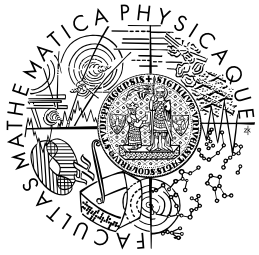

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Denis Drobný

Extracting Information from Database Modeling Tools

Department of Distributed and Dependable Systems

Supervisor of the bachelor thesis:	RNDr. Pavel Parízek, Ph.D.
Study programme:	Computer Science
Study branch:	Programming and Software Systems

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I hereby declare that I have authored this thesis independently, and that all sources used are declared in accordance with the "Metodický pokyn o etické přípravě vysokoškolských závěrečných prací".

I acknowledge that my thesis (work) is subject to the rights and obligations arising from Act No. 121/2000 Coll., on Copyright and Rights Related to Copyright and on Amendments to Certain Laws (the Copyright Act), as amended, (hereinafter as the "Copyright Act"), in particular § 35, and § 60 of the Copyright Act governing the school work.

With respect to the computer programs that are part of my thesis (work) and with respect to all documentation related to the computer programs ("software"), I hereby grant the so-called MIT License. The MIT License represents a license to use the software free of charge. I grant this license to every person interested in using the software. Each person is entitled to obtain a copy of the software (including the related documentation) without any limitation, and may, without limitation, use, copy, modify, merge, publish, distribute, sublicense and / or sell copies of the software, and allow any person to whom the software is further provided to exercise the aforementioned rights. Ways of using the software or the extent of this use are not limited in any way.

The person interested in using the software is obliged to attach the text of the license terms as follows:

Copyright (c) <year> <copyright holders>
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sub-license, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

In ..... date ..... signature of the author

Dedication.

Title: Extracting Information from Database Modeling Tools

Author: Denis Drobný

Department: Department of Distributed and Dependable Systems

Supervisor: RNDr. Pavel Parízek, Ph.D., Department of Distributed and Dependable Systems

Abstract: Abstract.

Keywords: key words

Contents

1. Introduction

A *database* is a collection of related data. By data, we mean known facts that can be computerized and that have implicit meaning as stated in literature instead Fundamentals of Database Systems [1]. We will consider that a database stores data relevant to an enterprise at a host that can be accessed via network. Databases became deep-rooted in every business. Independently of the field that a company is focused on we can enumerate many reasons for considering a database storage deployment a good idea. Let us show some examples of how databases are used through various business domains.

- Social Media
  Every piece of information that has ever been published on social media, from photo through a reaction or comment to friendship establishment, was stored somewhere and that place is a database. Usually the database that a social platform uses does its job in a background. Nevertheless there may occur events when the data storage reminds of its presence as it did on the most recent outage of Facebook. [2].

- Healthcare
  Easy accessibility of large amount of patient's data is a main reason to deploy a database at doctor's office or a healthcare organization [3]. High discretion is a requirement when managing data of such sensitiveness.

- Finances
  complete

- E-commerce
  Every company that sells products online should use a database. The bare minimum is to store offered products themselves and keeping track of purchases that were done by users.

And the list goes on.

A *data model* is a description of data, data relationships, data semantics, and consistency constraints.

A *database schema* defines how is the database described in a data model actually constructed, specifying types of fields from data model. Represents an instance of a data model.

There are multiple kinds of specific data models known. By Fundamentals of Database Systems [4] the main categorization of data models nowadays, according to how big is the abstraction used and on what type of user are aimed, is following:

- Conceptual Data Models (High-Level)
  Reproduces real world objects along with their relationships and should be close to how business end-users perceive them.

- Logical Data Models (Implementation, Representational)
  In the middle between the two other model types there are representational data models which on the one hand are comprehensible by end-users and on the other hand are not too abstract so that they can be used as documentation for an actual database implementation of the modeled data.

- Physical Level Data Models(Low-Level)
  In contrast to conceptual models the physical ones are tied with how data are stored physically at storage media showing all specific internal details that may be overwhelming in the case that the reader is a computer specialist.

Once the decision is made and the usefulness of a database for our thing is proved there may be still a long way until everything runs as expected and we can use all the advantages that such data storage brings. The database design phase comes in place then. By the nature of the problem, a top-down approach is usually followed. A database designer comes in handy for the process.

The first thing that is needed to do is to discuss with an expert in the domain what domain in order to identify and collect requirements for the designed system.

The debate should be translated to a conceptual data model of the future database by the designer.

Once the abstract model is created a movement towards its implementation is desired. That is when the development of a logical model takes place and the high level view is turned into system-specific model that is the logical one.

Finally, the organization of the database itself is figured out and captured in a physical model of the analyzed system.

Given the physical model a database should be possible to deploy straightforwardly.

A *diagram* is a graphical visualization of a data model.

A *data modeling tool* is a software that allows a database designer to create data models. End user may use the tools for interactive previewing of the models' diagrams.

Data lineage provides a picture of how data moves in some system across its components. It is a description of how data go from an origin through their transformations until they reach a destination. The ability of seeing graphically how data are used, what for, and what are the consequences of the usage in a system is a powerful tool for error tracing.

The process of development and deployment of a database consists of multiple stages as we have seen. At the beginning there is a high level view of why the database is needed and what purpose will it serve. Hopefully, in some time

the result is that the data described in the initial step are stored physically at some server. This way the data can be accessed and processed. What we want to achieve in this work is to make use of the individual steps taken during the design process, and make operations on data as transparent and traceable as possible even for business users that don't have a technical background.

1.1 Goals

- Develop a component that extracts metadata from database models that were created using SAP PowerDesigner
- Develop a component that extracts metadata from database models that were created using ER/Studio
- Provide a description by means of a programming language for a general scenario of metadata extraction from a data modeling tool output and passing the information to a data lineage tool
- Propagate data lineage acquired by analysis of how is database used and constructed to more abstract data models than is the physical one, to the logical and the conceptual models.

Introduction to each of the following chapters once the final organization is known

2. Databases

A standalone database is not very useful. To take the full advantage of it we need some means to define, create, maintain and control access to the database. That is purpose of a software called *Database Management System (DBMS)*.

2.1 Database Types

We already described why we want to use a database and roughly mentioned what are the pieces of data that we want to save there. Now let's take a look at what are differences between in database implementations and what to take in account when comparing database technologies. That may be helpful when choosing the best suitable option for some specific data set to store or to see how storing of great amount of structured information can be approached.

The basic division of databases types is simple and binary - they are either Relational or Non-Relational.

There are Database Management Systems build around both, Relational Database Management System (RDBMS)

2.1.1 Relational

A *Relational Database* is a set of tables. A table consists of rows (also records) and columns. We can see such table as an object whose attributes are represented by columns and instances by rows. The important thing is that relational tables tables carry both data that need to be stored by user and the relationships between the data as well. To store an atomic piece of data about instance a proper column is filled with a value. Whereas to capture a relationship between objects the concept of keys is used.
A *Key* is a subset of table's columns used for identifying a record.
A *Primary Key* is a Key that non-ambiguously identifies a record in table and is used when referring to the record.
A *Foreign Key* is a Key that uniquely identifies a record from a table (may be the same or a different one).
They are known also as SQL databases by the query language used in RDBMS for managing data.

The Most Used Relational Database Management Systems [1]

- Oracle
- MySQL
- Microsoft SQL Server
- PostgreSQL

[1] The database technologies usage statistics are based on data from the most up to date version of website db-engines.com [5].

- IBM Db2

Advantages [6] [7]

- Designed for managing structured data
- ACID compliance - database transactions are Atomic, Consistent, Isolated, Durable
- The technology is mature, well-established with large ecosystem and many developers have experiences with SQL and RDBMS
- Data integrity is enforced

Disadvantages

- Problems managing data that are unstructured or semi-structured
- Data is normalized in order to achieve reduction of data redundancy, therefore stored objects may not have one-to-one mapping with the tables that represent them in memory. Also meaning lots of expensive (in terms of speed) joins when fetching such objects.

2.1.2 Non-Relational

A *Non-Relational Database*, is any database that does not follow the relational paradigm. They are younger and were invented to overcome limitations that relational engines have. The ultimate aim is to be more effective when coping with Big Data - data that is fast growing and their structure may not be defined strictly (unstructured, semi-structured information) [8]. There are multiple ways that these requirements can be met so we will introduce more precise division [9]. They are also commonly referred as NoSQL databases as the opposite of SQL databases.

Detailed Description of the types?

Non-Relational Database Types The most used DBMS is listed with each type.

- Key-value stores [10]
  Redis
- Wide column stores
  Cassandra
- Document stores
  MongoDB
- Graph databases
  Neo4j
- Search engines
  Elasticsearch

Advantages [6] [7]

- Elastic scaling, new cluster nodes can be added easily
- No strict database schema is required, bigger flexibility when changing inserted data format

Disadvantages

- Weaker data consistency mode - BASE (Basically Available, Soft state, Eventual consistency) in contrast to stronger ACID in RDBMS
- Lack of built-in data integrity
- Join operation is hard and may be even not supported

2.1.3 Conclusion

By the described properties of the respective systems, hopefully, a reader has some image of in what situation is reasonable to use the more traditional Relational design or look around for one of the Non-Relational databases. To sum it up, if the ACID principle is required by a user and business rules should be enforced the SQL databases are the ones to choose. Enterprises should be cautious and their first choice would be a Relational Database. In contrast when storing heterogeneous data or big volumes of it, consistency is not a priority and the system is extensively distributed some of the Non-Relational databases may be the right one.

Move to the modeling chapter?

However, in this work we will focus only on databases that are of the Relational kind.

The main reason behind this is that since NoSQL Databases have flexible schema or are schema-less (there is no point in determining a database schema when data types of attributes or keys) modeling of these databases quite a new discipline and is hard to find an intersection among different approaches to NoSQL modeling. Also concepts of higher abstraction models are omitted. [11]

The thing to consider is that once a database is Relational we more or less know what to expect from it. The structure of these databases . So a tool that would extract metadata from relational data models is potentially more powerful as it can be applied to more database technologies than a similar tool aimed for some specific type of Non-Relational database.

Lastly, despite the Non-Relational may be growing in numbers and became a serious alternative, as it suits some use-cases better, the Relational still are, and in the near future will be, far more widely used.

2.2 Query Language

?

2.3 Means of Database Access

We have a database that stores some data, the data may be queried and modified via SQL statement in Database Management System. However this can be insufficient as third party programs, let's call them application programs, would want to access the DBMS. A solution is to provide them with an application programming interface (API) that provides a set of methods available in the programming language that the application program was written in, so it can use them. Most commonly when the API is called its implementation translates the request so that to a specific DBMS driver that it is passed after understands it and performs the desired action.

A *Connection String* is a textual information used to identify a data source and establish a connection with it. It it is made of pairs of keywords and values separated by semicolon, the keywords are parameters of the connection.

APIs to DBMS

- Open Database Connectivity (ODBC)
  General, language independent
- Java Database Connectivity (JDBC)
  The Java ecosystem
- ADO.NET
  .NET Framework

3. Database Modeling

The first question that has to be answered is what does data modeling brings us.

One may ask why it is necessary to develop some models before an actual database creation. But let us imagine building a house without solid design and documentation. It sounds a bit strange to hire construction workers straight ahead and tell them that we need a house that has 5 rooms, some toilets and expect a good result. Most probably some building would be produced, but we will agree that expectations and requirements of the later inhabitant could not be met properly. Surely there are good reasons why the usual steps are followed strictly. Let us move on from the analogy to the database domain.
When deploying a database from a scratch we may think of two short term advantages. Firstly, the time needed to have data stored somewhere would be much shorter and secondly the initial cost of the system could be lower.
But over time both of the advantages will, most likely, get outnumbered by problems that will begin to appear. Maintenance of a poorly designed system (or not designed at all) is expansive and leads to numerous outages.

Data modeling should lead to higher quality as is pushes to thorough definition of the modeled problem. Once we know what to solve and what is the scope, it is much easier to come with different solutions and justify which of the proposed approaches is the most suitable one.

Costs are reduced since during creation of a data model many errors are identified thus can be caught early, when they are easy to fix.

Data models form a nice piece of documentation that is understandable by each of the involved parties. When someone tries to understand the system, he can choose a data model on an appropriate level of abstraction that will introduce him the important aspects of the problem that suits his knowledge and qualification.

Models should make easy to track whether high-level concepts were implemented and represented correctly in the end and to determine it the system is consistent.

Also during the design process we may learn a lot about properties of the data that we need or have and will be stored. These information are crucial for choosing an appropriate type of database, whether to stick with a relational database if so which DBMS is the one for us, or to look for a non-relational one.

mention NoSQL modeling possibilities

3.1 Data Model Perspectives

Some time ago, in 1975, American National Standards Institute [12] first came with a database structure called Three-schema architecture. It is formed by:

- External Level
  Database as a user sees it, view of the conceptual level.
- Conceptual Level
  Point of view of the enterprise that the database belongs to.
- Physical Level
  The actual implementation.

The idea behind the structure is to create three different views that are independent of each other. For example change of the implementation that is tied with physical level would not affect any of the remaining levels if the structures remained the same. The important thing is that this structure is used to describe finished product, it does not say anything about the design process that leads to the product and should not be mistaken with the data model structure proposed earlier 1.

An important thing related to data modeling happened a year later, in 1976, when Peter Chen identified four levels of view of data:
(1) Information concerning entities and relationships which exist in our minds.
(2) Information structure-organization of information in which entities and relationships are represented by data. (Conceptual data model)
(3) Access-path-independent data structure-the data structures which are not involved with search schemes, indexing schemes, etc. (Logical data model)
(4) Access-path-dependent data structure. (Physical data model)
And he proposed *entity-relationship (ER) data model* that covers the highest two levels and may be a basis for unified view of data.
In that time three major data models were used - relational, network and entity set model. His aim was to bring a data model that would reflect real-world objects and relations between them naturally, while having advantages of all the three already existing models. The mission seems to be successful as years have proven the ER data model to be the most suitable one for conceptual data modeling. Moreover, ER data models are used most commonly in logical data modeling as well.
An extended version of ER data model was introduced later - *enhanced-entity-relationship (EER) data model*. The main change is that concept sub-classes and super-classes, known as inheritance or is-a relationship, between entities was brought.

Conceptual and logical data models are usually represented by ER data models. The question is what specific data model type is used for physical models. As the most low-level model type is tied directly with how a database is organized, physical models must obey the structure of database.

In the early days when navigational databases were trending, concretely hierarchical and network database, each of them was represented by corresponding data model. The key concept behind these databases was that records stored in

databases should be found by navigating through link between objects. There were some issues about it. The biggest problem was that application code was too dependent on how data were actually placed physically and changes in data structures had influence on code that queried the storage had to be rewritten. The advantage of navigational databases was their performance as following a link is much simpler operation than a join that is used instead in relational databases so they were considered better in terms of performance. Although, the efficiency was at price of inflexibility when reorganization of the storage was needed. Some solutions to this issue were proposed but they tended to worsen the performance. move to database chapter?

In 1969 Edgar F. Codd [13] brought the idea of relational database organization and the relational data model was born. already described

3.1.1 Conceptual Data Model

The purpose of a conceptual data model is to project to the model real-world and business concepts or objects.

Characteristics

Aimed to be readable and understandable by everyone.
Is completely independent of technicalities like a software used to manage the data, DBMS, data types etc.
Is not normalized.

A real world object is captured by an *entity* in conceptual model if our modeling domain is public transport then entity may be a bus or a tram. For further description of objects that we are interested in *attributes* are used, those are properties of entities, for example a license plate number would an information to store when describing buses. Only the important ones are listed. [1] Also *relationships* between objects are necessary to provide full view of the section of the world that a data model resembles. Having transportation companies in our data model it is really fundamental to see that a company may own some vehicles.

3.1.2 Logical Data Model

Keeping its structure generic a logical model extends the objects described in a conceptual data model making it not that easy to read but becomes a good base documentation for an implementation. Data requirements are described from business point of view.

[1] Definitions varies and in some literature can be even found that a conceptual entity lacks attributes. We assume that the entity can contain important attributes as it is more common interpretation and modeling tools have attributes support on conceptual layer as well.

Characteristics

Independent of a software used to manage the data or DBMS.
Each entity has the primary key.
Foreign keys are expressed.
Data types description is introduced (but in a way that is not tied with any specific technology).
Normalized up to third normal form.

Entities, *attributes* and *relationships* from a conceptual model are present on this layer as well. Relationships are not that abstract as before and keys that actually make relationship happen between entities are added as their attributes.

3.1.3 Physical Data Model

A physical data is a description of a database implementation so it is necessarily tied with one specific database technology as it should have one-to-one mapping to actual implementation. Its main message is to communicate *how* the data are stored.

Characteristics

Exact data types (DBMS specific) and default values of columns are outlined.
DBMS's naming conventions are applied on objects.
Constraints are defined (eg. not null, keys, or unique for columns).
Contains validation rules, database triggers, indexes, stored procedures, domains, and access constraints.
Normalization in order to avoid data redundancy or de-normalized if performance increase is reflected in the model.

Objects in physical models are created by related higher-level concepts. *Tables* should store records that corresponds to logical entities and *columns* represent previously described attributes in memory.

image that illustrating the division horizontally and vertically

3.2 Relations Between the Models

We described what the role of each of the layers in a database design process is. Now we will show that the data models are somehow connected vertically and what are the implications.

When talking about vertical divisions, we should think about how database design can proceed.

The basic approach is the *top-down approach* to database modeling. It is natural to start with a general idea what should a database store and what are the relations between stored object. End-user defines this high-level logic and as time goes importance of database designer grows until he is at full charge and develops a complete database. It is the most common case of database development when a client identifies a high-level need for a database and hires an expert in this domain to make it happen.

The other way to create full view of a database is the *bottom-up approach*. It can be harder to imagine what would be use-cases for this approach, but there are some problems that are bottom-up in nature. A nice real world example of bottom-up strategy is how doctors work. They start with "low-level" details such as symptoms and they're trying to build the whole image of patient's condition. So in the field of software data elements are firstly identified and after they are logically grouped to form bigger units, entities, and so on until the full hierarchy is known.

3.2.1 Maps-to Relation

In order to capture how high-level concepts are actually realized by more precise object a relation that we will call *maps-to* is used. The relation leads between objects that are semantically equivalent on different levels of abstraction, sometimes even mapping between objects on the same layer are allowed but we will not consider this, as we consider it be mixing two different concepts together - data modeling with data lineage. To be more define what we mean by semantically equivalent objects in data models is that we will assume maps-to edges solely source is model and target is model, entity and table, attribute and column, or the other way around. Following these mapping links is extremely useful when a person wants to gain an overall overview of the system and comprehend it. For example when a user sees a data table in physical model that has a technical name that obey some naming convention and due to normalization does not represent any object straightforwardly, he can follow mapping links that leads to higher layer providing greater abstraction over the implementation and the motivation why the table was created should be much clearer then. It is worth mentioning that usually the mapping relations between objects of different layers simple one-to-one relationships but the cardinalities may vary greatly. For example one logical attribute may be realized via multiple database columns. Normally more technical models are composed of bigger count of objects so one conceptual entity may be realized by multiple database tables in the end. Generally it is assumed that number of conceptual objects $<$ number of logical objects $<$ number of physical objects. It is natural that when capturing important high-level aims less entities is needed to express the intention but as we are getting closer to the implementation more necessary details come to play.

3.3 Construction of a Data Model

We tried to make clear what is a data model and show that there are good reasons to use them throughout the process of database design. Now we will take a look how someone developing a database can actually create those models. In fact, a data model could be created by hand using only paper and pen. It would definitely bring some of the benefits described above but to take the full advantage of modeling we will use *computer-aided software engineering (CASE) tools*. The tools are here to help with development of quality software. The CASE tools are divided into multiple categories, our interest will be focused on the one that deals with Business and Analysis modeling. Graphical modeling tools. E.g., ER modeling, object modeling. The main motivation behind using the tools is

that they facilitate creating and previewing data models. Here is an overview of different ways how a data model can be created using them.

3.3.1 Modeling

This way of creation is the most similar to the pen and paper method. A user builds a model manually by selecting what object should be created and bringing it to the particular model, then he provides details about the object, creates sub-objects or specifies relationships with different objects. Some tools do not allow creating an arbitrary model, but only the conceptual or logical models may be drawn like this. The reason behind not allowing user to create a physical data model out of scratch is that a physical model should either be the result of some process and be based on a model with higher level of abstraction(see the Generating section) and then adjusted or resemble a live database that and to be obtained by reverse-engineering (see the Reverse Engineering section).

3.3.2 Reverse Engineering

Reverse engineering, or alternatively back engineering, is the process whose aim is to find out principles of how things are done or works in a system that is already running and try to gain deeper understanding of the system. Applied to our domain the reverse engineering approach to creation of a data model means that a CASE tool connects to a database and brings every object found to the physical model that is created. relationships The model is an exact image of the database and one-to-one mapping between the model and database should be secured.

3.3.3 Generating

Given a data model on some level another one on different abstraction level can be derived from it. Modeling tools usually support translating objects to semantically equivalent ones either towards either greater smaller abstraction. Of course models created like this are not full-featured models but may be a better starting point for a database designer to takeover. For example when conceptual data model is arranged and logical model should be created based on it it is really helpful not to start from a scratch but to generate an outline of the logical one by generating from the conceptual. Then it may be reshaped into the desired condition more quickly. Generation sources and targets are in maps-to relationship implicitly.

3.3.4 Importing

Finally a CASE modeling tool may be able to import data models that were created using a different modeling software and recreate the data models.

4. Modeling Tools

Describe Diagrams
Bring in the chapters 3.3 and on source of metadata
    Describe metadata that are available, layers, object hierarchy

4.1 ER/Studio

4.2 PowerDesigner

5. Data Lineage

Data lineage brings a way of tracking data from its origin throughout the whole life cycle taking into account every process that manipulates the data until it reaches its final destination. It is like a telling the story of a piece of data including where does it come from and how it interacts with other data. It should provide answers for questions where the data in given solution come from, whether it can be trusted or not, how it gets from point to point and how the data changes over time in the analyzed system. Basically data lineage helps enterprises to gain deeper knowledge and understanding of what happens to data as it travels through various interconnected data pipelines[1] that the system consists of. This is overview of the system, that data lineage provides, is crucial when taking decisions about the infrastructure since the understanding of the consequences should more clear. Also it makes much easier to find errors in systems, since they can be tracked down from where the undesired behavior came to the surface to where the affected data originates. Surely somewhere between these two points the malfunctioning part is and thanks to data lineage the domain of suspicious operations should be reduced and visible. Therefore much time spent on solving issues should be saved. Data lineage is a discipline of *business intelligence*. define

To present data lineage a visual representation is most commonly used. Generally, we can think of the visualization as of a graph explain graph elements.

Having a reference point of interest we can divide data lineage into three types by what it captures. *Forward data lineage* inspects movement of data towards the destination, *backward data lineage* creates picture of what happened to data when traveling to the point from the source and the last type, *end-to-end data lineage* combines both approaches and shows the full flow of data from its source until the very end destination.

Other differentiation of data lineage is the business one versus the technical one. *Business data lineage* highlights only transformations and aggregation of data in a simplified way to the target business user, whereas *technical data lineage* displays precisely flow of physical data as is in underlying components (eg. applications) of the system is made of.

Use case - GDPR

5.1 Theory

Now we will focus on how data lineage can be created to describe lifespan of data that are coming from or being saved to an SQL database. To analyze flow of actual data, having access to quality metadata is fundamentally needed. *Metadata* are the data describing other data. The metadata we will use when analyzing a database are the likes of database name, names of tables, columns in tables, names of columns, procedures, data types etc. When we have these information describing all the records that can be stored in the database together with all SQL scripts that are used for management of the database we can reliably determine how the data flows once the database is being used.

[1] A pipeline is a set of elements manipulating and processing data where output of one element is input of another.

The idea of data lineage construction is as follows. First precondition is to have access to all metadata related to the database under analysis to have a clear picture of objects stored there. Then SQL queries that modify data are examined. They are stored in .sql files and usually a node is added for each of the files. We identify what tables and columns are the sources of input data for queries and where outputs of the operations are stored. Each input and output is represented by a graph node as well. Based on an analysis like this directed edges between the nodes we described are added to show dependencies. Inputs are connected with the query in such manner that every edges originates in of the input nodes and ends in the transformation node. Correspondingly, edges from query node to output nodes are made.

an oversimplified example where data lineage would not be much of use as its importance grows with system's complexity.

5.2 Manta Flow

Manta flow is a product of Czech startup company MANTA. It is a tool that automatizes data lineage creation by and analysis of programming code. It is able to cope with SQL, altogether with various of its sub-dialects, and Java. Uniqueness of the software is in its capability of handling code that is hardly readable by human. Thanks to this feature Manta Flow can automatically process databases consisting of millions of records and create a map of data flow across business intelligence environment - data lineage. Alternatively the data flow is not visualized directly by Manta but cooperates with third party data governance solutions like Informatica, TopQuadrant, Collibra, IBM Infosphere etc. where it is integrated.

Our aim to interconnect the component that is subject of this work with Manta Flow to enrich the data lineage that it produces by metadata that can be obtained from relevant data models and can bring better understanding of the system under analysis.

5.2.1 Supported Database Technologies

Currently Manta Flow

6. Analysis & Design of the Solution

6.1 Requirements

6.2 Analysis of the Problem

6.3 Architecture of the System

7. Implementation

7.1 Extensibility

Description of the common structure of the solution - what to do when a programmer wants to write a connector for another modeling tool.

7.2 Technologies

7.3 Testing

Conclusion

Bibliography

[1] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems (7th Edition)*. Pearson, 2015.

[2] Facebook blames 'database overload' for most severe outage in its history as users continue to report problems with Instagram and WhatsApp more than 14 hours after global meltdown.

[3] HealthCare.gov's Heart Beats For NoSQL.

[4] Abraham Silberschatz Professor, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill Education, 2010.

[5] DB-Engines Ranking.

[6]

[7]

[8] What Is A Non Relational Database.

[9] The Types of Modern Databases.

[10] What Is a Key-Value Database?

[11]

[12] SPARC-DBMS Study Group. Interim Report: ANSI/x3/SPARC Study Group on Data Base Management Systems, 1975.

[13] E. F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. *IBM Research Report, San Jose, California*, RJ599, 1969.

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 Building

A.2 User Documentation

A.2.1 Tutorials

A.3 Cooperation with Manta Flow