ASSIGNMENT OF BACHELOR’S THESIS

Title: SAP Hana database metadata extraction tool
Student: Ondřej Hlaváč
Supervisor: Ing. Jan Trávníček, Ph.D.
Study Programme: Informatics
Study Branch: Web and Software Engineering
Department: Department of Software Engineering
Validity: Until the end of summer semester 2020/21

Instructions

Static analysis of SQL scripts requires information about objects present in the relation database instance the script is working with. The structure of these objects, the amount of metadata to extract, and the way of extraction depend on the concrete database used.

1. Describe the SAP Hana database’s metadata structure and compare it to the Oracle's and MS SQL Server's.
2. Present an analysis of metadata needed to perform static analysis of scripts for the SAP Hana database. Check their availability in the SAP Hana database and identify their possible extraction methods.
3. Based on the analysis, design a tool to extract metadata from the SAP Hana database.
4. Implement a prototype of the tool capable of extracting the metadata into the data structures of the Manta project. Use appropriate software architecture to allow future use of the tool in the static analysis of SAP Hana scripts.
5. Design a data set to test the tool’s functionality automatically. Create a user manual of the tool.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.  
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.  
Dean

Prague October 1, 2019
Bachelor’s thesis

SAP Hana database metadata extraction tool

Ondřej Hlaváč

Department of Software Engineering
Supervisor: Ing. Jan Trávníček, Ph.D.

June 4, 2020
I would like to express my gratitude to everyone who helped me or supported me to complete my thesis. First and foremost, I would like to thank my supervisor Ing. Jan Trávníček, Ph.D., for his help, time and guidance. Invaluable help also came from the whole Manta team, notably I would like to thank the parser team leader Mgr. Jiří Toušek. My appreciation goes also towards my family, and friends for all their support, not only during the time I was working on this thesis but also during the whole university studies.
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”), in accordance with Article 2373 of the Act No. 89/2012 Coll., the Civil Code, I hereby grant a nonexclusive and irrevocable authorisation (license) to use this software, to any and all persons that wish to use the software. Such persons are entitled to use the software in any way without any limitations (including use for-profit purposes). This license is not limited in terms of time, location and quantity, is granted free of charge, and also covers the right to alter or modify the software, combine it with another work, and/or include the software in a collective work.

In Prague on June 4, 2020

................................
Abstract

Static analysis of SQL scripts requires information about objects stored in the relation database instance the script is working with. These objects are called database metadata.

This thesis focuses on metadata analysis in the SAP Hana database. It explores various database object types and discusses various ways to extract them. Then compares them to the metadata of Microsoft SQL Server and Oracle database. Results are used in the design and implementation of a metadata extraction tool. The tool itself is a part of the project Manta and integrated into its tool Manta Flow.

Manta Flow is used to analyze information systems for the data flow that then creates a visual representation using data lineage. This includes databases and their scripts. The extraction tool is a fully working metadata extractor working with Manta Flow.

Keywords  SAP Hana, metadata extraction, MANTA, database, metadata
Abstrakt

Statická analýza SQL skriptů vyžaduje informace o objektech uložených v instanci relační databáze, se kterou skript pracuje. Těmto objektům se říká metadata.

Tato práce je zaměřena na analýzu metadat v databázi SAP Hana. Popisuje typy databázových objektů využívaných ve skriptech toho databázového stroje a ukazuje jejich možné způsoby extrakce. Také je srovnává s metadaty databáze Microsoft SQL Server a Oracle. Výsledky této analýzy jsou poté využity u návrhu a implementace extrakčního nástroje. Tento nástroj je součástí projektu Manta a je integrován do nástroje Manta Flow.

Manta Flow je nástroj používaný k analýze informačních systémů pro jejich datové toky, ze kterých následně tvoří vizuální reprezentaci pomocí datových linií. Analýzou informačních systémů se rozumí jejich zdrojový kód, zejména SQL skripty. Výsledný extrakční nástroj z této bakalářské práce plně funkční extraktor nástroje Manta Flow.

Klíčová slova SAP Hana, extrakce metadat, MANTA, databáze, metadata
# Contents

<table>
<thead>
<tr>
<th>Motivation</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data lineage</td>
<td>1</td>
</tr>
<tr>
<td>Manta Flow</td>
<td>1</td>
</tr>
<tr>
<td>Goal</td>
<td>2</td>
</tr>
<tr>
<td>SAP Hana</td>
<td>2</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>3</td>
</tr>
<tr>
<td>1.1 Database metadata</td>
<td>3</td>
</tr>
<tr>
<td><strong>2 Analysis of the metadata</strong></td>
<td>5</td>
</tr>
<tr>
<td>2.1 SAP Hana</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Architecture</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Object types</td>
<td>6</td>
</tr>
<tr>
<td>2.1.3 System views</td>
<td>7</td>
</tr>
<tr>
<td>2.1.4 System schemes</td>
<td>8</td>
</tr>
<tr>
<td>2.1.5 Metadata access</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Microsoft SQL Server</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 Structure</td>
<td>9</td>
</tr>
<tr>
<td>2.2.2 System catalog</td>
<td>9</td>
</tr>
<tr>
<td>2.2.3 Information schema</td>
<td>10</td>
</tr>
<tr>
<td>2.2.4 Metadata functions</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Oracle database</td>
<td>10</td>
</tr>
<tr>
<td>2.3.1 Structure</td>
<td>10</td>
</tr>
<tr>
<td>2.3.2 View types</td>
<td>10</td>
</tr>
<tr>
<td>2.3.3 Summary</td>
<td>11</td>
</tr>
<tr>
<td><strong>3 Metadata extraction</strong></td>
<td>13</td>
</tr>
<tr>
<td>3.1 Database objects description</td>
<td>13</td>
</tr>
<tr>
<td>3.1.1 Schemas</td>
<td>13</td>
</tr>
</tbody>
</table>
3.1.2 Tables, collections and custom types ........................................ 13
3.1.3 Views .............................................................................. 15
3.1.4 Indexes ............................................................................. 15
3.1.5 Sequences .......................................................................... 15
3.1.6 Synonyms ........................................................................... 16
3.1.7 Graph workspaces ............................................................... 16
3.1.8 Procedures and functions ..................................................... 16
3.1.9 Libraries ............................................................................. 17
3.1.10 Remote sources ................................................................. 17
3.1.11 Adapters ............................................................................. 17
3.2 DDL scripts ........................................................................... 17

4 Design ....................................................................................... 19
4.1 Extracted metadata .................................................................. 19
4.2 Requirements .......................................................................... 20
  4.2.1 Functional requirements .......................................................... 20
    4.2.1.1 FR1: Metadata extraction ........................................... 20
    4.2.1.2 FR2: Filtering schemes to extract .............................. 20
    4.2.1.3 FR3: Saving DDL scripts on disk ............................... 20
    4.2.1.4 FR4: Saving metadata to Manta’s internal structures ... 20
    4.2.1.5 FR5: Filtering object types and DDL scripts to extract 20
  4.2.2 Non-functional requirements .................................................. 20
    4.2.2.1 NFR1: Integration with Manta’s internal tools .......... 20
4.3 Used technologies ..................................................................... 20
  4.3.1 Java with Spring framework ................................................ 21
  4.3.2 MyBatis ........................................................................... 21
  4.3.3 JUnit ............................................................................... 21

5 Implementation ........................................................................... 23
5.1 High level idea ......................................................................... 23
5.2 Class diagram .......................................................................... 23
  5.2.1 SAPHanaExtractor ............................................................... 24
    5.2.1.1 setDictionary / getDictionary .................................. 24
    5.2.1.2 setDdlOutputDirectory / getDdlOutputDirectory .... 24
    5.2.1.3 setOutputDdlTypes .................................................. 25
    5.2.1.4 setIncludedDbsSchemas / setExcludedDbsSchemas ... 25
    5.2.1.5 setExtractInvalid ...................................................... 25
    5.2.1.6 setExtractedDdlTypes ............................................. 25
    5.2.1.7 extract .................................................................... 25
  5.2.2 HanaDAO .......................................................................... 26
  5.2.3 DictionaryWriter .................................................................. 26
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>SAP Hana object structure</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Microsoft SQL Server object structure</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Oracle object structure</td>
<td>10</td>
</tr>
<tr>
<td>5.1</td>
<td>Extraction tool class diagram</td>
<td>24</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Core services of the SAP Hana system [12] .................. 7
C.1 Data type simplification ........................................... 42
Motivation

The last few decades have been very successful in the information technologies industry. Most people started using information systems every day and do not think twice before using one. This means that there are more and more information systems and they are getting more and more complex. That would not be a problem in an ideal world. But in reality, many systems are getting supported in a very quick manner that creates mismatches between the documentation and the actual code. These mismatches may have substantial effects when the information system needs to be serviced, extended again if there is a need to show which data gets affected by some action or enhancing data loss prevention.

Also, there has been a rising need for information security. With the introduction of a GDPR, companies have been forced to care what they store about their customers and how each change might change that. That was always a hard task and even more so today.

Data lineage

To mitigate these issues, some companies maintain data analysis department. Usually, it is a time-consuming and demanding task to find all the information movement. Fortunately, there is an easy way. It is called Data lineage. Data lineage is defined as a data lifecycle that includes the data’s origins and where it moves over time. This can be nicely represented in a graph.

Manta Flow

Manta Flow is a tool used to generate Data lineage. It creates graphs that show which data a script uses – meaning which tables, views, and others. It adds data sources and data consumers as vertices and connects them with
Motivation

edges when some interaction (usage) is detected in the script. Manta is the central hub of all data flows in organizations.

Goal

Most of the Data lineage creation can be automated, but for that, Manta Flow needs to know how data is stored. This is the main goal of this thesis: show useful metadata analysis of the SAP Hana database, design an extraction tool, and implement it. The extraction tool is to be compatible with the Manta Flow tool and uses its internal structures. Nevertheless, it can still be easily modified to work independently.

SAP Hana

Currently, SAP Hana is not supported by Manta Flow. But since Manta Flow supports many other extractors such as extractor for the Oracle database or the Microsoft SQL Server, they provide the structure and useful help.
This chapter introduces the reader to the meaning of metadata and its usual usage.

### 1.1 Database metadata

Database metadata is data that describes saved data in the database. Relational databases usually have precisely defined structure of data that can be saved there and metadata describes these structures\(^3\). There are exceptions – for example non-relational databases (sometimes called non-SQL databases) usually do not require knowing the structure of some data before storing it. And even then it is possible to describe the structure and thus get metadata.

There is a need for metadata for every database engine currently used. The simplest case of a metadata usage can be database administration tools. These tools need to know which database objects are currently stored in the database – usually, this means schemes, tables, or table columns. Most databases also include complex objects such as procedures, functions, or macros. There are countless database clients that support many databases. Database vendors usually want to appeal to as much audience as they can, so they provide a way how to access the metadata.

Database metadata usually consists of table types and non-table types of objects. Of course, the default usage of the table type is a database table. But that is not all – many databases use table structures for many other purposes. For example, both Microsoft SQL Server and Oracle database offer a table variable, SAP Hana does not. Yet SAP Hana still offers arrays that can act like table variables anyway.

All of these structures are defined as database objects making it very easy to process such information in an automated tool. Formally, a database object is any defined object in a database that is used to store or reference data.\(^4\)
Database objects are usually stored in a hierarchical manner – such as tables are stored under a schema. Object types are database dependent, meaning that they will probably differ across multiple database engines (their query language dialects). Different methods can be used to retrieve such objects. The most common method is using database views – called system catalog. Database views describe the objects saved as virtual tables – a user can write a query that lists all objects of interest. The other methods include internal functions that return some information about an object or a full creation script of the object (also referred to as DDL – Data Definition Language). Other methods of a database object description are uncommon and not used by SAP Hana, Oracle, or Microsoft SQL Server.
This chapter describes the metadata found in the aforementioned databases. First in Section 2.1 it describes what kind of metadata can be found in the database of the interest – SAP Hana, continuing with Section 2.2 for Microsoft SQL Server. Finally, in Section 2.3 it shows the metadata structure of the Oracle database.

2.1 SAP Hana

SAP Hana is a tenant in-memory database system as shown in Figure 2.1. It does not have a collection of databases as the Oracle or Microsoft SQL Server. Instead, it contains separate tenant databases that act like different database servers without common users. This means that it is not possible to extract metadata from another tenant database.

SAP Hana supports catalog views and metadata functions. The views are mostly located in the SYS schema and all the user schemes. But there are also several other system schemes.

2.1.1 Architecture

An SAP HANA tenant database consists of multiple servers, for example, name server, index server, preprocessor server, and so on. The databases in an SAP HANA system run different combinations of these servers. The most important server is the index server. It contains the actual data stores and the engines for processing the data and runs in every tenant database [13].

There are two modes that SAP Hana can be installed in. Single container and multiple container system. To keep everything simple, this thesis will presume the multiple container system as that is now the default one and the most restrictive one. Multiple container system does have one system database, but the system database does not store metadata from the other tenants [1].
Another interesting part might be the XS server that allows user-created applications to run. This might be used for example for a custom REST API endpoint. Other servers are stated in Table 2.1.

SAP Hana also uses column-store based storage for tables by default. The easiest way to think about this is that it uses indexes instead of storing tables by rows. That means that it compresses (in a way) data such that if there is any record with the same value, SAP Hana stores it only once. And then points to two rows. This enables very quick aggregation queries and enables the usage of storing the data in-memory. This also comes with disadvantages – notably slower access of full table listings. But considering the typical usage of the SAP Hana database, this should not be a problem. Problems with data integrity are solved by periodical snapshots to disk, meaning that using SAP Hana for transactions incorrectly may be a big problem of data loss.

2.1.2 Object types

While the documentation explicitly does not state which objects types the database uses, it is not hard to query the SYS.OBJECTS view. The only problem is that this query returns also a few objects that belong to the XS server, so they are omitted in the Figure 2.2. A more elaborate description of the objects is in Chapter 3.
2.1. SAP Hana

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name server</td>
<td>The name server, which runs in the system database only, owns the information about the topology of the SAP Hana system, including knowledge of the tenant databases that exist in the system.</td>
</tr>
<tr>
<td>Index server</td>
<td>The index server, which runs in every tenant database (but not the system database), contains the actual data stores and the engines for processing the data.</td>
</tr>
<tr>
<td>Compile server</td>
<td>The compile server performs the compilation of stored procedures and programs, for example, SQLScript procedures. It runs on every host and does not persist data. It runs in the system database and serves all tenant databases.</td>
</tr>
<tr>
<td>Preprocessor server</td>
<td>The preprocessor server is used by the index server to analyze text data and extract the information on which the text search capabilities are based. It runs in the system database and serves all tenant databases.</td>
</tr>
<tr>
<td>SAP Web Dispatcher</td>
<td>The Web Dispatcher processes inbound HTTP and HTTPS connections to XS classic services.</td>
</tr>
<tr>
<td>SAP start service</td>
<td>The SAP start service is responsible for starting and stopping the other services in the correct order. It also performs other functions, such as monitoring their runtime state.</td>
</tr>
</tbody>
</table>

Table 2.1: Core services of the SAP Hana system

2.1.3 System views

SAP Hana’s system views are separated into two categories: metadata and runtime views. Metadata views provide metadata about objects in the database, including options or settings that were set using a DDL statement. Runtime views provide actual HANA runtime data, including statistics and
2. Analysis of the metadata

Figure 2.2: SAP Hana object structure

Tenant database
  Schema
    Table
      Column
    Custom type
      Column
    View
      Column
    Index
    Sequence
    Trigger
    Synonym
    Graph workspace
    Procedure
    Function
    Library
    Remote source
    Adapter

status information related to the execution of DML statements. Runtime views start with M_ for monitoring. For a static script analysis, only metadata views are interesting.

2.1.4 System schemes

Unfortunately, the documentation does not specify all system schemes, but looking at the names, one can come up with a pattern. These schemes will need to be excluded from the extraction process by default. System schemes include: schemes that have the _SYS_, SYS_, and SAP_ prefixes; schemes that have the XSSQLCC_AUTO_USER_ prefixes; SYS; SYSTEM; HANA_XS_BASE; UIS; SAPDBCTRL; and TEL_ADMIN.
2.1.5 Metadata access

Compared to the Microsoft SQL Server, SAP Hana does not offer the Information schema, only system views, and metadata functions. Each schema has its system views and each tenant database offers a SYS schema that contains objects for all schemes.

As the result is to extract metadata of all objects in the tenant database, the only useful schema is SYS. This also brings an advantage of easier permissions setting – there will be no access to the data saved in the database.

The second most important thing for metadata extraction is a function called get_object_definition. This function returns a DDL definition for most objects.

2.2 Microsoft SQL Server

Microsoft SQL Server is a traditional relational database. SQL Server offers several ways of accessing the database objects. Notably, there are system catalog views, information schema, and metadata functions.

2.2.1 Structure

Microsoft SQL Server uses a common structure with multiple databases in one instance. Each database contains multiple schemes. Figure 2.3 shows the way objects are hierarchically structured.

Figure 2.3: Microsoft SQL Server object structure

```
Instance
  └── Databases
      └── Schemas
          ├── Tables
          │    └── Views
          │         ├── Procedures
          │         │    └── Functions
          │         └── ...```

2.2.2 System catalog

The system catalog is understood here as a schema called SYS in each database. This schema is read-only and contains views with the metadata. Microsoft recommends using this method of metadata access as it is the most efficient to obtain and work with. [8]

Some catalog views inherit rows from other views such as the tables views inherits from the objects view. [8]
2. Analysis of the metadata

2.2.3 Information schema

The information schema is a way of standardized access to the metadata for database engines. Unfortunately only a few implement it. Its disadvantage may be that it can not show all metadata. It is understood as a schema called INFORMATION_SCHEMA in each database.

2.2.4 Metadata functions

Metadata functions offer a quick way of retrieving some of the metadata information on the database objects. They are not as useful for actual database structure extraction because all metadata functions are nondeterministic. This means these functions do not always return the same results every time they are called, even with the same set of input values.

2.3 Oracle database

Oracle Database does not support the Information schema. It only uses the system catalog for its metadata access. Oracle does not use the common database structure as database objects are owned by users and as such schema can not exist without a user. If one wants to create a new schema, one has to make a new user.

2.3.1 Structure

![Figure 2.4: Oracle object structure](image)

2.3.2 View types

There are many system views and they have their own groups. Some of them are owned by user SYS. The groups are:

- Views with the prefix USER: The views most likely to be of interest to typical database users are those with the prefix USER.
2.3. Oracle database

- Views with the prefix **ALL**: Views with the prefix **ALL** refer to the user’s overall perspective of the database. These views return information about schema objects to which the user has access through public or explicit grants of privileges and roles, in addition to schema objects that the user owns. [10]

- Views with the prefix **DBA**: Views with the prefix **DBA** show a global view of the entire database. Synonyms are not created for these views, because **DBA** views should be queried only by administrators. Therefore, to query the **DBA** views, administrators must prefix the view name with its owner, **SYS**. [10]

Considering the view groups stated above, we can deduce that the most useful views would be the **DBA** prefixed ones as we need a complete structure of the database.

2.3.3 Summary

All 3 databases offer very similar ways of accessing the metadata. The common way is to use a system catalog views that contain almost all information needed. Unfortunately, only Microsoft SQL Server offers a standardised way to access its metadata. This results in sometimes big differences in system catalog structures and naming. The one thing that differs the most is the DDL extraction. While the SQL Server and Oracle database include DDLs in system views, SAP Hana does not always do that. There is not a single way of accessing DDLs in the SAP Hana as some objects can be extracted only with `get_object_definition` and some can not be extracted that way.
This chapter shows which metadata are stored and which method will be used. This includes elaborate description of all database objects stored in the SAP Hana database – specifically the index server.

The goal here is to extract the metadata and this can be achieved in two ways. The first is to use system catalogs discussed in the previous chapter. The second one is to extract all the DDL scripts of the objects and then analyze them and extract their properties. If the system catalog offers enough metadata information for the Manta Flow, it is the preferred way. Fortunately SAP Hana does offer what is required, so system catalog will be used. If it is not explicitly stated, all views are presumed to be in the SYS schema.

3.1 Database objects description

This section describes some of the database objects of interest for the metadata extraction.

3.1.1 Schemas

Schemas can be listed using SCHEMAS view. The only this that is interesting here is the column SCHEMA_NAME. They contain all of the objects that are extracted. Beware of the PUBLIC schema that still contains synonyms.

3.1.2 Tables, collections and custom types

Tables, collections and custom types reside in the TABLES view. They all have column(s).

Custom types have flag IS_USER_DEFINED_TYPE, so it is important to filter them from tables. Custom type is a table structure that does not contain any data and can be used as a variable type – same as a table.
3. Metadata extraction

Tables can be virtual. Then they have a property `TABLE_TYPE` set to value `VIRTUAL`. A virtual table is a table that belongs to a remote source and its data is not stored in the database. Columns are still stored same as in local tables.

Another important thing about tables are its columns. They can be found by joining a `TABLE_COLUMNS` or `TABLE_COLUMNS_ODBC` view. They can be joined on a unique table ID `TABLE_OID`. The difference between `TABLE_COLUMNS` and `TABLE_COLUMNS_ODBC` is that the ODBC version also shows “structure columns”. This is useful for collections as collections do not have any columns in the non-ODBC version. A collection is a table that stores a JSON encoded data. Thus, it does not have any columns, only JSON structures. The only way to extract the collection data is to join the ODBC tables view – then the database adds a column with the JSON data.

Tables are similar to tables in other databases except for the fact that the default persistent tables are column-stored (temporary tables are row-stored by default). Column-stored tables mean that SAP Hana stores all the columns of the table separately – and that enables it to compress the data (it does not store the same values multiple times) and to efficiently make aggregations.

There also exist history column type tables. It means that they can “time-travel” – one can see data that were stored in this table at any point in time. One can set timestamp this per query or per session and by time or by commit id.

So persistent tables can be one of these types:

- column (`CREATE COLUMN TABLE` / `CREATE TABLE`) – default
- row (`CREATE ROW TABLE`)
- history column (`CREATE HISTORY COLUMN TABLE`)

And temporary tables can be:

- global
  - row (`CREATE GLOBAL TEMPORARY TABLE` / `CREATE GLOBAL TEMPORARY ROW TABLE`)
  - column (`CREATE GLOBAL TEMPORARY COLUMN TABLE`)
- local
  - row (`CREATE LOCAL TEMPORARY TABLE` / `CREATE LOCAL TEMPORARY ROW TABLE`)
  - column (`CREATE LOCAL TEMPORARY COLUMN TABLE`)
3.1.3 Views

Views are very similar to tables. They can be listed using the VIEWS view. Views also define columns, so it is important to join the VIEW_COLUMNS view on a unique view ID VIEW_OID.

Views provide a wrapper on some SQL queries. They differ from other dialects a bit because they can have parameters. Parameters are simple variables that one can use inside the view query.

3.1.4 Indexes

Indexes are not extracted by the Manta extractors, but it is important to list them as a commonly used database object. Indexes are structures based on top of columns that are helping to improve performance on reading. Their metadata can be listed by the INDEXES view.

There are different kinds of indexes:

- BTREE
- CPBTREE
- FULLTEXT
- FULLTEXT HASH
- INVERTED HASH
- INVERTED VALUE

BTREE and CPBTREE indexes are used on row-stored tables. Others are used on column-stored tables. All indexes can be UNIQUE.

3.1.5 Sequences

Sequences behave similarly as in the SQL Server or the Oracle database. The metadata reside in the SEQUENCES view and include information about:

- Sequence name
- Start number
- Minimal value
- Maximal value
- Increment by number
- Whether the sequence is cycled
3. Metadata extraction

3.1.6 Synonyms

Synonyms are alternate names for tables, views, procedures, or sequences. The metadata can be accessed by the **SYNONYMS** view.

There are two kinds of synonyms – private or public. Public synonyms are saved in the **PUBLIC** schema and are accessible from all schemes even without explicitly stating the schema. Private synonyms are saved in other schemes. When accessing an object via synonym, one must have the privileges to the base object.

3.1.7 Graph workspaces

Graph workspace is a representation of a graph. It contains a definition of vertices and edges – source to those can be tables, views, or synonyms. This object will not be extracted, at least in the initial version. It requires separate post-processing apart from the SQL scripts.

After creating a Graph workspace, one can make two kinds of queries on them:

- GraphScript function
- openCypher pattern matching

GraphScript is a language built-in SAP Hana. It is used in functions – flag **LANGUAGE GRAPH** when defining an SQL function is required. The function, when used, behaves like a normal SQL function.

openCypher pattern matching is a query language – it behaves like a source to a normal SQL query. The keyword for that is **OPENCYPHER_TABLE**.

3.1.8 Procedures and functions

Procedures and functions are pieces of SQLScript code. Procedures can be called with a **CALL** statement and functions can be queried as in **SELECT**. They have parameters that are of IN, OUT, or INOUT types. There can also be a **RETURN** type parameter. Procedures and functions reside in the **PROCEDURES** and the **FUNCTIONS** view respectively. SAP Hana supports three languages for procedures – SQLScript (default), R, and Graph script. There are two kinds of functions – Table User Defined Functions and Scalar User Defined Functions.

Table functions can have table types (be it a table, anonymous type – defined in the function definition or custom table type) as parameters, and they always return tables – again anonymous, table type, or table.

Scalar functions can not have table parameters, only primitive types and they return variables of primitive types. If they return only a single variable, one can call the function as if it returned only the value. If the function returns two or more variables, one have to address it with dot variable name – e.g. function(1,1).variable1 and function(1,1).variable2.
3.2. DDL scripts

3.1.9 Libraries
Libraries are containers for functions and procedures written in the SQLScript language. They reside in the **LIBRARIES** view. Wrapped functions and procedures can be imported by another SQLScript function/procedure. They cannot be used in SQL queries.

3.1.10 Remote sources
Remote sources represent a connection to another database. They use adapters to connect (to even different vendor databases). Then one can define functions and procedures in one’s instance as virtual and they will be run on a remote database. Or one can define a virtual table that represents a table or a view on a remote source.

3.1.11 Adapters
Adapters are links to other sources of data. They can be written as an external library or just a simple database object. Metadata can be found in the **ADAPTERS** view. Unfortunately implementing connections to other databases will not be implemented. It would require using an external library, so current implementation will not include adapters.

3.2 DDL scripts
As mentioned before, some objects do not include their DDL definition in their views. These include:

- All tabular types – tables, custom types
- Remote sources
- Schemas
- Sequences
- Synonyms

For these, it is required to use the `get_object_definition` function. Otherwise, most objects do have **DEFINITION** column in their system catalog view.
This chapter describes the extraction tool design. The first part focuses on what data will be extracted from the database (refer to Section 3 for a more detailed view on database objects). The second part focuses on the functional, non-functional requirements of the tool, and used technologies.

4.1 Extracted metadata

As per capabilities of the Manta Flow, the following metadata will be extracted:

- Schema
- Table
- Virtual table
- View
- Remote source
- Custom type
- Table function
- Scalar function
- Procedure
- Remote function
- Library
- Sequence
4. Design

4.2 Requirements

The extraction tool is primarily designed to work with other components of the Manta project. But despite that, it still allows for simple changes to the data structures used as only one part of it uses the Manta common parts.

4.2.1 Functional requirements

4.2.1.1 FR1: Metadata extraction

The extraction tool shall connect to the SAP Hana database and be able to extract all the necessary information about database objects along with their create scripts (also called DDL scripts).

4.2.1.2 FR2: Filtering schemes to extract

The extraction tool shall allow for a way to choose which schemes will be extracted.

4.2.1.3 FR3: Saving DDL scripts on disk

The extraction tool shall allow for saving the extracted DDL scripts to be saved on disk.

4.2.1.4 FR4: Saving metadata to Manta’s internal structures

The extraction tool shall allow for saving the extracted metadata to internal structures (data dictionary) of the Manta tool.

4.2.1.5 FR5: Filtering object types and DDL scripts to extract

The extraction tool shall allow for filtering object types to extract and for which object types to extract the DDL scripts.

4.2.2 Non-functional requirements

4.2.2.1 NFR1: Integration with Manta’s internal tools

All parts of the extraction tool shall allow for easy integration into Manta’s structures preferably by using defined interfaces.

4.3 Used technologies

The extraction tool will use Java programming language with the Spring framework, and MyBatis ORM tool. Tests will be using the JUnit framework.
4.3. Used technologies

4.3.1 Java with Spring framework

This programming language was chosen primarily because it is used by all the parts of the Manta Flow and that enables easier integration with existing code. Another reason is that Java still offers a lot of features while remaining de facto industry standard for all enterprise-grade applications. And thanks to great optimizations made in the Java Virtual Machine (JVM), it remains one of the best performing non-native programming languages.

Spring is another tool that remains very popular despite slowly becoming quite old. It offers features like dependency injection and lifecycle management which greatly improve readability and manageability of the code.

4.3.2 MyBatis

Although MyBatis is not as widespread, it offers an advantage compared to other ORM tools for this purpose – hand-written queries. It is useful there as queries to system views may get quite complicated and require careful handling of permissions. MyBatis offers two ways of defining queries. First is right into the java definitions and the second is to the separate XML files. The second option is used there. Also, it allows for easy integration with Spring, so everything looks cleaner.

4.3.3 JUnit

This testing framework was chosen again as it is widely used in Manta’s component and being industry-standard nicely integrated with the Spring framework.
This chapter describes the implementation of the extraction tool – classes and objects used to represent the database objects in the database. First, it shows a simple class diagram, then a description of the classes and finally objects used.

5.1 High level idea

The idea behind the metadata extraction tool is that it should contain all definitions for interesting database objects and then be able to extract them. So the project has been split between model classes and the classes that contain logical parts. The following classes will then describe only the logical classes. For Manta Flow, there are also a few more projects that define the internal structure of the database objects, but they are not part of this thesis.

The reality is that the objects can be extracted quite easily, but the problem comes with their processing as they have dependencies. For that an extraction order has been made (explained in the description of the method setExtractedDdlTypes in Subsection 5.2.1.6) and a graph-based dependency manager has been made (refer to the description of the class DictionaryWriter in Subsection 5.2.3).

5.2 Class diagram

Figure 5.1 shows a class diagram of the extraction tool. There is a common pattern shared among Manta’s extractors. For example, all extractors have the interface TechnologyExtractor and its implementation (Technology is substituted here by the name of the database).
5. Implementation

Figure 5.1: Extraction tool class diagram

5.2.1 SAPHanaExtractor

This interface is implemented by SAPHanaExtractorImpl. This interface is common for every extraction tool in most of the functions, but there are differences. For example SAP Hana’s extractor adds function `setExtractInvalid`. Its purpose is to abstract the common extractor functions. Beware that there are prerequisites for running the extractor – refer to the user manual at the end of this thesis.

5.2.1.1 setDictionary / getDictionary

`setDictionary` is a prerequisite for extraction as it sets the output metadata store. Data dictionary is the data structure where metadata is saved. `getDictionary` just returns what was set by the `setDictionary`.

5.2.1.2 setDdlOutputDirectory / getDdlOutputDirectory

`setDdlOutputDirectory` is a function that sets the directory where DDL scripts will be saved. `getDdlOutputDirectory` just returns what was set by the `setDdlOutputDirectory`.
5.2. Class diagram

5.2.1.3 setOutputDdlTypes

`setOutputDdlTypes` sets types of which DDL definitions will be extracted. This notes which DDL scripts will be saved to the disk outside of Manta’s structures.

5.2.1.4 setIncludedDbsSchemas / setExcludedDbsSchemas

These functions set the filters of schemes that get extracted or ignored. It uses class `SchemaFilter` that comes from the Manta common package. It is provided with a pattern for schemes and is used when determining if each given schema will be extracted.

5.2.1.5 setExtractInvalid

This function sets a switch whether objects that have been flagged in SAP Hana as invalid will be extracted. This is noted in the system views for many objects.

5.2.1.6 setExtractedDdlTypes

This is a very important function that sets which object types will get extracted after calling `extract`. Same as `setOutputDdlTypes` it uses enum `DdlType` that contains entry for each type that can be extracted.

5.2.1.7 extract

This function triggers the process. The first step is to check whether the extractor has been set up. This means that the data dictionary is set and the DDL output dictionary is valid. Then it proceeds to set a global namespace with the name of the server.

After all initialization, the extraction process begins. First schemes are extracted and filtered according to `setIncludedDbsSchemas` and `setExcludedDbsSchemas`. After extracting each schema, objects contained in the schema are extracted without objects that have dependencies. That means these objects in their respective order:

- Custom types
- Tables
- Virtual tables
- Views
- Sequences
- Triggers
5. Implementation

- Libraries
- Functions (both scalar and tabular)
- Procedures
- Synonyms

This is done by calling function `extractTypes` that checks whether the type is present in the `outputDdlTypes` and then calls a method for extracting that type – e.g. `extractCustomTypes`.

After that Remote sources are extracted. Then public Synonyms come (Synonyms in the public schema).

The last step is to extract objects with dependencies using the dependency manager.

5.2.2 HanaDAO

Interface `HanaDAO` is implemented by the class `HanaDAOImpl` and is responsible for extracting the metadata from the database. It uses `MyBatis` for connecting to the database. It contains methods for bulk extraction or simple one object extraction. That is useful for dependency resolution. It also does some basic filtering for system objects – most notably in schemes.

5.2.3 DictionaryWriter

Interface `DictionaryWriter` is implemented by the class `DictionaryWriterImpl`. It acts as an abstraction for every SAP Hana object as it writes to the data dictionary of the Manta tool using Manta’s common structures. This caused some problems as some SAP Hana’s objects are unusual. For example, SAP Hana’s views have arguments or scalar functions that return multiple variables.

It also uses the `DependencyManager` to manage objects that have dependencies. Usually, these are type dependencies – the object has parameters that use some custom type. Be it a table or custom type. Or this may be a function/procedure with library dependency that needs to be extracted before. Most of the extraction cycles, this is not a problem as the extraction order has been made not to make unnecessary dependencies, but this may happen with objects that reside in a schema that is not being extracted. Then these objects get written to the `DependencyManager` and extracted one-by-one in the correct order.

5.2.3.1 write*

All functions that share this pattern are used for abstracting and writing corresponding objects to the Manta’s data dictionary. They get the database
object as an argument, check dependencies, and use Manta’s data dictionary structures to represent the objects. They also convert all the data types using the `DataTypeNameSimplifier` to simple representations – all integer types of all sizes get reduced just to numeric types, all string representations are character, floating-point representations are float and all date/time representations are just datetime. For a more elaborate overview, see Appendix C.

5.2.3.2 persistChanges

This is an important function because it saves all objects currently saved in the data dictionary (previously written by the `write*` functions) to the disk. It may look like the last step or not so convenient thing to do because it invalidates all references currently pointing to objects in the data dictionary (also called flushing the object cache). But it is a necessity when the database contains a very high number of objects. Then it frees the memory from already extracted and not used objects. The disadvantage is that it must be assured that there are no references to objects from the data dictionary. That is why this method is called after extracting every schema. There are still references to database objects that need to be extracted (objects with unsatisfied dependencies), but they are not written to the data dictionary yet.

5.2.4 CreateScriptWriter

This class takes care of writing the scripts to the disk. It acts like a simple wrapper on the File/Path Java API. First the `SAPHanaExtractorImpl` calls `makeSureEmptyFolderExists` and that checks whether the path is valid, exists and can be written to. Then it removes old scripts from any previous run. Finally `createFile` is called for each script that is to be saved.

5.2.5 AliasKeeper

`AliasKeeper` is a utility that takes care of file names. There can be instances when there are two objects with the same name or the name can contain characters that are not supported on common file systems. So this class stores the file names that have been used and creates a unique file name for each script by appending a numerical index to the name.
Testing

This chapter shows different testing scenarios that were used. The extraction tool now has JUnit tests that were designed to cover most of the functionality. They differ in their need to have the SAP Hana database connection. Having a testing database has shown a big difficulty when developing the extraction tool as the SAP Hana database is very much a memory-based, meaning that it stores everything in the memory and only saves snapshots to disk in some intervals. One of the goals of the testing was to verify the memory and time performance of the tool. That was not accomplished, because the database requires an incredible amount of memory when faced with bigger data sets.

The tests were divided into two groups – one that requires the database connection and one that does not. Tests HanaDAOTest and HanaExtractorTest require a working connection, AliasKeeperTest, DependencyManagerTest, and DictionaryWriterTest do not.

For testing purposes, a testing database has been created. It contains two schemes – TESTSCHEMA1 and TESTSCHEMA2. Then there are DDL definitions for all extracted objects. TestResources contain corresponding objects as they would look after extracting. HanaDAOTest and HanaExtractorTest then compare the extracted objects to the predefined ones.
Disclaimer: This is a module of the MANTA tool. It has dependencies on some common parts and can not be built without them. If there is a need to use this tool without any external dependencies, it will require modifications. This manual presumes usage of the Spring framework.

This manual covers a case when one wants to add this project as a dependency and use it.

Steps:

1. First add this project as a dependency in your project dependency manager (e.g. Maven)

2. Then set data source – add a Spring bean with identifier "dataSource". There specify username, password and the database address. An example configuration can be found in the file `datasource-Hana.xml` in the test project.

3. Set the data dictionary to save objects to using the function `setDictionary`.

4. Set schema names to extract/ignore using the functions `setIncludedDbsSchemas` and `setExcludedDbsSchemas`.

5. (Optional) Set which object types will get extracted using the function `setExtractedDdlTypes`.

6. (Optional) Set which object types will get their DDL extracted using the function `setOutputDdlTypes`.

7. (Optional) Set whether to extract objects flagged as invalid by the SAP Hana using the function `setExtractInvalid`.

8. Finally run the extractor calling the function `extract`. 
Conclusion

The objective here was to describe the structure of the metadata of the SAP Hana database and compare it with Microsoft SQL Server and Oracle, then check its availability and ways of extraction. It was also necessary to choose the best way of metadata extraction given the different approaches and designing the prototype extraction tool. The final goal was to implement this extraction tool, test it, and create a user manual.

All of the stated objectives were achieved. A way for all metadata extraction was found. It was needed to combine two ways of extraction to achieve this as not all Catalog views provide the definition of the objects. Then the extraction tool was successfully designed, implemented, and tested. Test data cover most of the functionality of the extractor.

Except for the integration of the extraction tool, there are not many changes needed. The only thing that can be substantially improved is the support of external sources. Other than that, this tool will be serviced as other extractors in the MANTA project for common changes and SAP Hana updates.

The next part of the static SQL script analysis is the development of the script parser and resolver that will use this extractor and metadata information. It is out of the scope of this thesis, but this should work as a basis and information ground for the development of such projects.
Bibliography


12. SAP. *SAP HANA Services* [online] [visited on 2020-04-08]. Available from: https://help.sap.com/viewer/6b94445c94ae495c83a19646e7c3fd56/2.0.03/en-US/f0e6eb689f5648899749389c0894fd25.html.

13. SAP. *Server Architecture of Tenant Databases* [online] [visited on 2020-04-07]. Available from: https://help.sap.com/viewer/%206b94445c94ae495c83a19646e7c3fd56/2.0.03/en-US/f9aba40d6c4c4ae48ce461db4d4d88.html.

Acronyms

**DDL**  Data Definition Language  
**JVM**  Java Virtual Machine  
**ORM**  Object Relation Mapping  
**SAP**  Systems Applications and Products in Data Processing  
**SQL**  Structured Query Language
Permissions

This appendix describes the permissions needed to run the extraction tool for the SAP Hana database.

B.1 Listing the permissions/privileges

To list all privileges of a user, select all entries for the user in views GRANTED_PRIVILEGES and GRANTED_ROLES as there are roles that enable a user to be a server admin (have access to everything).

B.2 Minimal permissions

For the extractor tool, minimal permissions are useful as the tool will possibly run on a system full of confidential data and it is necessary to ensure security and privacy in case of a breach and leaking the connection properties of the extractor tool.

For that, an object privilege SELECT for the SYS schema is ideal. It allows for metadata access, but can’t be used to access data stored in the actual tables.

To grant an access, just execute GRANT SELECT ON SCHEMA SYS TO <user>;.
Data type simplifications

This appendix shows a full list that the `DataTypeNameSimplifier` uses. See table C.1. This means that this list contains all simple data types and its corresponding Manta’s variants. Manta Flow needs only a rough image of the types as each database comes with its own types and since many databases allow for connecting to another data sources (even SAP Hana does this), only data features are needed.
## C. Data type simplifications

<table>
<thead>
<tr>
<th>Type</th>
<th>Simplification</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>DECIMAL</td>
<td></td>
</tr>
<tr>
<td>SMALLDECIMAL</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>FLOAT</td>
</tr>
<tr>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
</tr>
<tr>
<td>VARCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td></td>
</tr>
<tr>
<td>ALPHANUM</td>
<td></td>
</tr>
<tr>
<td>SHORTTEXT</td>
<td></td>
</tr>
<tr>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>NSTRING</td>
<td></td>
</tr>
<tr>
<td>CLOB</td>
<td></td>
</tr>
<tr>
<td>NCLOB</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>SECONDDATE</td>
<td></td>
</tr>
<tr>
<td>DAYDATE</td>
<td></td>
</tr>
<tr>
<td>LONGDATE</td>
<td></td>
</tr>
<tr>
<td>SECONDTIME</td>
<td></td>
</tr>
<tr>
<td>VARBINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td>BLOB</td>
<td></td>
</tr>
<tr>
<td>BINARY</td>
<td></td>
</tr>
<tr>
<td>BINTEXT</td>
<td></td>
</tr>
</tbody>
</table>

Table C.1: Data type simplification
Appendix D

Contents of enclosed CD

- readme.txt: the file with CD contents description
- impl: the directory with the implementation
  - _manta-connector-saphana-dictionary-extractor: sources of the extraction tool
  - text: the thesis text directory
  - _thesis: the directory of \LaTeX source codes of the thesis
  - _BP_Ondrej_Hlavac.pdf: the thesis text in PDF format