] \neq t_1[Y]$ and $t'_0[Y] \neq t'_1[Y]$.
The contradiction follows since $F \models X \rightarrow_H Y$.

4 Application to IBs

First, we note that temporal databases are clearly relevant in the context of IBs since the sensors output timestamped data and the time dimension is required to deal with several needs as:

- the sensor value at a given time instant,
- the averages of the temperature in an inhabitant's house during a given year,
- a report of the temperature averages among different granularities (hour, day ...),
- annual charts of temperature according to the users' requirements.

In fact, the use of semantic assumptions of temporal databases (presented in section 2.2.2) could be relevant in the following situations:

1. **Interval-based assumptions**: when dealing with sensors whose values are updated regularly, e.g., temperature sensor, luminosity sensor, etc. For example, Bob, a house inhabitant, wants to know the evolution of his bedroom's temperature the last night (e.g., 13 Sept 2015). However, Bob doesn't want to have a report containing a lot of

values of temperature, so asks for one temperature value per hour (e.g., the first value of each hour).

2. **Point-based assumptions**: when dealing with sensors whose values are less often refreshed, e.g., a sensor detecting the door's position (opened or closed). For instance, Carla, another house inhabitant, left her house and she is not sure if she closed the entry door. If the system doesn't use the *persistence* assumption, the query asking for the door's position at the time instant t_i may not return any tuple unless someone is opening (or closing) the door exactly at the same time instant t_i. The use of the *persistence* ensures a non empty answer containing the last valid door's position.

The second point that makes IB a relevant application field to apply our proposition is the fact that user's approximations can be easily gathered. In fact, the concrete effects of an approximation are meaningful for the inhabitants and, conversely, a building manager could very well ask the inhabitants to express approximations in natural language, before translating them into TFDs. Using such easily collected semantic information in the optimization of the monitoring system of an IB can enhance the services given to the inhabitants and the building manager.

Moreover, it is very commun for an IB manager to request some portion of the sensor data at different time granularities, e.g. *hour* instead of *second*. From our experience, such a simple demand is very recurring in the context of IBs. Indeed, we may encounter queries asking for the first temperature value of each hour, the first luminosity value of each minute, the last CO_2 value of each hour, the last door position of each day ... In fact, it is very useful to be able to get back a representative tuple of each time interval when switching from a time granularity to another. In our case, we make such task possible thanks to the interval based assumptions. Thus, these queries may be gathered in the class $\mathcal{Q}$ previously identified in Table 1. Queries of $\mathcal{Q}$ are meant to extract one tuple from each time interval, based on time granularities and interval-based assumption.

5 Experiments

In this section we detail how we used temporal databases and temporal queries in an IB scenario. We then expose the results we got by comparing the performance of the user's queries with forms derived from user-defined TFDs.

5.1 Implementation

In the following we present the DBMS and the data that we used to effect our experiments.

5.1.1 Temporal DBMS

Although many researches focused on temporal query languages, a general temporal SQL standardization is still missing and there exists an important gap between theory and practice.

We chose to experiment our approach using *Oracle 12C* in order to take advantage of its implementation of the *temporal validity* which supports point-based assumption *persistence* and facilitates the use of temporal attributes. In fact, *temporal validity*, in the *Oracle 12C* lexicon, is embodied by the type *period*, that is meant to allow the user to define valid times (useful to check the validity of the tuples), which are actually made out of two timestamps, i.e., *start* and *end*. Furthermore, the granularities of interval-based assumptions can be expressed thanks to the function *trunc* that returns a date truncated up to a specific unit of measure. For instance, let us consider a date d_0 ="20/09/15 12:15:55" corresponding to the date format "$dd/mm/yy$ $hh24$:mi:ss", where the parameters dd, mm, yy, $hh24$, mi and ss represent respectively day, month, year, hour, minute and second. The function $trunc(d_0,$"$hh24$"$)$ will return a value d'_0 corresponding to the date d_0 considered up to hours only, that is d'_0 = "20/09/15 12". It follows that checking whether two dates belong to the same hour can be achieved by comparing the results of the *trunc* function applied to both dates, with the format parameter set to the value "$hh24$". For instance, let d_1 ="20/09/15 12:25:55". Since $d'_1 = trunc(d_1,$"$hh24$"$)$ equals d'_0, we can conclude that d_0 and d_1 belong to the same hour.

5.1.2 Data

Consistently with the previous sections, we focus on temperature sensors. The schema that we will use in the following sections is: ***sensor(sid,owner,temperature,vt_start,vt_end)***, where *vt_start* and *vt_end* represent temporal validity.

At the acquisition of a new value from $sensor_i$ at the time instant δ_j, the underlying temporal database is filled and updated as follows :

1. the system binds the value with the temporal validity attributes, i.e., *vt_start* and *vt_end*: the *vt_start* attribute is affected the value of the time instant δ_j, while *vt_end* is set to the value δ_j+*frequency*, where *frequency* is a time interval associated to the sensor type,

2. the system updates the value of *vt_end* of the last previous tuple concerning $sensor_i$: its value is set to $\delta_j - \Delta_t$, where Δ_t is a constant value ($1ms$ in our setting). This is meant to express the fact that this last tuple was valid until a small period (lasting Δ_t) before the new tuple.

We use the following data set:

- *Synthetic data:* We simulated the operation of a varying number of temperature sensors (10, 100, ...), each sending data at a frequency of five minutes. We also varied the periods of collecting data (i.e., on the data stored over 1 year, 2 years periods, ...). We produced several tables with different number of tuples (10^6,10^7, ...).

5.1.3 Experimentation platform

We chose to do our experiments in a user-like technical environment. Thus, the experiments were undertaken on a personal laptop with the following hardware and software:
CPU: Intel ® Core^{TM} i7-4600U, **RAM**: 8 GB 1600MHz DDR3L 1DM, **HD**: 256 GB SATA-3, **OS**: Windows 7 Pro x64.

5.2 Ongoing Experiments

In the following we have different sets of experiments along which we introduce a user-defined TFD. First, using the generated data, we compare the execution time of a query Q_1, that asks for the first value of temperature of each sensor during each hour, and the execution time of a query Q_2 that takes into account user-defined TFDs (e.g., $sid \rightarrow_{hour} temperature$) and consequently only asks for one value per hour. In order to assess the benefits of the use of TFDs in optimization, we compared with an "optimal" query Q_3 based on a materialized view, in which there is only one value per hour for each sensor so that we had a reference time.

Then, we compare two different ways of implementing the persistence assumption : with or without the temporal validity attributes (that are specific to temporal DB). To do so, we compare the execution time of two queries, Q_6, implementing the persistence assumptions by making use of predicates in the WHERE clause, in an SQL fashion, and Q_7, exploiting the implementation of temporal validity of *Oracle 12C*.

In the following, the values of the execution time of the different queries are average values of several executions.

5.2.1 1^{st} set of queries

The query Q_1, as presented in Figure 5, asks for the first value of temperature of each sensor for each hour. At this point, we can optimize Q_1 by taking into account the available TFD, given by our use-case. Indeed, Bob considers that "it is sufficient to consider the temperature every hour instead of every 5 minutes". Bob's approximation can be represented using the following TFD: $sid \rightarrow_{hour} temperature$. According to this temporal dependency, for a given sensor identifier, i.e., *sid*, the value of temperature doesn't change all along an hour. As a consequence, the value of each temperature sensor in Bob's house during one granule of the time granularity *hour* doesn't change. Then, using such approximation, we can derive Q_2 from the query Q_1, that asks for the first value for each hour, using the user-defined TFD $sid \rightarrow_{hour} temperature$. In fact, we don't need to search for the first value of temperature, it is sufficient to select only one value per hour. The query Q_2 is written in the Figure 5.

We also create a materialized view, mv_sensor_first, containing the first value of the temperature of each sensor of each hour, in order to compare the execution time of the optimized query, Q_2, with the execution time of an artificially optimal query Q_3 (e.g., that would benefit from a dedicated index), that asks for all the tuples of the materialized view. The query Q_3

Q_1	SELECT s1.sid , s1.temperature , trunc(vt_start,'hh24') as time FROM sensor s1 WHERE owner='Bob' AND NOT EXISTS (SELECT * FROM sensor s2 WHERE s1.sid=s2.sid AND trunc(s1.vt_start,'hh24')=trunc(s2.vt_start,'hh24') AND s1.vt_start > s2.vt_start) ORDER BY time;
Q_2	SELECT distinct sid, temperature, trunc(vt_start,'hh24') as time FROM sensor WHERE owner='Bob' ORDER BY time;
Q_3	SELECT * FROM mv_sensor_first WHERE owner='Bob';
Q_{mv}	CREATE MATERIALIZED VIEW mv_sensor_first as SELECT sid , temperature, trunc(vt_start,'hh24') as time FROM sensor s1 WHERE NOT EXISTS (SELECT * FROM sensor s2 WHERE s1.sid=s2.sid AND trunc(s1.vt_start,'hh24')=trunc(s2.vt_start,'hh24') AND s1.vt_start > s2.vt_start) ORDER BY time;

Figure 5: The first set of queries

and the query Q_{mv}, that creates the materialized view, are presented in Figure 5.

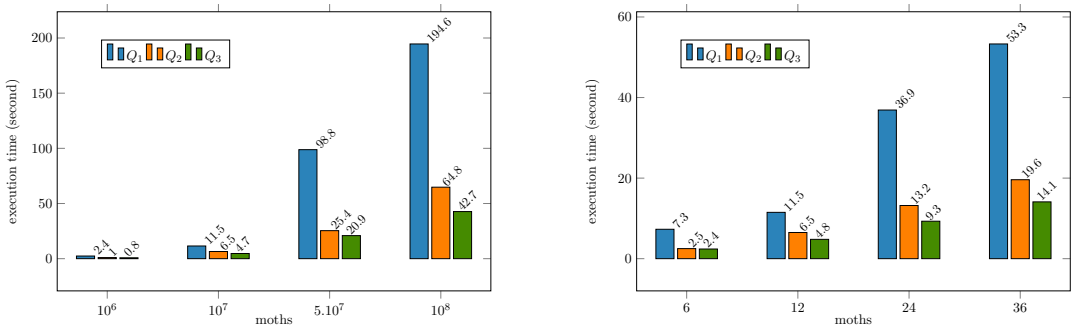
The queries, Q_1, Q_2 and Q_3, have been executed on different generated cases:

1. variation of the number of tuples: 10^6, 5×10^6, 10^7, 5×10^7 and 10^8,
2. variation of the number of months: 6, 12, 24 and 36,
3. variation of the number of sensors: 10, 100, 500 and 1000.

The obtained results are presented respectively in Figure 6a, Figure 6b and Figure 6c. For the range of number of tuples considered, the execution time of the optimized query Q_2 is cut down by a ratio of 2 to 3, compared to the initial query Q_1. The difference between the optimized query and the optimal one Q_3 is far smaller (less than 1.5). Moreover, an interesting aspect is the more tuples considered, the bigger the ratio between the execution time of Q_1 and Q_2, while the ratio between Q_2 and Q_3 remains steady. That seems to indicate the proposed optimization strategy is particularly promising for settings in which the volume of data is expected to grow fast.

5.2.2 2^{nd} set of queries

In a situation in which the *persistence* assumption is not considered neither by the DBMS nor by the user's query, a classical *SQL* query asking for the valid tuple at a given time instant t may return a void answer, unless a tuple occurs at the time instant t which is very unlikely. However, as explained above, when dealing with sensors, it is interesting to implement a *persistence* assumption strategy, and to consider that the valid value of a sensor at a time



(a) w.r.t. to the number of tuples

(b) w.r.t. to the number of months

(c) w.r.t. to the number of sensors

Figure 6: Execution time of Q_1, Q_2 and Q_3

instant t is the one that occurred at the time instant t' defined by: **(1)** $t' \leq t$, and **(2)** $\nexists t''$ such that $t' < t'' < t$. In this section we compare two SQL queries implementing the *persistence* assumption with and without the temporal features of the DBMS (Q_7 and Q_6 respectively). Those queries are written in Figure 7. The comparison of the execution time of the two queries, presented in Figure 8, shows that it is more interesting to use the temporal features of the DBMS in our case.

Q_6

```
SELECT *
FROM sensor s1
WHERE s1.vt_start= to_date('04/01/10 08:32','dd/mm/yy hh24:mi')
OR
( s1.vt_start < to_date('04/01/10 08:32','dd/mm/yy hh24:mi')
  AND NOT EXISTS
    (SELECT *
     FROM sensor s2
     WHERE s2.vt_start <= to_date('04/01/10 08:32','dd/mm/yy hh24:mi')
     AND s2.vt_start > s1.vt_start
     ));
```

Q_7

```
SELECT *
FROM sensor
AS OF PERIOD FOR valid_time to_date('04/01/10 08:32','dd/mm/yy hh24:mi');
```

Figure 7: The second set of queries

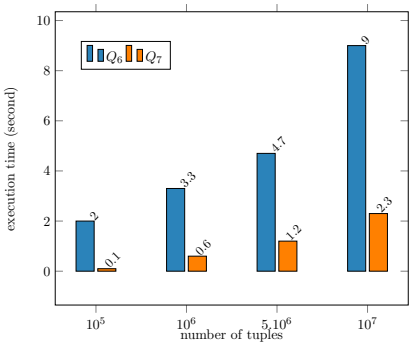


Figure 8: The execution time of Q_6 and Q_7

6 Related work

Semantic Query Optimization. Query optimization has always been one of the major concerns of the database field since System R [12] in 1979, the root of most major commercial RDBMS-optimizers. Following System R, a huge number of approaches has been suggested in the field of query optimization (e.g., semantic query optimization, index-based optimization, queries with user-defined functions optimization, ...). We are interested in the logical optimization. In the eighties, many researchers implemented different semantic optimization approaches [22, 7, 4, 13, 9] which basically used integrity constraints (such as FDs) to optimize the queries. The main idea of SQO is to identify the possible equivalences in order to rewrite the initial queries into new queries, syntactically different but semantically equivalent. Among the implemented SQO's techniques we can mention the detection of the unsatisfiable queries [22, 7], join removal or adding [4] and predicate removal or adding [13, 9].

Temporal Functional Dependencies. TFDs have been mainly introduced in order to constraint the temporal data in temporal databases. There have been an important number of articles aiming at defining and characterizing TFDs namely [17, 19, 8, 18, 20, 3].

The three first approaches [17, 19, 8] handle TFD without time granularity. In [17], the author defined a temporal relation as a temporal sequence of database states and extended each tuple with its updated version. He defined two classes of dependencies: static dependencies (i.e., standard dependencies that must be satisfied in each state) and dynamic dependencies (i.e., dependencies that relate every state to its successor). The data model in [19] was extended with a valid time which represents a set of time points. The author presented the suitable definition of FD in the presence of valid time and defines two classes of temporal dependencies: Temporal Functional Dependencies (TFDs) and Dynamic Functional Dependencies (DFDs). In [8], data contains two time dimensions: valid time and transaction time. The authors handle the problem of expressing the functional dependencies with such data.

Both [18, 20] handled multiple time granularities. The authors in [18] and in [1] defined

the time granularities and the different relationships between them. They defined the *temporal module schema* and the *temporal schema* as presented in section 2 as well as the definition of TFD that was used in the previous sections. In [20], the author extended the dependencies presented in [19] using time granularity and object identity which is a time-invariant identity that relates the different versions of the same object.

Interval-based data dependencies (ITFDs) [3] take into account the distance between two consecutive tuples, i.e., the associated valid time to each TFD.

Temporal databases and temporal query languages. Although many researches focused on temporal query languages namely [10, 11, 15], a general temporal SQL standardization is still missing. $SQL:2011$ includes parts of TSQL2 [15] which was a research work intending to present a temporal extension to SQL. Each DMBS provides specific temporal features. MQL^F is an abstract first order logic language that allows to be independent of the query languages' implementations and is therefore appropriate to define new SQO with TFDs. Load shedding techniques [16] developed in data stream management systems aims to reduce the system's charge, dropping excess load from the system by removing input tuples. Load shedding usually interferes with the physical plan of the query while our approach aims to rewrite the query at the logical level. Moreover, when the temporal database doesn't satisfy the TFDs, our approach leads to approximate the result, e.g., by selecting one tuple appearing in the concerned granule for a given sensor.

7 Conclusion

We have presented a semantic query optimization approach for a class of temporal queries in presence of a set of TFDs. We have proposed a rewriting algorithm where the key idea is to exploit background knowledge about temporal data to speed up query processing. We have applied our proposition to intelligent buildings (IB), which appears as a promising field of applications. We have argued that temporal databases are well-suited for IB, the time dimension being crucial when querying data from many sensors over long periods of time. We have also showed how inhabitants' preferences on their life condition in the building can be easily represented with TFDs. With respect to classical SQO techniques, we believe the availability of a large bunch of TFDs should revive SQO techniques. We have implemented and experimented our approach on synthetic IB sensor data. Our first results were promising and showed the validity of our approach.

For future works, we may need to limit the scope of TFDs, for instance, if the inhabitant Bob thinks that the corridor is representative of the whole house or if some TFD applies to a particular category of owners. In fact, such information are also easy to gather with a combination of spatio-temporal FDs [21] and Conditional FD [2]. Further work is also needed to tackle semantic query optimization for a more general class of queries.

References

[1] C. Bettini, S. Jajodia, and S. Wang. *Time granularities in databases, data mining, and temporal reasoning*. Springer, 2000.

[2] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 746–755. IEEE, 2007.

[3] C. Combi and P. Sala. Interval-based temporal functional dependencies: specification and verification. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):85–130, 2014.

[4] A. D'Andrea and P. Janus. Unisql's next-generation object-relational database management system. *ACM Sigmod Record*, 25(3):70–76, 1996.

[5] B. H. Genet. *Is Semantic Query Optimization Worthwhile?* PhD thesis, The University of Waikato, 2007.

[6] M. Hammer and S. B. Zdonik. Knowledge-based query processing. In *Proceedings of the sixth international conference on Very Large Data Bases-Volume 6*, pages 137–147. VLDB Endowment, 1980.

[7] A. Illarramendi, J. M. Blanco, and A. Goni. Making the knowledge base systems more efficient: A method to detect inconsistent queries. *Knowledge and Data Engineering, IEEE Transactions on*, 6(4):634–639, 1994.

[8] C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending existing dependency theory to temporal databases. *Knowledge and Data Engineering, IEEE Transactions on*, 8(4):563–582, 1996.

[9] B. G. Lowden and J. Robinson. Constructing inter-relational rules for semantic query optimisation. In *Database and Expert Systems Applications*, pages 587–596. Springer, 2002.

[10] S. B. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49(1):147–175, 1989.

[11] N. L. Sarda. Extensions to sql for historical databases. *Knowledge and Data Engineering, IEEE Transactions on*, 2(2):220–230, 1990.

[12] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM, 1979.

[13] S. T. Shenoy and Z. M. Ozsoyoglu. A system for semantic query optimization. *SIGMOD Rec.*, 16(3):181–195, Dec. 1987.

[14] R. T. Snodgrass. Temporal databases. In *IEEE computer*. Citeseer, 1986.

[15] R. T. Snodgrass. *The TSQL2 temporal query language*, volume 330. Springer Science & Business Media, 1995.

[16] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 309–320. VLDB Endowment, 2003.

[17] V. Vianu. Dynamic functional dependencies and database aging. *Journal of the ACM (JACM)*, 34(1):28–59, 1987.

[18] X. S. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical design for temporal databases with multiple granularities. *ACM Transactions on Database Systems (TODS)*, 22(2):115–170, 1997.

[19] J. Wijsen. Design of temporal relational databases based on dynamic and temporal functional dependencies. In *Recent Advances in Temporal Databases*, pages 61–76. Springer, 1995.

[20] J. Wijsen. Temporal fds on complex objects. *ACM Transactions on Database Systems (TODS)*, 24(1):127–176, 1999.

[21] J. Wijsen and R. T. Ng. Temporal dependencies generalized for spatial and other dimensions. In *Spatio-Temporal Database Management*, pages 189–203. Springer, 1999.

[22] S.-C. Yoon, L. J. Henschen, E. Park, and S. Makki. Using domain knowledge in knowledge discovery. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 243–250. ACM, 1999.

FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

NOM : CHARFI
(avec précision du nom de jeune fille, le cas échéant)

DATE de SOUTENANCE : 21/09/2017

Prénoms : Manel

TITRE : Declarative Approach for Long-Term Sensor Data Storage

NATURE : Doctorat

Numéro d'ordre : AAAALYSEIXXXX

Ecole doctorale : Ecole doctoral d'informatique et mathématique de Lyon (N° EDA 512)

Spécialité : Informatique

RESUME :
Nowadays, sensors are cheap, easy to deploy and immediate to integrate into applications. Having a set of sensors emitting data streams, our concern in this PhD thesis is to build a sensor database for applications that require long-term storage.
Since huge amount of sensor data can be generated, selecting only relevant data to be saved for further usage, e.g. long-term query facilities, is still an issue. For instance, a query asking for the temperature of a given house per day along a year may have to consider every temperature value sent from different sensors in every room every minute. In such cases, approximating data in order to reduce the considered values (i.e. relevant values with respect to the application requirements) enhances such query efficiency.
Given sensor data streams, the key idea is to consider both spatio-temporal hierarchies and Spatio-Temporal Functional Dependencies as first class-citizens for designing sensor databases on top of any relational database management system.
We propose an axiomatisation of these dependencies and the associated attribute closure algorithm, leading to a new normalization algorithm.
Then, we propose an annotation of attributes with aggregate functions allowing to specify which summarized values are of interest in the data. We can automatically both produce a concrete SQL DB and configure the data loading process.
We have implemented a prototype to deal with both database design and data loading. This allowed us to conduct experiments with, synthetic and real-life, data streams from intelligent buildings.

MOTS-CLÉS : Database design, Spatio-Temporal Functional Dependency, sensor data stream , spatio-temporal granularity hierarchies

Laboratoire (s) de recherche : Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS)

Directeur de thèse: Jean-Marc PETIT

Président de jury :

Composition du jury :
BIDOIT-TOLLU, Nicole Professeure des Universités, Université Paris Sud 11
WIJSEN, Jef Professeur des Universités, Université de Mons
BOUZEGHOUB, Amel, Professeure des Universités, Telcom SudParis
LAKHAL, Lotfi Professeur des Universités, Aix-Marseille Université
PETIT, Jean-Marc Professeur des Universités, INSA de Lyon
GRIPAY, Yann Maître de Conférence, INSA de Lyon
FOURTY, Nicolas Maître de Conférence, Université Pierre Mendes