

Diplomarbeit

Data Cleaning and Performance Tuning in the GAINS Model

ausgeführt am Institut für Informationssysteme
Arbeitsbereich für Datenbanken und Artificial Intelligence
der Technischen Universität Wien

unter der Anleitung von Univ.Prof.Dr.techn. Reinhard Pichler und
Dipl.Ing. Katrin Seyr als verantwortlicher Mitarbeiterin

durch

Maciej Piotr Makowski
Dr. F. J. Schicht-Gasse 5/20
A - 2340 Mödling

April 25, 2008

Abstract

This paper will discuss how data cleaning and performance tuning can positively affect a database driven application. Based on the example of the GAINS model the author will give various examples of actions taken that not only improve the application from the data modelling point of view, but also guarantee the correctness and completeness of the data presented by the model. Additionally it will be shown that the measures taken lower the maintenance effort and do not slow the model down. The introduced changes also set the stage for the introduction of new concepts like data warehousing or web services in the near future.

Zusammenfassung

Diese Arbeit wird zeigen, wie Datenaufbereitung (“data cleaning”) und Leistungs-optimierung (“performance tuning”) eine datenbankbasierte Applikation positiv beeinflussen können. Basierend auf dem Beispiel des GAINS Modells wird der Autor zahlreiche durchgeführte Maßnahmen anführen, die die Applikation nicht nur unter dem Gesichtspunkt der Datenmodellierung verbessern, sondern auch die Korrektheit und Vollständigkeit der durch das Modell präsentierten Daten garantieren. Zudem wird im Laufe dieser Arbeit gezeigt werden, dass diese Maßnahmen den Wartungsaufwand der Applikation verringern und das Modell nicht verlangsamen. Des Weiteren schaffen die durchgeführten Änderungen die Voraussetzungen für die Umsetzung neuer Konzepte, wie zum Beispiel Data Warehousing oder Web Services, in naher Zukunft.

Acknowledgments

I would like to thank Dr. Markus Amann and Dr. Wolfgang Schöpp at the International Institute for Applied Systems Analysis (IIASA) for allowing me to be part of the Atmospheric Pollution and Economic Development (APD) program over the past two and a half years. I know that I have already benefited a lot from this opportunity and will do so even more in the future. I would like to thank everybody in the team for giving me valuable feedback when taking time to answer my, not always comprehensible, questions. I would like to especially thank Dr. Schöpp and Dr. Fabian Wagner for the help with, and feedback on, my thesis.

I would also like to thank Dipl.Ing. Katrin Seyr and Univ.Prof.Dr.techn. Reinhard Pichler at the Database and Artificial Intelligence Group (DBAI) of the Vienna University of Technology not only for the help and feedback, but also for the possibility to write an external thesis.

It is only with the help of the above mentioned people that I was able to finalize my thesis and graduate.

I would also like to thank my family: Mamo, Tato, dziękuję za wieloletnie wsparcie i cierpliwość; Jędrek, dziękuję, że starałeś się studiować podobnie długo.

To my wife, Karolina: Thank you for your support and the motivation to finish what I started so long ago. I know I would not be where I am without you.

Last but not least, I would like to thank Kacper: You were never angry when I did not have time for you and my walks with you gave me the necessary breaks in working and inspiration.

Dziękuję. Danke. Thank you. Maciek.

Contents

1	Introduction	1
1.1	RAINS/GAINS Model	1
1.1.1	Evolution of the Model from a Software Engineering Point of View	1
1.1.2	Policy Relevance of the Model	2
1.2	Data Cleaning and Performance Tuning	3
1.2.1	Data Cleaning	3
1.2.2	Performance Tuning	4
1.3	Structure and Goal of this Paper	4
2	GAINS Basics	7
2.1	The Structure of GAINS Database Schemas	7
2.1.1	Regional Data Schemas	7
2.1.2	Central Data Schema	8
2.1.3	Central Web Schema	8
2.2	Basic GAINS Entities	9
2.3	Structure of a Scenario	9
2.3.1	Scenario-based Emission and Cost Calculation	9
2.3.2	Side Notes on Technologies	12
2.3.3	Activity Pathway	15
2.3.4	Control Strategy	16
2.3.5	Emission Vector	17
2.4	Sharing of Scenarios and Their Parts	21
2.4.1	Sharing of Scenarios	21
2.4.2	Sharing of Scenario Parts	22
2.5	User Management	22

2.5.1	User Groups	22
2.5.2	User Rights & Privileges	23
2.5.3	User Sharing	23
3	Current Challenges	25
3.1	Deployment of Data	25
3.1.1	Data Integrity	26
3.1.2	Long Query Execution Time	26
3.1.3	Runtime Optimization	26
3.2	Consistency in Data Modelling	27
3.2.1	Data Versions versus Version Owner	27
3.2.2	Applicabilities	28
3.2.3	Aggregation of Results According to Reporting Standards	29
4	Implementation	31
4.1	Key Issues to Solving Query Runtime Problems	31
4.1.1	Indexing of Data	31
4.1.2	Join Strategies	36
4.1.3	Other Pitfalls	38
4.2	Other Oracle Based Solutions to Query Runtime Problem	39
4.2.1	Materialized Views	39
4.2.2	Temporary Tables	41
4.3	Reorganizing Data Structures	42
4.3.1	Materialized Views for Data Deployment	42
4.3.2	Activity/Sector/Technology Combinations	44
4.3.3	Emission Vector Related Data	49
4.3.4	Applicabilities	65
4.3.5	Aggregation of Results According to Reporting Standards	73
4.4	Other Implementation Aspects	81
4.4.1	Calculation of Emissions	81
4.4.2	Initialization of Partial Emission Vectors	85
4.4.3	Display of Scenarios	90
5	Conclusion	99
5.1	Summary	99
5.2	Where to go from here?	99

5.2.1	Flexibility in Viewing	99
5.2.2	New Technologies	100

List of Figures

2.1	Structure of a GAINS scenario	10
2.2	Examples of BC1/PP_NEW implementation rates for SO ₂	14
2.3	Activity pathway list and data	15
2.4	Control strategy list and data	17
2.5	Emission factor data for SO ₂	19
2.6	Reduction efficiency data for SO ₂	20
2.7	Cost factors for all pollutants	20
4.1	Structure of the “act_sec_tech” table in “rains_data”	33
4.2	Execution plan for retrieving activity/sector/technology combinations relevant for PM	34
4.3	Execution plan for retrieving activity/sector/technology combinations relevant for SO ₂	35
4.4	Execution plan for retrieving activity/sector/technology combinations not relevant for SO ₂	35
4.5	Number of relevant activity/sector/technology entries in act_sec_tech table	36
4.6	Execution plan for strategy listing with left outer join	37
4.7	Execution plan for strategy listing with nested select	38
4.8	“regions_mv” materialized view	43
4.9	Structure of the “act_sec_tech_all” table in “rains_data”	45
4.10	Structure of the “act_sec_tech_to_poll” table in “rains_data”	45
4.11	Temporary tables for emission vector analysis	51
4.12	Tables for listing of emission vectors and user access to them	52
4.13	Database tables holding emission factors and removal efficiencies	57
4.14	Foreign key constraints on table “emiss_factors_abtd”	59

4.15	Tables holding applicability data	68
4.16	Tables holding definitions of SNAP and NFR codes	74
4.17	Relation between SNAP1 codes and GAINS sectors	75
4.18	Relation between NFR codes, GAINS sectors and pollutants . . .	78
4.19	Screen shot of results of CH ₄ emission calculation aggregated by NFR	81
4.20	Temporary database table for initialization of removal efficiencies	86
4.21	Overview of available emission vector data by pollutant and region	92
4.22	Overview of available vector data by pollutant	92
4.23	Database structure for assignment between scenarios and pollutants	93
4.24	Tables holding active user sessions and associated scenario infor- mation	97

Listings

4.1	PM relevant activity/sector/technology combinations	34
4.2	SO ₂ relevant activity/sector/technology combinations	34
4.3	Overview of activity/sector/technology combinations	36
4.4	Strategy listing with left outer join	37
4.5	Strategy listing with nested select	37
4.6	Create command for materialized view “region_mv”	43
4.7	Create command for mview log on “act_sec_tech_all”	46
4.8	Activity/sector/technology/pollutant combinations relevant for emis- sion factors	47
4.9	Activity/sector/technology/pollutant combinations relevant for cost factors	47
4.10	Check of activity/sector/NOC-technology combinations	48
4.11	Check of activity/sector/NSC-technology combinations	48
4.12	Tables having referential constraints to table “versions”	49
4.13	Generating commands to alter constraints on vector related tables	53
4.14	Command to retrieve unabated emission factors for NO _x	56
4.15	Command to retrieve removal efficiencies for NO _x	56
4.16	Command to retrieve abated emission factors	58
4.17	Incomplete but used emission vectors	60
4.18	Amount of abatement options per vector, pollutant and region . . .	60
4.19	Remove factor combinations of initialization mavericks	61
4.20	Percentage of missing emission factors	61
4.21	Abatement options for which emission factors are missing	62
4.22	Missing emission factors affecting calculation	63
4.23	Collecting data for needed applicability sets	66
4.24	Finding multiple values for applicabilities	67
4.25	Removing duplicate values for applicabilities	67

4.26	Analysis of sub and master applicability sets	69
4.27	Analysis of sub and master applicability sets	70
4.28	Removing redundant applicability sets	71
4.29	Collecting SNAP & NFR aggregations for NH ₃	73
4.30	Check of assignment between GAINS sectors and SNAP1 codes .	75
4.31	PL/SQL function to retrieve parent NFR code at a given level . . .	76
4.32	Check of assignment between GAINS sectors and NFR1 codes . .	77
4.33	View guaranteeing the completeness of the GAINS sector to NFR code relation	78
4.34	Emissions aggregated by NFR	79
4.35	View for emission calculation	81
4.36	Pollutant independent emission calculation	84
4.37	Retrieving all abatement options for removal efficiencies for NO _x	86
4.38	Initialization step setting initial constant removal efficiencies for NO _x	87
4.39	Checking of initialized values	87
4.40	Inserting of initialized values for missing abatement options . . .	88
4.41	Updating of necessary abatement options with initialized values .	89
4.42	Data for overview of scenario completeness	90
4.43	Analysis of overview of scenario completeness	91
4.44	User access to available scenarios	94

Chapter 1

Introduction

1.1 RAINS/GAINS Model

The **R**egional **A**ir **P**ollution **I**nformation and **S**imulation (RAINS) model was developed by the Atmospheric Pollution and Economic Development (APD) program, formerly called Transboundary Air Pollution (TAP), at the **I**nternational **I**nstitute for **A**ppplied **S**ystems **A**nalysis (IIASA) in 1983. RAINS provides a consistent framework for the analysis of emission reduction strategies, focusing on acidification, eutrophication, and tropospheric ozone. It was developed to help users understand the impacts of future actions - or inaction - and to design strategies to achieve long-term environmental goals (IIASA Options, 1998).

In 2005 the model was extended to meet the new needs of “pollution science” as well as modelling emissions by greenhouse gases. The extension of the scientific approach was also reflected in the new name of the model, namely **G**reenhouse **G**as and **A**ir **P**ollution **I**nteractions and **S**ynergies (GAINS). The first version of the model developed officially under the GAINS methodology was GAINS-Asia (European Commission - FP6, 2005).

1.1.1 Evolution of the Model from a Software Engineering Point of View

The RAINS model was originally developed as a stand-alone application. The first versions were developed in Fortran on mainframe computers (IIASA Options, 1993) followed by C (Witmuess, 1990) and C++ (Amann and Dhoondia, 1994)

as the programming language. Due to the model's popularity it soon became necessary to share the data with scientists all over the world. Because of the rising popularity of the World Wide Web it was decided to implement a web interface for the model. The nature of the C programming language necessitated a change in the underlying programming structure. The first version of the RAINS model available on the internet was developed in Perl in 1998. Although it only displayed the results calculated by the PC version on the web, it was already operating on an Oracle database. In 2001 the programming language changed again, this time to Java.

The model was initially developed for European countries. In 1994 the World Bank and IIASA started the RAINS Asia project (IIASA Options, 1993) using the application logic for Asian countries as well.

When the model became more popular, individual countries became interested in modelling their national pollution scenarios. During 2004 regional models for Italy and the Netherlands were developed. Even with the new versions, the model was still being developed on standalone schemas. This allowed easy modelling of certain regional aspects that had to be embedded into the model, but it also increased the maintenance efforts: Every change in the common structure of the model had to be introduced in each schema separately.

1.1.2 Policy Relevance of the Model

The GAINS model is popular because it can answer policy-relevant questions. For example, “How much would a migration from one technology to a more effective one cost and how much emissions would it save?”, or “What is the most effective way in terms of use of technologies to save emissions within a given budget?”.

Questions like this are answered with the help of the GAINS optimization module (Wagner et al., 2007). This paper will not only present an example of remodelling of data, in particular applicabilities, crucial to the optimization process, but also give various examples of data remodelling that ensure the correctness and reliability of input data for the optimization.

For example, the incompleteness of the emission vector, in particular of cost factors, can lead to wrong calculation results. If a particular cost factor is missing, it is assumed during the optimization process that this factor is zero. This will lead to falsely applying this “cheap” technology instead of other technologies and falsify the results of an optimization run.

The policy-relevance of the model is not only one of its biggest assets, but also one of the biggest challenges from the software and data engineering point of view. Due to the necessity for continuous development of the model, the development team is under constant pressure to provide new functionalities of the model. This paper will discuss many implementation aspects that are not considered state of the art in both software and data engineering, but in the context of constant time pressure it is understandable that they were implemented this way.

1.2 Data Cleaning and Performance Tuning

1.2.1 Data Cleaning

Due to the historical evolution of the model, the pollutant-specific development, and time pressure to keep up with unfolding policy-relevant questions, the underlying database of the GAINS model lacks a common, pollutant-independent data modelling approach. This leads to redundant storage of data, and occasionally even inconsistencies or contradictions in data.

Therefore the process of “data cleaning”, also referred to as “data cleansing” or “scrubbing” (Rahm and Do, 2000), is very important to the successful re-design of the GAINS model database structure. Data cleaning is most often used in connection with data warehouses, and there in particular with the extraction-transformation-loading process (ETL) (Rahm and Do, 2000), or the process of knowledge discovery in databases (KDD) in a wider sense (Fayyad et al., 1996).

Both scientific fields are very interesting, but there is a long way to go before the data structure of the GAINS model can be restructured into a well functioning data warehouse. Therefore we will only focus on data cleaning, and not discuss the concept of data warehouses in more detail in this paper.

The process of data cleaning itself was done mainly manually, without the help of tools. A large number of such tools is available, including ones for data integration which is regarded to be the most important aspect of a data warehouse (Calvanese et al., 1998), but the challenges of data cleaning in the GAINS model can be classified mostly on the schema level of single-source problems. Due to the multi-schema modelling approach there are, of course, some challenges on the multi-source level, but they are in the minority (Rahm and Do, 2000). Therefore the use of more sophisticated tools is not justifiable.

1.2.2 Performance Tuning

As the amount of processed data is rising constantly, efficient processing becomes even more important. Processing of SQL queries usually accounts for 60 to 90 percent of resources of an average relational database server (Irizawa and To, 2001). Basic SQL knowledge is relatively easy to gain, but highly complex and DBMS-specific optimizer algorithms, like those of an Oracle database, often tend to produce the most surprising performance-related results, especially when the queries are retrieving large amounts of data. Therefore writing functionally correct SQL maybe relatively easy, but to write performance efficient SQL is much harder (Gutjahr and Loew, 2002).

There are many other aspects of performance tuning that should not be disregarded when trying to solve performance problems of a database driven application. One of these aspects is the storage layout and I/O performance of Oracle databases (Loaiza, 2000), and data warehouses in general (Nicola and Rizvi, 2003). Also the question of the connection between the web application and the database (for example over JDBC) can have a positive effect on the performance of the whole application (Gutjahr and Loew, 2002).

Generally, over sixty percent of performance problems in database applications are due to poorly performing SQL statements, and the performance of at least thirty percent of all SQL statements can be significantly improved (Irizawa and To, 2001).

This paper will therefore mainly focus on performance tuning in terms of better SQL performance. We will show how join strategies and conceptual changes can impact the response times of the GAINS model. We will also demonstrate in various examples how an execution plan can help to show why SQL queries are performing poorly. Additionally we will discuss how proper indexing strategies can boost performance.

1.3 Structure and Goal of this Paper

Chapter 2 will give an introduction to the GAINS modelling approach, both from an environmental engineering, as well as from a database engineering point of

view. In chapter 3 we will discuss the current challenges, and chapter 4 will describe how these challenges were met by discussing the implementations. Chapter 5 will conclude this paper.

The goal of this paper is to discuss efficient data cleaning and performance tuning strategies and describe their implementation within the GAINS model. The actions described in this paper pave the way to reliable data contained in the model and fast sharing of this data. They also prepare the model for easy introduction of new scientific modelling ideas in the near future, like the introduction of new pollutants. Thanks to the restructuring, maintainability of the model is improved as well, and thus the possibility of mistakes in the future is minimized.

Furthermore these actions can be seen as a first step in the restructuring process during which the model can migrate to a data warehouse-based application that allows flexible and fast viewing of the data. This would also allow the implementation of a global GAINS model. Since air pollution does not stop at political borders, a union of all currently available regional GAINS models is desirable to allow the analysis of global emissions and their impacts.

Another aspect that might be considered in the future is the introduction of web services that could take the sharing of data to an application-to-application level. The GAINS model is currently part of a network of models. It relies on data calculated by other models, and also the results of GAINS calculations are used as input for other models. Currently this cooperation is the main subject of the European Consortium for Modelling of Air Pollution and Climate Strategies (EC4MACS) project funded by the LIFE program of the European Commission (EC4MACS, 2006). The existing interaction between the models would certainly benefit from the possibility to link the models on an application-to-application level.

Chapter 2

GAINS Basics

This chapter will give an overview of the basic data structure of both the GAINS model and the meta data needed to display the model through the web interface.

2.1 The Structure of GAINS Database Schemas

The database structure needed for storing the scientific information of the GAINS models and meta data needed to control the display of this information is stored in different Oracle database schemas.

2.1.1 Regional Data Schemas

The regional data schemas (for example “rains_europe”, “rains_asia”, etc.) hold the scientific data. Although all these schemas should be identical from the structural point of view, changes have been gradually introduced to reflect certain regional specifics.

In the previous versions of the web interface, these schemas have been also holding the meta data needed for the control of the display of scientific data (for example user information). As more and more regional versions of the GAINS model were introduced, more and more regional data schemas were needed. This led to an increase in synchronizing and updating problems because data that had to be identical, was stored in many places. This problem was addressed by creating two new schemas that would store the common information centrally. The following sections will discuss these schemas in greater detail.

Activity	Activity that can be measured
Sector	Economical sector in which activities take place
Technology	Technology that can be applied

Table 2.1: Basic GAINS Entities

2.1.2 Central Data Schema

The schema “rains_data” stores the basic information about the scientific structure of the GAINS model. This information includes the list of available GAINS activities, sectors and technologies, as well as allowed combinations of previously mentioned entities. More detailed information about these entities can be found in section 2.2.

Another very relevant and often referenced piece of information is the combination of activity/sector/technology combinations with the existing pollutants. This information has to be accessed and referenced by all regional data schemas as it is the basis for almost every calculation of the GAINS model.

Another aspect of the model that is stored in the “rains_data” schema is the transformation of allowed input data combinations to output combinations. This is necessary because the input data for the model can not be used directly for the emission and cost calculations, but has to be transformed according to given rules.

2.1.3 Central Web Schema

The schema “rains_web” holds all meta data needed for the control of the display of the scientific data stored in the regional database schemas. Examples for such data are user-specific data, the SQL commands needed to retrieve data or information about the display structure of the web interface.

Parts of the data have to be deployed back to the regional data schemas to allow the join of this data with particular entities of the scientific part. For example the sharing of scenarios is done by joining the users and their sharing relations together with the scenarios. This question will be discussed further in section 2.4.1.

2.2 Basic GAINS Entities

Table 2.1 displays basic GAINS entities. The term “activity” was introduced after the number of activity types was extended to the current set (“agriculture”, “energy”, “mobile”, “processes” and “VOC sources”). In early versions of the RAINS model there was only one activity type, namely the “energy” type. That is why the term “fuel” was used instead of “activity” (Alcamo et al., 1990).

The combinations of activities and sectors, as well as activities, sectors and technologies, form the basis of input data for the GAINS model. The allowed combinations are stored in the central “rains_data” schema. This data is then referenced from a high percentage of tables that hold the input data in the regional schemas.

In the following the term “abatement option” will be used synonymically with activity/sector/technology combination.

2.3 Structure of a Scenario

Scenarios are the heart of GAINS: The model is about developing different emission scenarios to show certain aspects and counter measures of pollution. Each scenario is thereby individually defined for the available GAINS regions. Depending on the granularity of available data, a GAINS region can be either a continent, a country or a part of a country (for example a province in China).

As you can see in figure 2.1 the definition of a scenario for each GAINS region consists of an emission vector, a control strategy and a pathway for each of the five activity types.

2.3.1 Scenario-based Emission and Cost Calculation

GAINS calculates emissions and the costs of reducing emission on a scenario basis. The details of these calculations will be discussed in this chapter.

Emission Calculation

The emissions for a given pollutant, GAINS region, and year within a given GAINS scenario are calculated according to the following equation (Klaassen et al., 2005).

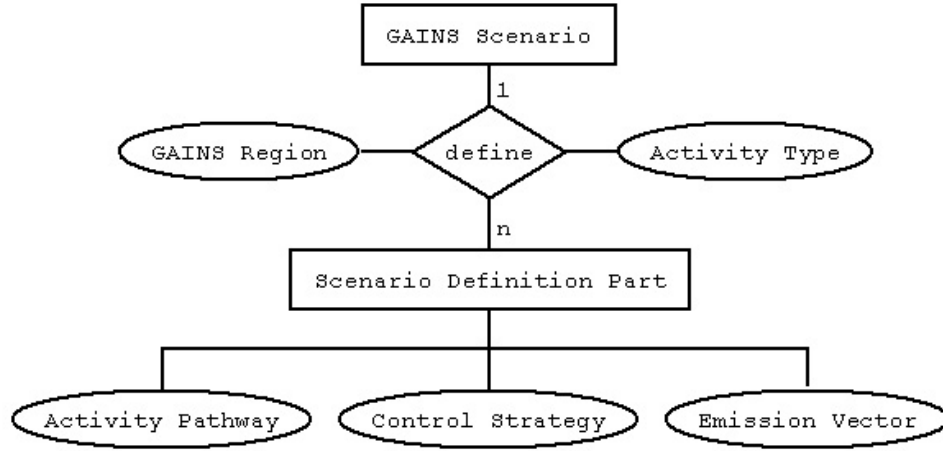


Figure 2.1: Structure of a GAINS scenario

$$E_{p,r,y} = \sum_{a,s,t} E_{p,r,a,s,t,y} = \sum_{a,s,t} A_{r,a,s,y} \cdot X_{r,a,s,t,y} / 100 \cdot ef_{p,r,a,s} \cdot (1 - \eta_{p,r,a,s,t}) \quad (2.1)$$

where

- p, r, y Pollutant, GAINS region, year,
- a, s, t GAINS activity, sector, abatement technology (option),
- $E_{p,r,y}$ Emissions of the specific pollutant p , GAINS region r , and year y ,
- $A_{r,a,s,y}$ Activity for a given GAINS activity/sector combination (a, s) ,
- $X_{r,a,s,t,y}$ Actual implementation rate ($0\% \leq X_{r,a,s,t,y} \leq 100\%$) of the considered abatement option ,
- $ef_{p,r,a,s}$ “Uncontrolled” (“unabated”) emission factor, and
- $\eta_{p,r,a,s,t}$ Reduction efficiency.

The following sections will explain how this formula is implemented in terms of database structure. All of the mentioned tables are stored in the regional database schemas.

Cost Calculation

Similar to the emissions, also the costs of reducing emissions for a given pollutant, GAINS region, and year can be calculated by the GAINS model according to the

following equation as stated in (Wagner et al., 2007):

$$C_{p,r,y} = \sum_{a,s,t} C_{p,r,a,s,t,y} = \sum_{a,s,t} \delta_{p,(a,s,t)} \cdot A_{r,a,s,y} \cdot X_{r,a,s,t,y}/100 \cdot cf_{r,a,s,t} \quad (2.2)$$

where

- $C_{p,r,y}$ Reduction costs of the specific pollutant p , GAINS region r , and year y ,
- $cf_{r,a,s,t}$ Unit cost factor of the considered abatement option, and
- $\delta_{p,(a,s,t)}$ Kronecker delta function that returns 1 if p is the primary cost pollutant for abatement option (a,s,t) and 0 otherwise.

There are two main differences between the cost calculation (equation 2.2) and the emission calculation (equation 2.1): First, unit cost factors are used instead of the emission factors. Unlike the emission factors, the cost factors are not pollutant specific, but only abatement option specific.

Since the interface of the model is designed in a way that the results of the calculations are displayed by selecting a pollutant, an assignment of costs to pollutants is necessary. Nevertheless the display of costs has to be done without double counting of multi-pollutant technologies. Therefore a hierarchy of pollutants for the purpose of cost calculation display was introduced.

This leads to the second difference in the cost calculation, namely the additional Kronecker delta function (Kronecker delta, 2007) ($\delta_{p,(a,s,t)}$) that prevents double counting of reduction costs. It returns 1 for the pollutant with the highest cost priority rating for a given activity/sector/technology combination, and 0 for all other pollutants relevant for this abatement option. This way reduction costs of multi-pollutant abatement options are only displayed once as required. This in turn means that the following equation has to be true for each abatement option:

$$\sum_p \delta_{p,(a,s,t)} = 1 \quad (2.3)$$

Information about the cost priority rating of pollutants is stored in the central “pollutant” table in the “rains_data” schema where each pollutant is assigned a unique “cost_priority” key. For example, NO_x is higher in the hierarchy than PM. Thus the costs for all technologies that control both NO_x and PM emissions are only accounted for and displayed in the NO_x costs, but not in the PM costs.

How the cost hierarchy information can be used for display and referential integrity purposes will be discussed in more detail in section 4.3.2.

2.3.2 Side Notes on Technologies

The list of available GAINS technologies contains two sets of “special” technologies with exceptional modelling rules and definitions. They are crucial to ranking all other technologies according to their cost-effectiveness, also referred to as “cost curves” (Klaassen et al., 2005), and other optimization related tasks. This section will explain these technologies in more detail.

“No Control - Technology” (NOC)

The “No Control” (NOC) technology is a special technology in that it is not a technology implementation rate, but the lack of an implementation rate. It shows which percentage of activities for a given GAINS activity/sector combination is not controlled by any technology for a given pollutant. In the following we will not distinguish the label “NOC” from its implementation rate.

By definition the removal efficiency of NOC equals zero for all activity/sector/technology combinations it is used in. This is why removal efficiencies for NOC technologies are not stored in the database. No activity/sector/NOC-technology combination is causing costs in the GAINS model.

The implementation rate of every activity/sector/NOC-technology combination is therefore not written into the database upon upload of data as it is for all other combinations, but calculated from the other activity/sector/technology combinations for a given pollutant as shown in equation 2.4.

$$NOC_p = X_{r,a,s,t=NOC_p,y} = 100\% - \sum_{t \notin \{NOC_p, NSC_p\}} X_{r,a,s,t,y} \quad (2.4)$$

where

NOC_p	NOC implementation rate for a given pollutant p ,
NSC_p	Implementation rate not suitable for control, and
$\sum_{t \notin \{NOC_p, NSC_p\}} X_{r,a,s,t,y}$	Implementation rates for all other abatement options

It can also be seen from this equation that the sum of all implementation rates of a given activity/sector combination is not allowed to be greater than 100 percent, as this would lead to a negative implementation rate for the NOC technology, which is not allowed.

“Not Suitable for Control - Technologies” (NSC)

The “Not Suitable for Control” (NSC) technologies are also not real technologies. They define which percentage of activities for a given GAINS activity/sector combination cannot be subject to any control. In other words, a given NSC technology defines the minimum NOC implementation rate for a given pollutant as shown in equation 2.5.

$$NSC_p = X_{r,a,s,t=NSC_p,y} \leq NOC_p = X_{r,a,s,t=NOC_p,y} \quad (2.5)$$

Since the consistency check of implementation rates has to be done already upon upload of data, it is necessary to reformulate equations 2.4 and 2.5 to show the dependency between the NSC implementation rates and all other rates uploaded into the system without involving the dynamically calculated NOC rate. This dependency is shown in equation 2.6.

$$\sum_{t \notin \{NOC_p, NSC_p\}} X_{r,a,s,t,y} \leq 100\% - X_{r,a,s,t=NSC_p,y} \quad (2.6)$$

Although NSC relevant combinations can be assigned to more than one pollutant, every activity/sector/NOC-technology can be constrained by at most one NSC implementation rate. It will be shown in section 4.3.2 how data consistency can be ensured so that this requirement is met.

Even though some of the NSC technologies contain the name of a pollutant (for example “NSC.PM” or “NSC.NOX”), this does not mean that they only constrain the NOC implementation rate of the given pollutant. This relation is stored in the “act_sec_tech” table which will be discussed in more detail in sections 4.1.1 and 4.3.2. “NSC.NOX” for example, sets the minimum for NOC_{NO_x} and for NOC_{NH_3} . Another example is the NSC technology “NSC.TRA” that can affect NOC_{CH_4} , as well as NOC_{NH_3} , NOC_{NO_x} , NOC_{PM} , and NOC_{VOC} .

The reason for the above mentioned multiple NSC technologies within the database is the fact that the implementation rates are not stored in a pollutant-related way. Therefore it would not be possible to assign multiple NSC values for the various pollutants if only one “NSC” technology existed.

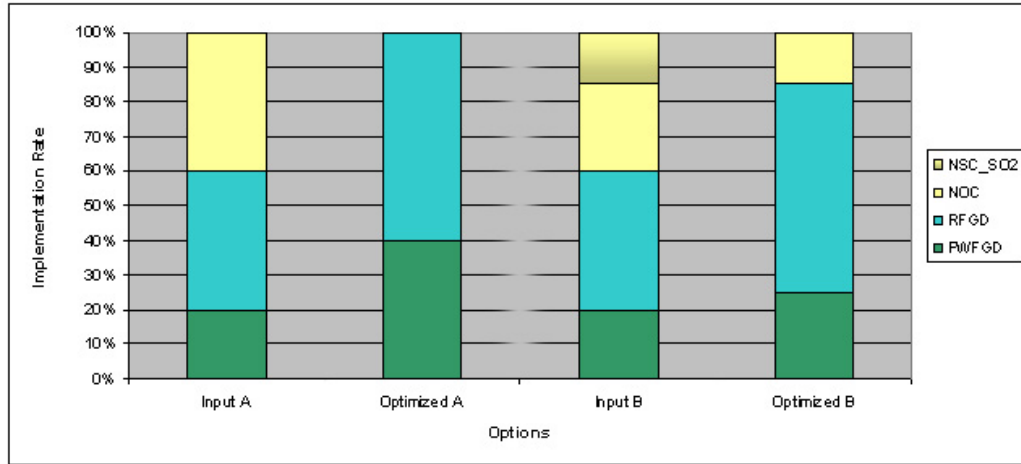


Figure 2.2: Examples of BC1/PP_NEW implementation rates for SO₂

Examples

To show how these special technologies are used, two simple examples of the GAINS optimization (Wagner et al., 2007) will be given.

First consider case A in which no NSC constraint is defined. In the baseline there are implementation rates of 40% for high efficiency flue gases desulphurisation (RFGD) and 20% for wet flue gases desulphurisation (PWFGD) for brown coal/lignite, grade 1 (BC1) use in new power heat plants (PP_NEW) as shown in the left bar in figure 2.2. According to equation 2.4, the implementation rate for NOC is therefore 40%.

Suppose now that the maximum application rate for RFGD is 60%. Then, the lowest SO₂ emissions are achieved by applying 60% RFGD and 40% PWFGD (second bar in figure 2.2).

In case B we assume the same input parameters, but the part of NOC (still at 40%) not suitable for control (NSC_SO2) is 15% (third bar). In this case the optimizer has to leave 15% uncontrolled according to equation 2.5 and decides to set the implementation rates to 60% for RFGD and 25% for PWFGD to achieve the lowest possible emissions under these constraints (fourth bar).

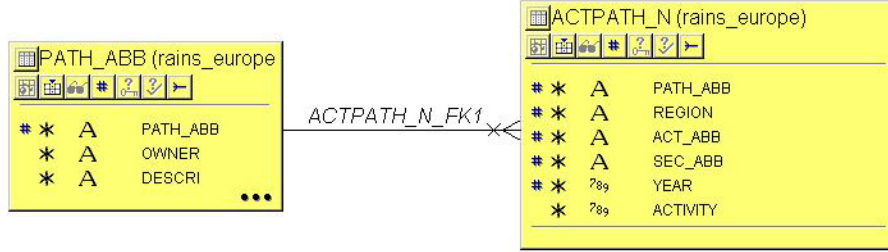


Figure 2.3: Activity pathway list and data

Conclusions

Although these “special” GAINS technologies form only a small percentage of all available technologies, they are crucial to the calculations of the model, especially for the optimization and cost curve calculation. Even though the rules and definitions presented in this section seem to be very straightforward, the implementation of some of them in terms of database constraints has proven to be difficult. Nevertheless, solutions to this challenge have to be found to assure the correctness of results calculated by the GAINS model.

2.3.3 Activity Pathway

The activities ($A_{r,a,s,y}$) mentioned in equations 2.1 and 2.2 are stored in the “act-path_n” table. The data is grouped into so-called “activity pathways” (also referred to as “pathways”) stored in the table “path_abb”.

As you can see from the database tables shown in figure 2.3 each of the activity entries is GAINS region, activity/sector combination, and year dependent. It is important to understand, that the model stores only combinations for activities larger than zero. This means that if a combination does not exist in a pathway, it is assumed to be zero. Because of this modelling constraint, a functionality to check the completeness of a pathway in terms of data amount cannot be implemented.

There are also other pathway related data that are stored in other tables, for example applicabilities which will be discussed in section 3.2.2. Other examples of pathway related data are beyond the scope of this paper.

2.3.4 Control Strategy

The second part of equations 2.1 and 2.2, namely the implementation rates of the considered abatement options ($X_{r,a,s,t,y}$) are grouped into so-called “control strategies”. As you can see from figure 2.4 the percentage of the implementation rate is stored per activity/sector/technology combination and year, but not for a given region. Although according to the equation, the implementation rate has to be GAINS region dependent. The control strategies are modeled in this way is because they can be assigned to different regions.

As you can see from figure 2.4 each control strategy can (and should) have an assigned primary region (in table “cons_n”). Nevertheless each control strategy can be assigned during the scenario definition to each region. This is also how the implementation rates are used in a region specific way, namely for the region for which the control strategy was assigned in a given scenario.

This approach has an important advantage when developing scenarios for new regions: During the initial phase of development, each GAINS region can have the same generic control strategy. After the initial definition phase of a scenario is finished, it is presented to national experts, who then take the generic strategy as the basis for regional strategies. These specific strategies are then introduced step by step for all GAINS regions of the scenario.

Another area of application for generic strategies is the modelling of groups of GAINS regions that have the same legislation. Larger countries, like China or Russia for example, are divided into multiple GAINS regions. All of these regions have the same legislation and since most of the application rates are based on current legislation, it is very common to use the same control strategy. Another development that can be observed, is the fact that within the European Union the legislation and thus the control strategies are becoming more and more similar. Some special scenarios are calculated with the same control strategy for all countries of the European Union.

Another type of scenarios that uses common control strategies for multiple GAINS regions are so-called “no control”, “maximum feasible”, or even global scenarios.

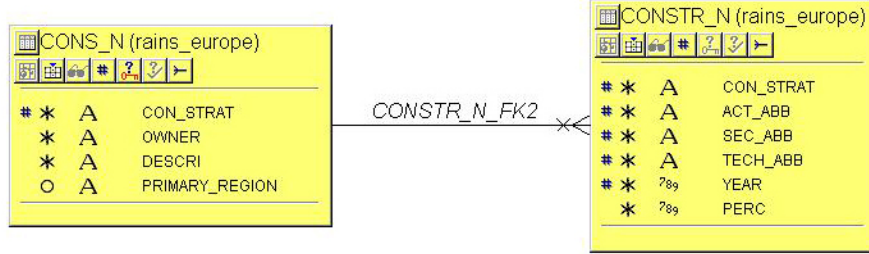


Figure 2.4: Control strategy list and data

2.3.5 Emission Vector

The calculations of emissions (equation 2.1) and costs (equation 2.2) can be seen as vector calculations. This is why the $ef_{p,r,a,s} \cdot (1 - \eta_{p,r,a,s,t})$ and $cf_{r,a,s,t}$ parts of the equations are referred to as the “emission vector”.

The storage of emission vector related data is more complex. This data is initialized through a routine that has to be triggered manually. The routine is pollutant specific. It takes data from different tables, calculates the emission vector related data and writes the results of the initialization process into specified tables.

Another difference in the emission vector, is the fact that all combinations (also zero values) are stored. Therefore, although the underlying database structure and the application logic of an emission vector is more complex, the completeness in terms of data amount can be checked.

Furthermore there is no single table listing all available emission vectors. The reason behind this might be the fact that there are two different types of emission vectors. First there are the so-called “version emission vectors” (the challenge of modelling “versions” will be discussed later in section 3.2.1). These vectors are developed by experts at IIASA and are guaranteed to be complete in terms of data. They are usually named after a certain report for which they were developed (for example “NECO5”) or a date (for example “Aug06”).

The second type are “user specific emission vectors”. These vectors can be created through the web interface by any user with appropriate access rights. Due to the complexity and amount of data related to an emission vector, every user is allowed to have only one emission vector. Since it cannot be expected that an average user has enough knowledge to collect all the data needed for the emission

vector to be complete, every user specific vector has to be “backed up” by a version emission vector when running the initialization. This way data can be taken from the complete version emission vector if it cannot be provided by the user. In the current version of the application this approach is not implemented, although it is an important requirement without which it is nearly impossible for a user to create their own emission vector.

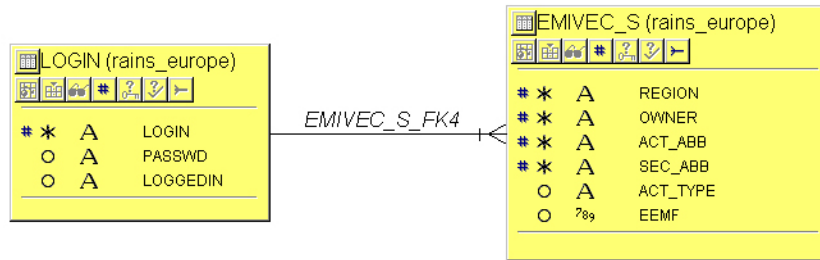
The duality in the modelling approach combined with the lack of appropriate modelling led to the previously mentioned fact that there is no table that would list all available emission vectors. Instead the “owner” column which is meant to reference the identifier of the emission vector, is in fact referencing the list of all available users stored in the “login” table. Examples of such references will be given in the following sections.

Further details about the challenge of modelling versions of emission vectors will be explored in section 3.2.1. Possible solutions for this problem will be discussed in section 4.3.3.

Emission Factors

The emission factors ($ef_{p,r,a,s}$ in equation 2.1) are - as mentioned in the previous section - calculated during the initialization process. In the case of SO_2 the data is written into a table called “emivec_s”. Because of the lack of a central list of available emission vectors, the “owner” column of the table “emivec_s” references the “login” table that holds all possible users of the application version (see figure 2.5). Thus the values in the column are referencing not only the users for whom an emission vector exists, but all users. The problem of referencing a user that does not have an emission vector is prevented by the application logic, but cannot be prevented by referential constraints within the database.

As can be seen from figure 2.5, the activity type is also included in the “emivec_s” table. Since the table is filled with data during the initialization process, the relation between the sector and the activity type is retrieved correctly from the “rains_data” schema. The redundancy in data is taken into account to save query runtime when information is needed during further steps of the emission calculation based on this table. The problem with this approach is that if the activity type would be changed, the initializations for all emission vectors would have to be run again, or the type would have to be adjusted manually in all tables as there is no single table which includes data of the emission vector for all

Figure 2.5: Emission factor data for SO₂

pollutants.

It is important to see that there is no referential integrity between the implicitly correlated columns “sec_abb” and “act_type” in the table. What cannot be seen from the structure of the table (and also not from the name of the table) is the fact that this table should only store activity/sector combinations relevant for SO₂. As explained later in section 4.3.2, the activity/sector combinations relevant for a pollutant, can change. Nevertheless such changes would not be reflected through referential constraints set between the tables, but would have to be enforced manually through the initialization process. Solutions to this question will be discussed in more detail in section 4.3.3.

Another fact that easily is observed from figure 2.5 is that the columns “act_type” and “eemf” can potentially be *NULL*. Again this is impossible because of the mentioned application logic, however the columns should be changed to be *NOT NULL* to prevent any possible problems already at the database level.

Reduction (Removal) Efficiency

Reduction efficiencies (also called removal efficiencies) are also part of the emission vector. The data is retrieved during the initialization process and stored in different tables for each pollutant. In the case of SO₂ the table is called “emivec_sr”. As you can see from figure 2.6, the structure of the table is very similar to the one used to store emission factors. The only difference is that - as already stated in equation 2.1 - the reduction efficiency is also technology dependent ($\eta_{p,r,a,s,t}$).

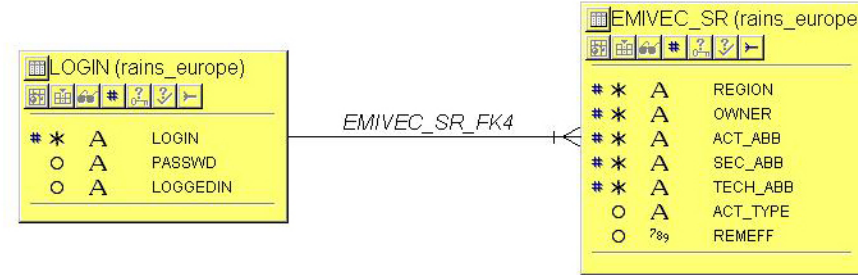
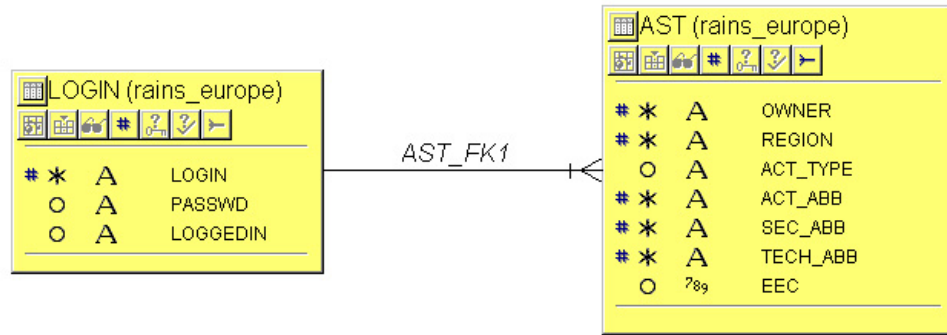
Figure 2.6: Reduction efficiency data for SO₂

Figure 2.7: Cost factors for all pollutants

Cost Factors

As noted in equation 2.2, cost factors ($cf_{r,a,s,t}$), unlike emission factors and removal efficiencies, are not pollutant specific. Furthermore, the structure of the “ast” table holding the cost factors for all pollutants is very similar to the tables holding the removal efficiencies as can be seen from figure 2.7. Also the considerations regarding *NULL* columns and the redundant “act_type” column are the same.

Conclusions

Although all database tables mentioned in this section are filled with data during the initialization process, the questions of referential integrity as well as complete-

ness of data are even more important than in other areas of the model.

Due to the complex process of gathering underlying data, most users rely on the emission vectors created and published through the web interface by the scientists working at IIASA. An average user only changes the pathway and control strategy data.

Because of this, the assumption that each emission vector is complete in terms of needed abatement options, seems to be even more important. If for example an emission factor is missing for a given abatement option, no emissions will be calculated even though corresponding activities and implementation rates were given by the user. While an expert might notice such “gaps” in the resulting emissions, an average user most likely will not.

Even though most of such inconsistencies have been discovered in the past, consistency of such large amounts of data should be ensured by the database where possible and the integrity of the model should not rely on manual checks.

2.4 Sharing of Scenarios and Their Parts

The scenarios and the mentioned region-specific parts of the scenarios (activity pathway, emission vector and control strategy) have to be shared in a controlled way. Therefore sharing rules that determine read and write access have to be defined and implemented.

2.4.1 Sharing of Scenarios

The access of a given user to a scenario is mainly determined by the owner of the scenario: If a given user shares read or write access with the owner of the scenario, then the given user has read or write access to the given scenario. How user sharing works, will be explained in section 2.5.3.

In addition there are also two other possibilities of how a user can be granted read access to a scenario. The first possibility is that the scenario is made public by the owner or an administrator. This way all users of the model can view the scenario. Usually a scenario is made public when its development is finished and every user should be have access to the results.

The other possibility is to give the user explicit access to a given scenario. This is mainly used during the development of a scenario when feedback from a

given group of experts is required. This permission only enables the user to see one particular scenario, but has no impact on the access to other scenarios of the owner of this scenario.

2.4.2 Sharing of Scenario Parts

As each scenario has an owner, the parts of the scenario (activity pathway, emission vector and control strategy) also have an owner. According to the rule of granting access to a scenario if sharing access with the owner, access to the parts of a scenario is determined by the ownership of the parts and sharing relations as well.

In addition, read access to parts is also granted if a given user has read access to any scenario in which this part is being used. The reason for this rule is the fact that a user has to be granted access not only to the results of a scenario, but also to the underlying data on which the calculation of the results is based.

2.5 User Management

Information about users of the model is stored centrally in the “rains.web” schema since a single user may access multiple versions of the model, but always with the same username and password.

2.5.1 User Groups

Groups of users have two functionalities: Users can be granted privileges (as described in section 2.5.2) and share data with other users (as described in section 2.5.3) through membership in groups.

There are two parameters that define how users share access to entities owned by them within the group: Read and write access. Both flags can be set independently for each group. This way users can either share read, or write access, or both. It is also possible that they do not share access to entities at all and the group is only used to grant privileges to all members of the group.

2.5.2 User Rights & Privileges

User rights regulate the access of users to the different functionalities of the GAINS model. These rights can be granted to a user personally, or to a group of users. Privileges assigned to a group are granted to all members of the given group.

The defined privileges are not hierarchical, in that the granting of one privilege does not cause other privileges to be granted automatically, although the definition of the privileges could indicate hierarchical dependency. For example the granting of the “user” privilege does not cause the automatic granting of the “viewer” privilege.

Furthermore there are two different types of GAINS versions: Public and private. The “viewer” privilege which is hierarchically seen as the lowest privilege, is granted to all users accessing the public versions automatically, whereas for private versions every privilege has to be granted manually.

2.5.3 User Sharing

Users not only want to access different functionalities of the model, they also want to have access to entities of the model created and owned by other users. Most of the entities have an owner. Sharing of entities is mainly done through the assignment of users to user groups.

Some entities, like for example the scenarios already discussed in section 2.4.1, can also be made public and thus accessible to all users. Another possibility also discussed in relation with scenarios, is granting direct access to entities.

The sum of all these rules is essential to the interface since it is the underlying basis for the display of almost every page of the model. The rules also become more and more complex. This allows for better control of user access to single entities of the model. Each additional rule is a potential loss in response time since even more data has to be parsed before retrieving a result.

A successful modelling strategy has to provide a way to combine all set requirements, while making the model respond in a time that is acceptable for the users.

Chapter 3

Current Challenges

As already mentioned in the previous chapter, the GAINS model is under constant development from a methodological point of view, as well as from the web interface's point of view. As the model develops, new ideas emerge and are implemented. These make the model and its display more precise, but each new feature has the potential to make the display of the model slower. The bottom line of many of the challenges is finding a way to have a more detailed level of modelling combined with a high speed response.

The other group of challenges include data and modelling integrity challenges which were for example already mentioned in the context of emission vectors. It is essential to be able to keep referential consistency within the model. Especially because the model handles large amounts of data, and the consistency cannot be monitored manually.

3.1 Deployment of Data

As already mentioned in section 2.5, all user related information is stored in the central schema "rains_web". This data has to be passed to the regional schemas to be able to join it with the scientific entities in terms of ownership among others. The same considerations apply to data stored in "rains_data" that defines the basic GAINS entities available and the relations between them.

In some cases simple views are enough. Especially if the underlying statements can be executed fast and the results of the views are not used by other views. In other cases simple views are not enough in terms of good performance. The

two major reasons why views are not the best structure to use, will be discussed in the following sections.

3.1.1 Data Integrity

Data integrity plays an important role, especially in large scale databases. It can be assured by setting referential constraints (foreign keys) between single tables. Such constraints cannot be set on views. Because of this, data calculated by views cannot be directly referenced in terms of referential constraints by other tables or views.

In case of the deployment of user-specific data this possibility is a large disadvantage because the user-specific data are further joined with the data of regional schemas. In some cases foreign keys can be set directly to the central “rains_web” schema. But this is only possible if the data is used in an one-to-one relation. As soon as the data is filtered in some way (for example only a subset of users has to be referenced) a foreign key cannot be set directly.

3.1.2 Long Query Execution Time

Another reason why views are not the optimal solution in many cases is the fact that essentially the underlying query of a view has to be executed every time. Oracle can optimize the execution of the query and is able to cache some results, but it does not store the results of the query completely.

Especially if the base query has a long runtime, and even more so if the underlying data changes relatively seldom, views can cause performance losses. Instead of working with precalculated data, the view has to retrieve the data every time upon request.

3.1.3 Runtime Optimization

A related challenge is the question of runtime optimization, not only for the runtime of views, but also of queries retrieving data based on views. In many cases properly set indices can help to optimize the runtime of queries by improving the time needed for joins between the tables. Unfortunately this strategy cannot be used when working with views because indices cannot be created on views. Views

use the indices of base tables when they are executed, but the results retrieved by a view cannot be indexed additionally.

Materialized views may be a solution as they combine the advantages of a table and a view. They will be discussed in more detail in section 4.2.1.

3.2 Consistency in Data Modelling

As already mentioned in previous chapters, the model continued and will continue to be developed for additional pollutants. The pollutants were not only introduced by different people, they were also introduced at different evolution stages of the model. The result of this is that some aspects of the model are inconsistent throughout the pollutants, sometimes even leading to contradictions in data.

3.2.1 Data Versions versus Version Owner

Section 2.3.5 already gave a short introduction of the definition of an emission vector. Unfortunately the already mentioned challenges are only one part of the data modelling issues to be resolved. This section will elaborate more on the fact, how the inconsistencies in modelling that were already mentioned arose.

The emission vector is a very large collection of data. In the very beginning of the development of the RAINS model, it was meant to be a collection of data that could be regarded as constant. But during the evolution of the model the emission vector data began to change. First it was changed when a larger update of data had to be done due to new scientific knowledge. The name of the vector was the date of introduction of the new version (for example “Apr04”, “Nov04”, etc.). The current status is that the emission vector for Europe for example is changed with every new generation of reports that have to be done for the European Commission (for example “NEC01”, “NEC02”, etc.).

Since the emission vector is not only a very large set of data, but also needs to hold data that is collected from various scientific fields, the emission vector has to be generated and maintained by more than one person. This multi-user aspect created a challenge of how to grant access to multiple users who need to work on the same data.

The solution was to create user accounts with exactly the same name as the emission vector. Through this approach everybody who was working on a given

emission vector, could log into the system as “a version of an emission vector” and be able to modify the data of the vector. This approach was soon extended to the ownership of scenarios and their parts by creating these entities under the “ownership of a version”.

Although this worked well for a time, it had one major drawback: Control over which person was editing which data could not be easily maintained anymore. Since everybody who had access to the version user accounts could access the model with the same log in, keeping track of who was editing which data could only be monitored by logging the IP addresses of the users.

Additionally a legal issue came up with the introduction of the new GAINS models: The acceptance of the disclaimer that was necessary in earlier versions of the models at every log in, was changed into a one-time step in the registration process. This way user accounts that were created by one person, but were used by many, who possibly never accepted the GAINS disclaimer, became a potential legal problem.

3.2.2 Applicabilities

The maximum of an application rate is referred to as the “applicability” ($X_{r,a,s,t,y}^{max}$) of the abatement option (Wagner et al., 2007). This requirement can also be written as shown in equation 3.1. The reason why this information is necessary, is the fact that some technologies cannot be used beyond a certain limit due to constraints in the “real world”. For example, a certain type of car or power plant may not have enough space to house a given technology. If an applicability does not exist for a given region, activity/sector/technology combination, and year, it is assumed to be 100%.

$$X_{r,a,s,t,y} \leq X_{r,a,s,t,y}^{max} \quad (3.1)$$

Based on 3.1 it can be deducted that the applicability of a given abatement option should have the same attributes as the implementation rate already mentioned in section 2.3.4. This means that applicabilities are not defined in a pollutant specific way, but for a given GAINS region, abatement option, and year.

In fact, the applicabilities are currently stored in several tables. Most of these tables are pollutant specific (by name), but there are also applicabilities stored across multiple pollutants. Some of the tables are activity pathway specific (not technology specific), others are emission vector specific.

The complete set of applicabilities for a given emission scenario is determined by the structure of the scenario, as explained in section 2.3. This means that for a given region the applicabilities are retrieved by pollutant from the pathway or the emission vector depending on the definition of the scenario for this region. As one can imagine, a union of all available applicabilities that is retrieved from many tables can lead to redundancy, and potentially even to inconsistencies.

Furthermore, as seen in equation 3.1, the consistency between the implementation rate defined in the control strategy, and the maximum of this rate set by the corresponding applicability has to be guaranteed. This distributed definition and storage of data that forms the previously defined constraint does not allow for any checks with simple database constraints. Especially because the possible inconsistencies do not only arise with each change in the region specific scenario definition, but also with every emission vector, pathway and control strategy change.

What makes the analysis easier in terms of amount of data, is the fact that applicabilities are only needed for scenarios for which cost curves have to be displayed through the web interface, or for scenarios that are taken as input for optimization runs.

3.2.3 Aggregation of Results According to Reporting Standards

Since the results of the RAINS (and also GAINS) models are used by the European Commission, the format in which data is presented has to follow given guidelines. An example of such reporting directives are the “Guidelines for Estimating and Reporting Emission Data under the Convention of Long-range Transboundary Air Pollution” published by the Economic Commission for Europe (Economic Commission for Europe, 2003).

According to these guidelines, the results for both emission and cost calculations have to be grouped into given categories. Currently GAINS results are reported by two standards: SNAP and NFR. The relation between the GAINS calculation results and the categories of the reporting standards is done based on activity/sector combinations. In addition these relations are stored in a separate table for each pollutant.

Initially reporting along with the mentioned standards was only requested for the results of European models. Therefore the fact that these tables were stored in the “rains_europe” schema was not a problem. However during the development of GAINS Asia it turned out that it would also be necessary to report the results

of the Asian models according to these standards and the needed tables were also copied to other schemas.

Another aspect of this modelling approach that possibly leads to the incorrect display of calculation results, is the incompleteness of assignments: If a given activity/sector combination is missing in one of the pollutant specific tables, but emissions or costs are calculated by the model for this combination and the given pollutant, the corresponding emissions or costs are not shown in the result tables. Such inconsistencies can most often be discovered when comparing the reported sums of emissions and costs with and without the aggregation by the reporting categories. A successful new modelling approach is needed to solve this problem.

How the described challenges can be solved, will be discussed in section 4.3.5.

Chapter 4

Implementation

4.1 Key Issues to Solving Query Runtime Problems

This section will discuss several key issues to solving query runtime problems in Oracle. It will give answers to questions such as “Why is the database responding so slow?” or “How can I make it run faster?”.

4.1.1 Indexing of Data

Proper indexing of data is key in preventing query runtime problems. Especially when dealing with large amounts of data, the lack of proper indexing will lead to long query execution times. Of course even when data is properly indexed, there are limits to what can be achieved in terms of runtime improvement, but proper indexing can have a significant impact on query runtime.

Indexing of Foreign Keys

In contrast to MySQL (MySQL, 2007) Oracle does not automatically create indices on columns referencing data through a foreign key. This is an important detail, since database developers used to MySQL or migrating databases from MySQL to Oracle might forget to create such indices. This in turn might lead to large performance losses. Indices on foreign key columns should always be defined unless one of the three following conditions is met (Kyte, 2005):

1. You delete rows from the parent table.

2. You update the constraint in the parent table to which the foreign key is set.
3. There are no queries that join the child table to any other table (especially the parent table) using the foreign key columns.

In most cases one of the above mentioned statements is met when you create a foreign key. The last rule should also be applied to columns where no foreign key exists: If you have an important execution path using a given column or combination of columns, you should most likely set proper indices to speed up access to data stored in these crucial columns.

Monitoring Query Execution and Index Use

Most database querying tools, like TOra (TOra, 2007) or TOAD (TOAD, 2007), offer information about the query execution plan (sometimes also called “explain plan”). Based on this underlying information you can track down access to all tables, including tables accessed indirectly through views. You can also see, whether tables are fully scanned, or accessed via an index. This helps to locate possible performance losses and add additional indices where suitable.

Examples of possible execution plans, will be given in the next section.

Why an Index Might Not Be Used

The possibility of successfully using an index is dependent on the following factors (Kyte, 2005):

1. The columns on which the index is set (also their sequence)
2. The columns which are selected by the query (also whether they are allowed to be *NULL* or not)
3. The way in which data is filtered or joined (through a function for example)

Even if indices are set and the query is written properly, the index might not be used because Oracle’s Cost Based Optimizer, the so called CBO (Oracle CBO, 2002), might “think” that using the index will be slower. In some cases the CBO can be wrong because its decision is based on outdated statistics. Such a case might for example arise if the amount of data in a table multiplies by a large factor within a short time. The optimizer decides based on a small amount of data

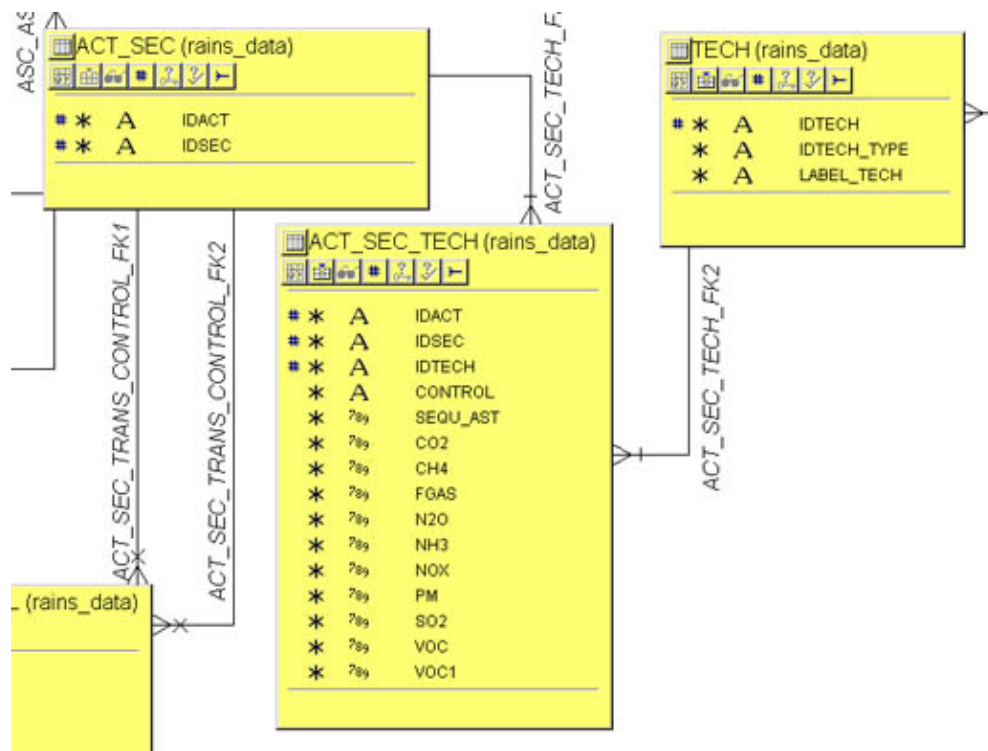
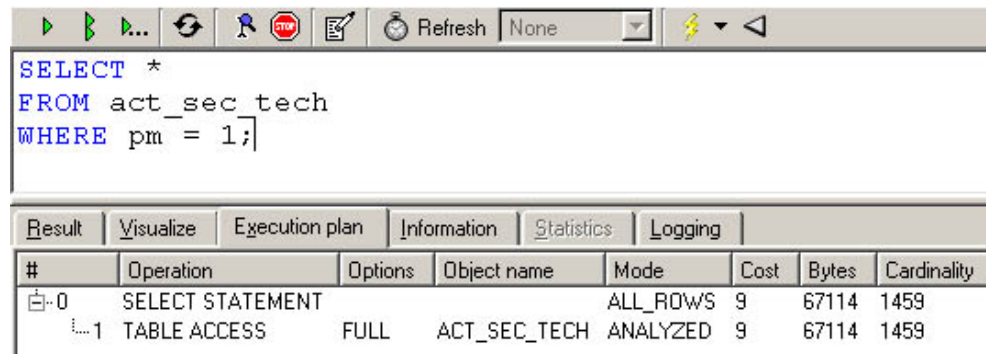


Figure 4.1: Structure of the “act_sec_tech” table in “rains_data”

and concludes that a full table scan is faster, whereas in fact using the index would be faster with the current amount of data. To solve this problem you have to gather (refresh) the table statistics so that the CBO can make its assumptions based on the current set of data. However it has to be said, that the CBO is very reliable if it is basing the decisions on statistics that are up to date.

In some cases when you have gathered the table statistics, set the index and written your query correctly, the index is still not used. The database table “act_sec_tech” (shown in figure 4.1) located in the “rains_data” schema is used as an example of this. It is one of the tables most often accessed in the GAINS model. It defines which of the known GAINS activity/sector/technology combinations is used for which pollutant.

As you see from the figure, the table joins available GAINS activity/sector combinations with available technologies. For each combination, a flag is set for each available pollutant in the respective column indicating whether this combi-



```

SELECT *
FROM act_sec_tech
WHERE pm = 1;

```

#	Operation	Options	Object name	Mode	Cost	Bytes	Cardinality
0	SELECT STATEMENT			ALL_ROWS	9	67114	1459
1	TABLE ACCESS	FULL	ACT_SEC_TECH	ANALYZED	9	67114	1459

Figure 4.2: Execution plan for retrieving activity/sector/technology combinations relevant for PM

nation is relevant for this pollutant (“0” means “no”, “1” means “yes”). What kind of impact this structure of the table has on the calculation and how the structure should be changed, will be discussed in section 4.3.2. In this section, the table will only be used as an example of index use.

Listings 4.1 and 4.2 look very much alike. In both cases we select all activity/sector/technology combinations that are relevant for the given pollutant. Both columns of the table (“pm” and “so2”) have the same kind of index. But if we look at the execution plans of the queries (figures 4.2 and 4.3) we see that in the case of SO₂ the index is used to retrieve data, and in the case of PM it is not.

Listing 4.1: PM relevant activity/sector/technology combinations

```

1 SELECT *
2 FROM act_sec_tech
3 WHERE pm = 1;

```

Listing 4.2: SO₂ relevant activity/sector/technology combinations

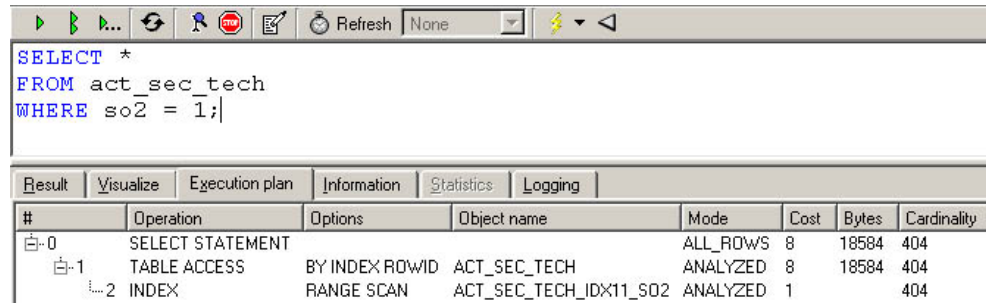
```

1 SELECT *
2 FROM act_sec_tech
3 WHERE so2 = 1;

```

Moreover, if you would like to retrieve all activity/sector/technology combinations that are not relevant for SO₂ (figure 4.4), you will see that in this case once again Oracle does not use the index created on the “so2” column.

The solution to this question lies partly in the structure of the table (the pollutants are arranged in columns) and partly in the amount of activity/sector/technology combinations defined per pollutant. The query showed in listing 4.3 gives an



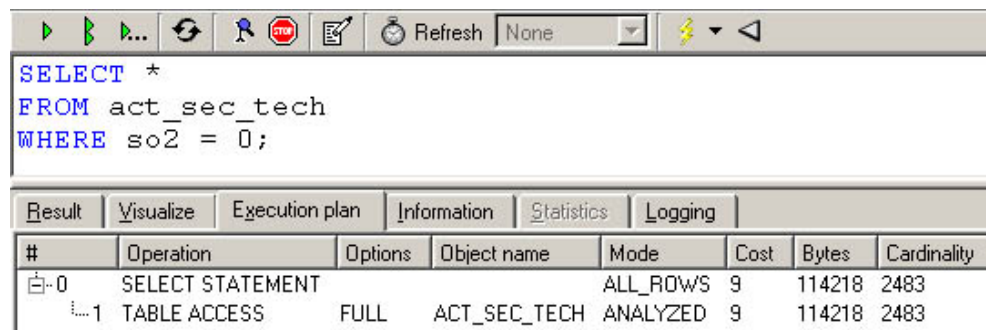
```

SELECT *
FROM act_sec_tech
WHERE so2 = 1;

```

#	Operation	Options	Object name	Mode	Cost	Bytes	Cardinality
0	SELECT STATEMENT			ALL_ROWS	8	18584	404
1	TABLE ACCESS	BY INDEX ROWID	ACT_SEC_TECH	ANALYZED	8	18584	404
2	INDEX	RANGE SCAN	ACT_SEC_TECH_IDX11_SO2	ANALYZED	1		404

Figure 4.3: Execution plan for retrieving activity/sector/technology combinations relevant for SO₂



```

SELECT *
FROM act_sec_tech
WHERE so2 = 0;

```

#	Operation	Options	Object name	Mode	Cost	Bytes	Cardinality
0	SELECT STATEMENT			ALL_ROWS	9	114218	2483
1	TABLE ACCESS	FULL	ACT_SEC_TECH	ANALYZED	9	114218	2483

Figure 4.4: Execution plan for retrieving activity/sector/technology combinations not relevant for SO₂

Result	Visualize	Execution plan	Information	Statistics	Logging
#	COMBINATIONS	PM_RELEVANT	SO2_RELEVANT	NOT_SO2_RELEVANT	
1	2934	1459 (49.73%)	404 (13.77%)	2530 (86.23%)	

Figure 4.5: Number of relevant activity/sector/technology entries in act_sec_tech table

overview of how many of such combinations are defined in total, how many are relevant for PM, how many for SO₂ and how many are not relevant for SO₂.

Listing 4.3: Overview of activity/sector/technology combinations

```

1 SELECT COUNT(*) AS combinations ,
2    SUM(pm) || ' (' || ROUND(SUM(pm)/COUNT(*)*100, 2) || '%' AS pm_relevant ,
3    SUM(so2) || ' (' || ROUND(SUM(so2)/COUNT(*)*100, 2) || '%' AS so2_relevant ,
4    COUNT(*)-SUM(so2) || ' (' || ROUND((COUNT(*)-SUM(so2))/COUNT(*)*100, 2) || '%' AS
   AS not_so2_relevant
5 FROM act_sec_tech ;

```

As you can see from figure 4.5 there are far more PM relevant combinations than SO₂. This is why the CBO does not use the index created on the column for the pollutant when retrieving data for PM, but uses it when retrieving data for SO₂. According to (Kyte, 2005) there is no exact threshold in terms of the percentage of selected rows below which an index is used and above which the index is not used, but a full scan of a table is performed. Nevertheless the rule of thumb is that this threshold lies somewhere between 10 and 20 percent of data. If you try to access larger amounts of data through an index, this will result in higher runtimes. This is why the CBO decides not to use the index on the “so2” column when we reverse the selection for SO₂.

4.1.2 Join Strategies

The way in which SQL commands are formed also has a significant impact on query runtime. Although, there are no golden rules, there are some guidelines that can be followed when writing queries. Also, as already mentioned in section 4.1.1, the execution plan can help understanding why a query might be running slowly.

Another short example will demonstrate how the performance of a query can be improved dramatically by a change in the join strategy between two tables. In

Result	Visualize	Execution plan	Information	Statistics	Logging			
#		Operation	Options	Object name	Mode	Cost	Bytes	Cardinality
0		SELECT STATEMENT			ALL_ROWS	95820	191180154	2896669
1		SORT	ORDER BY			95820	191180154	2896669
2		HASH	GROUP BY			95820	191180154	2896669
3		HASH JOIN	OUTER			2639	191180154	2896669
4		TABLE ACCESS	FULL	CONS_N	ANALYZED	7	78450	1569
5		INDEX	FAST FULL SCAN	CONSTR_N_FIDX2	ANALYZED	2532	46346704	2896669

Query executed (Duration 0:05.90)

Figure 4.6: Execution plan for strategy listing with left outer join

this example we are seeking a listing of all available GAINS control strategies with additional information on how many definitions each of the control strategies has. The “constr_n” table holds about 3 million records for about 1500 control strategies. Since we also want to have control strategies listed that have no definitions, we have to make a *LEFT* join between the tables:

Listing 4.4: Strategy listing with left outer join

```

1 SELECT c.con_strat, c.owner, c.primary_region, c.descri,
2       COUNT(d.con_strat) AS def_count
3 FROM   cons_n c
4       LEFT JOIN constr_n d ON (
5         c.con_strat = d.con_strat)
6 GROUP BY c.con_strat, c.owner, c.primary_region, c.descri
7 ORDER BY LOWER(c.con_strat);

```

The query executes in about 6.0 seconds. Even when looking at the execution plan (figure 4.6), there is no join visible that could be optimized. There is a full table scan on “cons_n”, but since we need all rows from this table, this is normal. The large table “constr_n” is “fast full” scanned on the foreign key index.

Still, the cost of grouping and sorting of the listing seems to be very high. Of course the grouping cannot be left out in this case. Unless you were able to leave the left join out (which is always a cost-expensive action). A solution to this would be a nested select to retrieve the amount of definitions (*NULL* values have to be set to “0” with the NVL function):

Listing 4.5: Strategy listing with nested select

```

1 SELECT c.con_strat, c.owner, c.primary_region, c.descri,
2       NVL((SELECT COUNT(*)
3            FROM rains_europe.constr_n d
4            WHERE d.con_strat = c.con_strat
5            GROUP BY d.con_strat), 0) AS def_count
6 FROM   rains_europe.cons_n c

```

Result Visualize Execution plan Information Statistics Logging								
#	Operation	Options	Object name	Mode	Cost	Bytes	Cardinality	
0	SELECT STATEMENT			ALL_ROWS	8	78450	1569	
1	Sort	GROUP BY NOSORT			15	16	1	
2	Index	RANGE SCAN	CONSTR_N_FIDX2	ANALYZED	15	53616	3351	
3	Sort	ORDER BY			8	78450	1569	
4	Table Access	FULL	CONS_N	ANALYZED	7	78450	1569	

Query executed (Duration 0:02.51)

Figure 4.7: Execution plan for strategy listing with nested select

```
7 | ORDER BY LOWER(c.con_strat);
```

The query with the nested select executes in about 2.5 seconds (in around 42% of the runtime of the left join query). Looking at the execution plan of the query (figure 4.7), it is visible that the runtime expensive left join could be left out and the grouping is now also unnecessary. Although the tables are still accessed over the same indices, there is far less data in terms of bytes that has to be joined. Also the count of data in the large “constr_n” table can be done by simply accessing the index.

As already stated, there is no golden rule on how to write joins. Sometimes you will find that a query unexpectedly works faster. The execution plan can give hints as to what is causing the runtime trouble, but it does not directly point to the actual problem. Also query performance can change with rising amount or changing of data dramatically at some point. Sometimes this might be caused by outdated statistics that have to be just refreshed, and sometimes you will have to rewrite the query completely.

4.1.3 Other Pitfalls

There are a few additional pitfalls that should be avoided. For example one should not use *ORDER BY* clauses within create commands of views. Of course the view will always be ordered in the proper way when looked at through a database tool, but it will also always be ordered when accessed in joins of queries. In most cases the results of such a query will not be ordered in the same way as the view was ordered. Therefore runtime is wasted on the ordering of precalculations. In fact this ordering can cause an even significant time loss, especially when the amount of data is large.

Another pitfall that should be avoided, is using *UNION* commands between parts of SQL commands that are unique by definition. Instead the *UNION ALL* command should be used to avoid unnecessary sorting of the values so that Oracle can make a complete union of all records of parts of the command. Especially for large amounts of rows this sorting can be very runtime consuming although it would not be necessary at all if the same result row cannot be found in more than one part of the *UNION* command by definition.

4.2 Other Oracle Based Solutions to Query Runtime Problem

4.2.1 Materialized Views

Deploying data with the help of materialized views is a common strategy. Most examples in common literature refer to deploying data over database links between remote databases, but deployment over different schemas within the same database is of course also possible. The following sections will give an overview of the basic idea of materialized views in Oracle. Examples of how materialized views can be used to cope with the previously mentioned challenges will be shown in section 4.3.1 and following.

For the sake of simplicity, the term “mview” will be used interchangeably with materialized view.

Introduction to Materialized Views

First of all it is important to understand that mviews are actually not views, but a special type of database table. Data is not fetched upon every select request, but is stored in the mview until the next refresh is triggered. Therefore an mview can also be seen as a cache from which a copy of data can be accessed quickly (Gupta and Mumick, 1995).

Mviews are especially suitable for precalculating data in cases when data in the base tables changes rarely compared to the read requests or if the execution of the underlying query takes a comparably long time.

Another feature that mviews have in common with tables, is the fact that it is possible to create indices on them. This is especially convenient if the data stored

in the mview is further joined with other tables to retrieve data. Examples for this feature will be given in section 4.3.1. According to indices, foreign keys can also be created that point to an mview. This way data stored in mviews can be referenced by other tables maintaining referential integrity between the data. An example for foreign keys which point to mviews will be given in section 4.3.2.

Refresh Types of Materialized Views

As already stated, mviews are not views, but have to be filled with data. There are two basic ways how mviews can be refreshed: “Complete” and “fast”. A “forced” refresh attempts to do a fast refresh, and if this is not possible, executes a complete refresh.

A complete refresh executes the underlying query and writes all data into the underlying table. If the query has a long runtime and only a small part of the fetched rows actually has changed, this approach is not very efficient. In such cases a fast refresh is more suitable.

This type of refresh only refetches the rows that actually changed in the mview (Gupta and Mumick, 1995). To provide information about which rows have actually changed, an additional structure has to be introduced: The materialized view logs (mview logs). These logs have to be created on every table that is used in a fast refreshing mview.

This is only one basic restriction that has to be met when creating fast refreshing mviews. There are also various restrictions on the structure of the underlying query. Some of these restrictions will be explained in more detail later. The bottom line for all these restrictions is that it has to be clear to Oracle how changes in the underlying tables affect the result of the query. This is why mviews cannot be based on views, but only on tables.

They can theoretically be also based on other mviews, but one has to be careful with this approach. Cascading refreshes of mviews are not executed in parallel but in sequence and can thus take a long time. Also when cascading mviews, if one of the mviews refreshes completely, all other mviews based on this mview will also do so, even if they were created to refresh fast. The reason for this is the fact that the complete refresh fills the mview log with all rows and the “fast” refresh has to fetch all rows from the log.

Conclusion

Although mvviews seem to be the perfect solution for precalculating results and thus saving runtime of queries, they actually have many restrictions. Especially for more complicated calculations as used in the GAINS model, most of the mvviews cannot be written to refresh fast. This is why mvviews will mostly be used for deployment of data between the central data and web schema and regional schemas. Since mvviews cannot be seen as the ultimate runtime problem solution, other solutions must be found.

4.2.2 Temporary Tables

Temporary tables are a special type of database tables (Managing Tables, 2002). They can hold “private” data, which means that data contained in these tables can be managed during the life time of a database connection, but will not be accessible any other established connection. For the GAINS application this would mean that data could be precalculated and stored in such a temporary table. This would create a user specific database cache within the temporary table thus minimizing the amount of time queries with long runtime have to be executed. Since also indices can be created on temporary tables, the precalculated data stored in these tables could be fetched faster and save even more runtime.

Temporary tables seem to have a lot of advantages, but they would only work, if the data in the temporary tables could be preserved over a user session and the user would be bound to a specific database connection. The first restriction is not a problem. With the *ON COMMIT PRESERVE ROWS* clause of the temporary table *CREATE* statement data in the temporary table can be preserved over the life time of a transaction. However, since data would get lost anyway when the connection to the database is closed, each database connection would not only have to be assigned to a user (and vice versa), but the connection would also have to stay alive from the user’s log in until log out (manual or caused by web session expiration).

This second restriction is problematic because database connections would have to be held open constantly and could not be reused by other users. It is also likely that the time they are really used by the user assigned to the connection, would only be a small percentage of the overall connection alive time.

Because of the relation of standby to overall time in a life time of a user bound

database connection, it was decided not to use user bound database connections. Instead the GAINS model uses connection pooling implemented by the Apache Software Foundation in the Commons project “DBCP” (DBCP, 2007). Although connection pooling can also have a positive impact on the performance of the system (Gutjahr and Loew, 2002), a more detailed discussion of this idea is beyond the scope of this paper.

Conclusion

Temporary tables are a useful tool, but only for caching data within the life time of one user request to the application. Using them for storing log-in-related data for each user is not an option because of the connection pooling. Therefore another method of caching user related data has to be implemented.

4.3 Reorganizing Data Structures

As already indicated in previous sections, the data structure of the GAINS model has been growing unsystematically over the years and has been developed by many persons. By analyzing the current structure of the data, some areas of improvement can be identified. A few of these areas will be discussed in this section. The topics shall give insight into how changing the structure will not only affect the integrity of data, but will also improve access time to data and simplify the data model altogether.

4.3.1 Materialized Views for Data Deployment

As mentioned in section 4.2.1 mviews can be used to deploy data between different databases, as well as between schemas within the same database. An example of data that has to be deployed to regional schemas, are the GAINS regions needed for display of calculation results within these schemas.

The list of all regions is stored within the “rains_web” schema in the “regions” table shown in figure 4.8. Access to a given region in a given database schema is controlled through the “region_options” table. This means that for a given schema, only the regions assigned in the latter table can be displayed.

The SQL command to list all regions according to the previously mentioned definition is shown in listing 4.6. The second nested select (starting at line no.

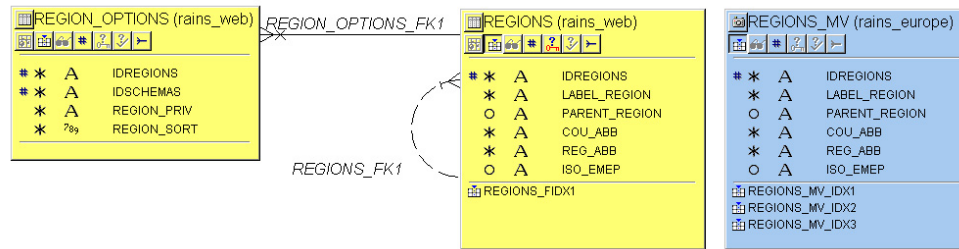


Figure 4.8: “regions_mv” materialized view

12) retrieves the name of the schema on which the mview is created from table “gui_schemas”. This table is very simple: It only has one row and one column. Of course the name of the schema could also be written into the query, but then the query would have to be changed for each regional schema.

Listing 4.6: Create command for materialized view “region_mv”

```

1 CREATE MATERIALIZED VIEW regions_mv
2   REFRESH FAST ON COMMIT
3   ENABLE QUERY REWRITE
4 AS
5 SELECT idregions , label_region , parent_region , cou_abb , reg_abb , iso_emep
6 FROM rains_web.regions r
7 WHERE EXISTS (
8   SELECT 1
9   FROM rains_web.region_options ro
10  WHERE ro.idregions = r.idregions
11  AND EXISTS (
12    SELECT 1
13    FROM gui_schemas s
14    WHERE s.idschemas = ro.idschemas
15  )
16 );

```

The mview is distinguished by an “_mv” at the end. Since mviews are treated by programs such as Tora or TOAD as tables in terms of display, it is recommended to name all mviews according to a unique convention so that they can easily be distinguished from tables. The mview is also created with an *QUERY REWRITE* option. This way the CBO can reduce I/O and processing time by optimizing access to the underlying resources and therefore returns results faster (Nanda, 2004).

To further optimize access time to data stored in the “regions_mv” materialized view, additional indices are created as shown in figure 4.8. The mview itself is not

very large in terms of the amount of table rows, but it will be accessed very often by SQL queries. So although the runtime improvement by creating the indices may not be very large if accessing data from the table alone, it will help when joining the table with other tables that have far more records.

Based on the same idea also the list of available users or the available activities, sectors and technologies can be deployed from the central “rains_web” and “rains_data” schemas to regional data schemas. How the just created mview can be used to assure referential integrity, will be discussed in more detail in section 4.3.3.

4.3.2 Activity/Sector/Technology Combinations

As already stated in section 4.1.1, the current structure of the “act_sec_tech” table (figure 4.1) impacts how indices on selected columns of the table can be used for different pollutants. Moreover referential integrity between the available activity/sector/technology combinations and the pollutants cannot be implemented because the pollutants are assigned in the columns of the table. It also has to be noted, that once a new pollutant is introduced, the structure of the table would have to be changed.

To solve the mentioned problems, new database tables have to be introduced. The first new table “act_sec_tech_all” (figure 4.9) combines available activity/sector combinations (table “act_sec”) with available technologies (table “tech”) and holds a list of all allowed abatement options.

The additional columns (“emiss_factor”, “cost_factor”, and “impl_rate”) are flags that define whether a given combination is needed for emission factors, cost factors and implementation rates. These columns will, for example, be used to achieve referential consistency for emission and cost factors. This idea will be discussed in more detail in sections 4.3.2, 4.3.3 and 4.3.4.

The retrieved combinations can further be combined with all available pollutants in table “act_sec_tech_to_poll”. This table will replace the “act_sec_tech” table. Through transposing the pollutants from table columns to values in rows of one column, the table now has more rows, but through purposeful indexing the access time to data stored within the table can be improved.

What is more important than access time, is the fact that the allowed activity/sector/technology/pollutant combinations can now be easily referenced by other tables. This possibility will play an important role for the next step, namely the re-

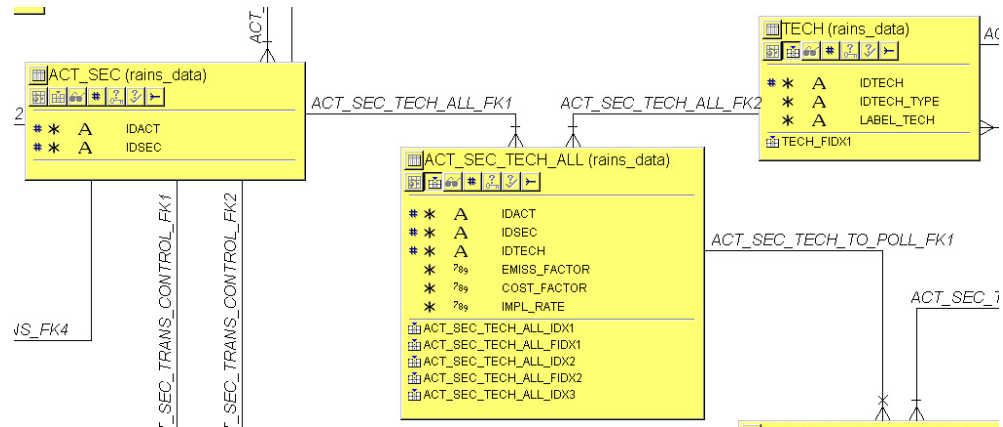


Figure 4.9: Structure of the “act_sec_tech_all” table in “rains_data”

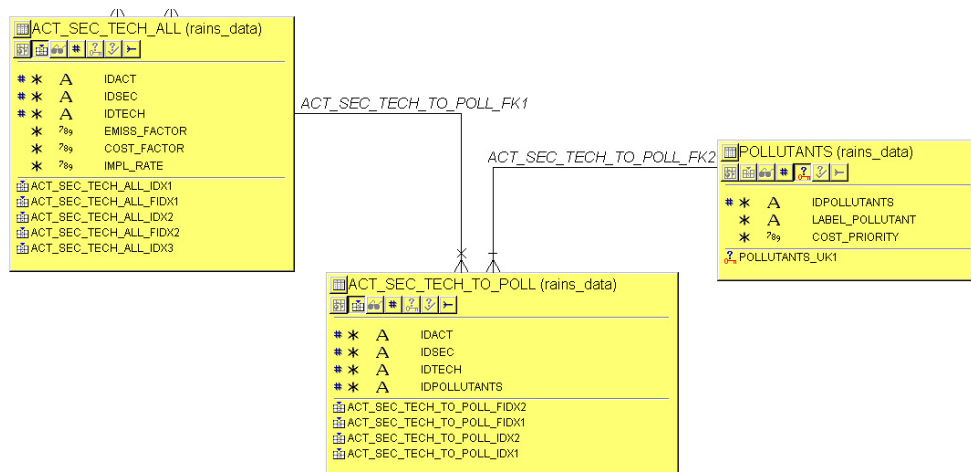


Figure 4.10: Structure of the “act_sec_tech_to_poll” table in “rains_data”

structuring of emission vector-related data that will be discussed in section 4.3.3.

Referential Integrity through Materialized Views

As mentioned in the previous section available activity/sector/technology combinations are used for different purposes. The above mentioned tables cannot assure referential integrity for all purposes. One possibility would be to create a new table for every needed purpose. This solution would lead to additional manual synchronization efforts, since the relation between abatement options and pollutants is the same independent of the purpose.

As already mentioned in section 4.2.1 foreign keys cannot only point to tables, but also to mviews. The structure of the SQL command for selecting the emission factor-relevant combinations (listing 4.8 line no. 5 and below) is fairly simple and therefore suitable for selecting data for a fast refreshing mview.

The command to retrieve the abatement options needed for cost factors is shown in listings 4.9. The command to retrieve implementation rate-relevant options looks the same, except for the flag in line no. 7. The difference between these commands and the command in listing 4.9, is that abated emission factors are pollutant specific, whereas cost factors and implementation rates of abatement options are not. Nevertheless it has to be assured that only options are selected that are assigned to at least one pollutant in table “act_sec_tech_to_poll”.

To be able to create the fast refreshing view, mview logs have to be created on the tables used in the underlying query. The command to create a log on the “act_sec_tech_all” table is shown in listing 4.7. Since the “emiss_factor” column is used in the subquery to filter combinations, the log has to be created not only with a *ROWID* and *PRIMARY KEY* option, but also with the *SEQUENCE* option on the columns used to filter the abatement options. The mview log on the “act_sec_tech_to_poll” table has to be created accordingly, except for the *SEQUENCE* option that is not needed.

Listing 4.7: Create command for mview log on “act_sec_tech_all”

```

1 CREATE MATERIALIZED VIEW LOG ON act_sec_tech_all
2   WITH
3     ROWID,
4     SEQUENCE (emiss_factor , cost_factor , impl_rate),
5     PRIMARY KEY
6   INCLUDING NEW VALUES;
```

Following this the mview for holding all emission factor relevant abatement

options should be created as shown in listing 4.8.

Listing 4.8: Activity/sector/technology/pollutant combinations relevant for emission factors

```

1 CREATE MATERIALIZED VIEW act_sec_tech_emiss_mv
2   REFRESH FAST ON COMMIT
3   ENABLE QUERY REWRITE
4 AS
5 SELECT idact, idsec, idtech, idpollutants
6 FROM rains_data.act_sec_tech_to_poll astp
7 WHERE EXISTS (
8   SELECT 1
9   FROM rains_data.act_sec_tech_all ast
10  WHERE ast.idact = astp.idact
11    AND ast.idsec = astp.idsec
12    AND ast.idtech = astp.idtech
13    AND ast.emiss_factor = 1
14 );

```

Listing 4.9: Activity/sector/technology/pollutant combinations relevant for cost factors

```

1 CREATE MATERIALIZED VIEW act_sec_tech_cost_mv
2   REFRESH FAST ON COMMIT
3   ENABLE QUERY REWRITE
4 AS
5 SELECT idact, idsec, idtech
6 FROM rains_data.act_sec_tech_all ast
7 WHERE cost_factor = 1
8 AND EXISTS (
9   SELECT 1
10  FROM rains_data.act_sec_tech_to_poll astp
11  WHERE astp.idact = ast.idact
12    AND astp.idsec = ast.idsec
13    AND astp.idtech = ast.idtech
14 );

```

How these mviews are used to assure referential consistency within the emission and cost factors, as well as the implementation rates of abatement options, will be discussed in later sections.

Consistency and Completeness of Data

Also within the “act_sec_tech_to_pollutant” table itself, completeness of data can be checked easier. An example is the check for completeness of activity/sector/-NOC-technology combinations. One of the conclusions that has to be drawn from the definition of the NOC technologies in section 2.3.2, is the fact that if

an activity/sector/technology combination exists for a given pollutant, also an activity/sector/NOC-technology combination has to exist. This requirement cannot be guaranteed by a database constraint, but it can be checked with the SQL command shown in listing 4.10.

Listing 4.10: Check of activity/sector/NOC-technology combinations

```

1 SELECT *
2 FROM act_sec_tech_to_poll astp
3 WHERE astp.idtech != 'NOC'
4 AND astp.idact NOT LIKE '%NV'
5 AND NOT EXISTS (
6   SELECT 1
7   FROM act_sec_tech_to_poll noc
8   WHERE noc.idact      = astp.idact
9   AND noc.idsec       = astp.idsec
10  AND noc.idtech       = 'NOC'
11  AND noc.idpollutants = astp.idpollutants
12 );

```

Another definition that can be checked through an SQL command, is the fact that an NSC technology can only set a minimum NOC implementation rate for a pollutant to which a NOC technology is assigned for the same activity/sector combination. In addition each NOC technology can only be controlled by, at most, one NSC technology. These requirements can be checked with the command shown in listing 4.11.

Listing 4.11: Check of activity/sector/NSC-technology combinations

```

1 SELECT *
2 FROM act_sec_tech_to_poll astp
3 WHERE astp.idtech LIKE 'NSC%'
4 AND astp.idact NOT LIKE '%NV'
5 AND (NOT EXISTS (
6   — check for not existing NOC option
7   SELECT 1
8   FROM act_sec_tech_to_poll noc
9   WHERE noc.idact      = astp.idact
10  AND noc.idsec       = astp.idsec
11  AND noc.idtech       = 'NOC'
12  AND noc.idpollutants = astp.idpollutants
13 ) OR EXISTS (
14   — check for multiple NSC to pollutant assignments
15   SELECT 1
16   FROM act_sec_tech_to_poll nsc
17   WHERE nsc.idact = astp.idact
18   AND nsc.idsec  = astp.idsec
19   AND nsc.idtech LIKE 'NSC%'
20   GROUP BY idact, idsec, idpollutants
21   HAVING COUNT(nsc.idtech) > 1

```

22 |));

4.3.3 Emission Vector Related Data

As already mentioned, the amount of data related to one emission vector is very large. What makes management of the data even harder, is the fact that it is spread across many tables. Furthermore the fact, that there is no single table holding a list of all available vectors makes the analysis of the database structure even more complicated. Therefore the list of all emission vector related tables cannot be retrieved by finding all referential constraints pointing to one table.

Finding Tables Holding Vector Related Data

The first step of analysis of the underlying data is finding the relevant database tables. As already mentioned in section 3.2.1, there are two types of emission vectors: Version and user specific. The list of available data versions is stored in the table “versions”, whereas all users that can log in to an application version running on a given database schema, are stored in the table “login”.

By looking at these two tables, an analysis of tables referencing them can be started. Since a rough estimation reveals that about 120 tables are referencing them with referential constraints, more detailed information about the tables has to be gathered. Listing 4.12 shows an SQL command that retrieves information about database tables having referential constraints to table “versions”. The same command can be executed for tables referencing the table “login”.

Listing 4.12: Tables having referential constraints to table “versions”

```

1 SELECT fk.owner, fk.table_name, fk.constraint_name, fk.delete_rule, t.num_rows,
2    ROWS.TO_STRING(CAST(Collect(fk.column_name) AS t_varchar2_column), ', ') AS
   fk_cols,
3    ROWS.TO_STRING(CAST(Collect(lpad(pc.position, 2, '0') || ': ' ||
4    pc.column_name) AS t_varchar2_column), ', ') AS pk_cols,
5    ROWS.TO_STRING(CAST(Collect(lpad(tc.column_id, 2, '0') || ': ' ||
6    tc.column_name) AS t_varchar2_column), ', ') AS tab_cols
7 FROM user_constraints fk
8 JOIN user_tables t ON (
9    t.table_name = fk.table_name)
10 JOIN user_cons_columns fc ON (
11    fc.constraint_name = fk.constraint_name)
12 JOIN user_tab_columns tc ON (
13    tc.table_name = fk.table_name)
14 LEFT JOIN user_constraints p ON (

```

```

15      p.table_name          = fk.table_name
16      AND p.constraint_type = 'P')
17      LEFT JOIN user_cons_columns pc ON (
18          pc.constraint_name = p.constraint_name)
19 WHERE fk.constraint_type = 'R'
20 AND EXISTS (
21     SELECT 1
22     FROM user_constraints t
23     WHERE t.constraint_name = fk.r_constraint_name
24     AND t.owner              = UPPER('rains_europe')
25     AND t.table_name         = UPPER('versions')
26 )
27 —AND t.num_rows = 0
28 GROUP BY fk.owner, fk.table_name, fk.constraint_name, fk.delete_rule, t.num_rows
29 ORDER  fk.owner, fk.table_name, fk.constraint_name;

```

The command is based on Oracle’s “Data Dictionary Views” (Oracle Data Dictionary Views, 2002). It retrieves information about the referential constraints and primary key constraints from the “user_constraints” view (type “R” and “P”). Information about the columns of the constraints can be retrieved from the “user_cons_columns” view, and information about the table columns can be retrieved from the view “user_tab_columns”. Another interesting question for the analysis is, how many rows each of the referencing tables has. This information can be retrieved from the “num_rows” column of the “user_tables” view.

To be able to analyze the retrieved information more easily, the columns of each of the tables, primary and foreign keys are transposed into a comma separated string with the PL/SQL function “rows_to_string”. Although it is a custom-made and not Oracle standard function, discussion of this is beyond the scope of this paper.

A combination of the lists of tables referencing both the “versions” and the “login” has to be filtered further. It also contains, for example, the tables “path_abb” (mentioned in section 2.3.3) and “cons.n” (mentioned in section 2.3.4) to maintain referential integrity between these entities and their owners. Furthermore the list does not contain some of the emission vector related tables, because a needed constraint was not set.

To gather more information for analysis of the tables the already mentioned initialization process of the emission vector has to be analyzed in detail. The initialization contains a set of SQL commands for every pollutant, and the amount of queries depends on the pollutant. All together around 140 SQL commands have to be analyzed.

By looking at the queries, two types of tables can be identified. First, tables

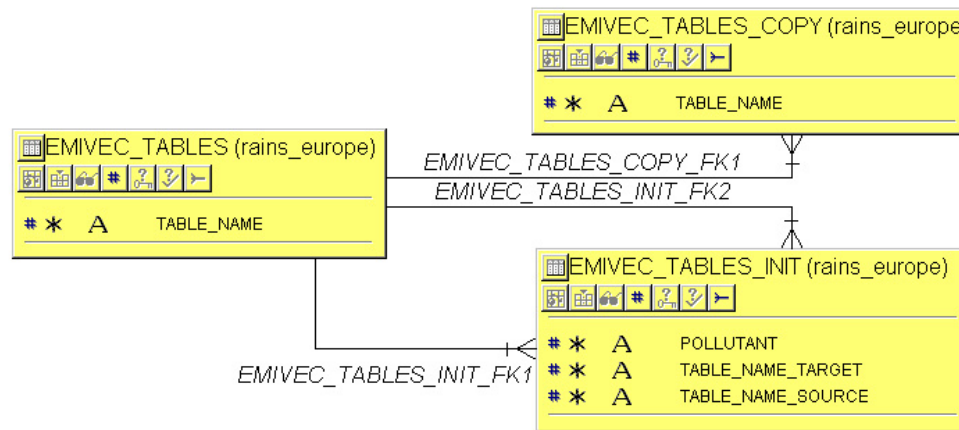


Figure 4.11: Temporary tables for emission vector analysis

holding input data for the emission vectors: Data stored in these tables is uploaded by experts. This data is then joined during the initialization process to retrieve data stored in output tables. The result of the manual analysis of commands is stored in the table “emivec_tables_init” as shown in figure 4.11. The table holds information about which table provides underlying data for which table for each of the pollutants for which an initialization can be triggered.

Another source of information retrieval is a script that was written to manually create a new emission vector. This script steps through a list of tables, selects data related to a given emission vector stored in this table, and copies it to the same table, but for a different vector. The results of the analysis of this script are stored in table “emivec_tables.copy” shown in figure 4.11.

By merging all the different names of database tables stored in the two previously mentioned tables, a list of all tables can be retrieved. This list is stored in table “emivec_tables” allowing the possibility to combine the results of the manual analysis with the results of the analysis of referential constraints. This combination leads to a list of 79 tables that hold vector specific data. It also reveals that two tables, namely “housing” and “amoemvec.n” were neither referencing the “login” nor the “versions” table. During the search process another ten tables could be identified as spare and were deleted from the system because they are not used anymore.

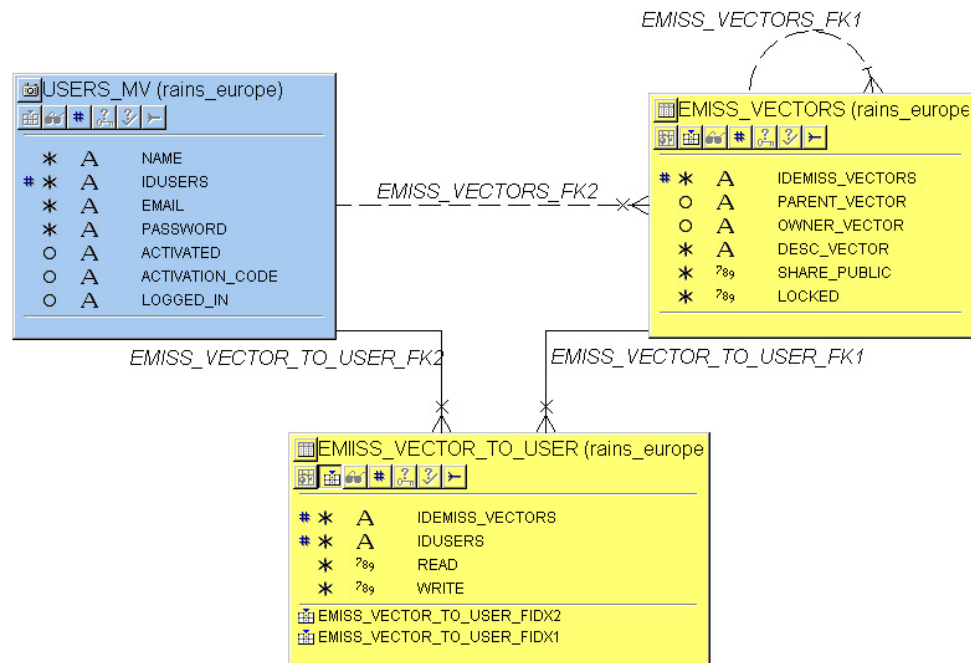


Figure 4.12: Tables for listing of emission vectors and user access to them

Creating a New Data Structure

As already mentioned, one of the origins of the modelling problem is the lack of a central table holding a list of emission vectors. This table has to satisfy both the version vector and the user vector constraint. As you can see from figure 4.12 the new table “emiss_vectors” satisfies this requirements.

Beside the unique name, each of the vectors can also have a “parent_vector”. For user specific vectors another vector can be assigned to provide data where the user is not able to provide them for her/his vector. In this case the owner of the vector has to be assigned. For version specific vectors, no owner is defined. The constraint on the owner column references the “users_mv” mvview that deploys the list of users stored in the “rains_web” schema to the regional schemas.

Another modelling request that might be implemented in the future because of the new table structure is that also a version vector is only defined partially. Imagine the following example: Starting from a given version vector, there is a need to change only a small percentage of values to satisfy the modelling requirements

for a new generation of scenarios. Copying the complete vector just for a couple of new values does not seem to be justifiable. In this case it would be possible to specify a parent vector for a version emission vector, use only the few new values from this vector, and fill up the rest of the values with values from the parent vector.

For user-specific vectors the access to the vector by other users is determined by access rules mentioned in section 2.5.3. For version vectors the read access is granted to all users through the “share_public” flag or through using the vector in a scenario that is made public (as mentioned in sections 2.4.1 and 2.4.2). Write access to a version vector is only granted if the vector is not locked. Locking a vector is important to provide consistency in scenario results after they have been published.

During the development phase of a vector it is important to grant write access to more than one user. Because of the multifaceted nature of the vector data has to be provided by many experts. Sometimes it is necessary to only grant read access so that certain users can only read and give feedback, but not change any values. The management of user access is done based on the “emiss_vector_to_user” table shown in figure 4.12. Read and write access of single users to selected emission vectors can be done by setting flags in the “read” and “write” columns.

Redirecting Referential Constraints

After creating the table which holds the list of available emission vectors, the references of the tables identified in section 4.3.3 have to be redirected. Since constraints of 79 tables have to be changed, an automated process is preferable. As already mentioned, a list of these tables is stored in the “emivec_tables” table. Therefore an SQL command can be written to build the commands necessary for altering the tables. This command is shown in listing 4.13.

Listing 4.13: Generating commands to alter constraints on vector related tables

```

1 SELECT f.table_name , f.constraint_name , f.fk_cols , 1 AS step ,
2   'ALTER TABLE ' || t.table_name || ' DROP CONSTRAINT ' || LOWER(f.
   constraint_name) || ';' AS command
3 FROM emivec_tables t
4   JOIN emivec_tables_potential f ON (
5     UPPER(f.table_name) = UPPER(t.table_name))
6 UNION ALL
7 SELECT f.table_name , f.constraint_name , f.fk_cols , 2 AS step ,
8   'ALTER TABLE ' || t.table_name || ' ADD CONSTRAINT ' || LOWER(f.constraint_name
   )

```

```

9 | || ' FOREIGN KEY ( ' || LOWER(fk_cols) || ' ) REFERENCES emiss_vectors (
    idemiss_vectors) ON DELETE CASCADE; ' AS command
10 FROM emivec_tables t
11 JOIN emivec_tables.potential f ON (
12     UPPER(f.table_name) = UPPER(t.table_name))
13 ORDER BY step, table_name;

```

Information about the names of constraints and the names of columns the constraints have to be created on, is dynamically retrieved from the “emivec_tables_potential” table. This table holds the results of the referential constraint analysis on the “login” and “version” tables retrieved by the SQL command in listing 4.12.

The results of the command in listing 4.13 can be exported to Microsoft Excel, where the columns not further necessary can be easily eliminated. The list of SQL commands can then be copied back to the SQL tool and executed. During the recreation of the constraints, remains of emission vectors that are not meant to be in the database any more, may be discovered in some of the tables. Since the referential constraints are created to cascade upon a delete, the easiest way to cope with this problem is to create the appropriate vectors in the “emiss_vectors” tables. After all constraints have been successfully re-established, vectors that are no longer used can be deleted and related data is eliminated by the cascading referential constraints.

Emission Factors and Removal Efficiencies

As mentioned in section 4.3.3 the analysis of vector related tables also included the analysis of the initialization process. This process was already also mentioned in section 2.3.5 when discussing the emission factors and removal efficiencies. For each pollutant the removal efficiencies and emission factors are stored in separate tables. These tables are the main target of data generated by the initialization. An analysis of the process reveals that the tables are not only named inconsistently, but also the way in which the emission calculation is done varies between the pollutants.

As mentioned in equation 2.1 emissions of the RAINS model are calculated based on emission factors and removal efficiencies. But the results of the analysis presented in table 4.1 reveal that for some pollutants there are no removal efficiencies, but only emission factors. What can also be seen from the table is the fact, that for PM the emission calculation is divided by fractions of PM.

The reason why removal efficiencies are missing for CH₄, CO₂, N₂O, and

Pollutant	Fraction	Factor Table	Rem. Eff. Table
CH ₄	CH ₄	emiv_ch4 (*)	—
CO ₂	CO ₂	co2emvec (*)	—
FGAS	FGAS	emiv_fgase	emiv_fgase
N ₂ O	N ₂ O	emiv_n2o (*)	—
NH ₃	NH ₃	emivec_a (*)	—
NO _x	NO _x	emivec_n	emivec_nr
PM	PM _{2.5}	emiv_fin	emiv_fir
PM	PM ₁₀	emiv_p10	emiv_10r
PM	PM _{TSP}	emiv_tsp	emiv_tsr
PM	PM ₁	emiv_pm1	emiv_p1r
PM	PM _{BC}	emiv_bc	emiv_bcr
PM	PM _{OC}	emiv_oc	emiv_ocr
SO ₂	SO ₂	emivec_s	emivec_sr
VOC	VOC	emiv_voc	emiv_vor

Table 4.1: Tables used for storing emission factors and removal efficiencies

NH₃, is the fact that for these pollutants the formula to calculate emissions (equation 2.1) was slightly adapted (Wagner et al., 2007):

$$E_{p,r,y} = \sum_{a,s,t} E_{p,r,a,s,t,y} = \sum_{a,s,t} A_{r,a,s,y} \cdot X_{r,a,s,t,y} \cdot e_{p,r,a,s,t}^{abated} \quad (4.1)$$

where

$$e_{p,r,a,s,t}^{abated} = e_{p,r,a,s} \cdot (1 - \eta_{p,r,a,s,t})$$

Since for CH₄, CO₂, N₂O, and NH₃ the “abated” emission factors ($e_{p,r,a,s,t}^{abated}$) are calculated and stored directly, the removal efficiencies are not needed anymore. The reason for this approach was that the abated emission factors can be measured, whereas the unabated emission factors and removal efficiencies have to be calculated from the measures that can be practically gathered.

To achieve consistency in the emissions calculation between the pollutants, the calculation should be based on equation 4.1 for all pollutants. Since the abated emission factors can be retrieved through a simple multiplication from the unabated emission factors and the removal efficiencies and both values are calculated during the initialization process, this change can be introduced quite easily.

The second goal that has to be achieved when redesigning the data structure of emission vector related data is the consistency of the data. The activity/sector combinations of unabated emission factors and activity/sector/technology combinations of removal efficiencies and abated emission factors available for a pollutant have to be consistent with the assignment of activity/sector/technology to pollutants defined in “act_sec_tech_to_poll” as mentioned in section 4.3.2.

As shown in figure 4.13 the new tables are extended by the “idpollutant_fractions” and the “pollutant” column. Of course the relation of a pollutant fraction to a pollutant is known and its storage in these tables is redundant, but it is necessary to assure referential integrity. Since the activity/sector/technology combinations are defined by pollutant and not by pollutant fraction, the referential constraint cannot be set on the “pollutant_fractions” column.

Another structural redundancy that can be observed in the tables shown in figure 4.13 is the “act_type” column. The reason for this redundancy is that this column is crucial for emission calculations done based on these tables. Since the data in the tables is precalculated anyway during the initialization process it seems to be reasonable to store this redundant information to save query runtime by not having to join the tables to the “sec” table in “rains_data”.

After the tables are created, they have to be filled with data from the tables listed in table 4.1. The SQL commands for retrieving the unabated emission factors and removal efficiencies for NO_x are shown in listings 4.14 and 4.15. For the other pollutants which have unabated emission factors the queries are adapted accordingly.

Listing 4.14: Command to retrieve unabated emission factors for NO_x

```

1 INSERT INTO emiss_factors
2 (idemiss_vectors, idregions, idpollutant_fractions, pollutant, idact, idsec,
   act_type, factor)
3 SELECT owner, region, 'NOX', 'NOX', act_abb, sec_abb, act_type, eemf
4 FROM emivec_nr;
```

Listing 4.15: Command to retrieve removal efficiencies for NO_x

```

1 INSERT INTO emiss_remeffs
2 (idemiss_vectors, idregions, idpollutant_fractions, pollutant, idact, idsec,
   idtech, act_type, remeff)
3 SELECT owner, region, 'NOX', 'NOX', act_abb, sec_abb, tech_abb, act_type, remeff
4 FROM emivec_nr;
```

In the next step the abated emission factors can be retrieved. As already mentioned for CH_4 , CO_2 , N_2O , and NH_3 they can be copied from the tables listed in

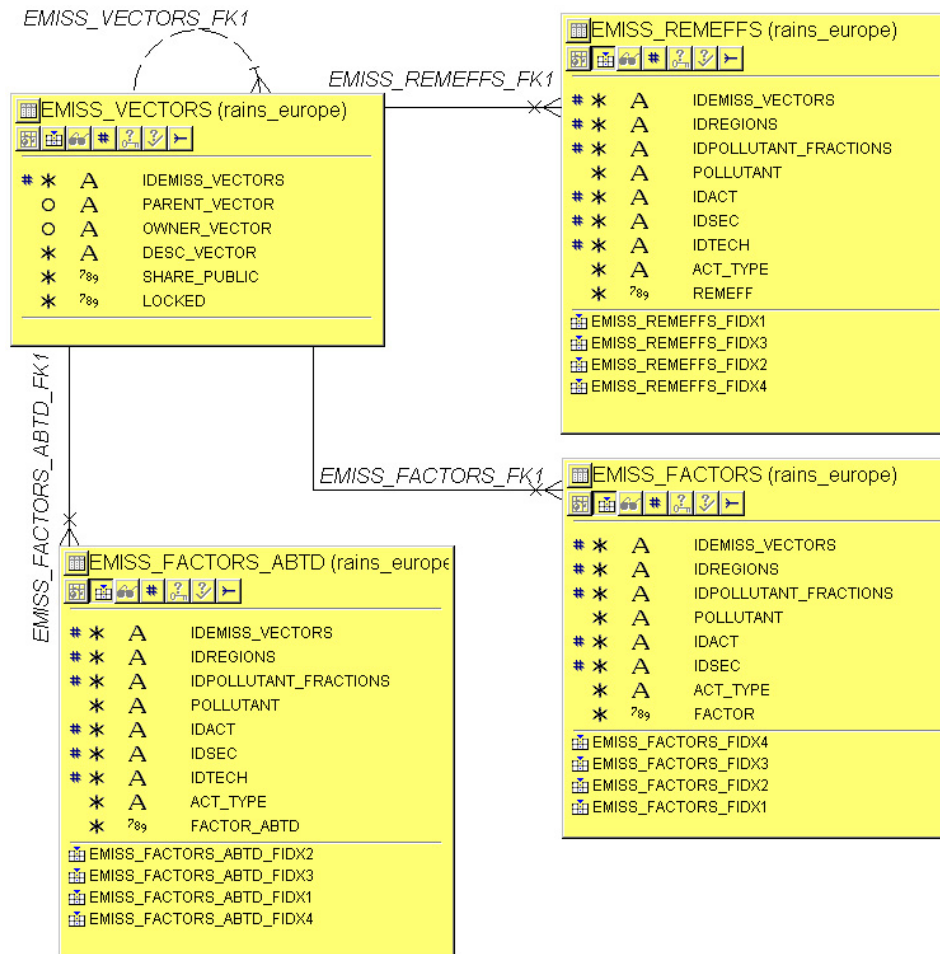


Figure 4.13: Database tables holding emission factors and removal efficiencies

table 4.1, whereas for the other pollutant they have to be calculated from the two just mentioned tables according to equation 4.1. This formula can be written in SQL as a *UNION* statement as shown in listing 4.16.

Listing 4.16: Command to retrieve abated emission factors

```

1 INSERT INTO emiss_factors_abtd
2 (idemiss_vectors, idregions, idpollutant_fractions, pollutant,
3  idact, idsec, idtech, act_type, factor_abtd)
4 SELECT idemiss_vectors, idregions, idpollutant_fractions, pollutant, idact, idsec
5   , 'NOC' AS idtech, act_type, factor AS factor_abtd
6 FROM emiss_factors
7 UNION
8 SELECT f.idemiss_vectors, f.idregions, f.idpollutant_fractions, f.pollutant,
9   f.idact, f.idsec, r.idtech, f.act_type,
10  f.factor*(1.-r.remeff/100) AS factor_abtd
11 FROM emiss_factors f
12 JOIN emiss_remeffs r ON (
13   r.idemiss_vectors      = f.idemiss_vectors
14   AND r.idpollutant_fractions = f.idpollutant_fractions
15   AND r.idregions         = f.idregions
16   AND r.idact             = f.idact
17   AND r.idsec             = f.idsec);

```

As shown in the first part of the *UNION* command, all unabated emission factors are joined with the “NOC” technology. As already explained in section 2.3.2, this special technology has a removal efficiency of zero by definition. Therefore the unabated emission factor equals the abated emission factor. In the second part the abated emission factor is calculated from the abated emission factors and the removal efficiencies as stated in equations 2.1 and 4.1.

In theory the statement could be written as an *UNION ALL* statement because of the runtime benefits mentioned in section 4.1.3 since the results of both parts of the command should not overlap. In fact the two parts of the query can intersect because the “emiss_remeff” contains removal efficiencies for NOC technologies although it should not. If this inconsistency in modelling leads to inconsistent results in terms of abated emission factors, the *UNION* would produce multiple abated factors for the same primary key column combination. This in turn would lead to an SQL error upon inserting these wrong factors into the database and the error would be discovered. Fortunately it turns out upon importing, that although there is an inconsistency in modelling of data, it does not lead to wrong abated emission factors.

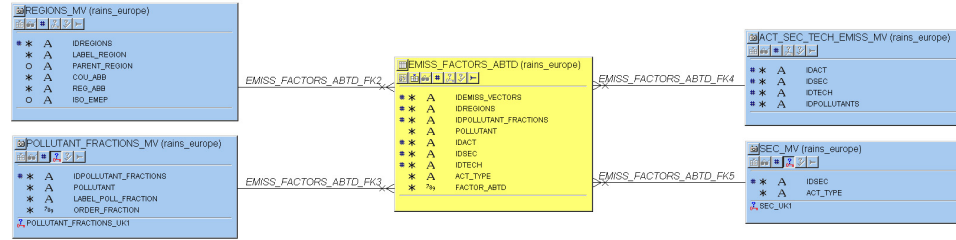


Figure 4.14: Foreign key constraints on table “emiss_factors_abtd”

Consistency of Emission Vector Related Data

As already mentioned the consistency of data within the emission vector tables is very important for the correct emission calculation. With the new structure of the initialization related output tables, the requirement of data consistency can now be guaranteed.

As shown in figure 4.14 the “emiss_factors_abtd” table references four mviews, namely “regions_mv” (already discussed in section 4.3.1), “pollutant_fractions_mv” (holding all pollutant fractions defined in “rains_data”), “sec_mv” (holding all GAINS sectors and the assigned activity types defined in “rains_data”), and “act_sec_tech_emiss_mv” (already discussed in section 4.3.2).

Upon the step of creating the mentioned referential constraints, many inconsistencies in the defined abatement options were discovered. These options were eliminated from the “emiss_factors_abtd” table. After setting the referential constraints, such inconsistencies will not occur any more.

Similar considerations have to be taken into account when creating the referential constraints for cost factors. The only difference is the fact that cost factors are not pollutant specific as already mentioned in section 2.3.5.

Completeness of Emission Vector Related Data

As mentioned in section 2.3.5 it is very important to understand that incomplete emission vectors can lead to errors in emission and cost calculation done by the GAINS model. Other than data integrity, completeness of data cannot be checked by database constraints. Therefore the application can only give different views indicating where data is missing.

The following examples of such checks and views are presented for abated

emission factors. Similar routines can be run for checking the unabated emissions factors and removal efficiencies, as well as cost factors. What also has to be noted, is the fact that such checks would have to be done for every pollutant independently if the emission factors were not stored in one table, but in a separate table for each pollutant, as before the restructuring.

The first check shall give an overview, of whether emission vectors exist that do not have any factors defined for a given pollutant and whether such vectors are used in any of the currently available scenarios. As shown in listing 4.17 the command uses all available pollutants, and fractions and the scenario structure to retrieve the needed data.

Listing 4.17: Incomplete but used emission vectors

```

1 SELECT DISTINCT v.idemiss_vectors ,
2   COUNT(DISTINCT p.pollutant), COUNT(DISTINCT s.scen)
3 FROM pollutant-fractions-mv p, emiss-vectors v
4   JOIN scenario-n s ON (
5     s.emv_owner = v.idemiss_vectors)
6 WHERE NOT EXISTS (
7   SELECT 1
8   FROM emiss-factors-abtd f
9   WHERE f.idemiss_vectors = v.idemiss_vectors
10  AND f.idpollutant-fractions = p.idpollutant-fractions
11 )
12 GROUP BY v.idemiss_vectors
13 ORDER BY 1, 2;
```

Already this first and sketchy test shows that five out of the eight currently available emission vectors are not available for at least one pollutant at all. Emissions calculations for the missing pollutants of the scenarios in which these vectors are used, will always return no result rows because of the missing factors. Such a matter of fact is of course not desired in terms of usability of the model, but the lack of the calculation results can be easily interpreted and is not misleading. Although not presented in this paper, a similar check can be run to analyze which regions are completely missing within which emission vector.

The next check shall give an overview of the amount of abatement options given for the defined emission vectors in each region. In the first step a view is created to hold the amount of activity/sector/technology combinations defined within all available vectors for all given regions and pollutants (first subselect starting in line no. 3 and following of listing 4.18) and the amount of needed combinations (line no. 9 and following).

Listing 4.18: Amount of abatement options per vector, pollutant and region

```

1 CREATE OR REPLACE VIEW emiss_factors_abtd_by_region AS
2 SELECT v.idemiss_vectors , p.idpollutant_fractions , r.idregions ,
3        NVL((SELECT COUNT(*)
4              FROM emiss_factors_abtd f
5              WHERE f.idemiss_vectors      = v.idemiss_vectors
6                  AND f.idpollutant_fractions = p.idpollutant_fractions
7                  AND f.idregions           = r.idregions
8              GROUP BY f.idpollutant_fractions , f.idemiss_vectors),0) AS amount_def ,
9        (SELECT COUNT(*)
10         FROM act_sec_tech_emiss_mv ast
11         WHERE ast.idpollutants = p.pollutant
12         GROUP BY ast.idpollutants) AS amount_needed
13 FROM emiss_vectors v, pollutant_fractions_mv p, regions_mv r;

```

This check is only run for the regions, which exist within a given emission vector. As shown this condition can be kept if only vector, region and pollutant fraction combinations are taken into account where at least one combination is assigned. Before making the next step of this check routine, abatement options of all of the just mentioned combinations will be eliminated, where less than 10% of the needed options are defined (listing 4.19). It is reasonable to remove the combinations because it can be assumed that such mavericks must have been created by mistake during the initialization process or manual copying.

Listing 4.19: Remove factor combinations of initialization mavericks

```

1 DELETE FROM emiss_factors_abtd f
2 WHERE EXISTS (
3   SELECT 1
4   FROM emiss_factors_abtd_by_region r
5   WHERE amount_def > 0
6   AND (amount_def/amount_needed)*100 < 10;
7   AND r.idemiss_vectors      = f.idemiss_vectors
8   AND r.idregions           = f.idregions
9   AND r.idpollutant_fractions = f.idpollutant_fractions
10 );

```

Now the completeness of the remaining emission factors can be checked as shown in listing 4.20. Line no. 6 can be commented in, and the comparison changed to view either all vector/pollutant fraction sets, incomplete, or complete ones.

Listing 4.20: Percentage of missing emission factors

```

1 SELECT f.idemiss_vectors , idpollutant_fractions ,
2        ROUND(AVG(100-(amount_def/amount_needed)*100),2) AS perc_missing
3 FROM emiss_factors_abtd_by_region f
4 WHERE amount_def > 0

```

```

5 GROUP BY idemiss_vectors , idpollutant_fractions
6 —HAVING AVG(100-(amount_def/amount_needed)*100) != 0
7 ORDER BY 1, 2;

```

It becomes apparent from the results retrieved by this command that only a small percentage of emission vectors is complete even for a single pollutant. The amount of missing emission factors may be very small for some of the vectors and pollutant fractions, but since it cannot be forgotten that the emission vector has to be complete, even a single missing factor can potentially lead to wrong emission calculations.

Therefore the next step of the analysis aims at finding all missing emission factors. Because of the long runtime of the query, the results are stored in a table (as shown in listing 4.21) so that further steps based on these results can be done faster.

Listing 4.21: Abatement options for which emission factors are missing

```

1 CREATE TABLE emiss_factors_abtd_fill_up AS
2 SELECT r.idemiss_vectors , r.idregions ,
3        r.idpollutant_fractions , ast.idpollutants AS pollutant ,
4        ast.idact , ast.idsec , ast.idtech , s.idsec_type AS act_type
5 FROM act_sec_tech_emiss_mv ast
6      JOIN (SELECT DISTINCT idemiss_vectors , idregions , idpollutant_fractions ,
7                          pollutant
8              FROM emiss_factors_abtd) r ON (
9        r.pollutant = ast.idpollutants)
10     JOIN rains_data.sec s ON (
11        s.idsec = ast.idsec)
12 WHERE NOT EXISTS (
13     SELECT 1
14     FROM emiss_factors_abtd f
15     WHERE f.idemiss_vectors      = r.idemiss_vectors
16     AND f.idregions              = r.idregions
17     AND f.idpollutant_fractions = r.idpollutant_fractions
18     AND f.idact                  = ast.idact
19     AND f.idsec                  = ast.idsec
20     AND f.idtech                 = ast.idtech
21 );

```

A quick *COUNT(*)* on both the “emiss_factors_abtd_fill_up” and “emiss_factors_abtd” reveals that around 290.000 combinations have to be filled up compared to around 2.300.000 that are already defined. This means that more than 10% of needed emission factors are missing although the analysis takes only vector/pollutant/region combinations into account for which at least one emission factor is already defined.

The reasons for such a high “incompleteness” rate are various. First, complete-

ness of the emission factors was never checked in such a systematic way before. Previously such a check has been quite time consuming because it could not be done by one query, partially because of the former structure of the “act_sec_tech” table, and partially because of the distribution of the emission factors over many tables.

Second, the emission vectors were developed over time. The available activity/sector/technology combinations as well as the assignment of the pollutants to them changes quite frequently. Since no overview of the missing combinations was given, the lack of combinations was hard to discover. Some of the missing combinations can be found in various vectors that are historically dependent on each other: Since an emission vector is initially created as a copy of an existing vector, undiscovered holes are also persistent in the new vector.

Third, some of the factors might be missing or be *NULL* because of the combination of former database tables that allowed *NULL* values for emission factors, missing input data and/or mistakes in the initialization process: A mistake in the initialization process or missing underlying data led to missing combinations or *NULL* values of the factors. Since the database structure did not prevent these values from being stored, the mistakes were not discovered.

Last, but not least, there is also a possibility, that the assignment between the abatement options and the pollutants within the “act_sec_tech_to_poll” or even the “act_sec_tech” table might be erroneous.

What has to be noted on the positive side, is the fact that these missing emission factors do not necessarily lead to mistakes in the emission calculation. They are errors in the modelling approach because the rule of completeness is not observed, but initially they are only potential mistakes of the model.

To check whether the missing combinations can lead to wrong results in the emission calculation, it has to be checked in which context in terms of the scenario definitions the incomplete vector/region combinations are used. This information can be retrieved according to the SQL command shown in listing 4.22.

Listing 4.22: Missing emission factors affecting calculation

```

1 SELECT f.idemiss_vectors , f.idpollutant_fractions , f.pollutant , f.idact , f.idsec ,
   f.idtech , f.act_type ,
2   ROUND(MAX(a.activity),2) AS activity_max , ROUND(MIN(a.activity),2) AS
   activity_min ,
3   ROUND(MAX(c.perc),2) AS perc_max , ROUND(MIN(c.perc),2) AS perc_min
4 FROM scenario_n s
5   JOIN emiss_factors_abtd_fill_up f ON (
6     f.idemiss_vectors = s.emv_owner

```

```

7 |   AND f.idregions    = s.region
8 |   AND f.act_type    = s.act_type)
9 | JOIN actpath_trans a ON (
10 |   a.path_abb        = s.path_abb
11 |   AND a.region      = s.region
12 |   AND a.act_abb_c   = f.idact
13 |   AND a.sec_abb_c   = f.idsec)
14 | JOIN constr_n c ON (
15 |   c.con_strat       = s.con_strat
16 |   AND c.act_abb     = f.idact
17 |   AND c.sec_abb     = f.idsec
18 |   AND c.tech_abb    = f.idtech)
19 | WHERE a.activity    != 0
20 | AND c.perc          != 0
21 | GROUP BY f.idmiss_vectors , f.idpollutant_fractions , f.pollutant ,
22 |   f.idact , f.idsec , f.idtech , f.act_type
23 | ORDER BY 1, 2, 3, 4, 5, 6;

```

Although around 290.000 emission factors are missing for all vectors to be complete for the defined regions, the query returns that only around 2.000 of these missing factors have an impact on the emission calculation. In addition a very high percentage of these combinations belong to older emission vectors that are rarely used in the model.

Nevertheless the missing emission factors for the abatement options have to be provided where needed. As already stated, very similar considerations apply also to cost factors as to the emission factors.

Conclusions

Although in theory the emission vectors should be complete for all regions and pollutants, in reality this requirement is not met. Many of the available vectors are defined only partially, both for regions and pollutants. Adding complete regional fragments or complete pollutant fragments to the vectors so that it is defined for all regions and pollutants, is not purposeful in terms of better quality of the results of the model.

Rather than that, information about the available regions and pollutants should be provided. Based on this data, the vectors should only be allowed to be used in scenarios for regions for which data is available. Also, the availability of pollutants for a given emission vector has to be reflected in the display of scenarios. How this requirement can be modeled and implemented in the interface will be discussed in more detail in section 4.4.3.

The most important aspect of restructuring the emission vector related data is

that the emission factors are now stored in only one table for all pollutants. The positive effects which this has on the emission calculation will be discussed in 4.4.1.

By the clean distinction between complete and partial emission vectors, the initialization process can also be rewritten to work properly for partial (user-specific) emission vectors. The significant feature of the model that will allow users of the interface to create their own emission vectors without necessarily having the expertise to gather all data on their own, will be discussed in more detail in section 4.4.2.

4.3.4 Applicabilities

As mentioned in section 3.2.2 the applicabilities are modeled in a very inhomogeneous way. This section will show how to achieve more consistency in data modelling and clean the existing data.

Collecting Applicabilities

Currently the applicabilities are stored in ten different database tables. The first step of analysis is to copy the data from the various tables into one table (“applic_collect_all”).

As shown in table 4.2, seven of these database tables are pathway and three emission vector related. The difference in the amount of available applicabilities within the tables is very large, two tables are completely empty.

The “applic_nh3” table is the only table, in which the values are not year-specific. To be able to analyze the values in a common way with all the other tables, the given applicability rates are assigned to every year available in the system.

As mentioned in section 3.2.2, applicabilities are maximum application rates. This is why the activity/sector/technology combinations of the applicabilities have to be consistent with the definition of available combinations assigned to implementation rates as discussed in section 4.3.2. Therefore, before further analyzing the collected data, consistency in terms of allowed abatement options for implementation rates is checked.

Another check that can be done immediately, is whether any values $\geq 100\%$ are stored. These values do not need to be analyzed further. Along with the idea

Purpose	Table name	Type	No. records
AGR	applic_agr	pathway	0
CH ₄	applic_ch4	pathway	7.056
CROSS	applic_cross	emission vector	13.272
FGAS	applic_fgass	pathway	2.122
N ₂ O	applic_n2o	pathway	0
NH ₃	applic_nh3	emission vector	95.175
NO _x	appln_n	pathway	372.299
PM _{2.5}	applfin	pathway	4.500
SO ₂	appls_n	pathway	235.520
VOC	applic_voc	emission vector	268.380

Table 4.2: Database tables holding applicability data

of not storing zero values for implementation rates, applicabilities that are equal to 100% also do not have to be stored in the database. Out of the around 1.000.000 values listed in table 4.2, around 1/3 can be deleted based on this rule.

Combining Applicabilities Into Sets

After the referential integrity is assured, the applicabilities can be grouped into sets. This is done by analyzing the scenario structure: The emission vector specific applicabilities are collected from the vector assigned in the scenario structure for a given region, the pathway related applicabilities by the assigned pathway.

Since applicabilities are only needed for scenarios for which cost curves can be displayed, this information from the “scen_master” table, which holds all available scenarios, is used to narrow down the analysis only for the relevant scenarios. The pathway related applicabilities are imported as shown in listing 4.23. The emission vector related values are imported accordingly. Since for now a set is created for every relevant scenario, the name of the scenario is used to identify each set.

Listing 4.23: Collecting data for needed applicability sets

```

1 INSERT INTO applic_set_data_imp
2 (idapplic_sets , path_abb , idregions , idact , idsec , idtech , idyear , applic_value ,
   source , table_name)
3 SELECT DISTINCT d.scen , a.path_abb , a.region ,
4   a.act_abb , a.sec_abb , a.tech_abb , a.year , a.applic ,

```

```

5  a.source , a.table_name
6  FROM scen_master sm
7  JOIN scenario_n d ON (
8    d.scen = sm.scen)
9  JOIN applic_collect_all a ON (
10   a.path_abb = d.path_abb
11   AND d.region = a.region
12  JOIN sec_mv s ON (
13    s.idsec = a.sec_abb
14    AND s.act_type = d.act_type)
15 WHERE sm.cost_curves = 1;

```

The temporary table “applic_set_data_imp” used for import of the data also stores the information from which of the single tables a given applicability was copied (columns “source” and “table_name”). After gathering all data for the relevant scenarios, the retrieved values have to be analyzed. Duplicate applicabilities for the same applicability set, region, abatement option and year can be eliminated without any further analysis. Inconsistencies in the values have to be analyzed and a decision made about which value should be kept.

Therefore the next step is to identify the multiple combinations as shown in listing 4.24. For each unique combination (set, region, act, sec, tech, and year) the minimum *ROWID*, as well as the amount of distinct and unique values is stored.

Listing 4.24: Finding multiple values for applicabilities

```

1  SELECT idapplic_sets , idregions , idact , idsec , idtech , idyear ,
2     MIN(ROWID) AS rowid_keep ,
3     COUNT(applic_value) AS all_values ,
4     COUNT(DISTINCT applic_value) AS d_values
5  FROM applic_set_data_imp
6  GROUP BY idapplic_sets , idregions , idact , idsec , idtech , idyear
7  HAVING COUNT(applic_value) > 1;

```

With the gathered information it is possible to remove the duplicate values and keep only the rows with the minimum *ROWID*. Since there is no primary or unique key on the table so far (and none can be created on the relevant columns), comparing the rows by the *ROWID* is the only possibility. Before duplicate values are deleted, as shown in listing 4.25, a backup of the not cleaned table is create so that it is possible to check the consistency of values after the cleaning process.

Listing 4.25: Removing duplicate values for applicabilities

```

1  DELETE FROM applic_set_data_imp a
2  WHERE EXISTS (
3    SELECT 1
4    FROM applic_set_data_dupl.tmp i
5    WHERE i.idapplic_sets = a.idapplic_sets

```

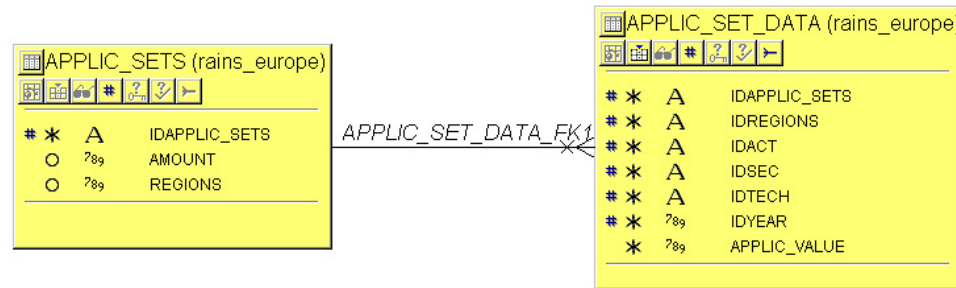



Figure 4.15: Tables holding applicability data

```

6 | AND i.idregions      = a.idregions
7 | AND i.idact          = a.idact
8 | AND i.idsec          = a.idsec
9 | AND i.idtech         = a.idtech
10 | AND i.idyear         = a.idyear
11 | AND i.d_values       = 1
12 | AND i.rowid_keep     != a.ROWID
13 | );

```

To delete multiple values, inconsistencies in the values have to be analyzed. Mainly it is of interest, from which tables shown in table 4.2 the inconsistent values have been copied. The analysis reveals that there are only inconsistencies between the applicabilities stored in tables “applic_cross” and “applfin”. Since the data in “applfin” is more reliable, the inconsistent values associated with “applic_cross” are deleted from the system.

Now it is possible to copy the applicabilities from the temporary table, to the “applic_set_data” table. As shown in figure 4.15 this table now has the same structure as the “constr_n” table discussed in section 2.3.4, except for the “idregions” column. As already discussed, the regional parameter for control strategies is retrieved through the scenario structure.

The table “applic_sets” holds all available applicability sets. The columns “amount” and “regions” are created just for temporary analysis purposes. They are be deleted after the analysis of applicabilities is finished.

Assuring Referential Integrity

In the next step, referential constraints have to be set to assure integrity that was initially provided upon collection of data as mentioned in section 4.3.4 also in

the future. Referential constraints are set to the mviews holding the available regions and implementation rate relevant activity/sector/technology combinations according to sections 4.3.2 and 4.3.3. In addition, a foreign key to the “years” table holding all available years is set.

Assuring Integrity with Implementation Rates

As mentioned in section 3.2.2 applicabilities are setting the maximum for corresponding implementation rates defined in the “constr_n” table. As stated in section 2.3.4 the region for which a control strategy is used, is not an attribute of the strategy itself, but is determined by the assignment of a control strategy to a region in a scenario.

Therefore the consistency between the applicabilities and implementation rates can only be checked on a scenario basis. To do a check for the current set of applicabilities and scenarios, the applicability sets have to be combined with the scenarios by their names (noting that the sets are currently named after the scenario they were collected from). In the future this check will be done by the relation between the sets and scenarios mentioned in section 4.3.4.

Listing 4.26 retrieves all combinations of applicabilities and implementation rates that do not match equation 3.1. Eliminating such inconsistencies cannot be done automatically because the decision whether the applicability value or the implementation rate should be changed, has to be made by experts who developed the applicabilities and the implementation rates that are causing the problems. Sometimes it will not even be possible to change specific values, but the creation of a new applicability set which does not cause any inconsistencies in terms of equation 3.1 will be necessary.

Listing 4.26: Analysis of sub and master applicability sets

```

1 SELECT s.scen, s.owner, d.region, c.con_strat, a.idapplic_sets,
2    a.idact, a.idsec, a.idtech, a.idyear,
3    a.applic_value, c.perc
4 FROM scen_master s
5    JOIN scenario_n d ON (
6        d.scen = s.scen)
7    JOIN applic_set_data a ON (
8        a.idapplic_sets = s.scen
9        AND a.idregions = d.region)
10    JOIN constr_n c ON (
11        c.con_strat = d.con_strat
12        AND c.act_abb = a.idact
13        AND c.sec_abb = a.idsec

```

```

14     AND c.tech_abb = a.idtech
15     AND c.year     = a.idyear)
16 WHERE c.perc > a.applic_value;

```

Removing Redundancies Between Sets

In the last step we want to analyze which of the retrieved applicability sets are redundant. As can be seen from the amount of defined applicabilities and regions, some of the sets seem to be identical. Also, some of the sets could be subsets of others that have more defined combinations. The SQL command to find the sub and master sets within the currently available applicability sets is shown in listing 4.27.

Listing 4.27: Analysis of sub and master applicability sets

```

1 CREATE TABLE applic_sets_master AS
2 SELECT sub.idapplic_sets AS idapplic_sets_sub ,
3        mast.idapplic_sets AS idapplic_sets_master ,
4        sub.amount AS amount_sub , mast.amount AS amount_master ,
5        sub.regions AS regions_sub , mast.regions AS regions_master
6 FROM applic_sets sub, applic_sets mast
7 WHERE ((
8     — find subsets
9     sub.amount          < mast.amount
10    AND sub.regions      < mast.regions)
11 OR (
12     — find equal sets
13     sub.amount          = mast.amount
14     AND sub.regions      = mast.regions
15     AND sub.idapplic_sets > mast.idapplic_sets
16 ))
17 AND NOT EXISTS (
18     — search for inconsistencies in values
19     SELECT 1
20     FROM applic_set_data d_sub
21     JOIN applic_set_data d_mast ON (
22         d_sub.idregions = d_mast.idregions
23         AND d_sub.idact  = d_mast.idact
24         AND d_sub.idsec  = d_mast.idsec
25         AND d_sub.idtech = d_mast.idtech
26         AND d_sub.idyear = d_mast.idyear)
27     WHERE d_sub.idapplic_sets = sub.idapplic_sets
28     AND d_mast.idapplic_sets = mast.idapplic_sets
29     AND d_sub.applic_value != d_mast.applic_value
30 ) AND NOT EXISTS (
31     — check for completeness of subset within master set
32     SELECT 1
33     FROM applic_set_data d_sub
34     WHERE d_sub.idapplic_sets = sub.idapplic_sets
35     AND NOT EXISTS (

```

```

36 SELECT 1
37 FROM applic_set_data d_mast
38 WHERE d_mast.idapplic_sets = mast.idapplic_sets
39 AND d_sub.idregions = d_mast.idregions
40 AND d_sub.idact = d_mast.idact
41 AND d_sub.idsec = d_mast.idsec
42 AND d_sub.idtech = d_mast.idtech
43 AND d_sub.idyear = d_mast.idyear
44 )
45 )
46 ORDER BY sub.idapplic_sets , mast.idapplic_sets ;

```

As shown in the *WHERE* clause of the master select, the condition for finding a subset of another set (called the master set) is split in two parts: The first part finds sets that are smaller in terms of amount of data than the master set; the second part finds all sets that are equal in terms of data and region amount, and classifies the one that is alphabetically larger as the subset. It would also be possible to remove the one which is alphabetically smaller, but due to the known naming conventions the first solution is more suitable.

In the first *NOT EXISTS* subselect (starting at line no. 10), we search for inconsistencies in the applicability values for the same region, activity, sector, technology, and year combination. If any inconsistencies can be found, a given applicability set cannot be the subset of another master set.

In the second *NOT EXISTS* subselect (starting at line no. 23) we check the completeness of the subset within the master set. Without this check it is possible that a given set is identified as a subset of another set only based on the fact that there are no inconsistencies in the applicability values. This does not mean that there cannot be applicabilities in the subset that do not exist in the master set.

By temporarily storing the results of the SQL command in the “applic_sets_master”, we can use the retrieved data to identify sets that can be removed because of redundancy: All sets that can be found in the temporary table, and are present in the subset column can be deleted from the system. The command to remove such sets is shown in listing 4.28.

Listing 4.28: Removing redundant applicability sets

```

1 DELETE FROM applic_sets s
2 WHERE EXISTS (
3   SELECT 1
4   FROM applic_sets_master m
5   WHERE m.idapplic_sets_sub = s.idapplic_sets
6 );

```

Using Applicability Sets

Since the relation between the applicabilities and the scenarios cannot be determined by the scenario structure after having unbound the data from the pathways and emission vectors, the relation has to be defined specifically. Since every scenario can only have one set, and this assignment is only needed for scenarios which are suitable for optimization and cost curves, a new column “applic_set” allowing *NULL* values within the “scen_master” table can be created.

If a set is given, the scenario can be used for these two purposes, otherwise not. In addition the possibility to add an applicability set can be set to an administrative privilege. Since the process of checking the relation between the applicabilities and implementation rates described in section 4.3.4 requires specific knowledge and data editing possibilities, this part of the model cannot be opened to average users anyway.

The previously mentioned check should also be embedded into the web interface so that it is possible to check consistency between the applicabilities and implementation rates for every scenario upon request. Since the possible inconsistencies change with every upload of affected control strategies as well as changes in the scenario structure, this check will have to be frequently performed for scenarios that are in the last phases of development.

Binding this check to certain actions done in the model (like the just mentioned implementation rate upload) is not recommended. Since checking the inconsistencies is only of interest beyond a certain development level of a scenario, constant notifications upon upload of data will be irritating to the users. Since the applicabilities are only needed for scenarios developed by experts from IIASA, triggering the check routine can be done manually when it is required.

Conclusions

We started with about 1.000.000 records in eight different tables. We did not reduce the amount of data significantly (only by about 20.000 records), but we managed to create 18 unique sets. It might be even possible to reduce the number of sets further, but this has to be done through manual inspection of the data by experts. These sets can be managed independently from the pathways and emission vectors. This is especially advantageous for pathways created by users that do not have the expertise to define applicabilities and thus should not have access to these values.

In addition we also assured consistency between the data used within one set. We have improved the referential integrity with other tables holding data that the applicabilities are referencing.

With the new structure it is also easily possible to maintain consistency between the applicabilities and implementation rates. This check would have had to be done for every single previously used table.

Another improvement based on the introduced changes is the exchange of applicabilities between the regional database schemas and the GAINS optimization module. Previously it was necessary to collect data from the single tables and analyze it according to the aspects discussed in this section. After introduction of the applicability sets applicabilities can be easily shared with the optimization module and the checks do not have to be done every time since each set is consistent within itself.

4.3.5 Aggregation of Results According to Reporting Standards

Collection of Data

For successfully solving the challenges discussed in section 3.2.3, the data distributed over the single pollutant-specific tables needs to be collected. Similar to the analysis of applicabilities mentioned in section 4.3.4 the data is copied from the single tables into one temporary table according to SQL commands similar to the one showed in listing 4.29 for NH₃.

Listing 4.29: Collecting SNAP & NFR aggregations for NH₃

```

1 INSERT INTO snap_nfr_import
2 SELECT 'NH3', act_abb, sec_abb, act_type,
3        snap1, snap1_name,
4        nfr_1, nfr_1_name, nfr_2, nfr_2_name
5 FROM rains_europe.asec_nh3_agg;
```

As can be seen from this command, the structure of the table is redundant in terms of defined SNAP and NFR codes: There is no reference to another table that would centrally store the available SNAP and NFR codes. All codes are listed multiple times with their identifier (columns “snap1”, “nfr_1” and “nfr_2”) and description (columns “snap1_name”, “nfr_1_name” and “nfr_2_name”).

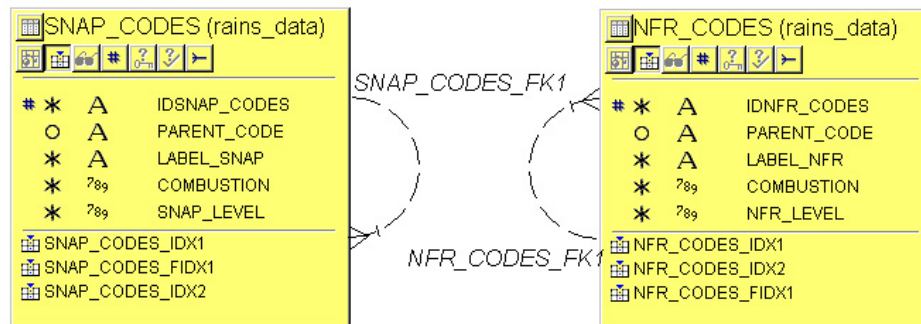


Figure 4.16: Tables holding definitions of SNAP and NFR codes

Data Definition of Reporting Standards

To get a unique listing of code identifiers and descriptions, separate tables for SNAP and NFR codes have to be created. An aspect that was not modeled before, but is very important to understand for the further steps, is the fact that both SNAP and NFR codes, are designed in a strictly hierarchical structure. This means that every node within the code tree has exactly one parent code (except the root node). As shown in figure 4.16 this fact is now also modeled in the structure of the database tables. Since the codes will be used for all available versions of the model, the tables are created in the “rains_data” schema.

Next, the tables have to be filled with data according to the official definition of reporting standards (Economic Commission for Europe, 2003). The data is available in files in comma separated value (CSV) format and can be imported into the database with the Oracle SQL*Loader (SQL*Loader, 2001).

In addition an artificial SNAP code “00” is introduced to create a single root node that does not exist in the official SNAP tree structure. All SNAP codes at level one (two digit identifiers “01” to “11”) reference this code in the tree structure (“parent_code” relation between the codes). Although the “parent_code” column can hold *NULL* values, the value is only allowed to be *NULL* for the root nodes of both trees. This requirement is assured by application logic.

The columns “snap_level” and “nfr_level” store the level of the given code within the code tree. This information is redundant because it could be retrieved recursively (similar to the PL/SQL function shown in listing 4.31) through the “parent_code” relation (assuming that the level of the root node equals zero). Since

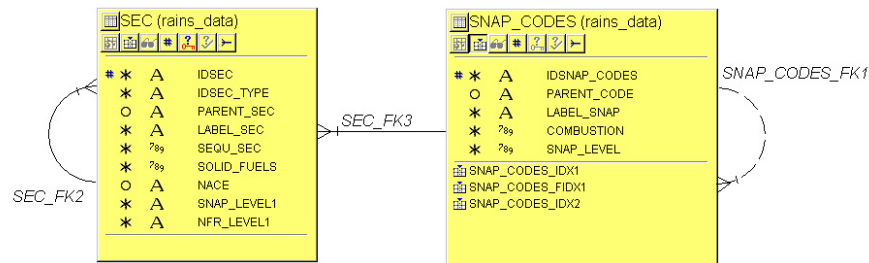


Figure 4.17: Relation between SNAP1 codes and GAINS sectors

the structure of the codes is static by definition, this would lead to unnecessary additional query runtime. Therefore it is better to store this information in database tables.

Assignment of GAINS Results to SNAP Codes

Based on the official list of SNAP codes, previously collected data can be analyzed. Already the initial visual analysis reveals, that the assignment of a SNAP code at level one (SNAP1) to a GAINS sector should be unique independently of the pollutant. Listing 4.30 retrieves all sectors and SNAP1 codes that do not follow this rule.

Listing 4.30: Check of assignment between GAINS sectors and SNAP1 codes

```

1 SELECT sec_abb, COUNT(DISTINCT snap1),
2    ROWS.TO_STRING(CAST(COLLECT(snap1) AS t_vvarchar2_column), ', ') AS snap1_codes
3 FROM snap_nfr_import
4 GROUP BY sec_abb
5 HAVING COUNT(DISTINCT snap1) > 1
6 ORDER BY sec_abb;
```

Since only a few mavericks are returned by this SQL command, they can be quickly corrected. Because the assignment of GAINS sectors is unique, the SNAP1 code can be stored within the “rains_data.sec” as shown in figure 4.17.

The modelling requirement of only assigning a SNAP1 code to each sector in the “snap_level1” code is assured by application logic.

Since the results of the GAINS models only have to be displayed aggregated by SNAP1 categories, the modelling is finished at this stage.

Assignment of GAINS Results to NFR Codes

The analysis of the “snap_nfr_import” table reveals that both the “nfr_1” and “nfr_2” columns hold values of NFR codes that cannot be found in the official listing of NFR codes. In addition, the digits in the names are not meant to indicate the first and second level of the NFR structure tree. Values of these columns are artificial aggregations of the official NFR codes: For example the artificial code “1.B.2.a,b” is used instead of a multiple assignment to codes “1.B.2.a” and “1.B.2.b”. Therefore it is necessary to first manually clean the assignment in table “snap_nfr_import” to get rid of the artificial codes. It is then necessary to dynamically compare the NFR codes among themselves, in particular their relation in terms of the “parent_code” relation. To be able to do this, a recursive PL/SQL function is created as shown in listing 4.31.

Listing 4.31: PL/SQL function to retrieve parent NFR code at a given level

```

1 FUNCTION get_nfr_parent_code (p_nfr_code IN VARCHAR2, p_at_level IN NUMBER)
2   RETURN VARCHAR2
3 IS
4   v_parent_code VARCHAR(15);
5   v_parent_level NUMBER(1);
6 BEGIN
7   — check the parameters
8   IF (p_nfr_code IS NULL
9     OR p_at_level IS NULL
10    OR p_at_level < 0) THEN
11     RETURN NULL;
12   END IF;
13
14   — get the parent code of the given code and its level
15   SELECT idnfr_codes, nfr_level INTO v_parent_code, v_parent_level
16   FROM rains_data.nfr_codes p
17   WHERE EXISTS (
18     SELECT c.parent_code
19     FROM rains_data.nfr_codes c
20     WHERE c.parent_code = p.idnfr_codes
21     AND c.idnfr_codes = p_nfr_code
22   );
23
24   — check whether the parent code could be found
25   IF (v_parent_code IS NULL) THEN
26     RETURN NULL;
27   END IF;
28
29   — check whether the found parent level is smaller than the requested level
30   IF (p_at_level > v_parent_level) THEN
31     — the parent at the requested level cannot be found any more
32     RETURN NULL;
33   ELSIF (p_at_level = v_parent_level) THEN

```

```

34  — the parent code at the requested level was found
35  RETURN v_parent_code;
36  ELSE
37  — search recursively
38  RETURN get_nfr_parent_code(v_parent_code , p_at_level);
39  END IF;
40 END get_nfr_parent_code;

```

Based on this PL/SQL function, the relation between the NFR codes and GAINS sectors can be examined as shown in listing 4.32.

Listing 4.32: Check of assignment between GAINS sectors and NFR1 codes

```

1 SELECT sec_abb , COUNT(DISTINCT GET_NFR.PARENT.CODE(nfr_1 , 1)) ,
2    ROWS.TO_STRING(CAST(Collect(Collect(GET_NFR.PARENT.CODE(nfr_1 , 1)) AS t_varchar2_column
3    ), ', ')) AS nfr1_codes
4 FROM snap_nfr_import
5 GROUP BY sec_abb
6 HAVING COUNT(DISTINCT GET_NFR.PARENT.CODE(nfr_1 , 1)) > 1
7 ORDER BY sec_abb;

```

Again a few mavericks in the data stored in “snap_nfr_import” have to be removed, but overall the relation between a sector and an NFR1 code can be seen as unique. Therefore also the NFR1 code can be added to the “sec” table as an attribute of each available GAINS sector (shown as constraint “sec_fk4” in figure 4.18). The same consideration as to the “snap_level1” column is applied to the “nfr_level1” column.

Other than the SNAP aggregation that is only needed at level one, the display of aggregated results by NFR standard has to be more precise. Therefore it is necessary to assign not only NFR1 codes to GAINS sectors, but also more detailed NFR codes.

Further analysis of the values gathered in the “snap_nfr_import” table reveals that this assignment can only be unique if it is not only done by sector, but also by pollutant. In addition, the assignment is not unique, but multiple assignments of NFR codes to a sector/pollutant combination are possible. The database structure needed to hold this relation is shown in figure 4.18.

As can be seen from the database structure, the relation between the “sec” and “nfr_codes” is modeled redundantly. This redundancy is introduced to guarantee completeness of the assignment. As already discussed in section 3.2.3, one of the goals of the new modelling approach was to guarantee that no data is “lost” during the aggregation process. This can only happen, if some parts of the displayed results cannot be assigned to a NFR category upon display.

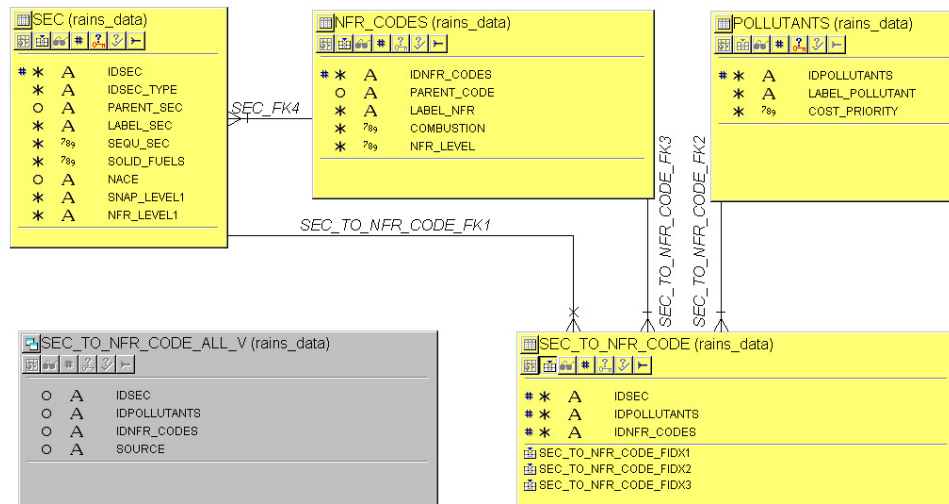


Figure 4.18: Relation between NFR codes, GAINS sectors and pollutants

Each time a new sector is introduced into the database, a second step, namely the assignment of the sector to the available NFR codes, has to be done for each pollutant. This assignment requires special knowledge and should be done by more than one person. Since it is possible to assign an NFR1 code uniquely for each sector upon creation of the sector, at least a very rough aggregation can be guaranteed for each available sector and no data is “lost” upon display of the results. As shown in listing 4.33 and figure 4.18 the view “sec_to_nfr_code_all_v” is created in which the complete relation can always be found.

Listing 4.33: View guaranteeing the completeness of the GAINS sector to NFR code relation

```

1 CREATE OR REPLACE VIEW sec_to_nfr_code_all_v AS
2   — all assigned sector/pollutant/nfr combinations
3 SELECT idsec, idpollutants, idnfr_codes, 'assigned' AS source
4 FROM sec_to_nfr_code
5 UNION ALL
6 — filled up by all needed combinations,
7 SELECT s.idsec, p.idpollutants, s.nfr_level1 AS idnfr_codes, 'filled' AS source
8 FROM sec s, pollutants p
9 WHERE NOT EXISTS (
10   — which cannot be found among the defined combinations
11   SELECT 1
12   FROM sec_to_nfr_code stnc
13   WHERE stnc.idsec = s.idsec
14   AND stnc.idpollutants = p.idpollutants

```

15);

Another benefit of assigning the NFR1 to each sector is the fact that the values that can be assigned on the detailed level for each pollutant can be filtered to be only the subtree of the already assigned NFR1 code. This way not only the consistency between the “sec_fk4” and “sec_to_nfr_code_fk1” relation is guaranteed, but also the assignment is completed more easily and faster since only a fraction of the codes can be chosen to be assigned through the graphical user interface.

Display of Results on Detailed NFR Level

The display of results aggregated by SNAP1 and NFR1 is not a big challenge. Since every GAINS sector can be uniquely assigned to one code, the results of the aggregation can be retrieved through a simple *JOIN* and *GROUP BY* statement.

As can be seen from figure 4.18 assignment of the GAINS sector to NFR codes on more detailed levels is not unique anymore. For example to provide information on which part of the calculated emissions belongs to a given NFR code, it would be necessary to provide fractions along with every sector/code/pollutant combination. To correctly model such fractions, they would also have to be region and year specific. Although such an assignment would theoretically be possible, it is not maintainable in practice and therefore not implemented.

Consequently the grouping of data retrieved for displaying the aggregated results cannot aim at single NFR codes, but has to aim at unique groups of these codes. Listing 4.34 shows the command to aggregate the results of CH₄ emissions for scenario “NEC_NAT_CLEV4” in Austria (“AUST_WHOL”) by the assigned NFR codes.

Listing 4.34: Emissions aggregated by NFR

```

1 SELECT nfr_code , year , SUM(emiss) AS emiss
2 FROM (
3   SELECT e.sec_abb , year ,
4     RAINS_DATA.ROWS_TO_STRING(
5       CAST(COLLECT(stnc.idnfr_codes) AS rains_data.t_varchar2_column) ,
6       ', or<br/>') AS nfr_code ,
7     SUM(e.emiss)/COUNT(DISTINCT stnc.idnfr_codes) AS emiss
8   FROM emiss.CH4_agg e
9   JOIN rains_data.sec_to_nfr_code_all_v stnc ON (
10    e.sec_abb = stnc.idsec )
11   JOIN rains_data.nfr_codes nc ON (
12    nc.idnfr_codes = stnc.idnfr_codes )
13   WHERE scen      = 'NEC_NAT_CLEV4'
14   AND idpollutants = 'CH4'

```

```

15 | AND region      = 'AUST.WHOL'
16 | GROUP BY e.sec_abb, e.year
17 | )
18 | GROUP BY nfr_code, year
19 | ORDER BY nfr_code, year;

```

What can be deduced from the SQL command and the underlying structure (shown in figure 4.18), is that one NFR code can be displayed multiple times in different groups of codes as shown in figure 4.19. As can be seen, 4.59 kt of emissions in year 1990 cannot be clearly divided into codes “1.B.2.a” and “1.B.2.b”, whereas 25.02 kt can be uniquely assigned to NFR sector “1.B.2.b”. For the analysis of the shown aggregated emissions in year 1990 the minimum and maximum emissions can be deduced as shown in the following table.

NFR sector	Min. Emissions	Max. Emissions
1.B.2.a	0 kt	4.59 kt
1.B.2.b	25.02 kt	29.61 kt

In the previous version of the modelling approach such a display combination was named “1.B.2.a,b” and the emissions were said to be 29.61 kt. The new approach is also not able to provide exact information for all single NFR sectors, but it shows more details about the aggregations where possible.

Conclusions

We have managed to identify the needed aggregations of results as being dependent only on the GAINS sectors and pollutants. By storing the relation between the sectors and codes at level one in the central “rains_data.sec” table, we have not only removed the redundancy in previously stored data, but also guaranteed completeness of the relation. Thus we can now guarantee that no data is “lost” when aggregating results according to the discussed standards.

In addition, we have minimized the maintenance effort which used to be much higher and led to inconsistencies in data when the aggregation information was stored in the regional database schemas. Therefore it is important to note that this solution is not only scientifically correct, but also maintainable. To provide a solution that would exactly allocate, for example, a given amount of emissions to NFR sectors on a detailed level, more detailed information would have to be provided by the model.

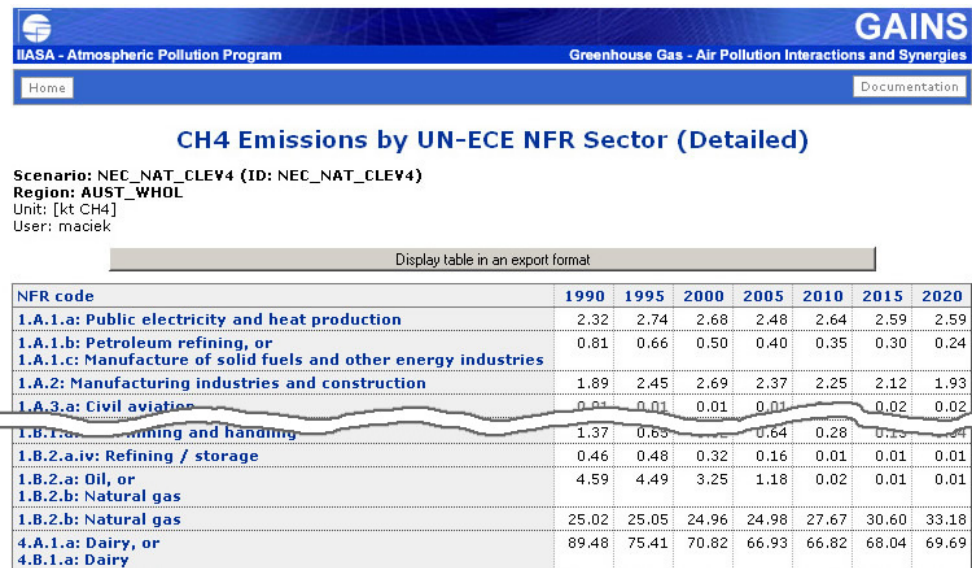


Figure 4.19: Screen shot of results of CH₄ emission calculation aggregated by NFR

4.4 Other Implementation Aspects

4.4.1 Calculation of Emissions

Due to the storage of emission vector related data in the set of tables mentioned in section 4.1 and the fact that for some pollutants abated and for other unabated emission factors were used, the emission calculation had to be done in a separate view for each pollutant. By using only the abated emission factors and introducing a new table “emiss_factors_abtd”, the emission calculation can be based on the same view for all pollutants. The *CREATE* statement for this view is shown in listing 4.35. This view does not show detailed emissions by technology as they are not needed for most user requests: The ten most frequently used functionalities of the model do not display results by technology. For detailed technology-specific emissions another view has to be created. This view will not be discussed in this paper.

Listing 4.35: View for emission calculation

```

1 CREATE OR REPLACE VIEW emiss_all_agg AS
2 SELECT scen, path_abb, idregions, year,
3        idemiss.vectors, idpollutant-fractions, pollutant,

```

```

4 | idact, idsec, act_control, sec_control, unit, act_type,
5 | activity, factor_noc, factor_rem_abtd, (factor_noc - factor_rem_abtd) AS
   | factor_impl,
6 | — calculate emissions from the implied emission factor
7 | activity*(factor_noc - factor_rem_abtd) AS emiss
8 | FROM (
9 |   SELECT s.scen, s.path_abb, e.idregions, y.year,
10 |    e.idemiss_vectors, e.idpollutant_fractions, e.pollutant,
11 |    e.idact, e.idsec, e.act_type,
12 |    NVL(ac.act_control, e.idact) AS act_control,
13 |    NVL(ac.sec_control, e.idsec) AS sec_control,
14 |    (SELECT unit
15 |     FROM rains_data.act_sec_all u
16 |     WHERE e.idact = u.idact
17 |     AND e.idsec = u.idsec) AS unit,
18 |    (SELECT activity
19 |     — transform activities stored in actpath_n with view actpath_trans
20 |     FROM actpath_trans a
21 |     WHERE s.path_abb = a.path_abb
22 |     AND s.region = a.region
23 |     AND e.idact = a.act_abb
24 |     AND e.idsec = a.sec_abb
25 |     AND e.act_type = a.act_type
26 |     AND y.year = a.year) AS activity,
27 |    e.factor_abtd AS factor_noc,
28 |    NVL(
29 |     — calculate the control strategy weighted sum of emission
30 |     — factors for given activity/sector combination
31 |     (SELECT SUM(perc*(e.factor_abtd - r.factor_abtd))/100.
32 |      FROM constr_n c
33 |      JOIN emiss_factors_abtd r ON (
34 |        r.idtech = c.tech_abb)
35 |      WHERE c.con_strat = s.con_strat
36 |      AND r.idemiss_vectors = s.emv_owner
37 |      AND r.idregions = s.region
38 |      AND y.year = c.year
39 |      AND c.act_abb = NVL(ac.act_control, e.idact)
40 |      AND c.sec_abb = NVL(ac.sec_control, e.idsec)
41 |      AND r.idpollutant_fractions = e.idpollutant_fractions
42 |      AND r.idact = e.idact
43 |      AND r.idsec = e.idsec), 0.) AS factor_rem_abtd
44 | FROM scenario_n s
45 |   JOIN scen_master_year y ON (
46 |     s.scen = y.scen)
47 |   JOIN emiss_factors_abtd e ON (
48 |     s.emv_owner = e.idemiss_vectors
49 |     AND s.act_type = e.act_type
50 |     AND s.region = e.idregions)
51 |   LEFT JOIN rains_data.act_sec_trans_control ac ON (
52 |     ac.idact = e.idact
53 |     AND ac.idsec = e.idsec)
54 | — get only the unabated emission factors
55 | WHERE e.idtech = 'NOC'
56 | )

```

57 | **WHERE** activity > 0;

The emission calculation using the NOC implementation rate is proportionally seen very time consuming, as can be seen from table 4.3 (column “old style”). The above mentioned query avoids this time consuming step by using the mean control strategy weighted emission factor, also called the “implied emission factor” (Wagner et al., 2007). By using this factor, the emission calculation shown in equation 2.1 can be reformulated as shown in equation 4.2.

$$E_{p,r,y} = \sum_{a,s} E_{p,r,a,s,y} = \sum_{a,s} A_{r,a,s,y} \cdot ef_{p,r,a,s,y}^{implied} \quad (4.2)$$

The implied emission factor can be calculated for a given GAINS activity/sector combination as shown in equation 4.3. The formula was reformulated using the relations already mentioned in equation 2.4.

$$\begin{aligned} ef_{p,r,a,s,y}^{implied} &= \sum_t X_{r,a,s,t,y} \cdot ef_{p,r,a,s,t}^{abated} / 100 \\ &= \sum_t X_{r,a,s,t,y} \cdot ef_{p,r,a,s,t=NOC}^{abated} / 100 \\ &\quad - \sum_t X_{r,a,s,t,y} \cdot (ef_{p,r,a,s,t=NOC}^{abated} - ef_{p,r,a,s,t}^{abated}) / 100 \\ &= ef_{p,r,a,s,t=NOC}^{abated} \\ &\quad - \sum_{t \notin NOC} X_{r,a,s,t,y} \cdot (ef_{p,r,a,s,t=NOC}^{abated} - ef_{p,r,a,s,t}^{abated}) / 100 \end{aligned} \quad (4.3)$$

The emission calculation based on the implied emission factor (column “single” in table 4.3) is much faster than the previously used emission calculation (column “old style”). The reason for this is the lack of time-consuming full table scans and outer joins in the execution plan of the underlying queries when using the implied emission factor.

What is even more important to see from this step, is the impact that this pollutant independent view has on the whole model and its interface: For the first time it is possible to calculate emissions of the model for all pollutants in exactly the same way. As can be seen from the SQL command in listing 4.36 due to the structure of the new view “emiss_all_agg”, the pollutant becomes a parameter within the query and can be used in the same way as the scenario and the region.

Region(s)	Pollutant(s)	“old style”	“single”	“multi”
Austria	CH ₄	13.40	0.39 (2.88%)	0.39 (2.88%)
EU27	CH ₄	84.45	8.54 (10.12%)	8.61 (10.20%)
Austria	CH ₄ , NO _x , SO ₂	30.85	0.87 (2.81%)	0.92 (2.99%)
EU27	CH ₄ , NO _x , SO ₂	129.86	19.41 (14.95%)	21.34 (16.44%)

Table 4.3: Query runtime comparison between emission calculation based the implied emission factor for single pollutant views and the multi-pollutant view “emiss_all_agg”, as well as previously used “old style” emission calculation (all times in seconds)

Listing 4.36: Pollutant independent emission calculation

```

1 SELECT e.idregions, e.year, e.pollutant, SUM(e.emiss) AS emiss
2 FROM emiss_all_agg e
3 WHERE e.scen = 'NEC_NAT_CLEV4'
4 AND e.idregions = 'AUST_WHOL'
5 AND EXISTS (
6   SELECT 1
7   FROM rains_data.pollutant_to_group pg
8   WHERE pg.idpollutants = e.pollutant
9   AND pg.idpollutant_groups = 'GHG'
10 )
11 GROUP BY e.idregions, e.year, e.pollutant
12 ORDER BY e.idregions, e.year, e.pollutant;

```

Therefore it is now easily possible to calculate, for example, the emissions of any group of pollutants. In this case the emissions of all greenhouse gases (CH₄, CO₂, N₂O, and FGAS) are retrieved by the definition of the group in table “rains_data.pollutant_to_group”. Previously it was necessary to calculate the emissions of all greenhouse gases in four different steps and combine them in one view. It would have also been necessary to introduce additional database structures, to show emissions for any other requests in terms of a new pollutant group. The new view saves redundancy in database structures and maintenance efforts.

For the sake of completeness it also has to be mentioned that the emissions are calculated in different units for different pollutants. For example to sum up the emissions for greenhouse gases, the emissions caused by single pollutants have to be converted into one unit. This unit is chosen to be kilotons of CO₂ equivalents emissions.

Table 4.3 shows that calculations based on the new multi-pollutant view (column “multi”) are up to two seconds and 10% slower for some of the calculation re-

quests for the previously mentioned scenario “NEC_NAT_CLEV4” than the same improved calculation approach with a view for each pollutant (column “single”). Nevertheless, this time loss is acceptable because of all the previously mentioned benefits.

To prevent possible differences in caching strategies, either of the database or the SQL client itself, the times were measured as the average of five executions of each query. The queries for retrieving data for multiple pollutants do not sum up and convert the emissions to a unique unit, but only retrieve the data emissions of the single pollutants.

4.4.2 Initialization of Partial Emission Vectors

As already mentioned in section 3.2.1, it is important to provide an initialization functionality that can also handle emission vectors that are not complete in terms of the underlying data. As discussed in section 4.3.3, each vector that is not complete, has to have a parent vector that is complete.

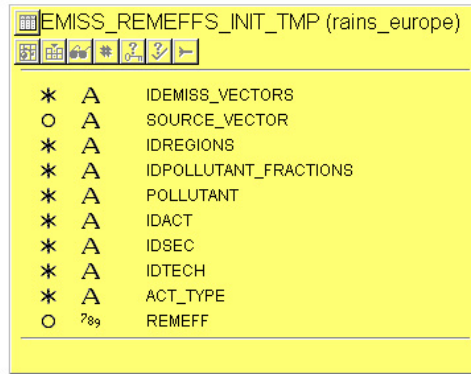
Additional Database Structures

For the initialization approach we will use temporary tables discussed in section 4.2.2. As shown in figure 4.20 a temporary table is created to hold the values retrieved during the initialization process. The structure of the table is basically identical with the structure of the “emiss_remeffs” table (figure 4.13), except for the additional column “source_vector” and the allowed *NULL* values in column “remeffs”. Similar tables can be created for the initialization of abated and unabated emission factors, as well as cost factors.

A temporary table is created with the *ON COMMIT DELETE ROWS* option. Since the initialization is done within one database transaction, it is not necessary to preserve the values in the table. Should it be necessary to hold values over various transactions, the concept of a temporary table would have to be abandoned. As already mentioned in section 4.2.2 data cannot be preserved in temporary tables because of the used database connection pooling.

Initialization of the Process

The initialization process is performed for a selected emission vector and a pollutant. In the initial step of the process, all possible combinations of needed removal



EMISS_REMEFFS_INIT_TMP (rains_europe)	
*	A IDEMISS_VECTORS
○	A SOURCE_VECTOR
*	A IDREGIONS
*	A IDPOLLUTANT_FRACTIONS
*	A POLLUTANT
*	A IDACT
*	A IDSEC
*	A IDTECH
*	A ACT_TYPE
○	789 REMEFF

Figure 4.20: Temporary database table for initialization of removal efficiencies

efficiencies are retrieved and copied into the temporary table as shown in listing 4.37. In the selected example the partial user-specific emission vector “maciek” backed up by values from the complete vector “NEC04” is initialized for Austria.

Listing 4.37: Retrieving all abatement options for removal efficiencies for NO_x

```

1 INSERT INTO emiss_remeffs_init_tmp
2 (idemiss_vectors, idregions,
3  idpollutant_fractions, pollutant,
4  idact, idsec, idtech, act_type)
5 SELECT 'maciek' AS idemiss_vectors, 'AUST.WHOL' AS idregions,
6        p.idpollutant_fractions, p.pollutant,
7        ast.idact, ast.idsec, ast.idtech, s.act_type
8 FROM act_sec_tech_emiss_mv ast
9      JOIN pollutant_fractions_mv p ON (
10         p.pollutant = ast.idpollutants)
11      JOIN sec_mv s ON (
12         s.idsec = ast.idsec)
13 WHERE ast.idpollutants = 'NOX'
14 AND ast.idtech != 'NOC';

```

As can be seen from the SQL command the columns “source_vector” and “re-meff” are not mentioned and therefore they are initialized with *NULL* values. Values of these columns will be set in later steps of the process.

Calculation of Values

After having gathered all necessary combinations of removal efficiencies, various steps of the initialization process have to be executed to gather the needed removal efficiency for each of the implementation options. An example of a step of this

process is shown in listing 4.38. This step sets the initial constant values for the removal efficiencies. The values are retrieved from table “cotecnox_v” with a primary key on columns “versions” and “tech_abb”.

Listing 4.38: Initialization step setting initial constant removal efficiencies for NO_x

```

1 UPDATE emiss_remeffs_init_tmp e
2 SET (source_vector, remeff) = (
3     SELECT c.version, c.eff_const
4     — value for selected emission vector
5     — (1) idemiss_vectors
6     — (2) parent vector
7 FROM cotecnox_v c
8 WHERE c.version = 'maciek' — (2) 'NEC04'
9 AND c.tech_abb = e.idtech
10 )
11 WHERE e.remeff IS NULL
12 AND e.idregions = 'AUST.WHOL';

```

The query is executed two times: In the first run the query attempts to retrieve the values from the partial vector “maciek”. In the second run all values that were not set before (are still *NULL*, line no. 11) are retrieved and set from the parent vector “NEC04”. The information from which emission vector the data was retrieved is written into the “source_vector” column.

Other initialization steps and queries have to be rewritten according to the same idea.

Checking of Calculated Values

Before the values can be copied from the temporary table to the “emiss_remeffs” table, it has to be checked whether all combinations have been initialized as shown in listing 4.39.

Listing 4.39: Checking of initialized values

```

1 SELECT *
2 FROM emiss_remeffs_init_tmp
3 WHERE remeff IS NULL
4 OR source_vector IS NULL;

```

The check is fairly simple: The SQL command tests only whether all needed combinations have been assigned a removal efficiency during the initialization process. In addition also checks whether the information about the source vector of the removal efficiency value was documented properly are performed.

If any rows are returned by this query, the initialization process can already be stopped at this stage and the failing combinations presented to the user of the web interface or reported as an exception to the development team. Additionally a logic that based on the privileges of the user can be implemented to make this decision.

Without an introduction of the temporary table it would not have been possible to store *NULL* values directly for the value of the removal efficiency. Therefore checking the correctness by the completeness of the initialization process would only have been possible after writing the values into the “emiss_remeffs” table.

Storing Values Persistently

After assuring the completeness of the data, the values can be copied to the “emiss_remeffs” table. The values are copied in two steps: First, all implementation options that are not available in the current working set are inserted as shown in listing 4.40.

Listing 4.40: Inserting of initialized values for missing abatement options

```

1 INSERT INTO emiss_remeffs
2 SELECT idemiss_vectors, idregions, idpollutant_fractions, pollutant,
3        idact, idsec, idtech, act_type, remeff
4 FROM emiss_remeffs_init_tmp tmp
5 WHERE NOT EXISTS (
6     SELECT 1
7     FROM emiss_remeffs r
8     WHERE r.idemiss_vectors = tmp.idemiss_vectors
9     AND r.idregions = tmp.idregions
10    AND r.idpollutant_fractions = tmp.idpollutant_fractions
11    AND r.idact = tmp.idact
12    AND r.idsec = tmp.idsec
13    AND r.idtech = tmp.idtech
14 );

```

This step guarantees the completeness of the initialized emission vector for the selected region and pollutant fraction. A *DELETE* command is not necessary because any disallowed combinations are removed automatically by cascading constraints if needed.

The second step sets the values in the “emiss_remeffs” table by overwriting them with values retrieved in the initialization process. As shown in listing 4.41 only the values that are different from the ones already stored are updated.

Listing 4.41: Updating of necessary abatement options with initialized values

```

1 UPDATE emiss_remeffs r SET
2 remeff = (
3   — get the value from the temporary table
4   SELECT remeff
5   FROM emiss_remeffs_init_tmp tmp
6   WHERE r.idemiss_vectors = tmp.idemiss_vectors
7   AND r.idregions = tmp.idregions
8   AND r.idpollutant_fractions = tmp.idpollutant_fractions
9   AND r.idact = tmp.idact
10  AND r.idsec = tmp.idsec
11  AND r.idtech = tmp.idtech
12 ) WHERE EXISTS (
13   — check which values are different and have to be updated
14   SELECT 1
15   FROM emiss_remeffs_init_tmp tmp
16   WHERE r.idemiss_vectors = tmp.idemiss_vectors
17   AND r.idregions = tmp.idregions
18   AND r.idpollutant_fractions = tmp.idpollutant_fractions
19   AND r.idact = tmp.idact
20   AND r.idsec = tmp.idsec
21   AND r.idtech = tmp.idtech
22   AND r.remeff != tmp.remeff
23 );

```

It is not necessary to give further details on the updated emission vector, region and pollutant fraction since the temporary table can only hold the set of removal efficiencies that was inserted by the SQL command in listing 4.37. By committing the updates in the “emiss_remeffs” table, the values in the “emiss_remeffs_init_tmp” table are automatically deleted.

Updating only the changed removal efficiencies and not all of them, is not only faster, but also enables the user running the initialization process, to view exactly how many values were updated and inserted during the process.

Conclusions

The most important success of the new initialization process is the fact that it is now possible to initialize not only complete vectors, but also partial vectors. Furthermore we have assured the completeness of the part of an emission vector that is initialized by filling up necessary abatement options that were not present before.

By choosing a temporary table as the stage for the initialization process, we have to lock the target table of the initialization process (“emiss_remeffs” in the mentioned examples) only for a much shorter time. The process can also be com-

pleted faster because only the initialized subset of abatement options is stored and manipulated in the temporary table.

In addition, only the values that have changed, have to be copied from the temporary table to the target table. This does not only save time, but also increases the usability of the process by providing exact information about which values have been changed during the last initialization run.

4.4.3 Display of Scenarios

In theory, each GAINS scenario should be available for all defined pollutants and regions. However as already mentioned in section 4.3.3 practice has shown that it is often only possible to provide fractions of data, either in terms of the available regions, pollutants or both.

To reflect this change in requirements, additional data has to be provided. Based on this data we have modified the display logic of the interface to fit the requirements.

Analysis of Scenario-To-Pollutant Relation

The analysis of the completeness of emission vectors can be extended by the information about scenario structures to conclude which scenarios can be displayed for a given pollutant. An analysis by manual visual comparison of the emissions and costs of currently about 150 scenarios for all nine pollutants would take very long. Since an overview of the available emission vector data can be presented very fast, this laborious process can be skipped.

To provide the data for the overview, first a view is created to gather the needed data. The command creating this view is shown in listing 4.42. It joins all available pollutant fractions with the scenarios as defined in the scenario structure (“scenario_n”) and the analysis of the emission vector data retrieved by the “emiss_factors_abtd_by_region” view (described in listing 4.18).

Listing 4.42: Data for overview of scenario completeness

```

1 CREATE OR REPLACE VIEW scenario_by_emivec AS
2 SELECT s.scen, s.owner AS scenario_owner, m.share_public AS public_scenario,
3        s.region, s.emv_owner, p.pollutant,
4        f.amount_def, f.amount_needed,
5        (f.amount_def - f.amount_needed) AS diff_amount,
6        ROUND(ABS((f.amount_def - f.amount_needed)/f.amount_needed)*100, 2) AS
        diff_perc

```

```

7 FROM rains_data.pollutant_fractions p
8 JOIN scenario_n s ON (1 = 1)
9 JOIN scen_master m ON (
10     m.scen = s.scen)
11 LEFT JOIN emiss_factors_abtd_by_region f ON (
12     f.idpollutant_fractions = p.idpollutant_fractions
13     AND f.idemiss_vectors = s.emv_owner
14     AND f.idregions = s.region);

```

Based on this view the data can be further conveniently filtered. Listing 4.43 shows a few basic filtering options that allow a manual visual check of the retrieved data.

Listing 4.43: Analysis of overview of scenario completeness

```

1 SELECT DISTINCT scen, scenario_owner, public_scenario,
2     region, emv_owner, pollutant,
3     amount_def, amount_needed, diff_amount, diff_perc
4 FROM scenario_by_emivec
5 —WHERE NVL(diff_perc, 100) != 0 — incomplete combinations
6 —AND public_scenario = 1 — public scenarios
7 —AND pollutant = 'SO2' — certain pollutant
8 —AND emv_owner = 'FCCC2000' — certain emission vector
9 ORDER BY 1, 4, 6;

```

After checking the analysis on correctness, the data has to be presented to the experts developing the scenarios. Only they can decide which scenarios are meant to be shown for which pollutants. Since most of the scientists are not database experts, it is not convenient for them to view the data directly through a database querying tool and check the data as shown in the previous listing.

Therefore the data has to be exported and presented in an external program. A convenient solution for this requirement is offered by Microsoft Excel. The complete data set retrieved by the SQL command shown in listing 4.43 is exported to a worksheet within a Microsoft Excel workbook. Based on this data a pivot table (Collins, 2006) can be created as shown in figure 4.21. In the pivot table view shown in this figure, the table is filtered to show only data for a given emission vector, a scenario owner, and a set of regions and pollutants.

By giving the scientists the possibility to easily filter the data and allowing flexible views on the data, conclusions can be drawn faster even though the amount of data is very large. For example it is easily visible that in figure 4.21 data for Moldovia (“MOLD_WHOL”) is missing in the “NEC05” emission vector and FGAS. Such a maverick can quickly be corrected.

Another pivot table can be created to show an overview of the average availability of pollutant specific emission factors over all regions of a scenario. As

Average of DEF %	OWNER	PUBLIC	REGION	R_PRIV	EMIVEC	POLL	CH4	CO2	FGAS	N2O
SCEN										
NAT_CLE_CEIL_EU_V5	janusz		0 AUST WHOL	viewer	NEC05		100.00	100.00	100.00	100.00
			GERM WHOL	viewer	NEC05		100.00	100.00	100.00	100.00
			MOLD WHOL	user	NEC05		100.00	100.00	0.00	100.00
			POLA WHOL	viewer	NEC05		100.00	100.00	100.00	100.00
NAT_CLE_CEIL_EU_V5 Total							100.00	100.00	75.00	100.00
NAT2006_CLEip_CEIL_EU_V5	janusz		0 AUST WHOL	viewer	NEC05		100.00	100.00	100.00	100.00
			GERM WHOL	viewer	NEC05		100.00	100.00	100.00	100.00
			MOLD WHOL	user	NEC05		100.00	100.00	0.00	100.00
			POLA WHOL	viewer	NEC05		100.00	100.00	100.00	100.00

Figure 4.21: Overview of available emission vector data by pollutant and region

Average of DEF %	POLL	CH4	CO2	FGAS	N2O	NH3	NOX	PM	SO2	VOC
SCEN										
NEC NAT_CLE4REV		0.00	78.84	0.00	0.00	85.09	98.56	98.83	97.52	98.09
NEC NAT_CLEV4		100.00	79.34	97.62	97.71	85.12	98.59	99.52	97.58	98.06
NEC NAT_EU_V HDV_V4		100.00	79.34	97.62	97.71	85.12	98.59	99.52	97.58	98.06
NEC NAT_OPT2		0.00	78.84	0.00	0.00	85.09	98.56	98.83	97.52	98.09
NEC PRIMES20_CLEV2		0.00	78.84	0.00	0.00	85.09	98.56	98.83	97.52	98.09
NEC PRIMES20_OPT2		0.00	78.84	0.00	0.00	85.09	98.56	98.83	97.52	98.09
NEC PRIMES90_CLEV2		0.00	78.84	0.00	0.00	85.09	98.56	98.83	97.52	98.09
NEC PRIMES90_OPT2		0.00	78.84	0.00	0.00	85.09	98.56	98.83	97.52	98.09
NEC PRIMESCOH_EU_V HDV_V4		100.00	97.92	97.62	100.00	86.39	100.00	99.62	99.89	98.97
Grand Total		33.33	81.07	32.54	32.82	85.24	98.73	99.07	97.80	98.18

Figure 4.22: Overview of available vector data by pollutant

shown in figure 4.22 the conclusions for the display of the scenarios shown in the given pivot table view can be drawn fairly fast: For scenario/pollutant combinations where the average of available emission factors per region is 100% or close to it, the scenarios will be displayed. For availability rates close to 0% the scenarios should not be displayed. Rates between 25% and 75% have to be explored in more detail, to see whether only a few regions are incomplete or complete, or whether the emission vectors used in this case have a systematic problem independent of the region.

The conclusion of this process is a matrix that flags a scenario/pollutant combination with 1 if the scenario shall be shown for the pollutant, and with 0 otherwise.

Assignment of Scenarios to Pollutants

Based on the results of the analysis done in the previous section, an assignment of scenarios to pollutants can be done in the database as shown in figure 4.23.

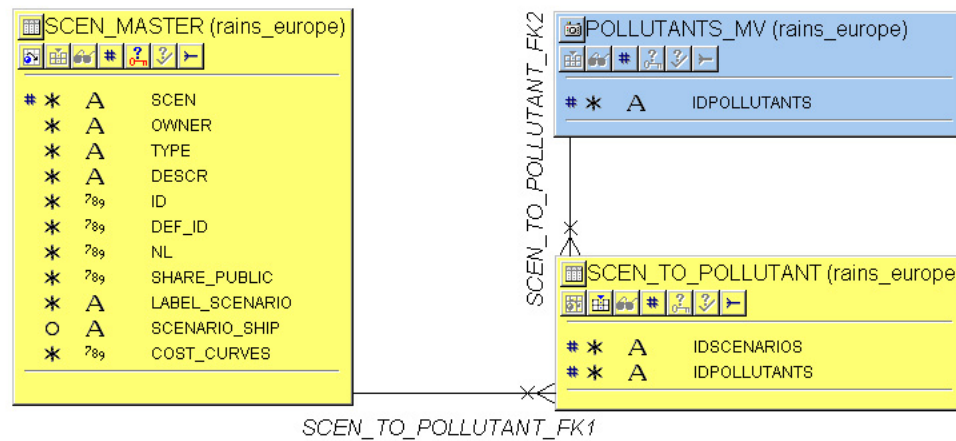


Figure 4.23: Database structure for assignment between scenarios and pollutants

Assignment of Scenarios to Regions

According to the assignment of scenarios to pollutants, also an assignment of scenarios to regions for which the scenarios should be displayed can be created and stored in table “scen.to.region”. It may not be possible to use scenarios that are complete in terms of the amount of regions for all functionalities of the application (for example for map display where scenarios have to be complete), but this option may be interesting for other aspects.

Experience has shown that creating emission vectors only for a given group of countries (for example European Union member countries) or even single countries is requested. To display results of such use cases, scenarios have to be created that use these emission vectors. Such scenarios had to be “filled up” for the rest of the GAINS regions available within the model to guarantee completeness in terms of regions.

Since information about the availability of regions for a given scenario is now existing, it is not necessary to define a scenario for all regions. To correctly display only the defined regions, the web interface of the model has to be developed in a way that allows only the selection of available regions for a given scenario and not all of them.

User Access to Scenarios

Several parameters determine whether or not a user is granted access to a given scenario. The requirements will be explained based on the SQL command that retrieves all necessary data to display the scenarios available for a given user in a given application version shown in the order in which they appear in listing 4.44:

- Table “scen_master” holds the list of all available scenarios,
- Table “scen_master_year” and view “gui_user_years” store information about which years a scenario is defined and which of these years a given user can see. This depends on the privilege needed to view data of a given year and the rights the user has in a given application version,
- Table “scen_to_pollutant” holds the assignment of scenarios to pollutants as mentioned in the previous section,
- Table “pitem_accesses” and view “gui_user_rights” determine which of the available pollutants a given user is allowed to see. As already mentioned, the display of the model interface for emissions and costs is pollutant driven. Since the available pollutants are in different development stages, access to the data can also be restricted by assigning appropriate privileges to the pollutants in table “pitem_accesses”,
- Tables “scen_to_group” and “scen_groups” aggregate the available scenarios into groups. Since there are around 150 scenarios currently defined in the European version of the model, a multi-level selection is more user friendly,
- The “share_public” attribute of a scenario, and tables “scen_to_user” and “gui_user_sharing” determine whether a given user has read access to a given scenario as mentioned in section 2.4.1,
- The last step is the check, whether a given scenario has a defined structure in table “scenario_n”.

Listing 4.44: User access to available scenarios

```

1 SELECT DISTINCT y.idvers , y.idusers , stp.idpollutants ,
2   s.scen AS idscenarios , y.year AS idyears ,
3   s.label_scenario , s.owner , s.type , s.share_public , s.cost_curves ,

```

```

4  y.idprivs AS priv_year,
5  sg.idscenario_groups, sg.label_sgroup
6 FROM scen_master s
7  JOIN scen_master_year smy ON (
8    smy.scen = s.scen)
9  JOIN gui_user_years y ON (
10    smy.year = y.year)
11  JOIN scen_to_pollutant stp ON (
12    stp.idscenarios = s.scen)
13  JOIN rains_web.pitem_accesses pta ON (
14    pta.pitem_idpitem = stp.idpollutants
15    AND pta.vers_idvers = y.idvers)
16  JOIN gui_user_rights r ON (
17    r.idprivs = pta.priv_idprivs
18    AND r.idvers = y.idvers
19    AND r.idusers = y.idusers)
20  LEFT JOIN scen_to_group stg ON (
21    stg.idscenarios = s.scen)
22  LEFT JOIN scen_groups sg ON (
23    sg.idscenario_groups = stg.idscenario_groups)
24 WHERE y.idusers = \$(idusers) — selected user
25 AND y.idvers = \$(idvers) — selected version
26 AND (s.share_public = 1
27  OR EXISTS (
28    SELECT 1
29    FROM scen_to_user stu
30    WHERE stu.idusers = y.idusers
31    AND stu.idscenarios = s.scen
32  ) OR EXISTS (
33    SELECT 1
34    FROM gui_user_sharing us
35    WHERE us.idusers = y.idusers
36    AND us.idvers = y.idvers
37    AND us.idowner = s.owner
38    AND us.read = 1
39  ))
40 AND EXISTS (
41  SELECT 1
42  FROM scenario_n d
43  WHERE d.scen = s.scen
44 );

```

Thus the previously mentioned assignment of scenarios to regions is not checked and retrieved in the query because currently a scenario is assigned to about 40 to 120 regions, depending on the version of the application. Therefore the amount of records retrieved per user would be multiplied by this factor. This would lead to an increase in query runtime upon insert, select and delete actions. This loss in efficiency is unproportionally high compared to the minimal runtime loss when joining in the regional information when needed.

The runtime of the just mentioned command is around 5 seconds. This is not very long for an average query runtime, but the information this query is retrieving, is needed on almost every page of the web interface, because most functionalities of the model are based on the scenario selection. The execution of the query upon every page refresh would lead to unacceptable page refresh times and therefore cannot be accepted.

Caching of User-specific Scenario Data

As mentioned in the previous section, some kind of caching strategy has to be provided so that the scenarios needed for a given user have to be fetched only upon login of the user into the model and not on every page refresh.

As discussed in section 4.2.2 temporary tables would theoretically be able to hold such information over the life time of a user log in. But this strategy cannot be applied because of the connection pooling. Another possibility to save runtime would be mvviews. Due to the complexity of the query shown in listing 4.44 and the restrictions for queries retrieving data for mvviews, this strategy can also not be implemented very easily. Since all native Oracle based solution seem to fail on this question, a custom-made solution to this challenge has to be found.

The database structure of the solution is presented in figure 4.24. Table “user_session_scenarios” holds all scenario related information retrieved for a given user session as shown in listing 4.44. The table does not have a primary key because of the fact that the *NOT NULL* are not unique. To create a primary key on the table, the columns “idscenario_groups” and “label_sgroup” would have to be eliminated from the table. Since the table is filled with data dynamically and the information is needed, a primary key is not created. Instead, indices are created for all important data filtering possibilities (the indices are not shown in figure 4.24 due to space considerations).

What is more interesting in the figure, is the logic behind the “user_sessions” table. It holds a unique combination of a web session identifier and a version. Furthermore it stores the user name bound to this session and the time when it was created.

Upon each login the information is written into this table and the needed scenario information is fetched. When the user logs out of the application, the corresponding user session is removed from the table and the associated scenario information is deleted by the cascading constraint “user_user_session” scenarios.

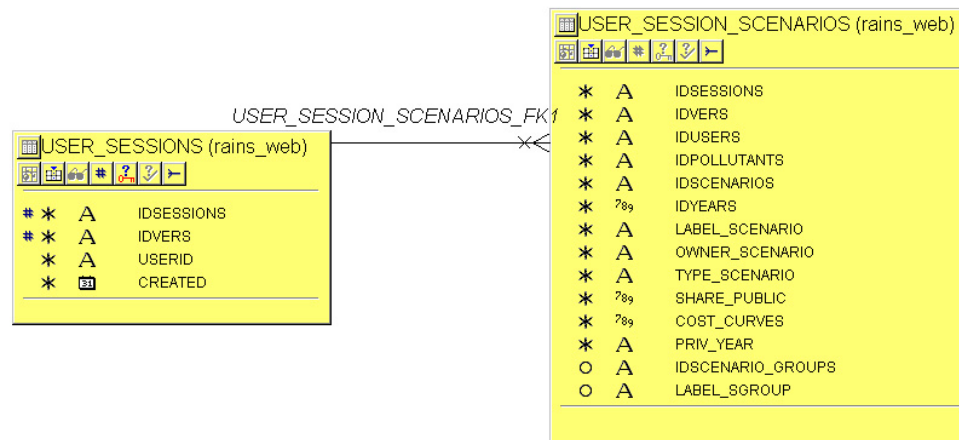


Figure 4.24: Tables holding active user sessions and associated scenario information

In many cases, due to various reasons, the user may not log out manually. Such abandoned sessions would unnecessarily stay in the system. Therefore garbage collection based on the expiring of a web session is necessary. This is done by implementing the `HttpSessionBindingListener` interface of the `javax.servlet.http` package (`HttpSessionBindingListener`, 2003) and cleaning values bound to a given web session, when the expiration of a web session is reported through the “value-Unbound” method of the interface.

Similar approaches could be used to cache other user session relevant information by storing it in the database for the life time of an user login.

Chapter 5

Conclusion

5.1 Summary

Based on the selected examples in chapter 4 we have shown that correct data modelling not only assures integrity of the available data itself, but also leads to the integrity of the results calculated and presented based on this data. The importance of integrity and consistency of data becomes even more important with the rising amount of data, since human checks, although made by highly skilled experts, become more and more difficult and thus unreliable.

By showing ways of making the underlying data structures independent of single pollutants and modelling data from a multi-pollutant point of view, we have eliminated many single points of failure. At the same time we have also assured that the suggested approach is not only correct from the data modelling point of view, but also returns results in a reasonable amount of time.

5.2 Where to go from here?

5.2.1 Flexibility in Viewing

Currently the model forces one hierarchical view on the data due to the structure of the interface: First a pollutant, then one scenario, and lastly a single region or a group of regions has to be selected. This approach was developed due to the pollutant specific development of the model. The flexibility of other views and

more flexible ways of drilling-down into available data was not possible because of the data structure.

Thanks to the remodelling of the emission calculation approach, the pollutants can now also be regarded as selection and query parameters similar to the scenarios and regions. They can also be flexibly grouped allowing a bigger variety of views on the available data.

These three basic selection parameters can now also be flexibly rearranged. This would allow not only the current selection sequence of pollutant, scenario and region, but also other sequences. It would even be possible to let each user decide in which order he would like to drill down into the data.

For example for some aspects it may be more interesting to select a given GAINS region, a few scenarios and a few pollutants. Assuming that a given user would like to compare three different scenarios for two different, arbitrarily chosen, pollutants, six different calculation requests would have to be sent with the current approach. After retrieving the data, the user would have to combine the data using an external tool. With flexible viewing approaches, all needed data could be retrieved in one request and presented in a single overview.

5.2.2 New Technologies

Thanks to the successful data cleaning, it will also be easier to analyze and investigate the possibility of managing and sharing data with the help of other technologies. Such possibilities could for example include a data warehousing concept. By implementing a data warehouse based on GAINS data it would be possible to allow a variety of views in a much faster time than it is possible with the current database structure. Such a data warehouse would also be capable of holding and displaying not only the currently available regional GAINS models, but also a global GAINS model. This would allow the analysis of global emissions and their impacts and would be an important milestone in the development of the GAINS model.

Another already mentioned aspect is the fact that the GAINS model is part of a large network of scientific models: Much of the data used as input for calculations of the GAINS model are provided by other models, and the results are further used as input for other models. The cooperation between these scientific models is also the main subject of the EC4MACS project funded by the LIFE program of the European Commission. This interaction could certainly benefit from developing

a network of web services through which it would be automatically possible to exchange data between the models.

Bibliography

- Alcamo, J., Shaw, R., and Hordijk, L. (1990). *The RAINS Model of Acidification*. Kluwer Academic Publishers.
- Amann, M., Cofala, J., Heyes, C., Klimont, Z., Mechler, R., Posch, M., and Schöpp, W. (2004). RAINS Review 2004 - The RAINS Model. Technical report, International Institute for Applied Systems Analysis IIASA. Available at <http://www.iiasa.ac.at/rains/review/review-full.pdf> (April 25, 2008).
- Amann, M. and Dhoondia, J. (1994). *Regional Air Pollution Information and Simulation - RAINS-Asia, User's Manual*.
- Calvanese, D., Giacomo, G. D., Lenzerini, M., Nardi, D., and Rosati, R. (1998). Source Integration in Data Warehousing. In *In Proc. of DEXA-98. IEEE Computer*, pages 192–197. IEEE Computer Society Press. Available at http://www.dblab.ntua.gr/~dwq/DWDOT-98_dwq.pdf (April 25, 2008).
- Collins, J. C. (2006). Microsoft Excel Pivot Tables. Technical report, Microsoft. Available at http://www.microsoft.com/dynamics/using/excel_pivot_tables_collins.msp (April 25, 2008).
- DBCP (2007). *Database Connection Pooling (DBCP)*. Apache Software Foundation. Available at <http://commons.apache.org/dbcp/> (April 25, 2008).
- EC4MACS (2006). European Consortium for Modelling of Air Pollution and Climate Strategies. Technical report, European Commission. Available at <http://ec.europa.eu/environment/>

life/project/Projects/index.cfm?fuseaction=home.createPage&format=p&s_ref=LIFE06%20PREP%2FA%2F000006&area=6&yr=2006&n_proj_id=3200&cfid=11667912 (April 25, 2008).

Economic Commission for Europe (2003). Guidelines for Estimating and Reporting Emission Data under the Convention of Long-range Transboundary Air Pollution. Technical report, United Nations. Available at <http://www.unece.org/env/documents/2003/eb/air/ece.eb.air.80.E.pdf> (April 25, 2008).

European Commission - FP6 (2005). GAINS-Asia - Greenhouse Gas and Air Pollution Interactions and Synergies with special emphasis on South-East and East Asia. Technical report, European Commission. Available at http://ec.europa.eu/research/fp6/ssp/gains_en.htm (April 25, 2008).

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, (17):37–54. Available at <http://www.aaai.org/aitopics/assets/PDF/AIMag17-03-2-article.pdf> (April 25, 2008).

Gupta, A. and Mumick, I. S. (1995). Maintenance of Materialized Views: Problems, Techniques and Applications. *IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and Data Warehousing*, 18(2):3–18. Available at <http://citeseer.ist.psu.edu/gupta95maintenance.html> (April 25, 2008).

Gutjahr, J. and Loew, A. (2002). Scalability and Performance: JDBC Best Practices and Pitfalls. Technical report, Sun Microsystems GmbH. Available at <http://citeseer.ist.psu.edu/552192.html> (April 25, 2008).

HttpSessionBindingListener (2003). *javax.servlet.http Interface HttpSessionBindingListener*. Sun Microsystems. Available at <http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/http/HttpSessionBindingListener.html> (April 25, 2008).

IIASA Options (1993). Acidification - Modeling Transboundary Air Pollution. *IIASA Options Winter 1993*. Available at <http://www.iiasa>.

ac.at/Admin/INF/OPT/options-backissues-scanned/opt93-4win.pdf (April 25, 2008) .

IIASA Options (1998). Cleaner Air for a Cleaner Future. *IIASA Options Summer 1998*. Available at <http://www.iiasa.ac.at/Admin/INF/OPT/options-backissues-scanned/opt98-2sum.pdf> (April 25, 2008) .

Irizawa, F. and To, R. (2001). Why is SQL Tuning Important to Your Business? The SQL Quality Challenge. Technical report, LECCO Technology. Available at <http://www.hkcs.org.hk/dbwp1805.doc> (April 25, 2008) .

Klaassen, G., Berglund, C., and Wagner, F. (2005). The GAINS Model for Greenhouses Gases - Version 1.0: Carbon Dioxide (CO₂). Technical report, International Institute for Applied Systems Analysis IIASA. Available at <http://www.iiasa.ac.at/Admin/PUB/Documents/IR-05-053.pdf> (April 25, 2008) .

Kronecker delta (2007). Kronecker delta. *Wikipedia*. Available at http://en.wikipedia.org/wiki/Kronecker_delta (April 25, 2008) .

Kyte, T. (2005). *Expert Oracle Database Architecture*. Springer-Verlag New York, Inc.

Loaiza, J. (2000). Optimal Storage Configuration Made Easy. Technical report, Oracle Corporation. Available at http://www.oracle.com/technology/deploy/availability/pdf/oow2000_same.pdf (April 25, 2008) .

Managing Tables (2002). *Oracle Database Administrator's Guide 10g, Release 2 (10.2), Managing Tables*. Oracle Corporation. Available at http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14231/tables.htm#i1006400 (April 25, 2008) .

MySQL (2007). *MySQL 6.0 Reference Manual - 12.5.6.4 FOREIGN KEY Constraints*. MySQL. Available at <http://dev.mysql.com/doc/refman/6.0/en/innodb-foreign-key-constraints.html> (April 25, 2008) .

- Nanda, A. (2004). Oracle Database 10g: The Top 20 Features for DBAs - Materialized Views. Available at http://www.oracle.com/technology/pub/articles/10gdba/week12_10gdba.html (April 25, 2008).
- Nicola, M. and Rizvi, H. (2003). Storage Layout and I/O Performance in Data Warehouses. In Lenz, H.-J., Vassiliadis, P., Jeusfeld, M. A., and Staudt, M., editors, *DMDW*, volume 77 of *CEUR Workshop Proceedings*. CEUR-WS.org. Available at http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-77/07_Nicola.pdf (April 25, 2008).
- Oracle CBO (2002). *Oracle9i Database Performance Tuning Guide and Reference, Release 2 (9.2) - Introduction to the Optimizer*. Oracle Corporation. Available at http://download-uk.oracle.com/docs/cd/A97630_01/server.920/a96533/optimops.htm (April 25, 2008).
- Oracle Data Dictionary Views (2002). *Oracle9i Database Concepts, Release 2 (9.2) - The Data Dictionary*. Oracle Corporation. Available at http://download.oracle.com/docs/cd/B10501_01/server.920/a96524/c05dicti.htm (April 25, 2008).
- Rahm, E. and Do, H. H. (2000). Data Cleaning: Problems and Current Approaches. *IEEE Data Eng. Bull.*, 23(4):3–13. Available at <http://sites.computer.org/debull/A00DEC-CD.pdf> (April 25, 2008).
- SQL*Loader (2001). *Oracle9i Database Utilities, Release 1 (9.0.1), Part II - SQL*Loader*. Oracle Corporation. Available at http://download-uk.oracle.com/docs/cd/A91202_01/901_doc/server.901/a90192/part2.htm (April 25, 2008).
- TOAD (2007). *TOAD - Tool for Application Developers*. Quest Software, Inc. Available at <http://www.toadsoft.com/> (April 25, 2008).
- TOrA (2007). *TOrA - Toolkit for Oracle*. Quest Software, Inc. Available at <http://tora.sourceforge.net/> (April 25, 2008).

- Wagner, F., Amann, M., and Schöpp, W. (2007). The GAINS Optimization Module as of 1 February 2007. Technical report, International Institute for Applied Systems Analysis IIASA. Available at <http://www.iiasa.ac.at/Admin/PUB/Documents/IR-07-004.pdf> (April 25, 2008).
- Witmuess, A. (1990). *Regional Acidification Information and Simulation Model - RAINS, Version 5.1, User's Manual*.