Jouni Laakso

Developing an Application Concept of Data Dependencies of Transactions to Relational Databases

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

18. February 2016

Helsinki Metropolia University of Applied Sciences

Abstract

Author Title Number of Pages Date	Jouni Laakso Developing an Application Concept of Data Depencies of Transactions to Relational Databases 73 pages + 5 appendices 18. February 2016
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor	Pasi Ranne, Senior Lecturer

Information systems usually use a relational database to store the application data. The relational database can be used outside of the scope of the application. The information systems has to verify the attributes to be the attributes of the transactions to the relational database. The integrity verification includes the verification of the atomicity of the attribute values and the form of their values matching the attributes type. Integrity verification includes the verification and the checking of the dependency constraints. The dependency constraints are usually other attributes the attributes are dependent on.

Applications are reprogrammed for different purposes. It has been noted that a complete information system and a new application program is not always needed in the most simple information systems. Sometimes a database query language is enough to use a relational database. For example an administrator of an application can remove and add users with an SQL-editor.

The thesis studies the automatic checking of the attributes of the transactions with consistency and integrity verification. The purpose was to develop a concept automatically checking the integrity and consistency of the applications attributes. With the help of the concept, the quality of the application should improve with the help of the reusable application components and with a generic application to be used in different purposes.

The application concept could be used in the simplest applications where program logic is not needed. The concept can be used in some part to replace the integrity verification of the relational database management system if the application does not use a relational database. It is also intended to use response messages helping the user to insert missing and mistyped attribute values based on the integrity verification.

Consistency and integrity in a relational database are based on relational theory. The relational theory has been developed to its current state mostly already in two decades starting from the year 1970. In the thesis, the algorithms providing the consistency has been researched and studied. The possibilities to develop an application concept based on the algorithms is evaluated with the relational theory.

Keywords	Relational model, consistency, integrity, schema.

Helsinki Metropolia University of Applied Sciences

Tekijä Työn nimi Sivumäärä Päivämäärä	Jouni Laakso Relaatiotietokannan Transaktioiden Tiedon Riippuvuuksien Sovelluskonseptin Kehittäminen 73 sivua ja 5 liitettä 18. helmikuuta 2016
Tutkinto	Master of Engineering
Koulutusohjelma	Tietotekniikka
Ohjaaja(t)	Pasi Ranne, Lehtori

Nykyiset tietojärjestelmät käyttävät yleensä aina relaatiotietokantaa tallettamaan sovelluksen tiedot. Relaatiotietokantaa voidaan käyttää myös yksittäisen tietojärjestelmän ulkopuolella erilaisilla tietojärjestelmillä erilaisiin tarkoituksiin. Yleensä aina järjestelmän muuttujat on tarkistettava tietojärjestelmän toimesta vastaamaan niitä muuttujia joita tietokannan transaktioissa voi käyttää. Tarkistuksiin kuuluu tietoalkioiden yksilöinti ja niiden tyyppiä vastaavan muodon tarkastaminen. Tarkistuksiin kuuluu jokaisen tietoalkion riippuvuuden olemassaolon tarkistaminen ja riippuvuus on yleensä aina jokin toinen muuttuja.

Sovellukset kirjoitetaan yleensä aina uudelleen jokaisen tietojärjestelmän kohdalla. On huomattu että aina ei tarvita kokonaista uutta tietojärjestelmää varsinkin kaikkein yksinkertaisimmissa sovelluksissa. Joskus pelkkä tietokannan kyselykieli riittää tietokannan käyttämiseen. Esimerkiksi sovelluksen pääkäyttäjät usein lisäävät ja poistavat käyttäjiä pelkällä SQL-editorilla.

Opinnäytetyössä on tutkittu eheystarkistuksen toteuttamista ohjelmallisesti. Tarkoituksena on tehdä sovelluskonsepti joka tekee eheystarkistukset automaattisesti. Konseptin avulla sovellusten uudelleen ohjelmoinnin tarve tulisi pienentyä ja siten ohjelmien laadun tulisi parantua uudelleenkäytettävien sovelluskomponenttien tai sovelluksen avulla.

Konsepti soveltuu yksinkertaisiin tietojärjestelmäsovelluksiin joihin ei tarvita ohjelmalogiikkaa. Joiltain osin konseptia voi käyttää myös korvaamaan relaatiotietokannan eheystarkistukset sovelluksissa jotka eivät käytä relaatiotietokantaa. Tarkoitus on myös selväkielisin syöttein auttaa sovelluksen käyttäjää syöttämään tiedot oikeassa muodossa jos automaattinen eheystarkistus ei ollut onnistunut.

Eheystarkistukseen on olemassa relaatioteoriaa joka on kehittynyt suurelta osin nykyiseen muotoonsa jo 1970 ja 1980 -luvuilla. Opinnäytetyössä on tutkittu soveltuvia algoritmeja soveltuvan eheystarkistuksen toteuttamiseksi ja arvioitu sovelluskonseptin toteuttamismahdollisuuksia relaatioteorian avulla.

Avainsanat	Relaatiomalli, yhtenäisyys, eheys, skeema.

Helsinki Metropolia University of Applied Sciences

List of Figures

Figure 1: Classification of the constraints..26
Figure 2: An imaginary example of the schema in Jacobson notation..........................28
Figure 3: A hypergraph of the example attributes of the schema.................................29
Figure 4: A derivation tree example...36
Figure 5: An UML activity diagram of forming the derivation tree.................................46
Figure 6: An UML activity diagram of the consistency verification................................47
Figure 7: A binary tree of determinants, a data structure of nodes and pointers...........48
Figure 8: Application schema files...54
Figure 9: Schema extensions...55
Figure 10: Research subjects...60
Figure 11: A stack model of a traditional application...63
Figure 12: A possible concept application stack model...63
Appendix 5, Figure 13: Test results, bytes read from a tree from 1 to 1000 attributes….1
Appendix 5, Figure 14: Test results, bytes read from a list from 1 to 1000 attributes......1
Appendix 5, Figure 15: Test results, integrity verification..2
Appendix 5, Figure 16: Test results, integrity verification from 1 to 100 attributes……....2
Appendix 5, Figure 17: Test results, forming the derivation tree.....................................3
Appendix 5, Figure 18: Test results, forming the derivation tree, 1 to 100 attributes.......3

List of Abbreviations and Acronyms

1NF	First Normal Form.
2NF	Second Normal Form.
3NF	Third Normal Form.
4NF	Fourth Normal Form.
5NF	Fifth Normal Norm or Project-Join Normal Form.
ACID	ACID is used to abbreviate atomicity, consistency, isolation and durability.
BCNF	Boyce-Codd Normal Form.
DDAG	Derivation Directed Acyclical Graph
DK/NF	Domain-Key Normal Form.
JSON	Javascript Object Notation.
LHS	Left hand side.
NoSQL	A term used to describe graph databases.
PJ/NF	Projection-Join Normal Form or 5NF.
RAP	A derivation sequence algorithm.
REST	Representational state transfer.
RHS	Right hand side.

Contents

Abstract

Tiivistelmä

List of Figures

List of Abbreviations and Acronyms

1 Introduction 1

1.1 Background 1

1.1.1 Three Tier Information Systems 1

1.1.2 Data Dependencies 1

1.1.3 Application Functions 2

1.1.4 Data Types 2

1.1.5 Descriptive Error Messages 3

1.1.6 Security, Roles, Ownerships and Rights 3

1.1.7 ACID in Database Transactions 4

1.1.8 Software Design Process 4

1.2 Research Plans 5

1.3 Focus and Scope 5

1.3.1 Data Dependencies 5

1.3.2 Application Schema 6

1.3.3 Tools 6

1.4 Research Method 6

1.5 Research Outcome 6

2 Background 8

2.1 Relational Model 8

2.1.1 Application Concept and Relational Model 8

2.1.2 Aggregates 8

2.1.3 Quantity in Relational Model 9

2.1.4 Keys of Attributes 9

2.1.5 Relational Integrity 10

2.2 Relational Algebra 10

2.2.1 Relational Modeling 11

2.2.2 Domain and Field 11

2.2.3 Cardinality 12

2.2.4 Tuples 12

2.3 Database Design 12

2.3.1 Terminology....13
2.3.2 Normal Forms....13
2.3.3 Functional Dependencies....15
2.3.4 Armstrong's Axioms....16
2.3.5 Completeness....17
2.3.6 Functional Dependency Set Closure....18
2.3.7 Cover, Nonredundant Cover and Minimal Cover....19
2.3.8 Closures of Attribute Dependencies....19
2.3.9 Multivalued Dependencies....20
2.3.10 Multivalued Dependency Inference Rules....21
2.3.11 Join Dependencies....23
2.3.12 Inclusion Dependencies....23
2.3.13 Null Values....24
2.4 Integrity Constraints....25
2.5 Empty Values....27
2.6 Schema Attributes....27
2.6.1 Schema Attribute Subsets....28
2.6.2 Transaction Order....29
2.6.3 Additional Application Attributes....30
2.7 Other Relational Database Constraints....30
2.7.1 Relational Database Integrity Policies....31
2.7.2 Other Constraints of Relational Database Software....31

3 Algorithms....32
3.1 Closure Algorithms....32
3.1.1 Linear Time Membership Algorithm....33
3.2 Multivalued Dependency Algorithms....33
3.2.1 Multivalued Dependencies in the Application Concept....34
3.3 Graph Algorithms....34
3.3.1 Derivation Tree Algorithm....35
3.3.2 Directed Acyclical Graphs....37
3.3.3 DDAG....37
3.4 Join Dependency Algorithms....38
3.4.1 Chase Algorithm....38

4 Application....40
4.1 Integrity Verification Algorithm Comparison....40
4.1.1 Algorithms....40

Helsinki Metropolia University of Applied Sciences

4.1.2 Comparison......41
4.1.3 Algorithm Decisions......41
4.2 Algorithms of Application Concept......41
4.2.1 Attribute Declarations......42
4.2.2 Derivation Tree with Multivalued Dependencies......42
4.2.3 Null Termination......45
4.2.4 Integrity Constraints......45
4.2.5 Writing Derivation Tree......46
4.2.6 Verifying Attribute Integrity......47
4.2.7 Technical Implementation......48
4.2.8 Compatibility with Normal Forms......50
4.2.9 Using Joins and Selections......51
4.2.10 Transactions and Repetition......52
4.2.11 Representing Attribute Values......53
4.3 Schema......53
4.3.1 Schema Definitions......54
4.3.2 Transaction Definitions......54
4.3.3 Determinants Definitions and Other Constraints......54
4.3.4 Data Type Definition, Visibility, Origin and Role......54
4.3.5 Error Message Definitions......55
4.3.6 Usability......56
4.4 Relational Concepts Summary......56
4.4.1 Configuring the Attributes......57
4.4.2 Functional Dependencies......57
4.4.3 Time Variance of the Transactions......57
4.4.4 Process Implementation......58
4.5 Other Considerations......58
4.5.1 Atomicity of Operations......58
4.5.2 Cloud Computing Environments......58
4.5.3 Adding Program Functionality......59
4.6 Summary......59
5 Results and Analysis......60
5.1 Research Outcome......60
5.1.1 Research Method......60
5.1.2 Source Material......61
5.2 Objectives......62
5.2.1 Applications Without Programming......62

5.3 Application Development 63
5.4 Test Results 64
5.4.1 Forming 64
5.4.2 Searching 64
5.4.3 Execution Times 64
5.5 Standards 64
5.6 Further Modeling 65
5.7 Further Research 65
5.8 Publications 65

6 Discussion and Conclusions 66
6.1 Discussions 66
6.1.1 Comparison 66
6.1.2 Novelty 67
6.1.3 Evolution 67
6.1.4 Restrictions 67
6.2 Conclusions 68
6.2.1 Intended Benefits 68
6.2.2 Benefits 68
6.2.3 Testing in Real Use 69

7 Summary 70

References 71

Alphabetical Index 74

Appendices
Appendix 1. Linear Time Membership Algorithm
Appendix 2. Multivalued Dependency Basis Algorithm
Appendix 3. Schema to Use in Examples
Appendix 4. Application Concept Schema Example
Appendix 5. Performance Tests

1 Introduction

Introduction describes the background of the thesis. The chapter describes the reasons lead to the thesis work, the more precise scope and the expected results. On the basis of the introduction, finally the objectives of the work can be evaluated with the results. The introduction should not change during the work.

1.1 Background

The background of the thesis is in the three-tier information systems developed and used in the public sector. The relational database is an integration tool used in the ICT-management departments. The relational database provides the information systems the necessary data in a usable form.

1.1.1 Three Tier Information Systems

Three tier information systems based on messaging systems like HTTP usually consist of an end user application, a session based application with a store of a session-based data for the application user and a data-store serving all the application users. The data-store, usually a relational database, has to be consistent and serve all of the application sessions at the same time. Attribute dependencies from the application and the applications session have to be fulfilled in every transaction to maintain the consistency to the relational database.

Usually in the traditional programs the database transactions are represented in fixed strings and they are embedded inside the application code. This prevents the reuse of the application code to other purposes. Automated verification of the data dependencies already in the application would allow the program code to be more reusable for different purposes.

1.1.2 Data Dependencies

The data must be identified to be the data in a relation to other data for the data in the database to be consistent after the transactions. The data of the transactions has to be identified, the data of the transaction has to belong to some relation in the database. The form of the data has to stay consistent.

For example an authentication of the user usually provides globally an identification of a person. The result could be an existing variable of a type of an authenticated user. The information can be used in identifying data relating to it to verify that the relevant data is present before allowing the execution of a transaction. An application session is identifiable by a session based mechanism identifying a browser that started the session. This is usually a cookie mechanism providing an identifiable session [1] and is usually a feature of the server applications. A session can be represented as a variable.

1.1.3 Application Functions

Sometimes application functions are necessary. Application functions should be implemented in a reusable way. Application functions are not going to be replaced in the thesis and in the application concept.

Quantity, cardinality and the one-to-many relations of data are an interesting and challenging characteristic of the application concept. Set theory has been used in the modelling of the relational theory.

Usually information is stored in the persistent background datastore. The application has done everything needed after the data is saved in the correct form. After this, the data is reprocessable. It is not necessary for the same application to reprocess the data. This proves that if the datatypes can be represented in the user interface similarly as in the database, functions are not necessarily needed if the data integrity can be ensured and the necessary data is provided.

Application functions need the defined variable data. In transactions to a relational database, all variable data of a transaction can depending on the applications design be considered to be in a relation to one another.

1.1.4 Data Types

Data types should be verified before saving the data-items to a database. From the user interface, usually the data has to be typed in correctly in the form of its type depending on for example the language, country or even the number type of the application. Datatypes can be polymorfic. The types form can be derived from other types. Types can be chained together to inherit properties from each other. In addition in some programs it is possible to transform data items from one datatype to another as a property of the type system. Transforming of the data can be understood to replace some programs functions. Functions can be understood as types [2, 18].

A data-item can have a lifetime in its type. A data-item whose time has exceeded, can be discarded or forgotten programmatically. A session timeout, for example, gives information on how long the session is saved in the server. An idea of forgetting was introduced with the concept of functors [3, 12].

The name tuple is used mostly to represent the attributes in the same relation [5, 68]. In theory, an attribute in the tuple can have values null, one or multiple values. An object can be interpreted as a tuple [2, 33].

The data types are an important feature of an application. User fed informations form and integrity has to be verified before using the data in the application. A data-type should be a part of the application schema.

1.1.5 Descriptive Error Messages

The principle of verifying the data dependencies in an application applies similarly to the error messages. Applications usually respond to the user if errors occur and these error messages can be instructions to the user to correct the error state in the application. If something is missing, for example in an HTML form, the application could be made to inform the user automatically what data is missing. The information of the missing data should be got from the data dependency information of the data items.

Error messages of applications after usual HTTP server functionality are data-type verification with instructions to correct the data items form and checking the dependencies of the data-items with descriptive error messages informing what data is missing or incomplite to complite the transaction.

Error or an exception can be in the concept of a type [2, 29]. Using datastreams of for example key-value pairs, the exeption can be carried as a type to the user interface.

1.1.6 Security, Roles, Ownerships and Rights

Application users have different roles and depending on the role, different priviledges to the data. In a database, it is possible to have read or write permissions to the data-items based on the ownership or role. Transactions may need an execution right based on the ownership or the role in an application.

Security is important when using the data to verify the dependencies. Some data-items may not be used in the user interface because of the security and the data not being public.

The origin of the data may have to be restricted: From the application configuration file only, from the database only or from the user interface only or from some other functions or sources. Restrictions where the data may be visible and used may have to be defined.

The user roles and the data security are an important part of the application schema. Data-items can depend on other data-items and this could be described in the application schema. For example, if a role does not exist in the session, some transactions are not allowed. A role can be used for example as a data-item among other data.

1.1.7 ACID in Database Transactions

The database has to ensure the consistency and the integrity of the data. Sometimes an abbreviation ACID is used to descibe atomicity, consistency, isolation and durability. Durability means the data has to be in the permanent store after the transaction is completed. consistency is the normal form of the data. Verifying the integrity of the data ensures the consistency. In designing applications, the most important of these features are the consistency, isolation and the atomic transactions. The database management system usually ensures the durability and concurrent transations.

Transactions in a database are made atomically. An atomic transaction is a transaction started and completed before other transactions. Atomic operation is a concurrent operation reserving or locking the transactions data in its use until the transaction is finished. This ensures the consistency in the database and prevents race conditions.

The term isolation is used to describe that the data is isolated to the use of the transaction. The race condition can be avoided if the data can be ensured to be isolated. For example every gathered sample in a statistical analysis is always new data and appending it is an insert operation. An end date or deadline of the data sample collecting ensures that the data is not accessed before the end date of taking samples. Every sample is isolated from the other samples.

1.1.8 Software Design Process

The application concept can not replace software design process, the requirements analysis or the designing of an application. The application has to ensure the data is updated without race conditions and that the transactions have succeeded. The restrictions of the relational database schema has to be taken into consideration.

1.2 Research Plans

It is possible to make an application schema describing what data-items are dependent on another data-items. The application would refuse to do actions to the data if it's schema defined dependencies are not met. The research should give an answear to the question about the relevant theory of the data dependencies.

An empty data-item or an empty set can not be an argument to a transaction if it is not allowed. A data-item may have dependencies to other data-items. The dependency is not met if the dependency is an empty data-item. For example in a relation the data-items to which all the other data-items are dependent on, should in many cases exist before the transaction is possible. Transactions need their data-items and an application has to have a schema describing them and a method to verify and check the dependencies before transactions.

The main theory considering the subject is relation theory, especially functional and multivalued dependencies, normal forms and the database design. A method to implement similar functionality as is in the relational databases to verify the dependencies with a pre-defined application schema should be researched and evaluated.

1.3 Focus and Scope

The purpose of the research is to provide the necessary theory to implement an application or an application framework to use the data dependency verification mechanisms with an application schema. The relational database provides a stable datastore. It also serves as an example in developing a similar concept. A case where the background database is not a relational database can be evaluated.

1.3.1 Data Dependencies

Transactions can be any transactions to the outside data, authentication functions for example or to other application functions. The data should be identified. Data outside the scope of the application and outside of the applications database should be proved to be the same data related to the applications data before it can be used in the application. Data identification is needed to identify the dependent data items as described in the database schema. Data can be moved between the user interface, the application and the database. The relevant data may have to be fetch to the application from other resources to verify the dependencies.

1.3.2 Application Schema

An application schema describing the applications data dependencies should verify the dependencies in the application before transactions to the database. The application could be a dependency checking application instructing the user of missing dependencies in using the application.

1.3.3 Tools

A library had been developed previously to read key-value -pairs in unordered fashion. The library provided similar functionality as in the RFC-2822 without the predefined variable names [4]. The library formed a linked list of the read key-value pairs to a buffer in the order of appearance to be searched again. The locations of the key-value pairs were indexed to be read again quickly.

Tree-structures such as JSON could be used because of the compatibility with the current browser software in many different terminal equipment. JSON is used also in most of the document database software. Tools may restrict the concept, not the requirements or the theory. Tools are otherwice left out of the scope of the thesis.

1.4 Research Method

The thesis studies possibilities and restrictions of the dependency verification in an application. The same concept as in the relational database design is used as an example. The research was made from publications and literature. Testing with a test program could prove the concept usable. It would be useful to program the final product ready once and test once. The tests are left out of the scope of the thesis.

After the research, the similar technologies of the concept and the relational database could be used to compare the functionalities to find similarities on both sides of their interface. If the possible tests are complited before the end of the thesis work, the results can be used to compare the concept to it's environment with the test results.

1.5 Research Outcome

The purpose of the application concept is to *help the user to input correct data* by checking that the data dependencies are fulfilled already in the application and informing the user of the missing and incompleted data. *Necessary theory* to describe the dependencies in the schema should be provided with the possible restrictions.

The theory to be used in the application concept of data dependency verification should be specified and evaluated. The thesis should provide a specification of requirements based on the theory applied to the concept. Thesis is also *a learning project* of the theory with a technical background.

The aim generally is to increase *program code reusability, software quality* and to *reduce the work needed* in the application development with reusable software. How much the application development can be automated can be evaluated. The application development process is not going to be replaced.

2 Background

To verify the integrity and consistency of the attributes used in the transactions to the relational databases, the relational model provides a good example with its supporting mathematics. The relational model itself is evolved during the research and the commercial development of the relational database management software.

2.1 Relational Model

The relational model is not entirely compatible with its mathematical model, the relational algebra. The functional dependencies are used both in forming the database schema and in the mathematical models verifying the integrity and consistency of the application attributes.

2.1.1 Application Concept and Relational Model

In the application concept the concepts of the relational model are the normal forms and the dependency constraints. An attribute may have a range of values and the transactions may have repetition. Special meaning of null values are important.

Transactions to a database are create, read, update, insert and delete operations. If transactions to a database is a delete, insert or update, repetition is needed. If a range of an attribute value is large, an operation is repeated to every value in the range of the attribute. Read operations return a range of values.

2.1.2 Aggregates

Aggregate values are not a part of the relational model. An aggregate in an attribute value is a larger construct to be updated once. An aggregate can be a file, for example. Aggregates in values are treated as atomic values and they usually are text strings, for example XML-files or JSON-files or other objects.

The name aggregate can be used in database terms of an attribute of a column consisting of all of the attribute values in the column or specific attributes of the resulting rows of a query for example. An aggregate is updated in only one operation even if it has more information. If an aggregate has many values, it is called a repeat group [5, 289].

The name aggregate is sometimes used in another contexts as an aggregate function or aggregates of attribute values. Aggregate functions make calculations to a range of values and the values are usually the values of a specific attribute.

2.1.3 Quantity in Relational Model

In the relational model, the relation and a foreign key represent a functionality to specify items in another table and keep account of the items in another. The relational model is not capable of representing quantities without identifying every relation of the values of the attributes.

The relational model does not apply to numbers and calculations as well as to relations of attributes. The identity of each item would be lost if it is substituted by a number or with other representation of a cardinality. Quantity and counting are not part of the relational model. If the relational model is not used properly, program code is needed to replace the relational model.

2.1.4 Keys of Attributes

In a relational table a key is used to distinguish the relations of attributes from other relations of attributes. A key can consist of one or several attributes. If the attribute is an attribute of an element in another relational table, it is called a foreign key. The foreign key connects two or more tables with the same attributes.

A superkey is an attribute or many attributes of the set of attributes in a relation that identifies the relation from other relations. A superkey has a constraint that prevents adding similar relations [5, 69].

A candidate key is a minimal superkey identifying the attributes in a relation. Sometimes a key is denoted to be a superkey that has not a subset of another superkey [10, 5]. The name candidate key is most used. A value of a super key is called a prime [10, 6; 36].

Primary keys are used in most database software and the use of primary keys has become accustomed. Primary key is a candidate key chosen to identify alone the attributes in a relation. Primary key cannot be null [5, 70].

The foreign key has the same attribute value in more than one relational table. The same value appears in an attribute in more than one relational table. In addition to other normal forms, in 4NF, 5NF and DK/NF, one of the values has to be a candidate

key of the respective relational table. The foreign key of the other attribute of the relation can be a composite key.

All keys except the primary key can be a composite key of more than one attributes. The software used may allow or restricts the use. A name dynamic relation constraint is sometimes used to denote a composite key [18, 11].

2.1.5 Relational Integrity

Relational integrity is defined differently in different sources. Usually the following four rules are used.

Entity Integrity

Primary key does not accept null value. The tuple or row of the relational table has to be identified. [5, 70]

Null Integrity

Null is usually used as a placeholder of incomplete information [5, 71]. Null may have other meanings.

Domain Integrity

The attributes of a domain should be in their valid range [5, 71]. A domain is a set of values an attribute can have or a range of values.

The same attribute value has to occur on both sides of the domain. The same attribute value occurring from one to many times in the multivalued foreign keys side has to occur once as a unique value in the other side, in the super key side (4NF) or in the candidate key side (BCNF) of the domain.

Referential Integrity

Foreign key is an attribute in more than one relational table. It is important that the foreign key is the same in all of the relevant tuples of the tables. Referential integrity states that the foreign key must be null or it must match a primary key of another table [5, 72]. Referential integrity constraints are key-based inclusion dependencies [12, 3].

2.2 Relational Algebra

The relational algebra is formed by the functional dependencies introduced in Chapter 2.3 below and the following definitions together. The functional dependencies are used

also in the schema design and the functional dependencies are discussed under the topic Database Design as in many text books. This chapter introduces some basic definitions: a domain, a field, a tuple and the cardinality of the one-to-many relations. Relational algebra does not represent aggregate functions. Otherwice the functional dependencies with their axiom systems together form the algebra.

2.2.1 Relational Modeling

Relational algebra is a mathematical algebra used in relational modelling. Most algebras can be derived from the rules of the set theory. In the set theory, if something is not proven by the rules of the theory, it is called naive [21, 11]. Naive set theory is a theory of sets and compared to the set theory, it is not axiomatized. Sometimes relational algebra operations introduced have been derived using the set theory [21, 7].

2.2.2 Domain and Field

Domain of an attribute is sometimes used to describe all of the values of the attribute [8, 205]. In relational model, domain of an attribute consists of the same attribute in different relational tables. The attributes has to have the values in the domain from the same range of values. A same attribute value in the same domain is also of the same type with the other attribute values.

A classical interpretation of a domain is the following where a class is described as a collection of sets: "One class is said to be 'similar' to another when there is a one-one relation of which the one class is the domain, while the other is the converse domain." [7, 11, 16]

A field is the domain and the converse domain together [7, 32]. The same applies to relational model where attributes in more than one relational table together form a field of an attribute.

In set theory, a domain relation has a direction. In the next formula, a relation between attributes is inside less than and greater than marks. It is an ordered pair. [8, 26]. In the set theory a domain of a relation is [8, 26]:

$$\{x \mid for\ some\ y, <x, y> \in d\}$$

Where d is a relation of a domain and <x, y> a group of related x and y. x has the similar meaning as the projection or an attribute has in the relational algebra. Range similarly [8, 26]:

$$\{y \mid for\, some\, x, <x, y> \in d\}$$

Where y denotes a range as in the selection or a range of relations in relational algebra.

2.2.3 Cardinality

"The 'cardinal number' of a given class is the set of all those classes that are similar to the given class." [7, 42]

An ordinal is an arithmetic count of the cardinals. A number usually means an ordinal of similars of a class. A class is a collection of sets. Cardinal numbers are the usually used numbers [7].

The relation of a domain of attributes between relational tables can be in one-to-one, one-to-many or in many-to-many relationships to each other. In the relational model, cardinality have been said to be the number of rows in the relation [5, 67].

In the concept of the foreign-key forming a domain, an attribute identifying a row in an another table may have multiple same values in an another table forming a range of the attributes. The count of same values is its cardinality.

2.2.4 Tuples

The same attributes in both relational tables can be seen to be in a relation to each other via the transitive functional dependency. A functional dependency can be at the same time a multivalued dependency. Intuitively it would be tempting to say that all of the attributes with a common domain are in the same tuple. In relational database theory, the name tuple is reserved to the meaning of the same relational table only. The values of the attributes are taken from the domains of the attributes to form a tuple or a record [11, 29].

2.3 Database Design

The concept of the normal forms of the database schema is introduced in Chapter 2.3.2. After this, the mathematical concept of the relational algebra is extended with the axiom systems and with some mathematical definitions such as a cover and a closure. The relational model is not complitely a mathematical concept and the integrity constraints of the relational model are explained here as part of the database design.

2.3.1 Terminology

Many different methods have been used to describe the formulas of the set theory and the relational algebra. In the thesis, a common syntax should be used. Formulas used are usually in the form of the set theory.

A term scheme is usually used to denote a relational table with a name, the names of the attributes, the sets of the attributes and the definition of the designated keys. A scheme can also be taken to mean the relation and a set of relation constraints. The relation can be a row of joined table for example or a row of a relational table. A scheme can be also called a relation. A relation can be for example a row of joined attributes of relatinal tables. Sometime a capital letter R is used instead of the name scheme. A relation can have a meaning (M), a primitive relation scheme as follows (PRS) and a set of constraits or conditions (SC) [18, 3].

Usually a letter *Ω* is used to denote the set of the attributes [18]. Δ is used to denote the domains of the attributes of the relation, the set of values. dom: $\Omega \rightarrow \Delta$ is a function associating each attribute to a domain [18]. In short, a scheme contains attribute names of a relation and a collection of functions to associate the attribute to it's domain values [18, 3]. In this way, a relation scheme RS is RS={ *Ω, Δ, dom, M, SC* } [18, 4]. A schema is all of the schemes together, a collection of the schemes.

2.3.2 Normal Forms

Normal forms are used in designing the data structure of the schema of the database. The purpose of the normal forms is to provide a mean to check that the database design is feasible and to aid in designing.

There are eight normal forms in order: unnormalized form is often abbreviated as UDF, First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), Fifth Normal form (5NF) or Project-Join Normal Form (PJ/NF) and Domain Key/Normal Form (DK/NF). If a schema is in a normal form, it is in all of it's lower order normal forms.

Normal forms up to BCNF are defined with functional dependencies [5, 66]. Normal forms from fourth normal form onwards are used to ensure the integrity with multivalued dependencies and to reduce update anomalies.

First Normal Form, 1NF

A schema is in 1NF is all it's attributes are atomic. Datatype verification of the

application software ensures the values are in the form of its type and values are atomic.

Second Normal Form, 2NF

A schema is in 2NF if it is in 1NF and if all its nonkey attributes depend on all of the key [5, 309] [10, 7].

Third Normal Form, 3NF

A schema is in 3NF if it's relational tables do not contain transitive dependencies between attributes [5, 301]. Other definition is, if nonprime attributes do not have transitive dependencies to a key in the relational table [10, 7].

Boyce-Codd Normal Form, BCNF

A schema is in BCNF if it's in 3NF and for every nontrivial dependency (or for every disjoint attributes X and Y) X→Y, X is a super key [5, 304] [10, 7].

Fourth Normal Form, 4NF

A schema is in 4NF if it's in BCNF and the following holds. If there exists a multivalued dependency, it has to be either trivial or the determinant of the multivalued dependency has to be a super key [5, 307]. Multivalued dependency A→→B is trivial in relation R if B is a subset of A or A union B is R.

Fifth Normal Form, 5NF or Project-Join Normal Form, PJ/NF

A schema is in 5NF and PJ/NF if and only if every join dependency in relation R is implied by the keys of R [5, 311].

Domain Key Normal Form, DK/NF

"Every constraint on the relation must be a logical concequence of the definition of keys and domains." [5, 298]. A 1 NF relation schema is in DK/NF if and only if it does not have insertion or deletion anomalies [17, 12].

Schema is in a domain-key normal form if in all of its relations the domain dependencies and the key dependencies are a logical consequence of every constraint in the relation [17, 11]. A domain dependency is described to be a dynamic constraint [18] of the domain and a key dependency is the attributes of a key [17, 5].

In the original publication, the form of DK/NF is described to be enough to verify the constraints from the 1NF. The schema is in DK/NF if it's in 1NF and every constraint

can be inferred from the key dependencies and the domain dependencies [17, 11]. Domain dependencies are defined to be the attributes dependencies in each entry of the tuple and the key dependencies the dependencies identifying the tuples.

Most database software has implemented only the key and foreign key constraints and not the functional dependencies in their data definition languages [17]. For example a commercial System R used by IBM did not support functional dependencies, instead it had only an index of the definitions of the unique keys [17, 11].

2.3.3 Functional Dependencies

Functional dependencies are used in designing the relational database schema.

Functional dependencies are dependencies of variables to other variables within one relational table. Foreign key can be considered included because it appears in the same table of relations. Functional dependencies can be used in some extent to verify if relations of attribute values are legal in the rules of the schema.

In the functional dependency, a determinant uniquely determines another attribute. Functional dependency is denoted as $A \rightarrow B$. It means

1. "A" determines "B"
2. "B" is functionally dependent on "A"
3. "A" is called a determinant.

Functional dependency is said to be trivial if B is included in A, $B \subseteq A$. [5, 290]

If a variable is determined by many variables, the determinants are called compound determinants. A partial functional dependency is the case where a determinant is only a subset. Full functional dependency is the case where the determinant is the second attribute but not a subset. Transitive dependencies are dependencies through an intermediating functional dependency. [5, 290]

If A determines B and does not exist, it does not determine B. The determinant has to occur before the item depending on it.

The use of the functional dependencies

The use of the functional dependencies are in synthetizing the database schema computationally and in aiding in designing the database schema. They provide an elegant mathematical model to inspect the properties of the relational schema [10, 1]. The original purpose of the data dependencies were to introduce data independence

by abstracting the database functionality from the database users [16][10]. The functional dependencies also give information on the dependency constraints of the attributes and of the properties of the normal forms.

The use of the functional dependencies in the application concept

The functional dependencies are needed to verify if all of the attributes determinants of the transactions are present to be able use the attribute in the transaction. The functional dependencies are needed in the other direction than they are usually needed in the designing of the schema or in determining other properties. The emphasis is in the left hand side of the definition of the functional dependency and the needed direction is from the right to the left. Instead of calculating a closure of all of the attributes dependencies as described later, a collection of all of the determinants determining an attribute is needed.

2.3.4 Armstrong's Axioms

Functional dependency inference rules or Armstrong's axioms were first introduced in a publication of W. W. Armstrong in 1974. The publication describes functional dependency rules consisting of four statements describing the inference rules and three statements to describe maximal elements [9, 2, 3]. The third statement contains a part of the lattice-theory, a property of semi-ordered sets. The resulting set of applying the dependency rules to the dependencies is called a full family of dependencies. [9]

The original four statements can be used to verify the completeness of other axiom systems [9]. A complete set is sometimes said to be also the following inference rules where letters F denote the original four statements without the third statement [9, 4; 5, 292, 293].

F1 Reflexivity

If every x in X is in relation to x in X the relation is reflexive. Reflexivity means the relation of X to itself and additionally a relation X to it's subsets [5, 292; 9, 2]:

$$If\ Y \subseteq X\ then\ X \rightarrow Y$$

F2 Transitivity

A relation is transitive if relations X to Y and Y to Z imply X to Z [5, 292; 9, 2]:

$$If\ X \rightarrow Y\ and\ Y \rightarrow Z\ then\ X \rightarrow Z$$

Union

A union [5, 293]:

$If\ X \rightarrow Y\ and\ X \rightarrow Z\ then\ X \rightarrow YZ$

Pseudotransitivity

A pseudotrasitivity [5, 293]:

$If\ X \rightarrow Y\ and\ YW \rightarrow Z\ then\ XW \rightarrow Z$

Decomposition

A decomposition [5, 293]:

$If\ X \rightarrow YZ\ then\ X \rightarrow Y\ and\ X \rightarrow Z$

F4 Augmentation

The augmentation rules [5, 292; 9, 2]

$If\ X \rightarrow Y\ then\ XZ \rightarrow Y$

$If\ X \rightarrow Y\ then\ XZ \rightarrow YZ$

Armstrong's axioms or these axioms are usually used as inference rules to derive all possible sets of functional dependencies of a schema and to form a minimal closure set from the closure set of all possible functional dependencies by removing the redundant dependencies. They are also important in proving another axiom system to be complete and compatible with the reference axiom system.

Union and decomposition of the rules are logical consequences of the other rules [10, 9]. A concequence is that the other four axioms are enough in applying the inference rules. Sometimes only three rules are enough [10]. Different axiom systems have been developed to be used in algorithms and solve different properties of the dependencies of the relational model.

2.3.5 Completeness

An axiom system is said to be complete if the Armstrong's axioms can be derived from the axiom system. Compiteness can be used in proving if a method of the functional dependencies is sound. In the original publication it is encouraged that the completeness of an axiom system should be proved with these axioms. The role is more of as a reference because many axiom systems have been shown to be compatible with the Armstrong's axioms with more or less rules.

The axioms provide the standard to be used to prove the completeness of other axiom systems. The set of above axioms in this thesis were originally published before the Armstrong's publication. [9, 4]

2.3.6 Functional Dependency Set Closure

A closure is a set where the results of the operations of its members are itself the members of the set.

The set of functional dependencies of a closure is a set consisting of all of the possible functional dependencies of the schema [10, 4]. It is sometimes said that F is a set of functional dependencies logically implied [36]. Different subsets of the set of all of the dependencies can have the same closure. Closure of a set of functional dependencies is usually denoted by a plus sign in the subscript on the right upper corner of the letter describing the set.

All of the possible attributes dependencies together form the closure. Closure of an attribute can be derived from the functional dependencies by a repeating algorithm [5, 295] [6, 21] [11, 165]. The following algorithm is altered to the example [11, 165][5, 295]:

```
Input: a set F of fd's and a set X of attributes.
Output: the closure of X under F.

1. closure := X
2. unused := F
3. repeat until no further change:
      IF W → Z ∈ unused AND W ⊆ closure
         THEN
             i. closure := closure ∪ Z
            ii. unused := unused -{W → Z}
      FI
4. output closure.
```

At start, all attributes are included in the closure by reflexivity. Closure is then formed by joining all attributes dependencies to the attributes dependencies in the closure with a union operation.

Because of transitivity, all of the attributes dependent on the previous determinant are included and after this, the ones it determines. This algorithm is slow because when the number of the attributes in the closure increases, more repeat cycles are needed to add the dependencies of the added attributes. A polynomial time algorithm for example can be found to solve this problem to be used instead of this one [18].

2.3.7 Cover, Nonredundant Cover and Minimal Cover

A cover is a union of all of the functional dependencies. The cover can be represented with semi-ordered sets. To avoid the definitions, it is easier to use inclusion: If all of the dependencies are included in a set, it is called a cover. [8, 50]

In the same relational schema it is said that two sets of functional dependencies ***covers*** each other if their closures are equal. If $F^{+}=G^{+}$ for the sets of functional dependencies F and G, F covers G and G covers F . [5, 295] Cover is also a cover of a set of functional dependencies that has the same cover as the cover of the original set of the functional dependencies [10, 5].

If F covers G and G covers F, functional dependencies in F are ***equivalent*** to the functional dependencies of G, $F \equiv G$ [36]. In set theory, equivalence relation is a relation that is reflexive, symmetric and transitive [8, 29]. In set theory, the direction is lost due to symmetricity. Symmetricity is not among the inference rules. In the case of the functional dependencies, the direction is lost if closures are used due to different possibilities of a cover and the Armstrong's axioms. By the definition, determinant has to occur first in the application.

Redundant dependencies may be removed from the closure if the resulting set of functional dependencies has no subset of the dependencies of the kind of the removed dependency [10, 4]. Redundant dependencies can be removed with inference rules [5, 295]. When all the redundant dependencies are removed, the result is called a ***minimal cover*** [5, 295]. The result is ***nonredundant*** if it does not contain any subset that also has the same cover [10, 4]. Minimal covers can be formed also by using for example Bernstein synthetization algorithm [20, 3][36].

A functional dependency is said to be in a canonical form if its right hand side is in a singleton form where the right side consists of only one attribute.

2.3.8 Closures of Attribute Dependencies

A closure is a set of dependencies. Every attribute's constraints are the determinants with the functional dependencies. A closure of an attribute is the set of all the attributes dependent on it. A term ***saturated set*** can be used or the term ***closed set*** [15, 3; 9, 3].

A closure is a union of all of its dependencies in the set of all functional dependencies. A closure can be derived from the dependencies by applying the Armstrong's axioms. It is said that the attribute implies all of it's dependencies [36].

A common notation to describe a closure is to represent it with a plus sign at the upper right corner of the attribute name.

An example

For example if F is a set of functional dependencies of all known dependencies of a schema:

$$F^{+}=\{A\rightarrow B, B\rightarrow C, C\rightarrow D, D\rightarrow B\}$$

It would be a minimal cover because all of the redundant dependencies have been removed or do not exist. All dependencies determined by attribute A from the set of dependencies of F^{+} is defined by a union

$$A^{+}=ABCD$$

By reflection A is dependent on A. B is dependent on A as in the dependencies in F^{+}. By transitivity, C is dependent on B and D is dependent on C. A result is that the closure of A are all the variables and the closure A^{+} is the union of all of the attributes. The same rules applies similarly to the other attributes:

$$B^{+}=BCD$$
$$C^{+}=CDB$$
$$D^{+}=DBC$$

Finding if an attribute or a composition is a key

If an attribute's closure of the dependencies contains all of the attributes of the relation, the attribute of the closure is a key [36].

Without the definition of a cover:

$$if\ X\rightarrow Y\ and\ \mathrm{X,Y}\in U.\ If\ X\rightarrow U,\ X\ is\ a\ key.\ U\ is\ a\ set\ of\ attributes.$$ [11, 163].

It is also possible to find designated keys [10].

2.3.9 Multivalued Dependencies

A multivalued dependency is between two attributes A and B where the other attribute B has a set of values. Another attribute (C) may have a multivalued dependency with the attribute with one value (A) and in this case, the ones with multiple values, C and B are independent of each other and in many-to-many relationship [5, 305].

Multivalued dependency is caused by two independent values, in this case B and C and denoted by:

$A \rightarrow\rightarrow B$

Multivalued dependency is trivial if B is a subset of A.

Multivalue dependencies may cause excess rows within a relational table. One method to test multivalued dependencies is a to select rows with joins between relational tables internal attributes. If the joining results excessive rows, a multivalued dependency is found. In the case of a multivalued dependency, the table should be decomposed into smaller tables where the determinant of the multivalued attributes are in both tables.

If functional dependency exists between attributes, it implies that the multivalued dependency exists between the attributes [24, 127]. Multivalued dependency on the other hand does not imply functional dependency.

Decomposition

When designing the schema of a relational database, attributes causing the multivalued dependencies should be decomposed into their separate tables. Choosing the attributes is called projecting. The projected tables and the original relation can always be joined together again [18, 76].

2.3.10 Multivalued Dependency Inference Rules

Multivalue dependency inference rules are the same inference rules listed as inference rules to replace the Armstrong's axioms with decomposition replaced by M4 below [5, 307]. Complementation, M7 is added to these [24, 130]. The completeness of the multivalued dependency axioms have been proven with the axioms of the functional dependency in many sources [24][18].

The following multivalued inference rule principles are used for example in the original definition of the chase algorithm. They are chosen from the main sources used in the thesis. The multivalued dependency inference rules are [24, 129]:

M1 Reflexivity

$X \rightarrow\rightarrow X$

M2 Augmentation

$If\ X \rightarrow\rightarrow Y\ , then\ XZ \rightarrow\rightarrow Y$

M3 Additivity

$If\ X \to\to Y\ and\ X \to\to Z\ then\ X \to\to YZ$

M4 Projectivity (decomposition)

$If\ X \to\to Y\ and\ Y \to\to Z\ , then\ X \to\to Y \cap Z\ and\ X \to\to (Z - Y)$

M5 Transitivity

$If\ X \to\to Y\ and\ Y \to\to Z\ , then\ X \to\to Z - Y$

M6 Pseudotransitivity

$If\ X \to\to Y\ and\ YW \to\to Z\ , then\ XW \to\to Z - (YW)$

M7 Complementation

$If\ X \to\to Y\ and\ Z = R - (XY), then\ X \to\to Z$ [24, 130] or

$If\ X \to\to Y\ ,\ then\ X \to\to R - Y$ [18, 80]

FM1 Functional dependencies and multivalued dependency

$If\ X \to Y\ ,\ then\ X \to\to Y$ [18, 80] [19, 5]

FM2 Mixed pseudotransitivity

$If\ \{X \to\to Y\ , Y \to Z\}\ then\ X \to Z - Y$ [18, 80] [19, 5]

The proof of inference rule FM2, the mixed pseudotransitivity can be found from the source above or from the original publication of C. Beeri, R. Fagin and J. Howard: "Complete Axiomatization of Functional and Multivalued Dependencies in Database Relations" [18, 80][19].

Complementation and projectivity with multivalued dependencies is not included in the listed inference rules of the functional dependencies.

Multivalued dependency is at the same time a functional dependency [18; 19]. The inference rules for functional dependencies can be used with these inference rules.

Coalescence is added to these rules with the already listed reflexivity and FM1 [24].

C1 Replication

$If\ X \to Y\ then\ X \to\to Y$ [24, 132]

C2 Coalescence

$If\ X \to\to Y\ and\ Z \to W\ , where\ W \subseteq Y\ and\ Y \cap Z = \emptyset\ , then\ X \to W$ [24, 132]

Rules from M1 to M7 are enough with multivalued dependencies and the rules from C1 to C2 can be used together with the inference rules of the functional dependencies [24, 133]. The FM1 and FM2 rules can be used as well.

These inference rules together with the inference rules of the functional dependencies are used to infere multivalued dependencies.

2.3.11 Join Dependencies

A join dependency joins the attributes of the relation. A joined relation is equal to the join of the projections of the attributes.

A join symbol is usually a bowtie sign, ⋈. A natural join is a union of tuples of relations R1 and R2. A join is as follows with the ordered pairs as previously with relations R1 and R2 and with a scheme of a relation α(R). With a ternary relation (and with subsets if needed) [8, 25]:

$$R = R_1 \bowtie R_2 = \{<x, y, z> \mid \textit{for some } x, y \textit{ and } z, <x, y> \in R_1,\ z \in R_2 \ \ \textit{and} << x, y>, z> \in R\}$$

$$\alpha(R) = \alpha(R1 \bowtie R2) = \alpha(R1) \cup \alpha(R2)$$ [6, 7]

A join is effectively a multivalued dependency as was shown in the original publication in 1977 preceding the domain-key normal form [28]. A multivalued dependency on the other hand is not necessarily a join dependency.

A multivalued dependency X→→Y holds in the relation R(X,Y,Z) if and only if R is the join of it's projections R1(X,Y) and R2(X,Z) [17, 4] [28, 5].

Lossless-Join Decomposition

If joining with natural join two attributes between two relational tables results excess rows, the two tables are lossy. A composition is a lossless-join decomposition if at least one attribute of the decomposed relations are dependent on the same attribute in both relations [5, 309].

2.3.12 Inclusion Dependencies

If the values of a column are included in another column, the columns are said to be inclusion dependent of one another. For example the domain attributes are like this. In relational model of IBM (RM) v2 the inclusion dependency has been assumed between foreign keys and its target primary key or between a union of all the foreign keys of the domain [13, 273].

2.3.13 Null Values

Null values are usually attributes not yet set. They are waiting to be given a value [5, 71]. Nulls can be placeholders and they can be set to mean different properties. If an attribute determines another attribute, intuitively the determinant can not be null when setting the attribute it determines. Other way of saying this is that the dependency of a value has to exist until the value can be set.

Usually null is thought to be either:

1. Non existent value or
2. Unknown

Null is used as a placeholder of a missing value.

Information about IBM DB2 implementation was explained in 1990 [13]. In the explanation, E. F. Codd encourages to interpret missing values by what the missing value means in terms of program functionality and suggests a possibility to define the meaning by the application programmer. Two kinds of placeholders for missing data can be used: ***missing-and-applicable*** or ***inapplicable***. These are abbreviated by A and I placeholders. [13, 197, 174]

Attribute can be ***withheld*** if a user does not have the permission to the data and the data would be shown as null values [14, 249].

In database software, null-placeholders are not interpreted in the type of the attribute. They are treated differently. Without a pre-set placeholder functionality of the database software, only the first null value would be different from the other null values. First added row may succeed with null value. The second and further additional added rows with a null value would fail because of the dependency rules and the data not being unique.

In set theory, an empty set is included in all sets and a nonempty set can not be included in an empty set [8, 11] . The determinant has to occur before the item dependent on it.

Inapplicable data

Inapplicable data is under a condition preventing the setting of the data. When a constraint restricts the altering or inserting the data, the data is said to be inapplicable. A constraint can be a database constraint and it can be any constraint outside of the database constraints. Some constraints can be programmed to the database with procedures and triggers and different referential integrity policies can be se as is described later in Chapter 2.7.

A usual example of an inapplicable data is a spouce relation [14, 248]. An attribute should always return a truth value of a condition. A person can not have a spouce if he is not married and the spouce value is inapplicable within the attribute without a relation to a marriage. 'Spouce' would imply marriage.

A 'married' truth value attribute could be set and 'spouce' could be set to be dependent on it. 'Spouce' attributes value can not be set before the dependency is added and the database software should restrict setting a value to spouce before setting a value to attribute 'marriage'. 'Marriage' determines 'spouce'.

Not-null constraint

Usually in database software and in standard SQL a NOT NULL constraint is used to restrain addition of a set of values if the values are in relation to a NOT NULL constrained attribute and it's values are empty. The functionality seems to be similar in IBM DB2 softwares I -placeholders [13, 174] except that the placeholder is usually hidden from the user and the constraint is set to the schema and to all of the attributes values, not to the individual values.

To aid in database design it is usually useful to set all the attributes to NOT NULL and remove the constraint if it's certain that the attribute can have a null value.

Primary keys and foreign keys

In DB2 as in many database software products, primary key can not have a null value of any kind [13, 176]. Foreign keys including any or all composite attributes can have a null value [13, 176]. This is the case with the most database software, for example PostgreSQL, Microsoft SQL Server, SQLite and many others. The primary key a foreign key relates to can not be null. Candidate keys and superkeys can have a null value.

2.4 Integrity Constraints

The used integrity constraints were introduced in Chapter 2.1.5. As was seen in the previous Chapters 2.3.12 and 2.3.13, the empty value has a special meaning in the relational model. The data should be restricted with the constraints of the database to keep the data consistent.

Relational Integrity

Relational integrity constraints have to be used in INSERT, DELETE and UPDATE operations. Relational integrity is in the data representation when the data is read from

its form. Relational integrity constraints are used to decide on individual attributes in the transactions if they are valid for transactions. INSERT, DELETE and UPDATE are done in SQL to one relational table at a time. In the schema, transaction has to be identified as a transaction modifying data.

Classification of constraints

The methods to ensure data integrity of input data are functional dependencies and normal forms. Normal form is a pre-defined schema description of the data model. Functional dependencies with multivalued dependencies can be used to verify the dependencies of the values and ranges of the attributes. A classification of constraints is in Figure 1 [18, 12].

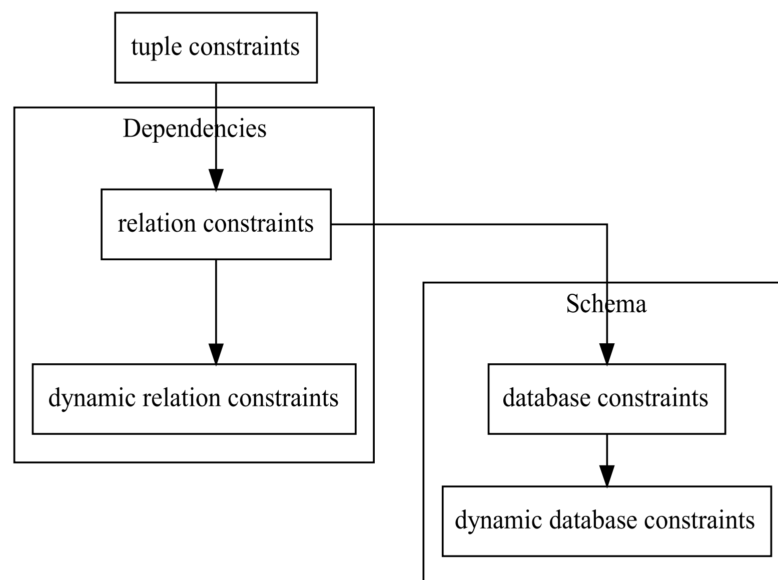


Figure 1: Classification of the constraints

The term dynamic relation constraint is used to represent truth -valued attributes and the term dynamic database constraint is used to represent the attributes under the database constraints [18, 12]. Every dependency is in a form of an attribute.

Types of algorithms needed

There are two possibilities. The functional dependencies and their properties can be read from the schema or the schema can be synthetized from the definition of the functional dependencies.

Different algorithms have been developed to form the closure structures of the dependencies and to find if a particular dependency is included in the closure. These algorithms are used in database design tools and as an aid in designing the schema.

It is interesting to see if the declaration of functional dependencies in the declaration of the variables is enough for the application functionality. Information is needed on what other properties have to be declared in the schema. In addition, an algorithm reading the schema and actually verifying the referential integrity is needed.

The schema provides the structure to declare dependencies and a form to use them efficiently. The normal forms should be used in designing the model of the structure. The relational model of a relational database schema can be used in desinging the application schema.

Data catalog

Originally the schema is meant to be stored outside of the application in a data catalog of the relational database to be used by anyone outside of the scope of the application [13, 5; 5, 67]. The schema is applied to the input data to maintain the data integrity by the database software. Schema descriptions of different database software vendors are in different form, there is no standard and sometimes a description can not be get from the database software. There are different methods, for example in Oracle RDBMS software a SQL procedure DESC is available giving descriptions of tables.

2.5 Empty Values

Empty values are important in determining if the value is valid in terms of it's dependencies. In the application concept, variables may occur in different order. By definition the determinant has to occur first meaning it has to be present in the same transaction.

If a dependency of an attribute value is not found due to an empty value with a ***not null*** attribute constraint, the attribute value cannot be used because its dependency is not met. Not null constrained variable without a value may not occur in a relation. This means additionally ***primary keys*** beause they can not be null. If the dependency is an ***inapplicable*** value, the dependency is not fulfilled. For example in the spouce example, without a marriage, the 'spouce' cannot exist without the 'married' dependency. If the dependency is to a ***missing-and-applicable*** data, the dependency is not fulfilled with that value.

2.6 Schema Attributes

An attribute defines allways a condition having a truth value. The constraints represented in previous Chapter 2.4 restrict the order the attributes may be used. A

serializable set of attributes of a transaction is the set of attributes a transaction can have. A schema may define many independent schemes consisting of attributes to be used in transactions.

2.6.1 Schema Attribute Subsets

An example of two schemas is illustrated in Figure 2 with imaginary transaction attributes.

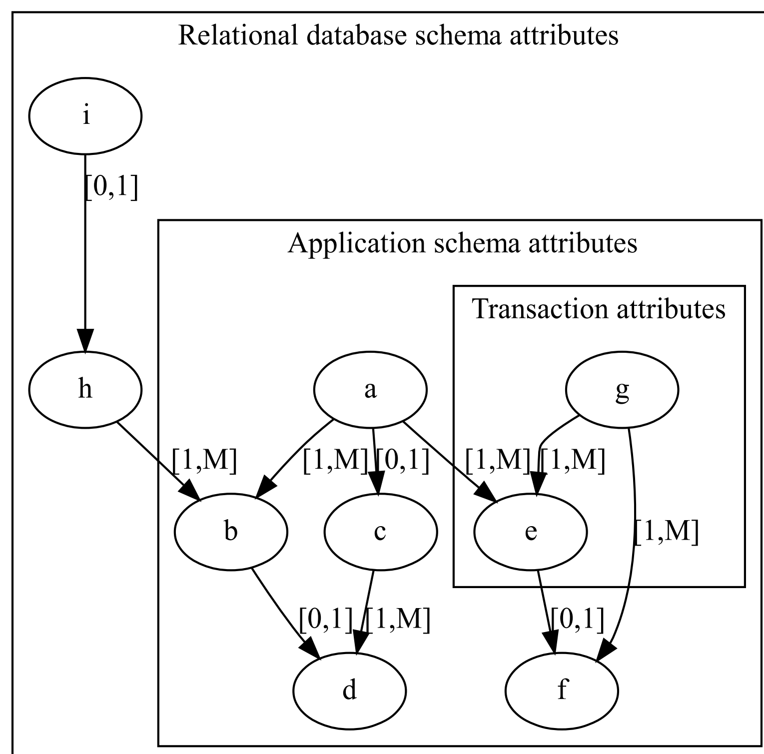


Figure 2: An imaginary example of the schema in Jacobson notation

Hypergraphs are sometimes used to represent attributes between relational tables. The attribute connecting the attribute sets (or relational tables) are common attributes to both attribute sets.

In hypergraph of the Figure 3, a capital letter denotes a determinant of a multivalued dependency and a smaller letter a functional dependency or a transitive functional dependency.

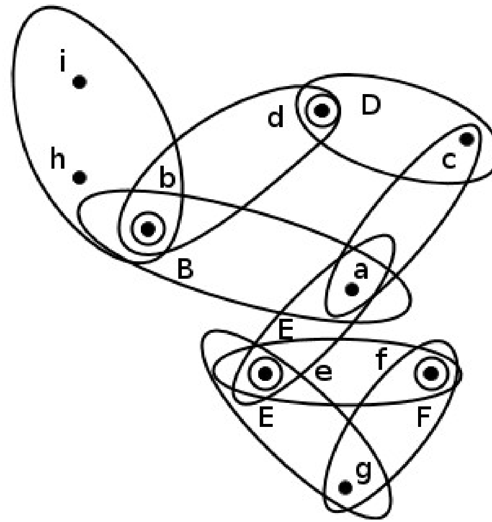


Figure 3: A hypergraph of the example attributes of the schema

The smaller letter symbol is the determinant giving the direction to the imaginary axel going trough the attribute sections or relatinal tables. The imaginary arrow would represent the preferred time variance of the occurrence of the variables, from the most important determinant towards the least important determinants. A dot with a circle around it denotes a multivalued dependency. A dot denotes a functional dependency. The imaginary arrows or axels are the dots and the dots with a circle and the direction is from a smaller letter to a capital letter.

2.6.2 Transaction Order

The domain integrity may be violated with a null value of the foreign key if the foreign key is allowed to be null. Otherwice the input of the variables should be started from the most significant determinants first. In the example of the Figure 3, the direction is the direction of the imaginary axels of the hypergraph. From smaller letters to the capital letters. On the image 3, first transactions to {i, h, b} or {b, d} then to {a, B}. Independently first {e, f} then {a, E}, {g, E} and {e, F}. Independently {a, c} then {b, d} and {c, D} . From these, attributes i and h would be needed from outside of the scope of the application. The possibilities for the transactions are the attributesets in the braces.

In the example schema in Appendix 3, Schema to Use in Examples, it can be seen that the relational tables are independent entities allowing many applications many purposes when a null termination is allowed in the multivalued dependencies.

The order of the transaction of the application concept is left to the application developer. The database schema should be read in designing the application and some of the constraints have to be taken into consideration when designing the SQL statements of the application.

2.5.3 Domain Integrity

The domain integrity should be taken into consideration in implementing the transactions. It is difficult to verify the domain integrity automatically in the application and it is easier with SQL statements. The domain attribute values should be from the set of the multivalued determinant. A good idea would be to use a SQL SELECT clause when inserting or updating a foreign key attribute or other application functionality to ensure the correctness of the attribute value.

2.6.3 Additional Application Attributes

It is possible to add functionality to the application with extra attributes. Extra attributes can be used to extend the applications beyond the relational database schema attributes. An example where this is needed is the authentication of the user of the application. Many times the authentication is outside of the scope of the relational database.

Application functions can be implemented with attributes or system attributes. A functions success would result an attribute. For example the session can be represented as an attribute the other attributes are dependent on. In these cases the transitivity is a certain feature and it does not divide the attributes into their own tables as in the relational database. The extended attributes can be treated as multivalued attributes without allowing a null value.

All of the variables dependent on extended system attributes should be configured in the variable declaration. In addition, transactions could be set to be dependent on the extended attributes. Some attributes as the session could be set to be automatically the determinant of every attribute.

2.7 Other Relational Database Constraints

In addition to the relational model, some relational database management systems have different policies to manage the consistency with the integrity constraints. The policies define how the database should behave under certain conditions. The available policies differ from vendor to vendor.

2.7.1 Relational Database Integrity Policies

Relational database software may have different additional policies to aid in designing the database schema functions. There are number of methods to choose from how the domain attribute in two relational tables should behave. The default is to allow a foreign key to be null or partly null with their values being partially specified: missing-and-applicable or inapplicable [12, 4].

Modification policies by default forbids deletions and updates if it creates one or more dangling tuples [14, 323]. This policy is called *restricted* [12, 4]. *Cascading* policy would delete or update in addition all referenced relation tuples [14, 321, 323] [12, 5]. The *set-null* or *nullify* policy would set the foreign key values to NULL [14, 323][12, 5].

2.7.2 Other Constraints of Relational Database Software

The transactions can be set as deferring instead of immediate to let the transaction process first and verify the dependencies after the transaction [14, 323]. Also a check constraint can be used (SQL CHECK -clause) with for example a selection to implement additional constraints [14, 328]. There are also other methods to add constraints. Constraints can be attribute-based or tuple-based [14, 327]. An assertion policy of the SQL standard can sometimes be used to implement any kind of a constraint [14, 337].

Some relational database sofware use triggers with the domain attributes, for example Sybase and Ingres, the predescessor of PostgreSQL [12, 5, 9]. The triggers are procedures to be launched with events, with every update or deletion for example. At the same time, the word trigger is used in another meaning. A deletion of a trigger attribute would result the deletion of an attribute dependent on it.

The deletion is always easier. As in the concept, updates and insert operations are important. The applicable existence dependency is the blocked dependency. If an attributes dependencies are not met, its use is blocked.

3 Algorithms

The neededed algorithms find the dependencies between the attributes of the application schema to verify the consistency of all of the needed attributes before a transaction. Many algorithms have been developed to be used with the rules of the relational algebra and the algorithms can be used in many purposes. The execution time of the algorithms is important because some of the algorithms become too slow to execute or even impossible if the number of attributes increases.

3.1 Closure Algorithms

Functional dependencies can be derived from the other functional dependencies with inference rules. A very slow algorithm found in most relational database text books was introduced in Chapter 2.3.6. Many faster closure algorithms have been developed. Here is a list of some of them with references to the publications.

An optimized version of the algorithm can calculate the attributeset in time relative to the multiplications of the attributes n to the number of functional dependencies p in the schema, O(np) [18, 70]. It is also possible to include the multivalued dependencies and the resulting algorithm computes the closure in still polynomial time [18, 83]. For example an alternative algorithm developed by Finnish researchers computes faster in still exponential time [25, 14]. The publication included an example of generating dependencies with the inference rules and it's complexity was left out of the scope of the publication.

An important method of forming the closure has been the derivation of the attributes. The derivation is a commonly used method in computing. Derivation trees have been used in proving some of the mathematical concepts. Linclosure algorithm from 1979 computes the closure with time complexity O(n) where n is the length of input [24, 66]. The graph algorithms are capable of forming the closure as well. Directed graphs or DAG:s and derivation DAG:s (1980) were developed by D. Maier and published in 1980 with a description of RAP derivation sequences developed in 1974 [24, 56][24, 66][24, 12].

A linear time algorithm in Chapter 3.1.1 was introduced in 1979 with a derivation tree algorithm or "a tree model of derivations of functional dependencies from other functional dependencies" [10, 9]. The membership algorithm itself computes with complexity O(F) where F is the size of the set of the functional dependencies [10, 17].

The most recent algorithm, a schema sythesis algorithm or a method without a given computable algorithm or a pseudocode was published in 2013. It still references the older sources and finds an optimal minimal cover from different possible orders of minimal covers [20]. It references Bernstein synthetization algorithm.

3.1.1 Linear Time Membership Algorithm

A membership algorithm verifies if a functional dependency is included in the closure of the functional dependencies. From two versions of the next linear time algorithm published in 1979 the second version uses less operations [10, 15]. The algorithm is in Appendix 1, a Linear Time Membership Algorithm [10]. This second algorithm is based on the derivation tree algorithm.

The algorithm builds a linked list of attributes of the left side values of the dependencies and attaches a pointer to it to all of the functional dependencies having the attribute in it's left side. A counter is attached to each functional dependency to show how many left side attributes there are in the dependency.

When an attribute is added to the DEPEND in the FIND_NEW_ATTR -loop, the attribute is removed from all of the left sides of the attributes list of functional dependencies. Whenever a functional dependency's counter becomes zero, its right side value is added to DEPEND [10, 16].

A proof of correctness of the algorithm can be found from the original publication [10, 16]. The algorithm is proved correct with induction of the length of the longest distance from the root to a derivation trees leaf and its complexity is shown to be linearly proportional to F, being O(F), where F is the count of loops of FIND_NEW_ATTR, the count of the functional dependencies.

The purpose of the algorithm is in the automatic schema synthesis. Membership algorithm can be used in redundancy tests and in finding the keys with the closure [10, 19]. In the original source, a 3NF schema synthesis algorithm is presented [27].

3.2 Multivalued Dependency Algorithms

Multivalued dependencies are at the same time functional dependencies. The foreign key may be multivalued in a domain where the same attribute of the domain is a key in another table.

If only functional dependency closure is needed, it can be derived with the previous

algorithms. If functional dependencies are needed with the multivalued dependencies, the inference becomes more complex because of the application of the multivalued inference rules. Instructions how to infere multivalued dependencies at the same time with functional dependencies can be found from publications [19, 12]. The rules of completeness can be used in analysing the algorithm [19, 6].

An algorithm to form a dependency basis of all multivalued dependencies with a closure algorithm of an attribute set is in Appendix 2, Multivalued Dependency Basis Algorithm [18, 77][19]. It calculates an attributeset closure and a set of multivalued dependencies, called a dependency basis. The algorithm computes the attribute set closure and the dependency basis in polynomial time. The execution time becomes slower when the number of the attributes increase. The proof can be read from the original publication [18, 83].

The algorithm forms the attribute set closure with the multivalued dependencies. It separates attribute sets to subsets. When changes are no longer made to the attribute set closure or to the dependency basis of the multivalued dependencies, the algorithm returns. For example the mixed pseudotransitivity (FM2, Chapter 2.3.10 page 21) is included in the rules to form the attribute set closure [18, 84].

This algorithm was published by the ACM in 1980, just a year later from the publication of the derivation tree and the linear time membership algorithm [19][10]. The original publication year of the algorithm with a publication is 1977 [10][19]. It was reviewed in 1989 with remarks to the other similar algorithms, for example from year 1979 in addition to the reviewers own from year 1983 [18][31].

3.2.1 Multivalued Dependencies in the Application Concept

Sometimes a functional dependency can be behind a multivalued dependency. The functional dependencies after the multivalued dependency should be added to the list of the determinants with the inference rules as is done in the algorithm in Appendix 2, a Multivalued Dependency Basis Algorithm.

3.3 Graph Algorithms

Graph algorithms preserve the order of the determinants and the determinants form a graph. The graph algorithms are as well able to form the closures of the attributes. Because the order of the determinants is preserved in the graph, the order can be used in a computer program to implement other functionality.

3.3.1 Derivation Tree Algorithm

The derivation tree is a graph model of determinants. It was used by C. Beeri and P. Bernstein in 1979 to prove mathematical properties of a linear time closure algorithm [10]. It occurred before in 1976 in the publication of P. Bernstein. The first publication year or author could not be verified because all of the references were not available. Derivation itself is a commonly used method.

A derivation tree (DT) is used in inferring the functional dependencies [10, 10]. A derivation tree is a method to verify if a functional dependency (FD) is included in a set of functional dependencies (F). A tree of dependencies is formed using the following three rules (with these exact words):

Rule 1. If A is an attribute then a node labeled with A is an F-based DT.

Rule 2. If T is an F-based DT with a leaf node labeled with A and the FD B1,. . .B, → A is in F, then the tree constructed from T by adding B1, . . . , B, as children of the leaf labeled with A is also an F-based DT.

Rule 3. A labeled tree is an F-based DT only if it so follows by a finite number of applications of Rules 1 and 2.

[10, 10]

The idea is to use only the augmentation and pseudotransitivity rules with a graph representation [10, 9]. All functional dependencies are found in the direction from the dependency towards its determinant. If in a dependencys tree X exists a determinant Y, then Y→X.

As a result, a tree with a root node A is formed. All of the leaf nodes of the root node form a list of attributes of the correspondent dependencies. The tree can be split into two sets of attributes from any nodes and the sets of the attributes can be reduced to individual dependencies with two inference rules [10]. The LHS can be split into multiple functional dependencies with decomposition and the RHS can be split into multiple functional dependencies with the union rule (or with the augmentation rule [10, 9]) [10]. A more in depth mathematical proof can be found from pages 10, 11 and 12 [10].

For example if the A:s determinants are {D4→D2, E1→D3, D3→B1, D2→B1, B1→A, C1→D1, D1→A}, the tree results for example the dependencies D4D2→B1A or D4D2→B1A, E1D3→B1A and C1D1→A. These can be reduced to the original form with the augmentation (LHS) and decomposition (RHS).

The tree is in Figure 4 where the root nodes are the attributes D2, D3, B1, D1 and A.

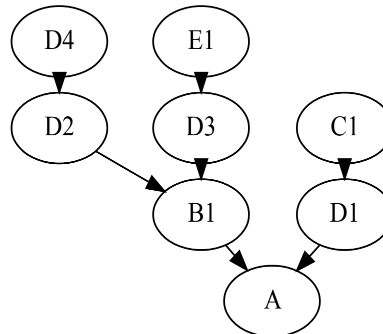


Figure 4: A derivation tree example

Another example of using a derivation tree can be found from the appendix of the publication of P. Bernstein, "Synthetizing Third Normal Form Relations from Functional Dependencies" from year 1976 [27, 19]. The example is not as clear as the previous one and it merely gives only a correct year to begin sourcing the original publication.

Completeness

The axiom system of the derivation tree with the following three rules have been proved to be complete:

Rule A1: (Reflexivity). If Y⊆X then X→Y.

Rule A2: (Augmentation). If X→Y and Z⊆W then XW→YZ.

Rule A3: (Pseudotransitivity). If X→Y and YW→Z then XW→Z. [10, 4]

The method of the derivation tree uses only the augmentation, pseudotransitivity and reflexivity and their derivatives only if they are needed.

Multivalued dependencies

The axiom system of the derivation tree method includes functional dependencies indicating that the multivalued dependencies are likewise included because multivalued dependency axioms are also complete.

From the application usage point of view, it is difficult to write the configuration without the multivalued dependencies. Extending the derivation tree with multivalued dependencies is needed.

Domains and null termination

Derivation tree does not include functionality known from the relational database software, the null termination of the relations and the allowed time variance of the variables. These have to be implemented in the algorithm verifying the attributes.

3.3.2 Directed Acyclical Graphs

RAP

RAP-derivation sequence is like the DDAG in the Chapter 3.3.3 below formed with the B-axioms as follows in the next paragraph. The DDAG is also a RAP-derivation sequence. The name comes from the order in which the following rules are applied (with these exact words) [24, 53].

1. The first FD is X→X.
2. The last FD is X→Y.
3. Every FD other than the first and last is either an FD in F or an FD of the form X→Z that was derived using axiom B2. [24, 53]

Examples can be found in the book of D. Maier, "Theory of Relational Databases" [24, 53].

A derivation sequence and the B-axioms

A derivation sequence is understood to be a set of functional dependencies belonging to the cover F^+ . Either the set of derivations is included in the cover or functional dependencies are added in the set from the previous functional dependencies in the set following one of the inference rules [24, 51]. Three B-axioms are used to derive derivation sequences and they are said to be complete axioms. In the source it is proven that Armstrong's axioms can be derived from these rules and because of this, the rules are complete [24, 51].

B1. Reflexivity: X→X.
B2. Accumulation: X→YZ and Z→CW imply X→YZCW.
B3. Projectivity: X→YZ implies X→Y.[24, 51]

The derivation sequences can be found with these rules [24, 51].

3.3.3 DDAG

DDAG is an abbreviation from derivation DAG or derivation directed acyclic graph. It is a directed graph without any cyclic reference from any node back to itself.

Rules to form the directed graph are (with these exact words):

R1. Any set of unconnected nodes with labels from R is an F-based derivation DAG.
R2. Let H be an F-based derivation DAG that includes nodes v1, v2, … , vk with labels

A1, A2, ..., Ak and let A1A2...Ak→ CZ be an FD in F. Form H' by adding a node u labeled C and edges (v1, u), (v2, u), … (vk, u) to H. H' is an F-based derivation DAG. **R3.** Nothing else is an F-based derivation DAG. [24, 56]

Initial node is the node added by R1 having no arrows pointing to itself. Essentially the DDAG is similar to the previous derivation tree algorithm without setting the root every time the attribute changes. Similar observations can be made as with the previous derivation tree. From the initial node, X→Y is true where X is the set of initial nodes and Y are some of the different nodes in the DDAG. With DDAG, the initial node is only one node while the derivation trees nodes can by the derivation tree rules be all of the root nodes. DDAG was introduced by D. Maier in 1980 after the derivation tree was introduced by C. Beeri and P. Bernstein with their other algorithms in 1979 [24, 70].

3.4 Join Dependency Algorithms

Multivalued dependencies are a good reason not to use entities or table names in the schema. The axiom systems presented do not know the join dependencies and the relations. The join dependencies introduced in Chapter 2.3.11 hold the relational tables together. It is possible to computationally show using an initial table and with a complex iteration if the attributes of the initial table after the iteration are in the resulting table. The chase algorithm in next Chapter 3.4.1 verifies if a join dependency or a functional dependency is included in the schema.

3.4.1 Chase Algorithm

A tableau is a matrix to be used in producing different combinations of initial values for different dependencies in a relation scheme [29, 7]. The purpose is to inspect different properties.

Chase algorithm computes an output value of true if the input join dependencies and functional dependencies are in the schema of all of the dependencies [18, 91]. Chase algorithm was first introduced by Maier, Mendelzon and Sagiv in 1979, "Testing implications of data dependencies". The algorithm was an exponential time algorithm and it has further been developed to compute in polynomial time. An early introduction of the polynomial time implementation can be found from the publication of C. Beeri and Y. Vardi, "The Proof Procedure for Data Dependencies" [26].

With functional and multivalued dependencies the chase algorithm is time consuming exponentially to the input variables size and because of this it is better to use the

functional dependency and multivalued functional dependency algorithms independently because they are faster [18, 89]. The basic chase algorithm is NP-hard.

Chase algorithm forms a tableau from the given join-dependencies to an initial one-table presentation. Functional dependencies are added similarly. Initially the tableau has distinguished values. Placeholders are used if a value in a table is not in the set of distinguished values of it's attribute.

Chase algorithm works by either equating variables or by adding rows to the initial table τ(J). With the join dependencies, if similar rows are found with:

$$\begin{array}{c} \text{For every join dependency } Y_i \bowtie ... \bowtie Y_l \text{, for all i,j, } 1 \leqslant i, j \leqslant l \\ \text{there exists a row} \\ l_i[Y_i \cap Y_j] = l_j[Y_i \cap Y_j] \end{array}$$ [18][26]

And with the functional dependencies, if similar rows are found with:

$$\begin{array}{c} \text{For every functional dependency } X \rightarrow Y \text{; for all i,j, } 1 \leqslant i, j \leqslant l \\ \text{there exists a row} \\ l_i[X] = l_j[X] \end{array}$$ [18][26]

They are added to the tableau τ(J) by combining the distinguished variables of the found rows. If a tableau has a final row not containing any placeholders and with only distinguished variables, the join dependency is implied and a value "true" can be returned.

An example of a description of the algorithm with a formal definition can be found from a book of J. Pareadens, P. De Bra, M. Gyssens and D. Van Gucht, "The Structure of The Relational Model" [18, 90]. More information on chase algorithm can be found from the book of S. Abiteboul, R. Hull and V. Vianu, "Foundations of Databases" [11, 175] and from the book of D. Maier, "Theory of Relational Databases" [24].

Chase is a tool to verify the database schema design. It can be used for many purposes [29]. In the application purpose it would be useless. It would be slow with large sets. It can prove properties of functional dependencies and multivalued dependencies with the joins. Chase can also prove the equivalence of different schema closures [29, 11]. The resulting tableau would be similar with equivalent closures of the two schemas.

4 Application

The chapter applies the algorithms and the background theory to the application concept. The chapter describes how the method was applied starting with the description of the possible algorithms, decisions made in the choosing the algorithm and with an UML diagram describing the algorithm. The algorithm in the UML diagram is further extended with multivalued dependencies and with the relational model. Finally, the compatibility with the normal forms is evaluated and the schema and the different schema files are described.

4.1 Integrity Verification Algorithm Comparison

The source material was used to form a scope of the known dependency algorithm concepts. Direct answear was not found how to verify the integrity of the transactions attributes and the theory had to be applied to the problem. Only some sources contain information about the dependency concepts. The approach is usually more towards the normal forms. A good source was a book of J. Pareadens, P. de Bra, M. Gyssens and D. van Gucht, "The Structure of The Relational Model" [18, 126].

4.1.1 Algorithms

Many found algorithms could be used to verify the referential integrity of the application concept. The most important criteria in the evaluation is the execution time because the algorithms are slow. In some occasions the software can do part of the work in the background before the application is used.

One criteria was the software library used. The library was able to read a tree structure into a form of a binary tree without being able to write a new tree. An application is more simple if it contains fewer parts.

The use of the algorithms is not used only in the purpose of the application concept. Researched algorithms are used in many purposes. A schema synthesis, the automated formation of the schema seems to be the most important reason why the algorithms were developed. One of the algorithms was a manual algorithm to be used in forming the closure.

4.1.2 Comparison

All except the graph based algorithms use the dependencies in the form from the determinant towards its dependent attributes and the dependencies are stored as attribute pairs. The need to verify the existence of the attributes determinants is in the opposite direction from the dependency towards the determinant.

To find fast the determinants of an attribute, a graph based solution is the fastest and simplest alternative. A graph saves the paths of the determinants and stores the fastes way to find them. A graph based algorithm forms a closure as well.

Closure algorithms exist and they can be used for the purpose. If set operations are needed, they slow down the program. To find determinant pairs from the sets would add complexity and increase the search time. From the closure algorithms, the most usable is the closure algorithm with the multivalued dependency basis. It is slower than the others. As was shown, the multivalued dependencies are sometimes required [19].

A short description of all of the sources of the six different algorithms to form a closure of the functional dependencies is in the Chapters 3.1 and 3.3. In addition to the previous example of a slow closure algorithm in Chapter 2.3.3, these are the better alternatives. One of the listed closure algorithms is a multivalued dependency basis algorithm. Two graph based algorithms are listed in the Chapter 3.3.

4.1.3 Algorithm Decisions

The derivation tree can be implemented with a binary tree and the already existing software can be used. Derivation tree uses different roots compared to DDAG and it is for this reason a better alternative. The derivation tree was chosen. An already proven mathematical theory exists to confirm the usability.

In short, the derivation tree algorithm seems to be more developed than the DDAG or RAP algorithms because of its roots. Both have a complete axiom system [10, 4][24, 51]. From these two axiom systems, augmentation of the derivation tree is missing from the B-axioms.

4.2 Algorithms of Application Concept

The design of the program applies the theory. The chapter describes the design decisions made starting from the configuration of the attributes and ending in the UML diagrams describing how the relational model known from the relational database

management systems can be used with mutlivalued dependencies and the derivation tree algorithm.

4.2.1 Attribute Declarations

Because of the augmentation M2 and F4, the attributes of the application can be declared one by one. Because of the additivity M3 and decomposition, the determinant attributes determining the attribute can be in a list format.

A determinant may be a composite key. An attribute may be used only as a determinant or as a composite determinant. The composite is constructed with multiple determinant declarations. If a composite key is read, it is known that another key is required from the determinants of the attribute.

4.2.2 Derivation Tree with Multivalued Dependencies

The derivation tree algorithm in Chapter 3.3.1, page 35 does not take into consideration the multivalued dependencies. The multivalued dependency is multivalued only to one direction. Towards the determinant it has only one value and this is the direction of the the derivation tree. The multivalued dependency can be added to the tree structure as a node. When reading the tree, the configuration file can be read to know if the node is a multivalued dependency. The nodes of the path have to be read in the original order, always the next determinant of the previous attribute.

In the relational database, the existence of a foreign key is a condition to the existence of the respective key value within the same domain. This is because of the domain integrity. More mathematical evaluation with three numbered sections can be found below with two evaluating definitions. The use of the empty values may need a mathematical proof if the properties are not well known. The use of the multivalued axiom FM2 may not be clear with just mentioning it. The idea is to traverse the referential path from determinant to determinant in order. The three sections are numbered below.

Definition: If Z, Y and X are *singular (disjoint)* and *Y is not empty*, a functional dependency $Y \rightarrow Z$ exists and a multivalued dependency $X \twoheadrightarrow Y$ exists, then $X \rightarrow Z$.

Proof: If Y was the last added determinant after Z and X is multivalued to Y, with the mixed pseudotransitivity rule FM2 from page 21, $X \twoheadrightarrow Y$ and $Y \rightarrow Z$ implies $X \rightarrow Z - Y$.
1. If $Y \notin Z$, $X \rightarrow Z - Y$ becomes $X \rightarrow Z$.
2. If $Y \in Z$. If Z is replaced with WY: $X \twoheadrightarrow Y$ and $Y \rightarrow WY$ implies $X \rightarrow WY - Y$.

With F4 decomposition:

a) $X \twoheadrightarrow Y$ and $Y \rightarrow Y$ implies $X \rightarrow Y - Y$. This results $X \rightarrow \emptyset$, Y and $X \twoheadrightarrow Y$. Because an empty set is included in all sets and because of reflexivity of X, the result is X, Y and $X \twoheadrightarrow Y$.

b) $X \twoheadrightarrow Y$ and $Y \rightarrow W$ implies $X \rightarrow W - Y$. The result would be $X \rightarrow W$ because $Y \notin W$.
Also with the union inference rule. $X \rightarrow \emptyset$ and $X \rightarrow W$ becomes $X \rightarrow W$.

Because the derivation tree attributes are single at the right side, Y is never a subset of Z, $Y \notin Z$. The right hand side is decomposed and the attributes can be evaluated individually.

The same proof can be applied to the next multivalued dependencies from the last multivalued dependency with replacing $X \twoheadrightarrow Y$ and $Y \rightarrow Z$ with $X \rightarrow Z$. In the derivation tree, the found determinant has to be added in parallel with the previous as long as a functional dependency is found and there are no more determinants. □

The same can be verified with M5, $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $Y \twoheadrightarrow Z - Y$. With singular nonempty values, $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$ becomes $Y \twoheadrightarrow Z - Y$ and with removing reflexivity $Y \twoheadrightarrow Y$, the result is $Y \twoheadrightarrow Z$.

In fact, in all of the possible transactions where multivalued dependency may be needed in verifying the integrity: delete, update and insert operations, the attributes affected are one relational tables attributes only. All of the attributes after the multivalued dependency are in an another relational table if the schema is of a good design. The $X \rightarrow Z$ is needed only to identify the domain and if only an identification is required, a primary key value is enough in the same tuple with transitivity from Z without necessarily verifying the existence of all of the other variables of the other relational table after the multivalued dependency.

The empty value may have to be examined more. Using multivalued dependencies with the derivation tree algorithm in the case when the Y value is empty.

Proof: **3.** If $Y = \emptyset$, $X \twoheadrightarrow Y$ is true because an empty set is a part of every set. With multivalued complementation $X \twoheadrightarrow \Omega\text{-}Y$. If in the previous formulas Y is replaced with $\Omega\text{-}Y$: $Y \rightarrow Z$ is not true, $\Omega \rightarrow Z$ from Y is not true, $\Omega\text{-}Y \rightarrow Z$ is not true and $X \rightarrow Z$ is true. $X \twoheadrightarrow Y$ is still a multivalued dependency, Chapter 2.3.9 page 20 [18, 78]. $Y \rightarrow Z$ is not true. □

In verification with the dependencies of the variables with a schema, if the schema is in 3NF, all transitive dependencies have been removed within a table. If the schema is in BCNF, all of the determinants have to be at least a super key.

More about the special multivalued dependency $\emptyset \twoheadrightarrow Y$ was referenced from the early

definition of the domain-key normal form [17] [28, 11]. If Z→→Y and Z=∅, the multivalued dependency has to be either trivial or a cartesian product R(Y,Z) of R(Y) and R(Z). A join forms a cartesian product. For example a relational table is a cartesian product. A join or a row of the table joins the identity between the values.

If Ω from a relational table of attribute Y can identify Z in another table or if X can identify Z, null could be allowed with the referring Z value. Otherwice the tuple with the empty foreign key can be only left “dangling” with a null value and may be updated later [14, 323]. This would still violate the domain integrity. The other attribute in the domain has to be either unique or a primary key [14, 319]. The third condition in the proof would violate the referential integrity [14, 323]. Both domain and referential integrity is violated if a null value is inserted. Referential integrity allows the violation if the foreign keys are allowed to be null [5, 72]. This is the default behaviour with the relational database software. The unidentified values cannot be verified with the determinants if the determinants cannot be identified.

> *If a multivalued attribute is empty and it is allowed to be empty, the computational referential integrity checking from the multivalued dependency should be stopped to the foreign key if the next determinant is not found from the transaction definition and from the attribute values.*

An example

In the previous example in Chapter 3.3.1 and in Figure 4 the following attributes were used with the following functional dependencies: E1 D3 → B1 A , C1 → D1 A and D4 D2 → B1 A. If only one path is chosen, for example the path E1 D3 → B1 A and it is assumed that the functional dependency is instead a multivalued functional dependency: E1 D3 →→ B1 A, with substituting Y with B1, Z with A and X with D3 in the following sentence from Chapter 4.2.2 in page 42: If Y→Z exists and a multivalued dependency X→→Y exists, then X→Z, becomes: D3 → A (the result of the formula), B1 → A, D3 →→ B1 and E1 → D3.

The order of verifying the attribute A:s integrity is as follows: First A, then B1. After B1 a multivalued dependency is known to exist with D3. Now it is known that D3 → A, D3 →→ B1 and because D3 is a determinant of a multivalued dependency, it is a domain attribute. D3 is a foreign key in the same relational table with the attribute A and the domains determinant attribute D3 is in a relational table with E1. As in the referential integrity in Chapter 2.1.5, the foreign key may be null with A and B1 if it is assumed to be filled in later. If D3 is null, the verification is stopped. This is the null termination. If D3 is not null, integrity of D3 must be verified and after this, the integrity of E1. All of the remaining paths from the attribute A should be verified similarly.

4.2.3 Null Termination

It is possible to insert values later by filling in the missing domain attribute field as was described earlier. The attributes of the entities or the relatinal table have to exist together. The null termination with the domain attribute is usually described to be an agreement. In the research, a good mathematical proof was not found. As in domain integrity, a foreign key may be null violating it's referential integrity [14, 323].

From the example, all possible occurrences in time variance are: first independently B1 or D4, after this A if B1 was inserted and D2 if D4 was inserted. If D2 is the determinant of the B1 and this B1 was inserted, as a domain attribute, D2 has to be filled in in both of the tables.

An attempt in 1984 was to connect the functional dependencies to propositional logic [30]. The equivalence between the propositional logic and the functional dependencies was shown only syntactically and the system was not sound or complete as was mentioned in the same publication [30, 5]. With the rules of the publication, the previous examples dependencies would become conditional with the proposed 'or' and this is againts the rules of the functional dependencies.

4.2.4 Integrity Constraints

The verification of the attributes referential integrity was described in previous chapters. The attributes are the transactions attributes and their determinants. In following the domain to another relational table as in the previous chapter, the determinant of the key attribute of the domain has to be verified with each variables other declared integrity constraints.

As in the entity integrity, a primary key or a composite primary key has to be chosen from the candidate keys to provide identified values for transactions. A configured key could also be a superkey. Attribute declared as a key may not be empty. The concept can not automatically ensure that unique primary keys are used. Uniqueness should be provided with the application functionality, otherwice the transactions to the relational database may fail.

Null integrity has to be declared in the configuration file. An attribute may have a null value as discussed earlier in Chapter 2.5 page 27.

As in the domain integrity, the foreign key has to be a value from the values of its referenced key. The value has to be the value referenced. The attribute the foreign key

refers to has to be unique or a primary key [14, 319]. Database management systems allow the foreign key to be null. This is a special condition of the domain integrity violating the referential integrity. A null terminates the relation. If the foreign key is not null, it has to be one from the key values. Sometimes this can be ensured only by selecting the foreign key value from the determinants. The domain integrity has to be ensured with the applications own functions or with relational database triggers or other policies. An example would be to select the foreign key value from the values of the determinant of the domain in a join between two or more tables.

4.2.5 Writing Derivation Tree

Every attribute "A" read from the set of attributes forms its own derivation tree. If the new attribute is found in the derivation tree, it is set as a new root of the next derivation tree. Writing the derivation tree is illustrated in Figure 5.

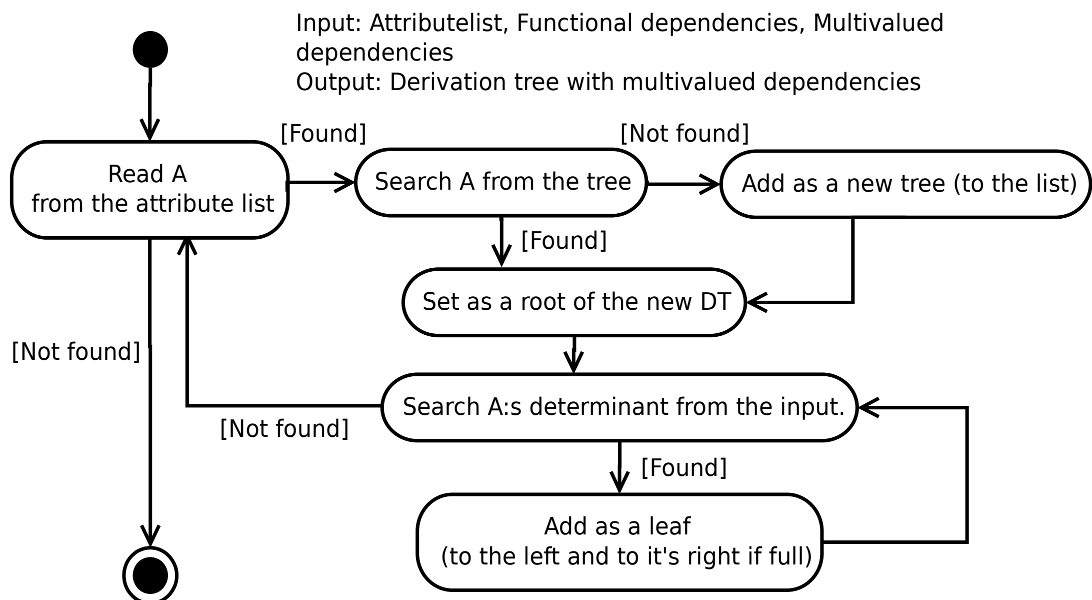


Figure 5: An UML activity diagram of forming the derivation tree

All of the determinants are in the derivation tree after the dependent attribute. After the derivation, all of the determinants of the attribute are found from the tree structure after the attribute.

Derivation with the Multivalued Dependencies

The determinants of the multivalued dependencies are added as normal determinants. Adding the multivalued dependencies with the FM2 rule in Chapter 2.3.10 would add redundant functional dependencies to the tree at a lower level than the determinants

after a functional dependency. The three rules to form the tree are complete without the multivalued dependencies.

4.2.6 Verifying Attribute Integrity

A mathematically correct algorithm does not take into consideration the time variance and the different entities of the attributes and the possible order of using them. Mathematically all of the determinants including the multivalued determinants have to be verified to exist.

The process to verify the referential integrity of the attributes is in Figure 6. This is the relational model the relational database software uses. The foreign key of the domain is usually allowed to be null. The value is assumed to be filled in later if it is needed. This is the null termination of the relation. Foreign keys do not exist in the derivation tree. The domain attribute is recognized to be a domain attribute if it is in a multivalued dependency to some determinant. All of the attributes are verified from the given attributes of the transaction.

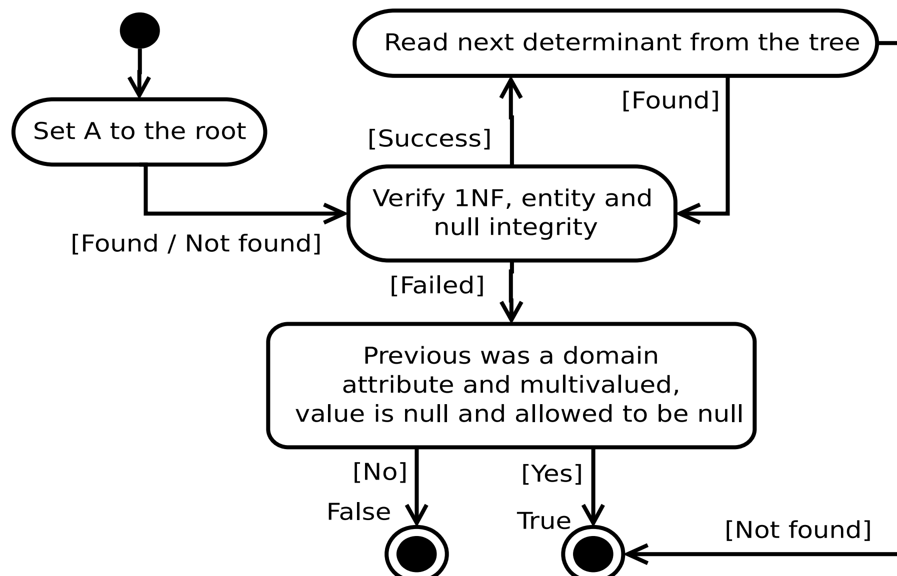


Figure 6: An UML activity diagram of the consistency verification

The algorithm in Figure 6 stops if after a domain attribute, the determinant is not found. This is the null termination. Otherwice all determinants have to exist. Null termination is in effect if the attribute has a multivalued dependency towards the next determinant.

Every path from the derivation tree root has to be verified in the order of the determinants because of the multivalued dependencies. Description can be found from the Chapter 3.3.1 and 4.2.2 on pages 35 and 42.

Inside the transactions attributes, all of the determinants of the attributes have to exist. The domain attribute is recognized to be a domain attribute if it is in multivalued dependency to some determinant.

4.2.7 Technical Implementation

Consider the derivation tree is a binary tree where the determinants of an attribute are on the left side. The first leaf to the left has in the right side (and all the nodes in the right have in their right side) all of the direct determinants of the attribute A. All nodes have their determinants in the left side. If a multivalued dependency occurs between nodes, the multivalued rules of the verification can be used with the leaf in the list starting from the node on the left. The data structure is given in the Figure 7.

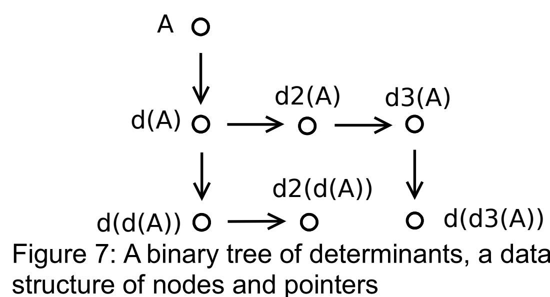


Figure 7: A binary tree of determinants, a data structure of nodes and pointers

In iterating through the datastructure, every time when moving to the left, the transitivity is increased. Because the referential paths have to be read in order, the order is first left, then right. All of the attributes of a transaction must be checked. A failing integrity constraint should prevent the transaction and result an error message describing the missing values.

The domain integrity is difficult to be verified without being able to select from the determinants of the multivalued attribute. Again, this would be easier with a selection in the SQL-statement. The checking of the domain integrity is left to the programmer or to the program designer to implement in an application.

The algorithm verifying the attribute integrity does many reads from different files. The transaction definitions with the attribute names of the transactions are in its own file. Derivation tree is in it's own file and attribute definitions are in the configuration file. The input variables are read according to the information from the transaction definitions. This results four different files or input streams. Indexing the files attribute locations first with a linked list before using the file reduces read operations. As test results show, forming the derivation tree is fast.

An option if null termination is used could be set. Without the option, all of the determinants would be verified as with the original derivation tree.

JSON representation format was chosen as the format of the derivation tree because of its support in todays web browsers. Javascript is included in all of the web browsers from the desktop computers to the mobile phones and smart TV:s. Javascript libraries provide support for the Javascript Object Notation and JSON is widely used in for example document databases. A JSON representation of the derivation tree of the example in Chapter 3.3.1 page 35 is in Listing 1 below.

```
{
   "A":  {
          "B1": {
                "D2": {  "D4":  { } },
                "D3": {  "E1":  { } }
                },
          "D1": { "C1": { } }
   }
}
```

Listing 1: A derivation tree example from the figure in page 36 in JSON notation

The derivation tree format cannot include other definitions of the variables because they would be mixed to the variable names. Reserved names would add complexity and the usage of the configuration file would become complex.

JSON format was not easy to read with the test software and writing the format was not simple. An error in JSON configuration can prevent the program to execute correctly. A separate syntax checker may be needed or the JSON formatted text should be machine written.

Derivation trees can be formed by writing it to a file during the program startup. It is possible to compare the timestamps of the previous derivation tree file to the configuration file and omit the new derivation tree creation if it is not necessary. In the best case, when an own process is given to a requesting call, all of the files would be ready in memory in usable form with the pre-built attribute location indexes.

When an attribute has been checked, information of it can be saved in the derivation tree to remember it. When the same named attribute is encountered, the information of the already made verification can be used to omit a new verification of the same attributes. The same is achieved if for every transaction, the derivation tree is a ***read-attributes-only-once*** tree. When an attribute's integrity is already verified, integrity verification is done also to the next attributes integrity tests for the verified attributes.

This applies to only one transactions attributes at a time. Also, in setting the new root, the new root has to be found every time even if it was already verified once.

4.2.8 Compatibility with Normal Forms

The derivation tree algorithm with multivalued dependencies is not able to synthesize the schema. It could synthetize the schema up to the 3NF if it was made to do so. The schema syntax is nevertheless compatible with the other normal forms.

1NF and 2NF

In the algorithm, checking the attribute values form to be the form of the type provides atomic values and functional dependencies are added as in 2NF to the configuration files.

3NF

If the relational database schema is in 3NF, it should not have transitive dependencies in its relational tables. Every time the transitivity increases, the transitive determinant has to occur in two different tables and form a domain.

BCNF and 4NF

The key information of 4NF and BCNF can be configured in the application configuration files and the entity integrity can be verified in the runtime.

5NF

After a 5NF decomposition, the same determinant is in both of the decomposed tables and does not cause any exceptions in the mathematical rules of the functional dependencies of the algorithm.

DK/NF

It could be anticipated that in the domain-key normal form where the form is a concequence of the domain constraints and the keys and if every dependency between the relational tables is a multivalued dependency, the algorithm could be used if the form of the database is in the domain-key normal form. In short, the DK/NF form can be used with the schema. The domain constraints and the key constraint can not be verified with the application concept because the rows of the relational database are not available in the verification as they are with the relational databases. The DK/NF form can be used because the schema has the required elements.

4.2.9 Using Joins and Selections

With relational databases it is possible to join tables with key values in different table combinations. It is possible to select from the join with an attribute value. The identification of the relation in data modification of a transaction can be get from a selection of a join between two or many relational tables. In this case the attribute value of the selection can be a determinant identying the relations to be modified. The join dependencies can be used in the integrity verification if the names of the attributes are known.

> *Attribute names in the selections should be the same as in the relational database schema to verify the referential integrity of the attributes of the transaction and to use the same names in the SQL statements.*

This is not restricted and without any automatic program writing the configuration file, attributes can be chosen erroneously without a notification from the software. It is possible to include the attributes in the configuration as only attribute names without a value. An example is in Listing 2 using the example schema in Appendix 3. In addition in the example, the domain integrity is ensured with joining the tables together with the domain attribute. In the example, the attribute manager_id is the employee_id used in a different purpose. The example adds a new manager to every department in the location identified by attribute country_name. Country names are usually unique enough to do this or the attribute can be selected with a SELECT statement to get the correct value of the primary key. Primary key country_id is included in the statement.

```
-- Adding a new department manager with country_name.
UPDATE departments
        SET manager_id=$manager_id
        WHERE employees.@employee_id = departments.manager_id AND
                employees.department_id = departments.@department_id AND
                departments.location_id = locations.@location_id AND
                locations.country_id = country.@country_id AND
                country.country_name = $country_name;
COMMIT;
```

Listing 2: An example of an SQL statement configuration

A dollar sign '$' in the image represents a value from the application variable. The '@' sign represents a variable name without a value. These signs can be parsed with an application and the result are the attributes country_id, location_id, employee_id and department_id. The signs means that the integrity check can be omitted and it is trusted that the database has these values. At-sign '@' here represents the datastore

as an origin and dollar sign '$' represents the application. A third sign could represent the configuration file or other source. The hash sign '#' is used as a comment sign in some relational database software and its use is not wise. The entity or the relational table names are not used by the application.

The list of joins can be long. To use the attribute names in the integrity verification, the attributes joined together can be marked with an at-sign to be included in the verification as bypassed attributes.

For the SQL transaction to be valid, it should be ensured that the needed variables are in the WHERE clause and not in for example an ORDER BY -clause. Additionally the attributes should not be inside comments. To verify that the SQL-statements are correct at this point is left to the application programmer and the SQL-statements are not parsed. Parsing text files is slow and the attributes of the transactions definitions could be in a configuration file separately.

More at-signs signs mean more possible errors from the datastore because of the verification of the existence of the attribute is left to the datastore. On the other hand only an identification of the attributes of the transaction is enough. The success of the action should be provided and an error message could be sent if an error status can be get from the transaction. Otherwise the application should always check the success somehow with an another action.

4.2.10 Transactions and Repetition

The definition of a transaction has to include information if it modifies data to know that the integrity verification is needed. Otherwice, the origin has to be configured as in Listing 2. If the attributes are parsed from the SQL text, the application should be able to parse SQL comments.

Individual application programs should implement the correct order of the transactions with their application logic: For example with a series of web pages and with a Javascript application.

Transactions should return success or fail to be able to send a correct error message to the application user interface. Otherwise the application should verify the success with its other functionality.

To know what attributes are needed in repetitions, some kind of a relation is needed between the attributes. In HTTP GET and POST methods, the attributes are individual name-value pairs. They may be unordered if the application does not order them. A

name-value pair can have a range of different values containing an ordered list of the values of the attributes if this is the application protocol. One method to group together attribute values is to use a dot notation to represent the group and its attributes. HTTP supports this. Also a JSON aggregate can be used.

> *The item count of the attribute range has to be the same for all of the attributes in a transaction.*

Before every transaction, all of the individual values integrity has to be verified. If an attribute is multivalued, it may have repetition with all the different values of the other attributes of the tuple.

> *The same value can be repeated if the attribute is a multivalued attribute, a foreign key of the domain. The same value can not be repeated if it is a primary key, candidate key or a superkey.*

Attribute value ranges can be inserted in a repetition if the key and the domain attribute constraints allow.

4.2.11 Representing Attribute Values

The usual choice of today is to use the REST model (representational state transfer) where the information of the representation is delivered with the data to the user interface.

JavaScript libraries JQuery and AJAX can be used to get the needed variables from a server application to the user interface in some form. JavaScript is today the choice because all of the web browsers in the different terminal equipment supports it.

If the server software is used, predefined web pages can contain tags or other identifications to place the content of the variables with the needed repetition in columns and rows. For example PHP is a programming language using this kind of a concept to add tags in the web pages.

4.3 Schema

At the time of the writing, the structure of the schema is not defined yet. The schema is described in different files and they and some of the requirements are described next. A schema should describe the application attributes, the transactions and the derivation tree. The schema should be a tool to program the application.

4.3.1 Schema Definitions

As in in Figure 8, the schema definitions are divided into three files.

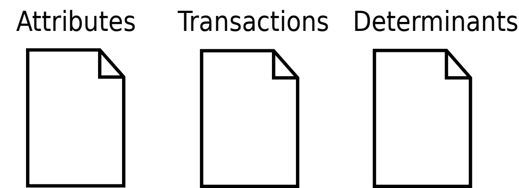


Figure 8: Application schema files

The definitions of the files are in the next chapters. The attribute types can be further extended into an own file. Further the roles can be in an own file.

4.3.2 Transaction Definitions

Every attribute can be declared individually as in a programming language. Mathematical rules do not show why relational tables or entities collecting together attributes should be declared in the application configuration.

Definitions of entities are not needed in the declaration of the variables. The entities help in reading the functional dependencies from the relational database schema if a relational database is used. The entities have to be declared in the declaration of the transactions as was defined in the previous chapters.

Definitions of transactions have to include: the list of the attributes of the transaction and an information if the attributes are modified in a transaction. The definitions should include a module name, return value name and a parameter text.

4.3.3 Determinants Definitions and Other Constraints

Application variable configuration should incude definitions of attribute constraints, for example not-null constraint and the information if the attribute is a key. The determinants have to be listed and multivalued determinants similarly. The derivation tree is derived from the attribute definitions.

4.3.4 Data Type Definition, Visibility, Origin and Role

Data types can be declared in the declaration of the variables or in a separate file. Regular expressions offer one possibility to check the form of a value. Usually it is important to verify the length of the variable separately to prevent memory leaks. To

distinguish the attribute values is to use the same attribute names everywhere where the attribute is the same.

It would be convenient to represent everything as a datatype to avoid unnecessary functions in the case when functions are needed, including type transformations. Polymorfism of different data types can be implemented with recursion. An attribute can contain an attribute name of the types basetype and both of these can be verified during the type verification.

An array can be specified with some kind of a method. It is possible to use a delimiter such as a comma between the different values. A bypass character is used when the delimiter is needed. The array can be a JSON array for example where the brackets declare the use of an array value.

Data may be originated from the application configuration, the user interface or from the database as was described in Chapter 4.2.9. Visibility can be restricted. These can be different configuration files to avoid unnecessary reads during the program runtime. Allowed role permits or denies the usage of the attributes. The schema files can be extended with extensions such as roles and types.The two extending files are shown in Figure 9.

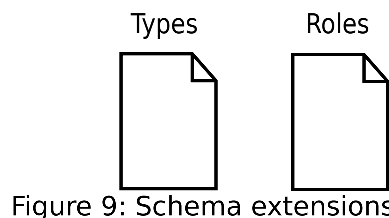


Figure 9: Schema extensions

A role can be a dependency to an attribute if the attribute is given from a trusted source. If the database is used with a username and password combination, the roles of the datastore can be used to finally permit or deny the transactions.

4.3.5 Error Message Definitions

As in the introduction, error messages can be carried as a type to the user interface. An example application schema can be found in Appendix 4. If an error variable exists, it can be carried to the user interface as an aggregate or as named variables and printed in its place in the user interface if the variable exists. The printing can be done with for example Javascript functions if an HTML user interface is used or with server software. The error messages may cumulate and many error messages may be sent if the transaction does not stop when the first error has been encountered.

4.3.6 Usability

As in a defined relation scheme, RS = { *Ω, ∆, dom, M, SC* } in Chapter 2.3.1, the application schema currently defines only one scheme:

Ω	*Attributes are defined individually*
∆	*The set of the domains of the attributes is the defined type of the attribute*
dom	*The function associating the domain to the attribute is the unique attribute names and the type*
SC	*The set of the constraints are the relational integrity constraints*
M	*The meaning is defined in the error messages*

All of the attributes are not connected together with the functional dependencies. The schema definitions could be divided into named schemes describing the entities as in the relational tables. The meaning with the entity on the other hand is not similar. Entity names would help in designing the schema. Entity names do not have a purpose in the application program as a function and the attributes would overlap in the schema. Attribute names can be represented with a dot notation where the first part represents the entity name. Some database management software uses this notation. Attribute names are not restricted.

The dom -function associating the attributes domain to the attribute should be a function describing the attribute. Now it is described as a type. Short named text descriptions can be added to describe every attribute. The set of constraints in the original meaning should be to describe the attribute with a sentence. The sentence should describe a condition [18, 5]. The meaning is now in the application schema constraints similarly as the database constraints are with the database management systems [18, 10].

Multivalued dependencies was chosen to be in the schema to be able to understand the schema structure more easily. Multivalued dependencies are between the relational tables. Removing multivalued dependencies with the inference rules cannot be expected from an application programmer.

4.4 Relational Concepts Summary

Three models of the relational integrity verification were considered:

- Mathematical: all determinants have to be verified
- Relational Model: null termination is possible at multivalued dependencies
- Third option: the most important determinant is enough to identify the selection

These options could be set as an option to the test program and the use can be tested in practice. Mathematically simply all determinants have to exist as in the original derivation tree system. In the relational model, the null termination is possible if the domain attribute is not yet known. The third option is only practical and from using of the SQL language as was shown. Only the most important determinant is needed in the SQL statement from the application.

4.4.1 Configuring the Attributes

The schema could be formed with an application user interface where the texts would ease the development and document the attribute definitions at the same time. The attribute definitions has to be more precise than with the relational database. Defining only the primary key is not enough.

4.4.2 Functional Dependencies

The configuration of the attributes can be based on the information, what information should be given from the application user interface. Thinking about the municipal services for example and the use of the paper forms and the fact that the electronic forms have not yet completely replaced the paper forms, the attribute configuration could be based on the information what attributes is required by the application. The attribute configuration should be based on what is needed for the purpose of the application and not reflecting only the relational database schema. The functional dependency is not only a formal method, it is a method to describe the attributes and the application functions. The original ideology in defining the meaning of the functional dependency may have been the same [18, 64][16].

4.4.3 Time Variance of the Transactions

The possible time variance of the different attribute subsets has been discussed in Chapter 2.6.1 and later with the integrity verification algorithms in Chapter 4.2.7. The domain attribute has to have a NOT NULL restriction if the order of the transaction is restricted. The domain attribute is defined in the schema with a multivalued dependency. This subject is more related to the application planning. The application planning cannot be replaced.

4.4.4 Process Implementation

The simplest application is just a form to be filled by an authenticated user for example. To implement a process where the other transactions are dependent on the results of the previous transactions depends on the application design. The existence of the domain attributes can be used to define if the needed attributes are usable with the NOT NULL constraint. Also an identification attribute selected in a previous transaction can be used as a dependency in the next transaction. An insert, an update and a delete operation should have configurable operations to output error and the response.

4.5 Other Considerations

4.5.1 Atomicity of Operations

The attributes of a transaction is a serializable set of attributes allowing them to be modified once. A datastore used has to be able to serve concurrent transactions from many server and client application instances. The attributes modified in a transaction are the attributes of a relational table serialized to one atomic transaction and the ones selecting them.

If atomicity is needed in an application as an application function, atomicity can be ensured in many ways. Usually a lock ensures that an aggregate is not accessed before transactions are complete. It is possible to use version stamps [38, 38:45]. Some document datastores implement a similar multiversion concurrency control (MVCC). Unlike with the relational databases overwriting the modified data, the older versions of the attribute or the aggregate versions are saved and the version is compared in the transactions.

4.5.2 Cloud Computing Environments

Cloud computing is today the growing trend in implementing software services. The application can be installed as a service and it is then called Software as a Service (SaaS). The requirements usually are at least: The cloud has to be able to scale the application to different nodes, the fault tolerance of the nodes has to be taken into consideration and the load balancing between the nodes has to be implemented.

If an application session is needed between the client and the server, the state of the session has to be saved to identify the client and the clients session data. The state

between the client and the server can be stored in an extra HTTP key-value called a cookie. The client can be redirected to the same server where it was before using the information in the next requests.

Another option is to distribute the session information between all of the servers and redirect the client to any server. Using a separate database for the sessions, the instance itself does not have the session data being stateless. The session identification acts like a protected password across the application nodes within an encrypted SSL connection. An authentication framework can be used to identify the caller of the transaction.

4.5.3 Adding Program Functionality

Sometimes the programming can not be avoided. Transformations of the attributes may be needed and sometimes the order of the transactions has to be more restrictive if it cannot be restricted with only the attribute definitions, with the authentication for example. Adding program functionality outside of the application concept can be made if the source of the attributes is trusted. The input of the the program has to be known and the source of the output has to be trusted.

4.6 Summary

The graph algorithm was chosen based on the research and it was described with the UML diagrams. Multivalued dependencies was known to be applicable to the axiom system used and the multivalued dependencies was added to the described algorithms. To be able to program the schema, the user interface needed the multivalued dependencies. The null termination of the relational model was included and a possibility to use only mathematical concept was evaluated. An identity after the relational table attributes in the transaction was mentioned and a method to use the algorithm with the SQL statements with joins was evaluated.

5 Results and Analysis

The chapter compares the results to the objectives from the introduction in Chapters 1.2 and 1.3. The purpose is to verify that the work was as was specified comparing it to the objectives. A test program was impelemented with the help of the UML diagrams and the execution time was evaluated in Chapter 5.4.

5.1 Research Outcome

The research was made from the other research publications. In information system science, most information is in the publications and less information is in the books. Overall the material may be the most read material in the relational theory.

5.1.1 Research Method

Thesis subjects and the timeline of the research of the algorithms is shown in the Figure 10.

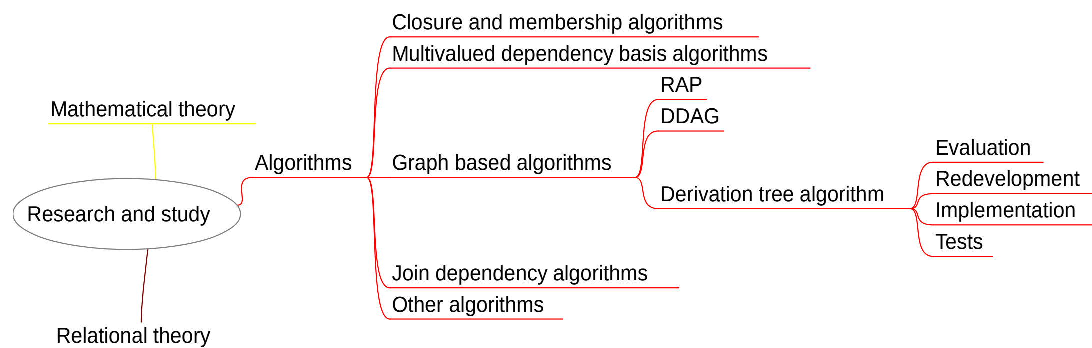


Figure 10: Research subjects

The research of the theory was made studying the subject at the same time. The mathematical theory is a special subject under mathematics and it was mostly introduced in the publications. The learning objective was met.

The needed theory was collected and described. Different algorithms were found in different sources and documented. The theory was applied to the concept with choosing an algorithm and developing it further with multivalued dependencies and the relational model.

Overall process can be described with these words in order: Research and study of the background theory, research and study of the algorithms, choosing an algorithm and developing a computer program component based on it, testing and evaluating the made algorithm and evaluating the work done.

5.1.2 Source Material

The research of the algorithms to use in the application concept was started from the text books and the publications and the fundamental theory was listed as the theory section of the thesis. Source material was searched partly with search engines and keywords from the library databases. Most material was found by reading the research of others referencing the source material. The referenced source material of these was again read. Some publications were found with reference information to these. All of the read material is not included in the reference list.

Some sources were the original and first publications [9][17][16][10]. The original research is sometimes conducted in other context in which the theory is used today and reading the original publications sometimes reveals some more information on the subject. For example the idea of a data catalog was already in one of the earliest publications [16][5].

It was good to understand some of the importance of the lattice theory to get the scope of the needed theory [9]. This information was from the original publications and books about lattice theory. Nothing lead to the lattice-theory when researching the subject again and the original publication stated that no more of the lattice theory is needed after the basic definitions of the lattices or the semiordered sets.

Previous relational theory research similar to the subject of this thesis was not found from the Universitys own listings. An undergraduate thesis was read from the library of Helsinki University because of the availability, the subject matched and the supervisor of the work was known from a publication [35]. Some of the finnish vocabulary was taken from the text and compared even if the text was written using a typewriter.

Some courses today have public video distributions. Material of at least two different university courses was found [36][37]. Third video was included because of its good explanation of atomicity with NoSQL databases [38].

The age of the source material and the relevancy

The majority of the source material of the relational model is more than thirty years old. Still most of the newer research still reference the older source material [5][20]. The

newer material has not been able to replace the fundamental ones. Not until recently, a new research has shown results of a new proposed normal form. The most important theories of the relational theory were developed before year 1985. The last normal form, the domain-key normal form was published in 1981. The newest version of an algorithm taken to the source material is from year 2013 [20].

The research of the algorithms from the source material

The searches did not clearly give results on the graph algorithms for the purpose. It is possible that this kind of material does not exist or it is more difficult to find. The found algorithms had to be improved with the multivalued dependency rules because a better alternative graph algorithm was not found.

5.2 Objectives

The objective was to help the user to input the correct data. The data dependencies should be verified and checked before the transactions. The goal was to increase the program code reuse, software quality and to reduce the work needed in software development. The concept is capable of returning an error code and error messages of the application schema based on the concept of functional dependencies. The schema should include more information than in the relational database schema. All of the definitions of the determinants have to be given, not only the primary keys.

5.2.1 Applications Without Programming

Application program code usually finds relationships between the attributes, checks the attribute values form and verifies that the needed attributes are present. Application programs ties the attributes together with a somekind of a condition. The condition can be anything a programming language can offer. These operations in principle return the value of the condition: true or false if the attribute values fullfill the given constraints of the program code.

Some of the program functions can be replaced using the attributes. Functional dependencies can be attributes with a condition having a value true or false. If an extra condition exists in the application, the condition can be an attribute in the schema describing the condition. If the program code is not needed to verify the condition, the application programming is not needed.

One of the thesis objectives was to reduce the need to program new applications. Simplest applications do not have to be programmed. The conditions between the

attributes with programming is not always needed. If programming is needed, the programming can be an addition to the typesystem. Attribute values may be derived from the other attribute values.

Libraries to check the attribute values form with it's type definitions are seldom used. The typesystem results some program code and it can be a reusable library. One reason to program applications is to create a typesystem of the application.

In the three tier information systems the relational database is separated in the third tier, usually to an own server. The network models (for example the OSI-model, Open Systems Interconnection) do not describe the saving of the data to a physical datastore and do not apply very well to the concept. The application concept changes the use of the relational database management system as in Figures 11 and 12. The application concept uses directly the relational model and the typesystem is used as an application program if it is needed.

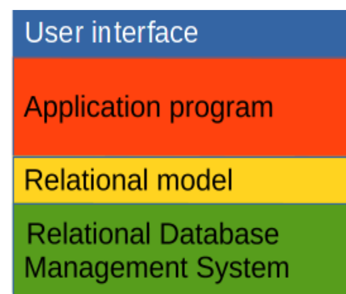


Figure 11: A stack model of a traditional application

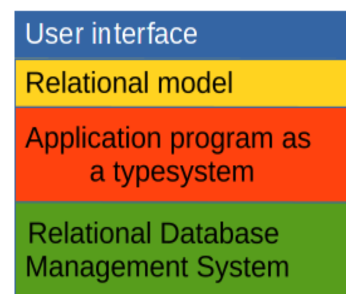


Figure 12: A possible concept application stack model

The role of the join dependencies have more weight in the physical storage. The relational database management systems have a typesystem. Only the most primitive types are available. The relational model of the relational database is still important to verify the interconnected form of the data, to reduce the storage size and to use the relational model to search faster with the different transactions in the form of the schema.

5.3 Application Development

After the thesis and the definition of the relational concepts, the overall design of the application program using the concept has to be made. The transaction definitions and the application program using the relational database in addition to the user interface has to be programmed.

5.4 Test Results

In during the research, a first version of an integrity verification test program was developed based on the derivation tree algorithm. The source code or the programming was not included in the scope of the work. The test results in Appendix 5 validate the work done and show how the results compare to the expectations.

5.4.1 Forming

The execution time of forming the derivation tree is linear as in Figure 17 in Appendix 5. The O-value of forming the tree is $O(k \cdot x)$ where x is the count of the attributes and constant k is proportional to the count of the determinants. The tree is searched every time when the new root is set and the searching time increases when the attributes increase. Otherwice the algorithm is linear.

5.4.2 Searching

The graphs in Appendix 5 shows the test results of two different searches from the derivation tree in four figures. The derivation tree grows in the test when the attributes increase. If attributes have more determinants, the search is slower. The O-value of the search algorithm is practically $O(k \cdot x)$. Constant k is relative to the number of the determinants the attributes have.

5.4.3 Execution Times

To compare in what class the application performace can be, a small inaccurate comparison with a runtime compiled scripting language PHP. If the PHP program size is 400 kilobytes, estimately 400 kilobytes is read in the runtime compiler. Number of the transaction attributes is typically less than 20. From the test results, comparable sizes are 44 kilobytes read with the programmed integrity verification. The integrity verification of 20 attributes with the test setup would be 85 milliseconds. The performance is reasonable and the execution time is not too large.

5.5 Standards

The SQL standard does not define the data catalog. Many relational software database vendors have tools to represent the schema of the relational database. These tools are

different and the form they output is not interoperable or usable. From the application point of view, the relational databases schema usually does not describe all of the needed determinants of the application and information of the data catalog of the relational database would not be enough.

5.6 Further Modeling

After the research of the relational part of the concept, modeling the application further would benefit the application development. Algebra is used to model mathematical systems and it can be used to model computer programs as well. For example the concept of functors uses the concept of an algebra with the relational theory. The application concept developed uses declared attribute names while the functors use the definition of a category. The relationship between the typesystems and the relations could be researched more to model the application concept further.

5.7 Further Research

The relational model could be used with the NoSQL databases. Usually the relational database is needed with the NoSQL database. The relational database properties could be attached to the NoSQL databases providing the consistency and integrity of the attributes. The research of the thesis represents some of the needed theory.

It was mentioned that graph algorithms were not found for the purpose of the application concept. The graph algorithms found are good enough for the purpose at the moment. New research could try to find alternative algorithms if they exist.

5.8 Publications

The concept of using the functional dependencies to verify the attribute integrity was introduced between years 1970 and 1981. It is only mentioned later in some publications [18]. New publications are not easy to find. It is possible that the mathematical concept may not be mathematically as important or as worth researching as is the schema synthesis where the algorithms are primarily used. The importance is more in the information systems development than in the mathematical concepts.

The consistency and the integrity verification with the graph based algorithms could be introduced again in a publication. The intention would be to gain more interest in the research of the integrity verification with a short summary of the most important research collected in the thesis.

6 Discussion and Conclusions

The purpose of the chapter is to validate. The application concept is compared to the surroundings and to the expectations led to the development. The results are compared to the real use of the possible application. The chapter serves providing the information how the application concept evaluates in its possible environments.

6.1 Discussions

There were some discussions during the verbal presentations of the thesis and some with the supervisor. The conclusions here are from the discussions and partly from the subject only. In the following is presented first a comparison with the relational database management system, evaluation of the reknowness of the concept and the restrictions of using the concept. The benefits are evaluated with the conclusions in the Chapter 6.2.

6.1.1 Comparison

Integrity verification of the relational databases

Relational databases use primarily projection and selection with a well formed schema to ensure the database integrity. The join dependencies are included in the tools of the database. In the application, the integrity verification has to be done with less available tools.

It has been shown that if a table has been divided into separate tables for a reason as in the Chapter 2.3.11, a multivalued dependency exists between the attributes of the tables. Mathematically the multivalued dependency is enough to ensure the integrity of the data. The relational database uses join dependency for this purpose and the application concept has to use the definitions of the multivalued dependencies.

Other products available

Some object-based databases have libraries with an API with more functionality. Some frameworks such as Java Hibernate reduces the object-relational impedance between the object-based programming language and the relational database. Some research have been done with relational model and a NoSQL product called MongoDB [32].

6.1.2 Novelty

It is difficult to find evidence of research done to provide information how to verify the integrity with the functional dependencies and with it's other derivatives. It can be true that a graph based algorithm can be easily formed with the available rules of the relational algebra and mathematically there is no reason to publish new material about it. It can also be assumed that programming database products tens of years ago was more producive than the programming of the ever changing applications. This could explain the lack of motivation to research the subject. The previous research was made from mathematical subjects and the subject of the applications is in information system science or engineering. New research of integrity verification in an application has not begun or it is not popular.

6.1.3 Evolution

The research of the relational model has started from the applications. The first step was to separate the relational database management system from the application. The relational database software have evolved during many years and at the same time the relational model has been improved. The evolution of the fundamental relational model now seems to be stopped. When the use of the relational databases has become popular and when it is used in most of the information systems today, it can be asked if more emphasis should be put to the automatic consistency and integrity verification of the attributes as a part of an application.

6.1.4 Restrictions

The incapability to verify the domain integrity

As with every application using a relational database, the domain integrity cannot be verified in the concept because the application does not have the data of the relational database. The domain integrity has to be ensured to be existing in every implemented application somehow by the programmer using the concept. The transactions should ensure the domain integrity.

The incapability to verify the uniqueness of the keys

The uniqueness of the determinants can not be verified with the application because the data of the relational database is not available. This is the case with every application using a relational database. The uniqueness of the determinants or the

uniqueness of the key values has to be provided somehow in the application functionality and it is left to the programmer using the concept. The unique primary keys have to be ensured to be included in the transactions.

Application development

The schema design is application development. Every definition has to be well planned and an easy way to implement application program do not exist. In some cases the concept is capable of replacing the need to program new applications but this is due to the application design. Everything is not possible without programming and making a mistake affecting the consistency of the data is possible.

6.2 Conclusions

The intended benefits can be written down again. It is not however clear if the database transactions can be used as freely as the concept suggest. The use of the concept should be tested with real applications before the evaluation can be done complitely, The cloud computing has become a trend and the parallelism is important.

6.2.1 Intended Benefits

Intention was to develop a software acting as a guard before the relational database preventing unintended use of the application. Implementing an application with a configurable file should result a higher quality service with a faster development time. It is not desirable to just deny the use of the database. Descriptive error messages are used in the declaration of the variables. The error message should inform what dependency is required to use the variable.

6.2.2 Benefits

If there would be no relational database in the background, just any database providing atomic transactions, the application concept could be able to provide database integrity with it's relational model. NoSQL -databases have been an increasing trend for many previous years. Only the relational databases have been able to ensure the relational consistency.

The research has found the two axiom systems to help in developing a relational concept to existing graph and NoSQL databases if the relational model is needed. These databases are usually schemaless and the use of a relational model has to be

added to the application using them if the relational model is needed.

Todays cloud computing demands parallel programs. The concept can separate the transactions in different processes in the cloud. The transactions should trust the data to be correct and the consistency of the data has to be verified before the transactions.

6.2.3 Testing in Real Use

The concept has not been used with a relational database or a real application. The application concept should be tested in real use to finally know if the concept is useful and to describe in what circumstances it can be used. The restrictions have to be known and taken into consideration when the application is planned. Some tricks may have to be used in planning the application. Comparing to the programming of the applications, the tricks may be the same. Verifying if a transaction has succeeded has to be taken seriously. A manual should be written at the same time to list the possible solutions and workarounds if these have to be used.

The integrity verification before transactions to a database can be an individual program in a network of nodes as described in Chapter 4.5.3. The application concept does not have to be restricted to the simplest applications if program code can be added. The components have to have a trust relationship with each other.

7 Summary

The thesis covers a subsection of the theory of an application concept checking the attributes of the transactions with consistency and integrity verification. The mathematical model was researched and studied with the relational theory.

The found algorithms were documented. The best alternatives were chosen. Graph based algorithms were chosen because they preserve the order of the determinants and the order can be used for the other purposes. They are as fast as the fastest membership algorithms and the use of the set operations is not needed. Set operations would add complexity and hinder the program.

To use the application schema, the graph based algorithm was extended with the multivalued dependencies. An UML diagram was documented to describe the algorithm with the domain attributes. The algorithm was further implemented in a test program. The test results were included to show how the research connects to the outside world and how it compared to the requirements of the thesis.

The graph based algorithm was extended with the multivalued dependencies because the relational model has the concept of a domain. Extending the attribute checking with integrity verification with the multivalued dependencies was needed to use the similar functionality as in the relational databases. Because the graph based algorithms preserve the order of the determinants, the multivalued dependency rules can be used directly in the algorithm verifying the consistency and checking the attribute values. The mathematical proof is otherwice provided in the source material.

The connectivity to the outside application environment, a document datastore or an authentication service for example was left open. Trusting the source of the data is important to prevent the manipulation of the data. The use of the session identifier with other system attributes as was intended from the beginning was described.

The found algorithms and their sources were documented and the choices were described. Adding the multivalued dependencies was described with the rules of the axiom system of the multivalued dependencies. An application concept to automatically verify the consistency and integrity was developed. The algorithms were documented and the application concept was described.

References

1. A. Barth. HTTP State Management Mechanism [RFC 6265]. University of California, Berkeley, 2011. Internet Engineering Task Force (IETF); URL: http://tools.ietf.org/html/rfc6265, USA, 2011. Accessed 15. March 2015.

2. L. Cardelli. Typeful programming. Digital Equipment Corporation, Palo Alto, California, USA, 1989. Springer-Verlag; IFIP State of the Art Reports Series, URL: http://www.lucacardelli.name/Papers/TypefulProg.pdf, USA, 1989. Accessed 15. March 2015.

3. D. Nelson, B. Rossiter, M. Heather. The Functorial Data Model - An Extension to Functional Databases. University of Newcastle, Newcastle, United Kingdom, 1994. University of Newcastle upon Tyne; URL: http://www.cs.ncl.ac.uk/publications/trs/papers/488.pdf, Newcastle, United Kingdom, 1994.

4. P. Resnick. Internet Message Format [RFC 2822]. Qualcomm Incorporated, San Diego, California, USA, 2001. Internet Engineering Task Force (IETF); URL:https://tools.ietf.org/html/rfc2822, USA, 2001.

5. S. Sumathi, S. Esakkirajan. Fundamentals of Relational Database Management Systems. Springer; Delhi, India, 2008.

6. P. Kanellakis. Elements of Relational Database Theory. Department of Computer Science, Brown University; Rhode Island, USA, October 1989 URL: ftp://ftp.cs.brown.edu/pub/techreports/89/cs89-39.pdf. Accessed 15. March 2015.

7. B. Russell. Introduction to Mathematical Philosophy. 1919. The MacMillan Co.; New York, USA, 1920.

8. R. Stoll. Set Theory and Logic. Dover Publications Inc., New York United States, 1961, 1965, 1979.

9. W. Armstrong. Dependency Structures of Data Base Relationships. University of Montreal, Quebec, Canada. IFIP North Holland Publishing Company; Amsterdam, Holland, 1974.

10. C. Beeri, P. Bernstein. Computational Problems Related to the Design of Normal Form Relational Schemas. ACM; ACM Transactions on Database Systems, Vol. 4, No. 1, USA, March 1979.

11. S. Abiteboul, R. Hull, V. Vianu. Foundations of Databases [online]. Addison Wesley; URL:http://webdam.inria.fr/Alice, 1995. Accessed 16. March 2015.

12. V. Markowitz. Referential Integrity Revisited an Object-Oriented Perspective. Lawrence Berkeley Laboratory, Berkeley, California, USA. 16:th Very Large Databases Conference; Proceedings, Brisbane, Australia, 1990.

13. E. Codd. The Relational Model for Database Management: Version 2. Addison-Wesley Publishing Company; USA, 1990.

14. J. Ullman, J. Widom, H. Garcia-Molina. Database Systems: The Complete Book. Department of Computer Science, Stanford University. Prentice Hall; New Jersey, USA, 2002.

15. W. Armstrong, C. Delobel. Decompositions and functional dependencies in relations. ACM; New York, USA, 1980.

16. E. Codd. A Relational Model of Data for Large Shared Data Banks. IBM Research Laboratory, San Jose, California, USA. ACM; Communications of the ACM, USA, 1970.

17. R. Fagin. A Normal Form for Relational Databases That Is Based on Domains and Keys. IBM Research Laboratory, USA. ACM; ACM Transactions on Database Systems, USA, 1981.

18. J. Paredaens, P. de Bra, M. Gyssens, D. Van Gucht. The Structure of the Relational Database Model. European Association for Theoretical Computer Science, Monographs on Theoretical Computer Science. Springer-Verlag; Berlin Heiderberg, Germany, 1989.

19. C. Beeri. On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases. Princeton University. ACM; ACM Transactions on Database Systems, Vol. 5, No. 3, September, USA, 1980.

20. A. Waseem, S. Hussain, Z. Shaikh. An Extended Synthesis Algorithm for Relational Database Schema Design. Hamdard University, University of Karachi, FAST-NU University, Karachi, Pakistan. ACM; New York, USA, 2013.

21. H. Enderton. Elements of Set Theory. University of California, Los Angeles, California. Academic Press; New York, USA, 1977.

22. M. Weiss. Data Structures and Algorithm Analysis in C. Florida International University, USA. Addison Wesley; California, USA, 1997.

23. D. Rutherford. Introduction to Lattice Theory. University of St. Andrews, Great Britain, University of Notre Dame, Indiana, USA. Oliver & Boyd Ltd; Edinburgh, Great Britain, 1966.

24. D. Maier. Theory of Relational Databases. Oregon Graduate Center, USA. Computer Science Press; Maryland, USA, 1983.

25. H. Mannila, K. Räihä. Design by Example: An Application of Armstrong Relations. University of Helsinki, Helsinki, University of Tampere, Tampere, Finland, 1985. Academic Press Inc.; Journal of Computer and System Sciences, USA, 1986.

26. C. Beeri, M. Y. Vardi. The Proof Procedure for Data Dependencies. The Hebrew University of Jerusalem, Israel. ACM; USA, 1984.

27. P. Bernstein. Synthetizing Third Normal Form Relations from Functional Dependencies. University of Toronto, Canada. ACM; ACM Transactions on Database Systems; USA, 1976.

28. R. Fagin. Multivalued Dependencies and a New Form for Relational Databases. IBM Research Laboratory. ACM Transactions on Database Systems; USA, 1977.

29. C. Beeri, A. Mendelzon, Y. Sagiv, J. Ullman. Equivalence of Relational Database Schemes. The Hebrew University, Jerusalem, Israel, University of Toronto, Toronto, Ontario, Canada, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, Stanford University, Stanford, California, USA, 1980. Society for Industrial and Applied Mathematics; 1981.

30. Y. Sagiv, C. Delobel, D. Stott Parker, R. Fagin. An Equivalence Between Relational Database Dependencies and a Fragment of Propositional Logic. 1981.

31. Y. Sagiv. An Algorithm for Inferring Multivalued Dependencies with an Application to Propositional Logic. University of Illinois at Urbana-Champalgn, Urbana, Illinois, USA, 1979. ACM; USA, 1980.

32. G. Zhao, W. Huang, S. Liang, Y. Tong. Modeling MongoDB with Relational Model. Sun Yat-sen University, South China Normal University, Guangzhou, China, URL: http://doi.ieeecomputersociety.org/10.1109/EIDWT.2013.25. IEEE; Conference Publishing Services, USA, 2013.

33. B. Babock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and Issues in Data Stream Systems. Stanford University, California, USA. University of Southern California; URL: http://infolab.usc.edu/csci599/Fall2002/paper/DML2_streams-issues.pdf, Los Angeles, USA, 2002.

34. C. Date. An Introduction to Database Systems. IBM Editorial Board, USA, 1994. Addison-Wesley Publishing Company; New York, USA, 1995.

35. E. Palosuo. Relaatiomallin Normaalimuodot ja Funktionaalinen Riippuvuus (sivulaudaturtutkielma). University of Helsinki, Finland, 1983. University of Helsinki; Helsinki, Finland, 1983.

36. G. Boetticher. Graduate Database Course [Video]. University of Houston, Clear Lake, USA. Youtube; URL: https://www.youtube.com/watch?v=jMB6UC_ItN4, USA, 2011. Accessed 19. November 2015.

37. J. Widom. Course: Introduction to Databases, 4. Multivalued Dependencies, 4th Normal Form [Video]. Stanford University, Engineering, Stanford, USA. Stanford University; URL: https://lagunita.stanford.edu/courses/Engineering/db/2014_1/about, Stanford, USA, 2012. Accessed 23. March 2015.

38. M. Fowler. NoSQL Distilled to an hour by Martin Fowler [Video]. NoSQL matters Conference in Cologne, Germany 11. December 2013. Youtube; URL: http://www.youtube.com/watch?v=ASiU89Gl0F0, 2014. Accessed 23. March 2015.

Alphabetical Index

1NF........13, 50
2NF........14, 50
3NF........14, 50
4NF........14, 50
5NF........14, 50
ACID........4
Additivity........22
Aggregate........8
Aggregate function........9
Aggregate values........8
AJAX........53
Armstrong's Axioms........16
Array........55
Assertion policy........31
Atomic........4
Atomicity........58
Attribute closure........19
Attribute integrity........47
Augmentation........17, 21, 36, 42
B-axioms........37
BCNF........10, 14, 50
Boyce-Codd Normal Form........14
Candidate key........9
Canonical form........19
Cardinality........12
Cascading policy........31
Chase algorithm........38
Check policy........31
Classification of the constraints........26
Closure........18, 34
Closure algorithm........32
Cloud computing........58
Coalescence........22
Complementation........21, 22
Completeness........17, 36
Composite key........10
Concurrency........29
Consistency........4
Constraint........56
Cover........19
DAG........32, 37
Data type........2, 55
Database design........12
DDAG........37
Decomposition........17, 22
Deferring policy........31
Definitions........54
Dependency basis........34
Derivation sequence........37
Derivation tree........32, 46
Derivation tree algorithm........35
Determinant........15

Directed acyclical graph....37
DK/NF....14, 50
Domain....11, 36, 56
Domain integrity....10, 29, 30
Domain Key Normal Form....14
Domain-key normal form....62
Durability....4
Empty set....43
Empty values....27
Entity integrity....10
Equivalent....19
Error messages....3, 55
Field....11
Fifth Normal Form....14
First Normal Form....13
Foreign key....25, 29
Fourth Normal Form....14
Functional dependency....15
Graph algorithm....34
Hypergraph....28
IBM....15, 23, 25
IBM DB2....24, 25
IBM Relational Model v2....23
Inapplicable....27
Inclusion dependencies....23
Integrity constraints....25, 45
Integrity verification....40
Isolation....4
Jacobson notation....28
Javascript....49, 52
Javascript Object Notation....49
Join dependency....23
Join dependency algorithm....38
JQuery....53
JSON....6, 49, 55
Keys....9
Lattice-theory....61
Linclosure....32
Linear time algorithm....32
Linear time membership algorithm....33
Lossless-join decomposition....23
Many-to-many relationship....20
Membership algorithm....32
Minimal cover....19
Missing value....24
Missing-and-applicable....27
Mixed pseudotransitivity....22
Multivalued complementation inference rule....43
Multivalued dependencies....20, 36
Multivalued dependency algorithm....33
Multivalued dependency basis algorithm....34
Multivalued dependency inference rules....21
Multivalued pseudotransitivity inference rule....42
Multiversion concurrency control....58
MVCC....58
Natural join....23

Nonredundant........19
Nonredundant Cover........19
Normal form........12, 13
NOT NULL........25
Not-null constraint........25
Null integrity........10
Null termination........36, 45
Null values........24
Nullify policy........31
Ordered pair........11
Ordinal........12
Ownerships........3
PJ/NF........14
Placeholder........24
Polymorfism........55
Primary key........9, 25
Prime........9
Primitive relation scheme........13
Priviledge........3
Project-Join Normal Form........14
Projectivity........22
Pseudotransitivity........17, 22, 36
Quantity........9
Range........8, 9, 10, 11
RAP........32, 37
RAP-derivation sequence........37
Redundant dependency........19
Referential integrity........10
Reflexivity........16, 21, 36
Relation........13
Relational algebra........10
Relational concepts........56
Relational database integrity policies........31
Relational database policies........30
Relational integrity........10, 25
Relational table........9
Replication........22
REST........53
Restricted policy........31
Restrictions........67
RFC-2822........6
Roles........3
Schema........27, 28
Scheme........1, 13
Second Normal Form........14
Security........3
Set-null policy........31
Singleton form........19
Software design........4
Superkey........9
System R........15
Tableau........39
Third Normal Form........14
Three tier........1
Time variance........57
Transaction........4

Transaction order....44
Transitivity....16, 22
Trigger....31
Tuple....3, 12
Type transformations....55
Union....17
Usability....56

Appendix 1, Linear Time Membership Algorithm

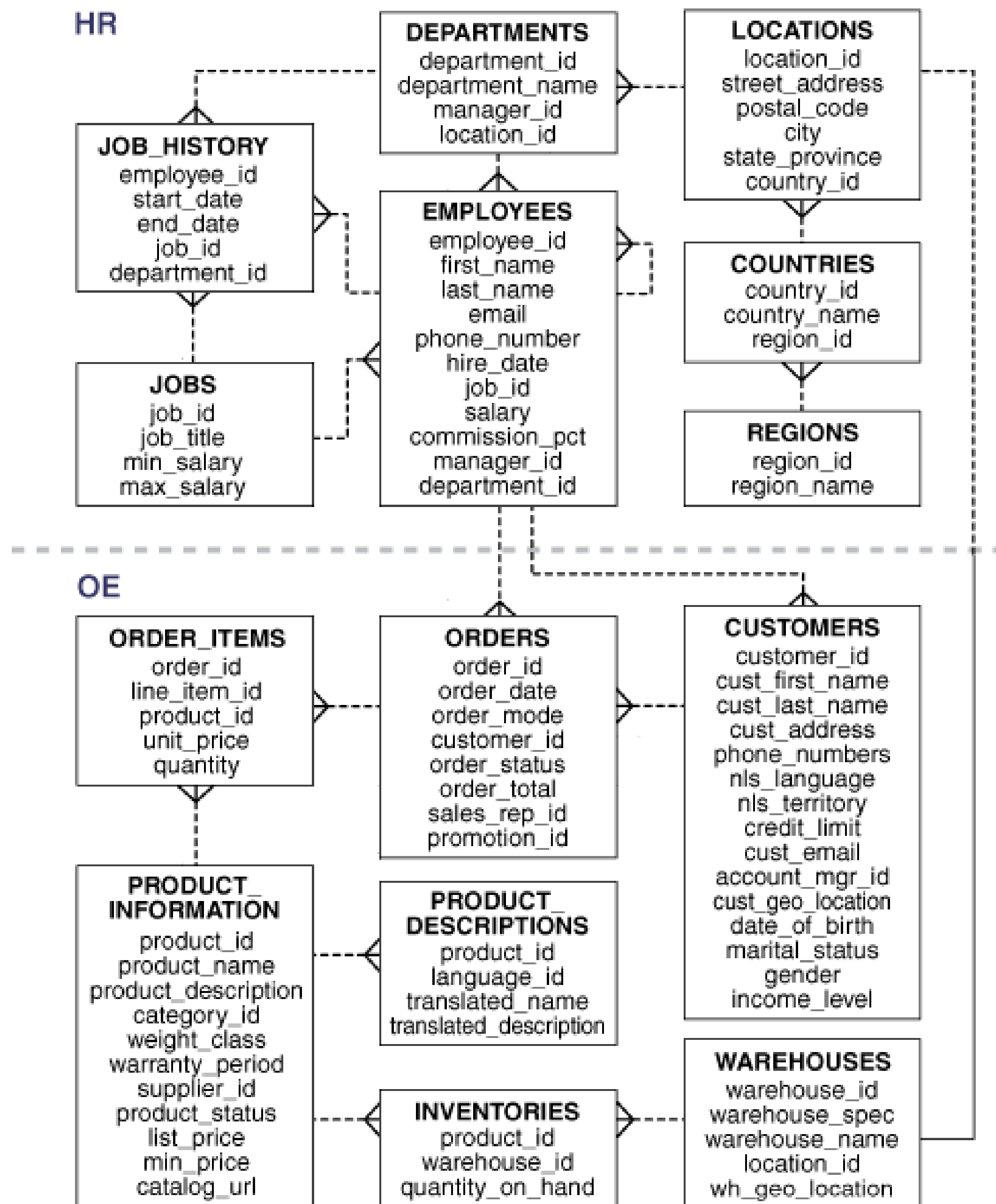
```
// A LINEAR TIME MEMBERSHIP ALGORITHM, PAGE 33,[10].
// from 0 to m attributes
// from 0 to n functional dependencies

begin
      INITIALIZE:
            do i = 1 to m;
                  ATTRLIST[m] = 0
            end;

            do i = 1 to n;
                  COUNTER[i] = 0;
                  do for each j ∈ LS[i];
                        ATTRLIST[j] = ATTRLIST[j] ⋃ {i}; //i, FD:s
                        COUNTER[i] = COUNTER[i] + 1
                  end
            end;
            DEPEND = X; // X it's self, reflexivity
            NEWDEPEND = DEPEND;

      FIND_NEW_ATTR:
            do while (NEWDEPEND ∉ 0);
                  select NEXT_TO_CHECK from NEWDEPEND;
                  NEWDEPEND = NEWDEPEND-{NEXT_TO_CHECK};

      CHECK_FDS:
                  do for each i ∈ ATTRLIST(NEXT_TO_CHECK);
                        COUNTER[i] = COUNTER[i] - 1;
                        if(COUNTER[i] = 0)
                          then do for each j ∈ RS[i];
                              if (j ∉ DEPEND)
                                then begin
                                    DEPEND = DEPEND ⋃ {j};
                                    NEWDEPEND = NEWDEPEND ⋃ {j}
                            end
                        end
                  end CHECK_FDS
            end FIND_NEW_ATTRS;

PRINT:      if A ∈ DEPEND
              then
                  print "YES"
              else
                  print "NO"

end
```

Appendix 2, Multivalued Dependency Basis Algorithm

```
// AN ATTRIBUTESET CLOSURE AND A MVD DEPENDENCY BASIS,
// PAGE 33,[18, 82].
// Ω is a set of attributes, SC a scheme, attribute A

Input:   X ⊆ Ω and SC, set of MVD:s and primitive relation scheme

Output:  X̄ (attributeset closure), DepB(X) (dependency basis)

Method:  var OLDX, NEWX, XPLUS, DBU, DBV, W : set of attributes;
             OLDD, NEWD, DEPBX : set of sets of attributes;
         NEWX := X;
         NEWD := {{A}|A∈X}∪{Ω-X}; // attributes X and the rest

repeat   OLDX:=NEWX;
         OLDD:=NEWD;
         for each U→V in SC do
               DBU:= ⋃{W|W∈NEWD & W⋂U≠∅}; // LHS attributelist
               DBV:= V-DBU; // removes LHS attributes from the RHS
               if DBV≠∅
                 then
                      begin
                        NEWX:=NEWX ∪ DBV; // FD attributeset closure
                        // Subset separation: (NEWD-DBV)⋃ DBV
                        NEWD:={W-DBV|W ∈ NEWD & W-DBV≠∅}
                                                          ⋃{{A}|A∈DBV}
               end
         od
         for each U→→V in SC do
               DBU:= ⋃{W|W∈NEWD & W⋂U≠∅}; // LHS attributelist
               DBV:= V-DBU;
               if DBV≠∅
                 then
                      for each W in NEWD do
                         if(W⋂DBV≠∅) and (W⋂DBV≠W)
                            then // Subset separation
                                 NEWD:=(NEWD-{W})⋃{W⋂DBV, W-DBV};
                      od
         od
until    (NEWX:=OLDX) and (NEWD=OLDD); // until no further change
XPLUS=NEWX;  // closure, X̄
DEPBX:=NEWD; // subsets, dependency basis
return( XPLUS, DEPBX );
```

Appendix 3, Schema to Use in Examples

A sample schema from Oracle RDBMS 11g "Database Online Documentation", Chapter 4 "Application Development", Section "Sample Schemas", http://docs.oracle.com



Appendix 4, Application Concept Schema Example

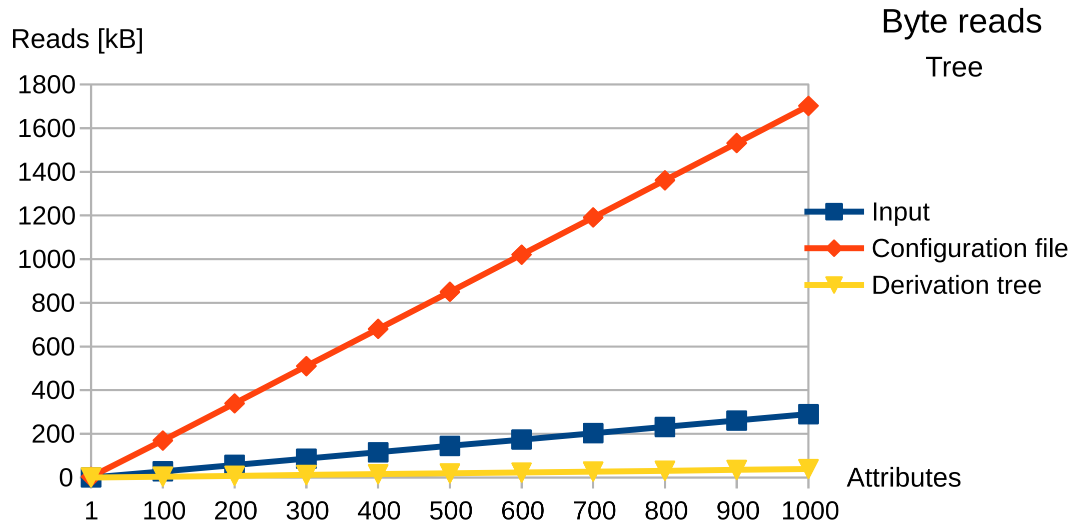
Example attribute definitions of the data of Figure 17 in the application concepts format.

```
{
   "A": {
          "primary_key": "true",
          "type": {
            "maxlength": "13", "regexp": "^[ -~]+?$",
            "matchcount": "1"
          },
          "error": {
            "dependencies": "A has no dependencies.”,
            "type":
              "Error: Characters were not in ASCII encoding."
          }
        },
   "B": {
          "determinant": "A",
          "determinant": "H",
          "multivalued_to": "A",
          "multivalued_to": "H",
          "type": {
            "maxlength": "13", "matchcount": "1",
            "regexp": "^[\x01-\x7E\x80-\xFF]+?$"
          },
          "error": {
            "dependencies":
              "Error: To use B, attributes A and H are needed.",
            "type":
              "Error: Characters were not in ISO8859-15 encoding."
          }
        },
   "C":  {
          "determinant": "A",
          "type": { "maxlength": "13", "matchcount": "1",
            "regexp": "^[\x01-\x7E\x80-\xFF\x{20AC}]+?$",
            “basetype”: “A”
          }
        },
   "D":  {
          "determinant": "B",
          "determinant": "C",
          "multivalued_to": "C",
          "type": { "maxlength": "13", "matchcount": "1",
            "regexp": "^[\x01-\x7E\x80-\xFF\x{20AC}]+?$" }
        },
   "E":  {
          "determinant": "A",
```

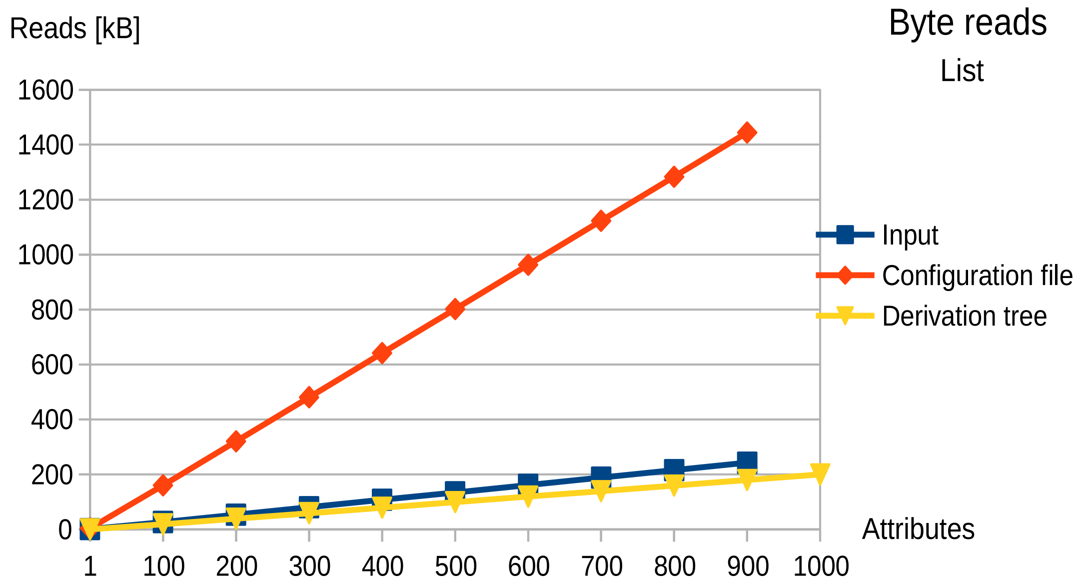
```
            "determinant": "G",
            "multivalued_to": "A",
            "multivalued_to": "G",
            "type": { "maxlength": "13", "matchcount": "1",
              "regexp": "^[\x01-\x7E\x80-\xFF\x{20AC}]+?$" }
          },
    "F":  {
            "determinant": "E",
            "determinant": "G",
            "multivalued_to": "G",
            "type": { "maxlength": "13", "matchcount": "1",
              "regexp": "^[\x01-\x7E\x80-\xFF\x{20AC}]+?$" }
          },
    "G":  {
            "primary_key": "true",
            "type": { "maxlength": "13", "matchcount": "1",
              "regexp": "^[\x01-\x7E\x80-\xFF\x{20AC}]+?$" }
          },
    "H":  {
            "determinant": "I",
            "type": { "maxlength": "13", "matchcount": "1",
              "regexp": "^[\x01-\x7E\x80-\xFF\x{20AC}]+?$" }
          },
    "I":  {
            "primary_key": "true",
            "notnull": "true",
            "type": { "maxlength": "13", "matchcount": "1",
              "regexp": "^[\x01-\x7E\x80-\xFF\x{20AC}]+?$" }
          }
}
```

Appendix 5, Performance Tests

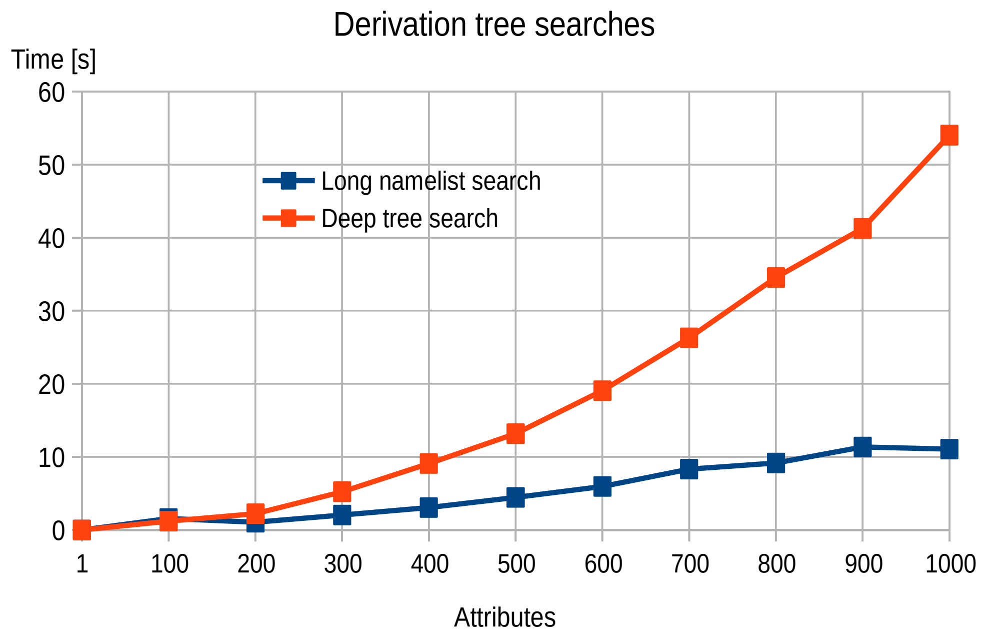
Results



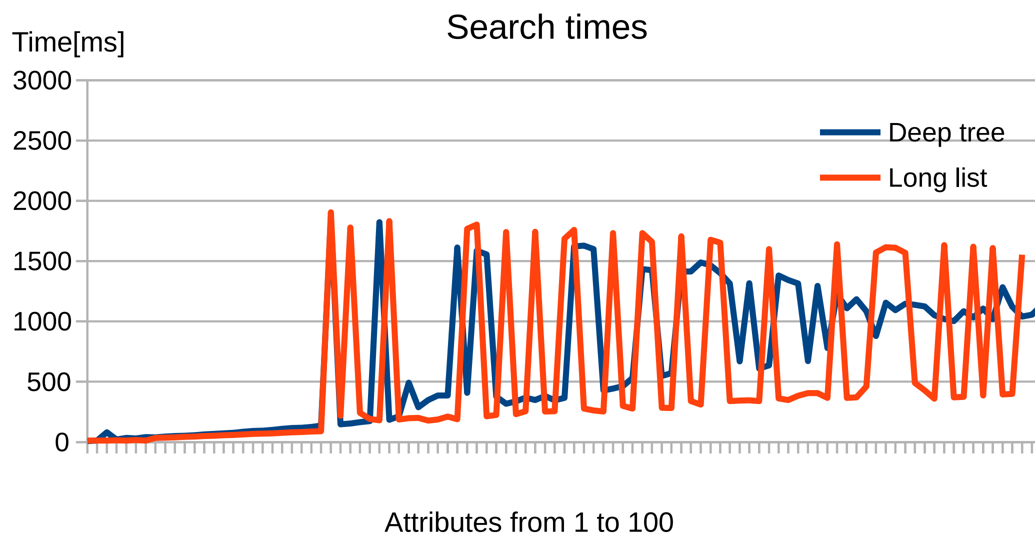
Appendix 5, Figure 13: Test results, bytes read from a tree from 1 to 1000 attributes



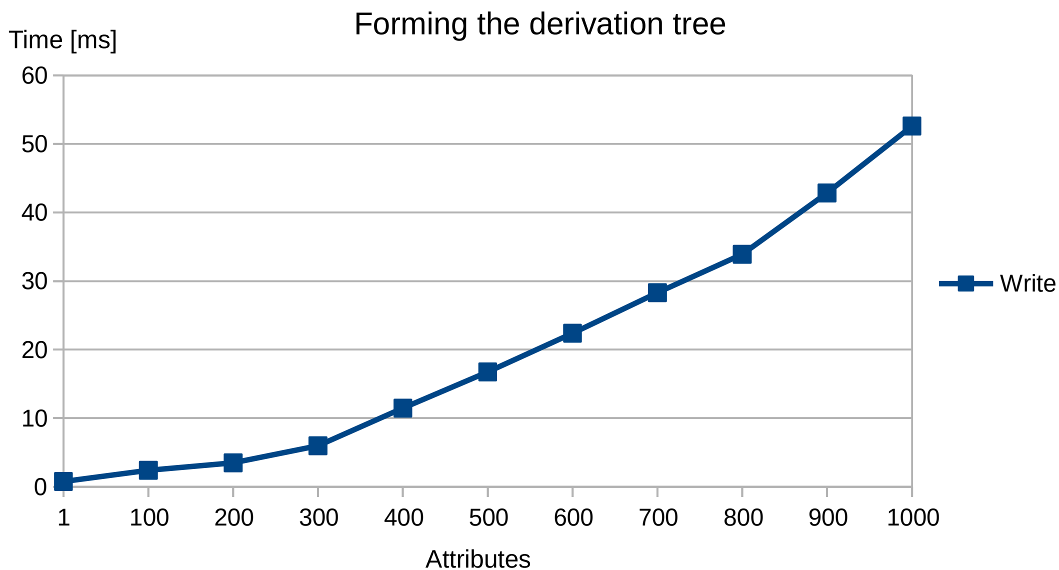
Appendix 5, Figure 14: Test results, bytes read from a list from 1 to 1000 attributes



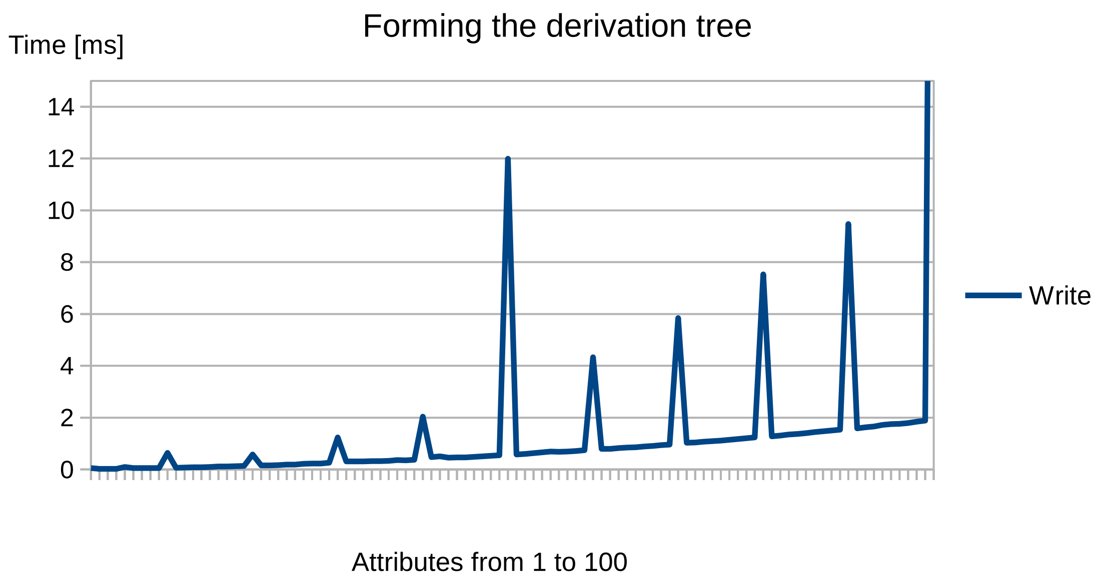
Appendix 5, Figure 15: Test results, integrity verification



Appendix 5, Figure 16: Test results, integrity verification from 1 to 100 attributes



Appendix 5, Figure 17: Test results, forming the derivation tree



Appendix 5, Figure 18: Test results, forming the derivation tree, 1 to 100 attributes

Test environment

The tests were made with a desktop computer. The one-processor computer with two arithmetic logic units had other load at the same time with the tests affecting the results. The results are not accurate. The derivation tree was as large as the input variables and in the chart the search time increases because the derivation tree grows. In the axis named “attributes” with every value, four attributes was searched at a time. Typechecking was included in these results with regular expressions. The integrity verification resulted many reads from the configuration file and less from the derivation tree file. Indexing before the tests was not used.

Results

Verifying the integrity of one hundred attributes with typechecking with regular expressions shows variation in search times as in Figure 15. Variation exists because of other load was on the same machine at the same time. The test data may have been periodically different showing spikes as in Figure 13. The first version of the test programs may still have had programming errors in them explaining the spikes. The test program was in it's first form.

Test Arrangement

Namelist search searched and verified the four attributes from the end of the list. The tree search verified all attributes from the start of the tree list.

Files were formed with a shell script and contained the configuration file with the attributes with number prefixes in their names. The tree file was formed only from attributes A B H and I in a form of a tree. Bytes read was read from the output of the program.

Test programs were compiled into native operating system environment from C -program code.

Integrity verification, 16 and 15:

Command line 1 (tree):

```
cat ./data/input.txt | time ../cbi -J -D -B -b $BUFFERSIZE -l "A B H
I" -d ./data/depth_dt_file.json.${SERIES} -c ./data/conf.json.$
{SERIES} 2>> ./results.${SERIES}
```

Command line 2 (list):

```
cat ./data/input.txt | time ../cbi -J -D -B -b $BUFFERSIZE -l "1A 1B
1H 1E" -d ./data/dt_file.json.${SERIES} -c ./data/conf.json.$
{SERIES} 2>> ./results.${SERIES}
```

Forming the derivation tree, Figures 14 and 13:

Command line:

```
cat ./data/conf.json.${SERIES} | time ../dtree -b "${SERIES}0000" -J
./data/dt_file.json.${SERIES} 2>> times.dtree
```

Test Programs

```
[~/Documents/Source/derivationtree]$ ./dtree

Usage:

    Creating a derivation tree file:
    <input, attribute configuration> | ./dtree [-h] [-J] <dt file>

    Verifying an attribute:
    ./dtree [-h] -r [ -R <root attribute> | -L <list> ] [-a <attribute>]
    <dt file>

    -h  help (this message).
    -r  verify attributes instead of writing the dt-file.
    -R  root of integrity verification.
    -L  list of roots to verify.
    -a  attribute.
    -u  use unfolding with <dt file>.
    -d  debug messages.
    -J  use JSON format instead of default configuration
          file format (delimiters '{','=',';' and '}').

    Outputs a derivation tree from input variable declarations. First
    options are the default values if the declaration is missing. Foreign
    key accepts nulls by default if notnull declaration is missing.
    “determinant_to” may be declared multiple times.

    An attribute can be multivalued towards it's determinants. If the
    attribute is multivalued to the determinant, "multivalued_to" is
    used.

    Program verifies in runtime if an attribute is a multivalued
    determinant.

JSON, special characters '"' with escape character '\':

{
    "<attribute name>": {
      "determinant": "<attribute name>", ... ,
      "multivalued_to": "<determinant name>", ... ,
      "primary_key": "true|false|partial",
      "notnull": "false|true",
      "type": {
        "maxlength": "<integer number>", "matchcount": "<count>",
        "basetype": "<attribute name>",
        "regexp": "<regular expression>"
      }
      "error": {
      "dependencies": "<error message with dependency information
        description>",
      "type": "<error message with type format description>",
          "code": "<error code or an array of codes or ... >"
```

```
      }
    }, ...
}
```

Default, special characters '=' and ';' with escape character '\':

```
<attribute name> {
  determinant=<attribute name>; determinant=...;
  multivalued_to=<determinant name>; ... ;
  primary_key=true|false|partial; notnull=false|true;
  type {
            maxlength=<integer number>; regexp=<regular expression>;
            matchcount=<count>; basetype=<attribute name>;
  }
  error {
            dependencies="<error message with dependency information
            description>";
            type="<error message with type format description>";
            code="<error code or an array of codes or ... >";
  }
} ...
```

```
[~/Documents/Source/derivationtree]$ ./cbi

Usage:

      Verifies attributes in list:

      <input, attributes and values> | ./cbi [-h] [-J] [-u] \
            [-s] [-b] [-D] [-T] [-B] -d <derivation tree file> \
            -c <configuration file> -l "<list>"

      -h    Help.
      -d    Derivation tree file.
      -c    Attribute declaration file.
      -l    List of attributes in parenthesis.
      -N    Do not use null termination, verify every determinant.
      -u    Use unfolding in <dt file>.
      -J    JSON format.
      -D    Print demo messages.
      -T    Print read trees.
      -B    Print benchmark information.
      -I    Build configuration file index before operation.
      -n    No typechecking.
      -t    Time it.
      -b    Buffer sizes of all three caches.
      -s    Block sizes of all three caches.

      Return values under 50 are success, above:
      80    Entity integrity failed.
      81    Null integrity failed.
      82    Type integrity failed.
      83    Value was missing (reflexivity failed).
      84    Determinant not found.
      90    Root was not found.
```

Forming the derivation tree

```
[~/Documents/Source]$ cat conf.json | ./dtree -J ./dtree.json
[~/Documents/Source]$ cat ./dtree.json
```

{ "A": { }, "B": { "A": { }, "H": { "I": { } } }, "C": { "A": { }}, "D":
{ "B": { }, "C": { } }, "E": { "A": { }, "G": { } }, "F": { "E": { },
"G": { } } }