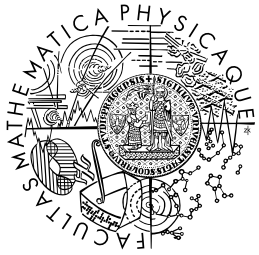

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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 should reflect database organization and at the same moment related higher-level concepts should be transformable to physical level. *Tables* should store records that corresponds to logical entities and *columns* represent previously described attributes in memory. Commonly schemas[2] are present. A *schema* is basically container for tables that logically groups them. Users have usually schemas assigned and can access only the tables contained in those schemas.

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.

[2]Plural of the word schema is schemata but in literature about database design the word schemas is used

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 a 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 precise 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

The main feature of modeling tools is to capture metadata about data models that can be created using them and previewed. The tools use diagrams to present data models to their users.

Bring in the chapters 3.3 and on source of metadata

Describe metadata that are available, layers, object hierarchy or it is a technical detail

4.1 ER/Studio Data Architect

ER/Studio Data Architect is a data modeling and database architecture tool by IDERA, Inc.

ER/Studio allows creating logical and physical data models.

Logical model is realized by an entity-relationship data model, whereas physical models are relational data models.

4.2 PowerDesigner

PowerDesigner is a software for data modeling owned by company SAP SE.

The tool supports conceptual, logical and physical data models. The first two are of extended-entity-relationship data model type and physical is relational data model.

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 IGC 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

Among other technologies currently Manta Flow is able to scan these are the supported databases it can handle and thus are relevant for us:

- Oracle Database
- Microsoft SQL Server
- SAP ASE (Sybase)
- Hive
- IBM Netezza
- IBM DB2
- PostgreSQL
- Amazon Redshift

- Greenplum

5.3 Data Lineage in Modeling Tools

what are the limitations and why we want to avoid it

6. Analysis & Design of the Solution

We will work towards a piece of software that will obtain metadata from data models created using modeling tools ER/Studio and PowerDesigner and the solution will be able to connect to Manta Flow and bring data lineage to objects that have greater extent of abstraction than the physical ones which are currently the only objects supported by Manta.

The solution, which we will refer to as *Metadata Extractor* in this text, is comprised of two separate parts. The first one processes ER/Studio's data models, whereas the second one works with PowerDesigner.

6.1 Analysis of the Problem

We already presented that the modeling tools are capable of creating data models. These models are then saved into files from which they usually are reopened to be modified or previewed. We will use them for reconstruction of the objects and information contained in the data models. In order to do so we need to determine what objects we will be looking for in the files and how they are related to each other. In chapter 3 about database modeling we introduced the standard layout of every data model type. We will quickly review the basic skeleton of each model type once again.

In conceptual data model is focused mainly on entities which may have attributes. An entity may be related to other entities.

On logical layer also entities with attributes can be found and the entities may have relationships.

Physical data models are made of tables. A table belongs to a schema and is composed of columns.

These are the objects we must find in the file formats to recreate the main object hierarchy.

Then we will need to figure out how the models in maps-to relation refer to each other across levels of abstraction. For example how a logical model and a physical model, being the realization of the logical one, are tied together.

6.1.1 File Format

Firstly, we will have a look at what is the output of the analyzed modeling tools, what information are stored there and how it is done.

ER/Studio

The modeling tools uses its custom file format. The file stores plain text and it is made up of many tables. A table in this context is a CSV (comma-separated values) structure but as there is not only one CSV table identifying name is also included. So by table we understand its name, definition of columns (or fields)

and records. escaping in CSV How more complex objects can be stored in files like this is not clear from the first sight and it required some work to get the idea behind it. insert a table here An attentive reader may find these terms, that we used for describing the data structure, familiar since we already used them where introducing relational databases. And he would be true. When investigating these tables we can notice columns that are shared, meaning relation between the tables that have them in common. This is pretty much how primary and foreign key work in relational databases. Now we face the challenge of reverse-engineering the tables to rebuild the composite objects that are deconstructed and saved in the tables. It would be quite exhausting to try to restore the relationships by hand, so we will develop a little tool that will help us to get on overview of the format that will work on one hand with tables' metadata to find the intersecting columns as well as with the records to see whether there are similarities in data stored in different tables to uncover relations between them and the core concepts behind the file format. Further details and ideas related to the ER/Studio file reverse-engineering tool can be found in the chapter subsection 7.1.1

Surely, when we want to load an arbitrary file a component that parses it will be needed. We mentioned that the file of our interest is basically a sequence of CSV tables. The question stood whether to reach out for an existing CSV parser or to develop a tailor made one. We took the second option why and how we did so is described further in subsection 7.1.2.

Once we understand the structure and can tell how the data we are seeking for are stored we can reconstruct them. In one .DM1 file related data models are stored. Let's call these models a solution. An *ER/Studio solution* is set of data models, describing a problem on both logical and physical levels (the two layers are only that ER/Studio supports). In a solution one logical model must be present whereas 0 to N physical ones are supporting it. We can imagine why ER/Studio behaves like this. The motivation may be that we have a problem (if there is no challenge, no data modeling is needed) that is obligatory described by the logical model. Possibly we worked out the way to solve it and that is when physical models are present. Note that the actual storage may be distributed and the corresponding databases can be of different technologies, that is why more than one physical model is allowed in a single solution.

PowerDesigner

In the case of PowerDesigner we will be handling files with three types of extensions - .pdm, .ldm and .cdm. They stand for physical data model, logical data model and conceptual data model respectively. All of them are XML (Extensible Markup Language) based file formats. more about the XML format - elements, tags we will use/need

Since we have three different output file types from the modeling tool it is easy to see that the logic of how data models are saved varies from ER/Studio's approach. While ER/Studio groups data models into solutions every model created in PowerDesigner is saved independently. Set of files that are currently opened in PowerDesigner form its state. Such state is called a workspace here and can be saved into a .sws file, but these files do not bring us anything interesting. The information captured stores only what files were at some time opened in the

environment and does not tell anything about logical links between the captured files.

When parsing XML files there are basically two major ways we can face the problem.

The first approach is SAX (Simple API for XML) that is an event-driven parser, which process an XML document sequentially by a single pass. By default the processing is state independent and handlers are triggered when an event occurs. It is a simple (for some cases may be even too simple) and lightweight parser.

On the other hand we have a family of DOM (Document Object Model) parsers. They load an XML file into a full AST (Abstract Syntactic Tree) structure. This way of file processing is both more memory and time consuming but translates everything stored in the parsed file into data structure straightforwardly. Then we can conveniently work with the tree-like result structure where nodes represent parts of the processed document.

In the next chapter we will describe what we want to retrieve from the PowerDesigner data model files. We will see that the objects and their properties are quite complex and composite using a DOM parser will be much more suitable and doing XPath queries over a DOM document is nicer than having to store a context manually, what would be needed with SAX.

detail, move: One more extension to mention is .xdb, that is definition of a specific DBMS that is used for reverse-engineering databases into physical models.

6.1.2 Metadata to Collect from Data Models

Now it is time to identify which metadata to collect in order to bring data lineage to the conceptual and logical level. The bare minimum is to be able to reconstruct high-level entities to have at least something to visualize data flow between according to knowledge gained in later stages. But we will aim to bring as much information as possible and try to make use of every relevant (meta)datum saved by a modeling tool. In this section we will discuss what specific types of objects can we obtain and what are means to describe these objects even further then by the basic definition using the tools. On the other hand we will not pay attention to most relations in entity-relationship model. This is given by nature of Manta Flow, it is not a modeling tool thus it does not work with them and links between objects are used solely to represent dependencies determined by data lineage. The only relationship type from data models we will need to cope with is the inheritance relation. It is present in enhanced-entity-relationship models. The exception is made because if the is-a links are not captured, entities' structure are be not described completely and their attributes may be missing. Other categories of metadata we will not extract are the ones that describe some constraints on the actual records saved in database themselves. We will work exclusively with database metadata and don't have access to what is really saved there thus we cannot neither monitor nor enforce anything on the database entries. That is why likes of keys and data types defined in data models will not be in our domain of interest.

Conceptual & Logical Data Model

Here we list objects that appear in both conceptual and logical data models, together with what additional information about them can be inserted by user.

ER/Studio

CDM

ER/Studio does not support conceptual data models.

LDM

justification of each of property?

- Owner
  Owner is a concept equivalent to a schema - it is a container for logically related entities. Every entity belongs to an owner.
- Entity
  An entity has the following properties
  - Name
    Identification of the entity.
  - Attributes
    Attributes assigned to the entity.
  - Definition
    Further description of the entity. Plain text or RTF (rich text format).
  - Note
    Notes are used when a documentation about the entity is generated. Plain text or RTF.
  - Where Used
    Shows objects that are in maps-to relation with the entity. Those which were created by generating.
  - User-Defined Mappings
    Shows objects that are in maps-to relation with the entity. These mapping are user defined. They can contain description of a relation, but we will not fetch the text as Manta Flow does not support attributes on mapping edges.
  - Owner Owner is a concept equivalent to a schema - it is a container for logically related entities. The entity belongs to an owner.

    #### Properties We Will Not Extract

  - Permissions
    By owner assigned to the entity permissions are realized.

  - Keys
    As we work exclusively with metadata we cannot enforce key constraints on the actual records in database, thus there is no need for them.
  - Relationships
    Manta Flow does not support them.
  - Constraints
    The same reason as with the keys above.
  - Naming Standards
    The property is used when creating/generating data models but as we don't modify or add anything that would need to apply naming convention to, it is irrelevant for our case.
  - Data Lineage
    We already mentioned that we want to create a real data lineage not just something that was drawn by user because it could have nothing to do with how the data actually flows.
  - Security Information
    There would not be much of use if extracting security information since they are currently not supported in Manta Flow.
  - Attachment Bindings
    Manta Flow does not support attaching external pieces of information like ... to objects.

- Attribute
  - Name
  - Definition
  - Notes
  - Where Used
  - User-Defined Mappings

    Properties We Will Not Extract

  - Datatype
  - Default
  - Rule/Constraint
  - Reference Values
  - Naming Standards
  - Compare Options
  - Data Lineage
  - Security Information
  - Attachment Bindings
  - Data Movement Rules

Objects We Will Not Extract

- Relationships

PowerDesigner

Conceptual and logical data models in PowerDesigner have so much in common that we will propose unified view on what may be stored in them. The properties/object that are specific for either of them are marked with information in brackets saying "CDM/LDM only".

- Data Item (CDM only)
  A data item holds an elementary piece of information, which is given by some fact or a definition in a modeled system. It may or may not be present as a modeled object. Data items can be attached to entities to form their attributes. It is a datum that may seem relevant and is possible to capture at first but later may be not used as no entity needs it in the end.

  - Name
  - Code
  - Comment
  - Keywords

  #### Properties We Will Not Extract

  - Data type
  - Length
  - Precision
  - Domain
  - Stereotype
- Entities

  - Name
  - Attributes
  - Code
  - Comment
  - Keywords
  - Parent Entity

  #### Properties We Will Not Extract

  - Number
  - Generate
  - Identifiers

  - Rules
  - Stereotype

- Attributes
  - Name
  - Code
  - Comment
  - Keywords
  - Parent Entity

Properties We Will Not Extract

  - Data type
  - Length
  - Precision
  - Domain
  - Primary Identifier
  - Displayed
  - Mandatory
  - Foreign identifier (LDM only)
  - Standard Checks
  - Additional Checks
  - Rules
  - Stereotype

- Inheritances

Objects We Will Not Extract

  - Parent Entity
  - Child Entity

Objects We Will Not Extract

- Relationships
- Identifiers
- Associations and Association Links (CDM only)
- Domains

Physical Data Model

ER/Studio

- Physical Model
  connection, type of data model
- Schema
  - Name
  - Tables
- Table
  - Name
  - Columns
  - Schema
  - Definition
  - Note
  - Where Used
  - User-Defined Mappings

    #### Properties We Will Not Extract

    Technical properties, many of them are effective only on some specific technologies (Organization adjusts only behavior of Netazza tables). describe every?
  - Storage
    Specifies storage option
  - Dimensions
  - Properties
  - DDL
    Code to create the table.
  - Indexes
  - Foreign Keys
  - Partition Columns
  - Distribute Columns
  - Distribution
  - Organization
  - Partitions
  - Overflow
  - Constraints
  - Dependencies

  - Capacity Planning
  - Permissions
  - PreSQL & Post SQL
  - Naming Standards
  - Compare Options
  - Data Lineage
  - Security Information
  - Attachment Bindings
- Column
  - Name
  - Definition
  - Notes
  - Where Used
  - User-Defined Mappings

    Properties We Will Not Extract

  - Datatype
  - Default
  - Reference Values
  - Naming Standards
  - Compare Options
  - LOB Storage
  - Data Lineage
  - Security Information
  - Attachment Bindings
  - Data Movement Rules

Objects We Will Not Extract

- View

PowerDesigner

- Tables
  - Name
  - Columns
  - Code
  - Comment
  - Keywords
  - Schema

Properties We Will Not Extract

- Number
- Generate
- Dimensional type
- Type
- Indexes
- Keys
- Triggers
- Procedures
- Check
- Physical Options
- Preview
- Lifecycle
- Stereotype

- Columns
  - Name
  - Code
  - Comment
  - Keywords
  - Table

Properties We Will Not Extract

- Data type
- Length
- Precision
- Domain
- Primary Key
- Foreign Key
- Sequence
- Displayed
- With default
- Mandatory
- Identity
- Computed
- Column fill parameters
- Profile

  - Computed Expression
  - Standard Checks
  - Additional Checks
  - Rules
  - Stereotype
- Users, Groups, and Roles

Objects We Will Not Extract

- Primary, Alternate, and Foreign Keys
- Indexes
- Views
- Triggers
- Stored Procedures and Functions
- Synonyms
- Defaults
- Domains
- Sequences
- Abstract Data Types
- References
- View References
- Business Rules
- Lifecycles

6.1.3 Recreating of Modeled Objects

We defined what are the objects and their properties that we will try to obtain from data models. The objects live in the common environment of a data model, therefore they must be organized in some hierarchy. The above enumeration of the objects may help reader to see that the very basic layout of objects of a model is resembling a tree-like structure. The reason is that the basic skeleton of a data model goes like shown on the figure figure. A file can store one or more data models, the models may have several owners, each of them may own zero or more entities/tables which are comprised of none or multiple attributes/columns. Surely further relations between the objects will come to play, like inheritances or mappings discussed later in the subsection 6.1.4, making the diagram of actors in the system more complex.

These objects can be seen then as nodes of the tree, whereas their properties are attributes of the corresponding nodes.

We will be building the tree from top to bottom. Firstly we will reconstruct the root standing for a file, then link it to its children, data models, and so on and so forth.

ER/Studio

Each type of the objects is defined in some table that defines its type as such table is used for storing all instance of the type. It has an id relative to its table used for identification among other realizations of the same type. Basically all links to other objects are done using foreign keys the only thing is to know from which tables the keys come from. So if an object has a reference to its, if we stick with the tree terminology, parent we can get it by looking at what is the id being referenced and identifying the reconstructed object using this information and as we are descending down the tree the object is already loaded and we can plug the child in.

PowerDesigner

XML files form a tree structure by definition what makes storing hierarchy of objects with the same nature very much natural and straightforward. This way a parent object of a child is simply its predecessor in layout of XML elements. Other properties of an object are stored as child elements as well or attributes of the object's representation. Also in this case creating our resulting tree structure top-down makes sense. As first, on our way from the XML root, build objects higher in the hierarchy and only if a parent is build we examine its children.

6.1.4 Maps-to Relation

Once we identified objects across the data models it would be handy further explanation to know which ones are related even though they are not defined at the same level of abstraction.

We will deal only with mappings of objects which are not at the same level of abstraction. Some modeling tools allow mapping, for example, a logical entity to a logical entity but it is unclear what is the meaning of such construct, since we have relationships available for defining relationships like that. Possibly it could indicate that the objects are used identically as they are implemented by a single database table, but that is what data lineage describes precisely and will be brought by Metadata Extractor.

To be specific only the following mappings we will extract:

- An entity to a table or another entity.
- An attribute to a column or another attribute.

ER/Studio

We already listed two types of mapping relation that entities, tables, attributes and columns in ER/Studio can dispose of. In fact their meaning is the same, the

only difference is that the where used mappings are generated automatically and the user-defined are drawn by user. We assumed that all the objects in maps-to relation are in the very same solution but there is also an option to create a mapping to objects that are defined in different .DM1 files. It can be done using the Compare and Merge utility in ER/Studio whose functionality is to synchronize a model with another model/live database/SQL file. Among other operations that keep the pairs in sync there is the mapping creation option. We are interested in the first scenario where models may come from two solutions. The compared models' objects are listed side by side and mappings can be created between pairs of them. These mappings are referred to as universal.

Our focus is on how mappings are saved in an ER/Studio solution.

We will look at what is required to do in order to extract the mappings. Let's start with the seemingly easier case of mappings between objects inside the same solution. After some analysis we found a table defining them. It is named Where_Used_PD. In the table there are four crucial attributes namely id_A, id_B, Meta_Table_A, Meta_Table_B. The first two attributes are foreign keys to tables where the mapped objects are defined. The second pair of columns defines a type of the object so that we know to which tables we should look for the keys that are referenced. The meta tables also allows us to check if the objects are actually compatible with each other.

At the first sight solving the universal mappings may appear more difficult as it looks like we will need to search for object in different solution than the one that is analyzed and reconstruct them. But the way it is really solved in ER/Studio is much simpler. We don't need to go anywhere else as the external objects referenced by a mapping are saved in the solution as well. They are described briefly in a table called External_Mapped_Objects by XML structures. Also a table Universal_Mappings using the same concepts as Where_Used_PD allows us to reconstruct them easily. type-checking

PowerDesigner

By the nature of how PowerDesigner saves every data model into a separate file to resolve mappings will be not as straightforward as in the case of ER/Studio. So every mapping we take into account is an external one, using the terminology introduced above. There is a further division into two categories. Similarly as in the first tool, the mappings may be either generated or user-defined.

Before we will go through how they are represented we must mention the way objects taking part in the relation are identified. Every standalone object in PowerDesigner has a unique identifier stored in its attribute named ObjectID. This sequence of characters (string) is used when referring to an object in the XML.

When an object is created out of an existing one it is reflected in the structure of the object's definition. In such case an element History is present where all the

ids of objects that made an impact on the objects creation are listed there date as well.

User defined mappings are stored as separate relations. A composite XML element describe one relation the name. The structure is formed by a pair of mapped entities/tables may be empty and if their underlying attributes/column are in maps-to relation as well they are contained as in children elements of the mapping element name.

So we are reading a file where the mappings are defined but we are only able to reconstruct a single object in the relation out of two. The only property of the second one known is its id. We will need to find the object corresponding to the id in the file where it is defined in order to gather all the required metadata about it. Thus in situation like this a data model file are somehow dependent on other(s). To learn about the needed files there is an XML element Targets. When a model is generated from a file a dependency is created in both of the data model files, the one that was generated just like in the one that it was generated from. In other words, if we imagine an oriented graph where a file is a node and an edge leads from file a to file $b \iff b$ is listed as a target file of a, then bidirectional edges are created when models are created by generation. Whereas when user-defined mapping is created from an object in a source model a to an object in a target model b, only the $a \rightarrow b$ edge is created and the b file has no knowledge about the mapping. We must solve how a resolution of the foreign objects will be done. As we have nothing but a target object's id, not even information what file does it from a naïve approach would be hugely inefficient. It would go search for every demanded id across all targets. But that potentially leads to a great amount of file opens as well as having to reconstruct the same objects over and over, leading to a big time overhead. Surely we can improve this solution by collecting the ids and postponing the resolution to the and so when processing a single files its target would be opened only once and the reconstruction of the object would take place one time as well. But that is still expansive in terms of time. If we went in a different direction and processed each of the model once, then stored all the objects that may be referred to and once all the data models on input are loaded, resolve mappings. Logic like this would decreased count of reconstructions and file openings to ideal amount but eventually, if number of inputted data models would be too big the size of memory claimed by Metadata Extractor could become unbearable. To achieve a solution that would have advantages of the both naïve approaches we need to split the set of input files into disjoint subsets that represent the smallest group of logically tied files. We will transform all of the unidirectional edges in the dependency graph described above into bidirectional and will find connected components, that will be our searched logical groups. Therefore we can make the resolution at the end as storage requirements for objects of such subset should be reasonable enough. Based on the assumption that the far most common use-case is having three data models - logical, conceptual and physical (or few physical ones).

But there is one thing remaining that may cause some problems. The basic scenario how a user will behave is that he will works with PowerDesigner data models in some directory, for example C:/PowerDesigner/Project/ and once he wants to let them analyze by Metadata Extractor he drags them to a different

directory, that is used as input for our tool. This way, the paths pointing to targets of models became are not correct since they have no reason to be updated and still depend on the files in C:/PowerDesigner/Project/. We want to work only with the files that user explicitly marked as to process, those are only the ones that are present in the input directory. Also if this problem is not thought of, it may cause undesired and unexpected behavior. For example we have file a referring to b in input but a change of external object in C:/PowerDesigner/Project/b would have affect on a. So we will try to come up with a fallback for this situation and will try to deduce by the former path the one in input folder. First thing to do, we will try the ideal scenario and check whether the target is in the input directory. If yes, we are done with this one. example If not we will assume similar structure of the both directories as well as that the names of files were not changed.

6.1.5 Database Connections

Modeling tools usually can make connections to a databases. It is useful for multiple reasons. For example already familiar reverse-engineering would not be possible without this ability since metadata are fetched directly from a database in order to capture its most up-to-date state. Also one of the features of modeling tools is keeping data models and databases that are tied with them in sync, so basically there is a mechanism for comparing actual state of a database with a model.

Manta Flow also extracts metadata of databases for needs of data lineage creation. It is done via JDBC connections.

What we will do is not accessing metastores [1] of databases. Getting metadata directly is not really straightforward as each database technology has its own specifics - type of metadata and their organization varies greatly. Instead we will make use of the fact that Manta does has connectors that do the job for us, stores the metadata in its own local database and has unified API for getting the metadata independently of database engine.

And why would we want to request another metadata when that is just what we are extracting from physical data models? Because we are interested in data lineage which it is created by Manta Flow based on the real metadata of physical objects that are present in database. The simple view is that at the moment when we ask for the objects from Manta Flow's metastore, the analysis of data flow has already taken part, thus there are data lineage edges leading between the objects. To bring together both features of Manta Flow and our tool, that brings more metadata and links to higher abstraction data models, we need to merge equivalent physical objects which come from the both sources - from Manta's extraction as well as from our tool. So only if we are on the same page and we know what database at which server the modeled objects belong to we can ensure correct pairing of the objects and data lineage.

The details we use for identification of a database instance are the following:

- Database Type (Technology)
- Database Name

[1] shortly for metadata storage

- Server Name
- Schema Name
- User Name

is the list complete?
All the above can be stored in one property called *connection string*.
To this set of properties we will refer as a *connection*.
As each physical data model describes a single database (or its subset) we need exactly one connection for each processed physical model in order to achieve what we described just above.

ER/Studio

Databases whose models are created in ER/Studio can be reached only by ODBC drivers present in the used machine.

We cannot really work with that, so we will leave it up to a user to define connection parameters by hand. A .ini files is used for that where sections are named by physical models that they correspond to and connection details are specified inside a section. we can get type describe somewhere the format precisely

PowerDesigner

In PowerDesigner a user has multiple options for connecting to a database to choose from. Either ODBC or JDBC connection may be used. We would like to have the ability to find out what connections with what parameters were used for connecting to a database corresponding to a physical model. So a great help would be if there was a trace left after every each connection is made. We cannot enforce it but in PowerDesigner these traces can be created when connecting using .dsn or .dcp files. The tool has nice user environment for creating or using connections to a database where a user is guided through set up nicely and can test if he did set up everything correctly.

The .dsn files are definitions of ODBC connection containing parameters for an ODBC driver and stores all the interesting information we would like to have. The drawback of this file format is that it varies from a technology to technology. It has a structure of an .ini files but the properties representing the same things may be called differently. That means we would need to have a parser for each supported database engine.

On the other we have .dcp files. They can store information about native DBMS connection or about JDBC connection. The nice thing about them is that they are not that flexible and once we know whether we deal with native or JDBC respectively we know what exact structure expect. It is also a file consisting of property=value map. There are couple of properties common for both types, like description and user name. Then the most important property of a JDBC .dcp file is JDBC connection URL - in other words connection string which should sufficiently define a connection. In case of native DBMS variant Server Name

along with Database Name are crucial in order to identify a database we are connecting to by the setup.

But there is a problem that is common for both of the approaches, namely that there is no link between the connection file and a model that is result of reverse-engineering of the connection. So we will need to create a workaround. how it is solved, also .ini file can be the solution

6.2 Requirements/Desired Features

The overall goal is to collect those metadata that will allow us to recreate physical, logical and conceptual modeled objects with all attributes that may be interesting when shown in data lineage.

In order to achieve this goal we must answer these important questions:

1. Identify what data models the modeling tools work with, what objects are contained in the data models, how they are organized and what metadata can be obtained that are relevant to be brought into data lineage. Analysis
2. Find out what is the format of files that the tools save data models in. Together with how the data we assumed interesting in 1) can be reconstructed. Analysis
3. Design a data structure suitable for storage image of the modeled objects described in 1). Model
4. Determine how the file format can be parsed. Parser
5. Construct the data structure. Resolver
6. Build a graphical representation of the data structure. Data Flow Generator
7. Plug the data model representation built in previous steps into Manta Flow and bring data lineage to conceptual and logical level. Data Flow Generator

how models are mapped to each other

6.3 Architecture of the System

Metadata Extractor consists of two major parts where the first one handles ER/Studio models and the second one processes PowerDesigner files. Further division of each of the parts is that they are split the system into four modules. Then the most high-level view of the system's architecture is following:

- Model [2]
  Is a read-only description of a data model source. On one hand it reflects the raw structure of a file so no information is left out when compared to the source. On the other hand it allows reading access to the modeled

[2]There is a naming collision but here we don't refer to any data model but a data structure that reproduces objects stored somewhere, which one of these two possible meanings we use should be clear from context.

objects we are interested in that were reconstructed in convenient fashion. Independent of Manta

- Resolver
  Is the part where the logic of construction of objects from a file is hidden and loading of the model6.3 is done. Independent of manta

- Data Flow Generator
  Creates a graph representation of a model 6.3. Communicates with Manta Flow via DataFlowQueryService and connections.

- Manta Flow
  The external part capable of crating data lineage on database objects.

A figure showing cooperation of the most components

7. Implementation

7.1 ER/Studio

7.1.1 File Reverse-Engineering Tool

.. implementation of the tool where described precisely, could be a script but will be handy and we'll reuse parser and structures

7.1.2 Parser

implemention Using an existing parser is not suitable why..

7.2 Extensibility

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

7.3 Technologies

- Maven
- Java 8
- Spring
- JUnit 4

7.4 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

ER	Entity-Relationship
EER	Enhanced-entity-Relationship
CDM	Conceptual Data Model
LDM	Logical Data Model
PDM	Physical Data Model

A. Attachments

A.1 Building

A.2 User Documentation

A.2.1 Tutorials

A.3 Cooperation with Manta Flow