TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

IDK40LT

Aleksandr Lapuškin 134290IAPB

OPEN-SOURCE SOLUTION FOR SCHEMA AND DATA MIGRATION BETWEEN TWO POSTGRESQL DATABASES

Bachelor's thesis

Supervisor: Martin Rebane
MSc
Lecturer

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

IDK40LT

Aleksandr Lapuškin 134290IAPB

AVATUD LÄHTEKOODIGA TARKVARA SKEEMI JA ANDMETE MIGREERIMISEKS KAHE POSTGRESQL ANDMEBAASI VAHEL

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc
Lektor

Tallinn 2016

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Aleksandr Lapuškin

21.05.2016

Abstract

Open-source solution for schema and data migration between two PostgreSQL databases

A large part of developing new or maintaining existing software is finding, analysing and removing defects, also known as bugs. Sometimes these defects are only reproducible using real-world data or configurations. However, often obtaining or installing this data on a developer's workstation is difficult to accomplish or requires a custom-made solution for each project.

This thesis describes a proposed solution to this problem in the form of a desktop application, which would allow users to easily migrate the structure and data of a single PostgreSQL database instance to another one, requiring minimal configuration to work.

A working GUI-based application, providing simple migration functionality between PostgreSQL database instances was written using the Java programming language.

This thesis is written in English and is 36 pages long, including 4 chapters, 9 figures and 7 tables.

Annotatsioon

Avatud lähtekoodiga tarkvara skeemi ja andmete migreerimiseks kahe PostgreSQL andmebaasi vahel

Oluline ja suur osa tarkvaraarenduse või -haldamise protsessist on defektide otsing, analüüs ja parandus. Nende parandamise käigus arendajad tavaliselt peavad defekti reprodutseerima, et oleks võimalik aru saada, mis on selle põhjus. Kuid väga tihti defektide reprodutseerimiseks peavad arendajate keskkonna andmed olema identsed selle keskkonnaga, kus defekt ilmnes. Kahjuks andmete kättesaamine on tavaliselt suhteliselt raske ülesanne ja iga projekt peab leiutama oma lahenduse.

Käesoleva bakalaureusetöö eesmärk on avatud lähtekoodiga tarkvara loomine andmete ja andmebaasiskeemi migreerimiseks ühest PostgreSQL andmebaasist teise andmebaasi, selle teoreetiline kirjeldus ja tehniline dokumenteerimine.

Tulemusena realiseeriti graafilise liidesega Java rakendus, mis võimaldab migreerida skeemi ja andmed ühest PostgreSQL andmebaasist teise andmebaasi, viies kasutajapoolse konfigureerimisvajaduse miinimumini.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 36 leheküljel, 4 peatükki, 9 joonist, 7 tabelit.

List of abbreviations and terms

ANSI	ANSI, which stands for the "American National Standards Institute", has served as coordinator of the U.S. private sector, voluntary standardization system for more than 90 years. ANSI coordinates the U.S. voluntary consensus standards system, providing a neutral forum for the development of policies on standards issues and serves as a watchdog for standards development and conformity assessment programs and processes. [1]
Deployment environment	A deployment environment or tier is a computer system in which a computer program or software component is deployed and executed.
GPL	The GNU General Public License is a free, copyleft license for software and other kinds of works. [2]
GUI	In computer science, a graphical user interface or GUI, is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation. [3]
Instance	A running process (usually a running program)
Java	Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.
JDBC	The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases – SQL databases and other tabular data sources, such as spreadsheets or flat files. [4]
PostgreSQL	PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. [5]

Schema	The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases). [6]
SQL	Structured Query Language is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system [7]
View (relational database)	In database theory, a view is the result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object. This pre-established query command is kept in the database dictionary. Unlike ordinary base tables in a relational database, a view does not form part of the physical schema: as a result set, it is a virtual table computed or collated dynamically from data in the database when access to that view is requested. [8]

Table of Contents

List of figures

List of tables

Introduction

Nowadays, when developing software, most developers are aware of the fact that their software is usually located in different deployment environments all at once. Usually several such environments are used. All of them serve a different specific purpose, but can be generally thought of as a sieve or a gate. Each preceding environment helps filter out defects in the software before the next environment. Each environment's configuration and state can also differ.

Below is a table describing the most common set of environments typically in use: [9]

Table 1 Popular deployment environments

Environment	**Description**
Local	Developers' own workstations
Development or Trunk	Development server
Integration	Continuous Integration build target, or for developer testing of side effects
Test or QA	This is the stage where unit testing, interface testing is performed. Quality analysis team make sure that the new code will not have any impact on the existing functionality and they test major functionalities of the system once after deploying the new code in their respective environment(i.e. QA environment)
Stage or Pre-Production	Mirror of production environment
Production or Live	Serves end-users/clients

Sometimes a developer is notified of a defect that is only present in one of the other environments, but is not normally reproducible on his own workstation. A common source of such defects is the Production environment, where end-users often input their own information, sometimes creating complex edge-cases or simply using the product in an unexpected fashion. In such cases, a developer might require that his own environment has the same state as the one where the defect originated from. Usually defects are accompanied by a report, mentioning which actions led to the discovery of the defect. While these instructions can help to match the machines' states, they are not always

sufficient. Sometimes the defect source environment's data is also required to be able to successfully reproduce the defect on the developer's workstation.

In such cases the developer might face a problem. Depending on the size and complexity of the system, he might be able to create the data manually. However, as the size and complexity of the database structure grows, it becomes increasingly more difficult and time-consuming. In other cases, scripts or other solutions might have already been created by other developers, which help copy the data to a developer's workstation. However, these solutions are almost always tailored to the underlying structure of the database, very often become quickly outdated and require constant maintenance if the database structure is prone to frequent changes. Again, this maintenance becomes time-consuming and, as a result, a small defect might cost the business a large sum of money to fix.

The goal of this thesis was to develop a standalone desktop application that would enable the user to connect to two PostgreSQL databases, view their structure, select which data and structures should be migrated and then begin the migration process.

1. Overview of currently available migration solutions

This chapter will attempt to show the differences between the currently available data and schema migration solutions. Below is a table providing a quick overview of the existing solutions. When reviewing the migration sources, only databases are considered, not any kind of other sources, such as files.

Table 2 Comparison of available migration solutions

Name	Migration From	Migration To	Open-source	Free
MySQL Workbench	Microsoft SQL Server, Microsoft Access, PostgreSQL, Sybase ASE, Sybase SQL Anywhere, SQLite, and more [10]	MySQL [10]	Yes	Yes
Advanced Query Tool	At least Oracle and DB2 [11]	At least Oracle and DB2 [11]	No	No
SQL Import/Export Wizard	SQL Server, Oracle, DB2 and any other source with an ODBC driver [12]	SQL Server, Oracle, DB2 and any other source with an ODBC driver [12]	No	Yes
Oracle Data Pump	Oracle [13]	Oracle [13]	No	Yes
MySQL Red Gate Comparison Bundle	MySQL [14]	MySQL [14]	No	Yes (for non-commercial use)

In the following chapters a small overview of each solution will be provided.

1.1 MySQL Workbench

MySQL Workbench is available under GPL. The tool itself has many other features apart from database migration, but that is the only feature that we will be analysing.

The tool supports a number of database vendors including Oracle, PostgreSQL and Microsoft SQL Server, but only allows migration to MySQL. This is a limiting factor and, while understandable, is biased towards the users using or switching to MySQL. It allows both schema and data migration, but requires some initial setup, such as having the required database ODBC drivers present. The interface is stylish, but a bit confusing to use at first.

1.2 Advanced Query Tool

It was difficult to actually understand which vendors are supported by this tool, seeing as almost no information is provided on their website. However, at least Oracle and DB2 are mentioned as being supported.

Advanced Query Tool has a limited functionality evaluation period of 30 days and a license cost of 45 USD. Its list of features is long and it also lists database migration as one if its features. However, it was not possible to test this functionality with the available evaluation license.

All in all, this tool seems to be good, judging by its list of features, but seeing as almost no concrete and specific information is provided on their website, it's difficult for developers to justify obtaining a license for it, especially considering that the evaluation license provides almost no functionality.

1.3 SQL Server Import and Export Wizard

This tool was unexpectedly unbiased in terms of supported database vendors, providing migration from and to databases provided by several vendors, such as SQL Server, Oracle, and others. However, it required a fair amount of setup in order to actually perform any kind of migration.

1.4 Oracle Data Pump

Oracle Data Pump allows users to migrate both schema and data from one Oracle database to another. While it is highly sophisticated, it is severely limited in terms of supported database vendors.

1.5 Red Gate MySQL Comparison Bundle

Allows users to compare and migrate the schema (no information available regarding data migration) between two MySQL database instances. Free for non-commercial use. Highly sophisticated, but limited in terms of supported database vendors.

1.6 Overview of comparison results

After reviewing the available solutions, it is clear that there are no open-source solutions that would allow simple and quick data and schema migration for PostgreSQL.

As such, it is clear that PostgreSQL is at a disadvantage when it comes to existing migration solutions.

2. Main Results

2.1 General description of the proposed solution

The general outline of the proposed solution is a GUI-based Java desktop application, which will allow the user to connect to two PostgreSQL databases, view their schemas, select the tables that the user wishes to migrate from one database (hereby referred to as the Source) to another (hereby referred to as the Target) and then actually perform the migration process. The process itself should require minimal configuration by the user.

2.2 Used software

- Java programming language
    - Highly versatile multi-purpose programming language
    - Many community-developed libraries
    - Powerful interface for database connectivity
- pg_dump – a command-line utility for extracting schema objects and data from a PostgreSQL database
- pg_restore – a command-line utility for importing schema object and data into a PostgreSQL database
- JavaFX – a Java library for building modern desktop user interfaces
    - This is a more modern way of building desktop GUIs than other pre-existing solutions, such as Swing. There are, of course, existing solutions for building browser-based GUIs, but using them would mean also using a backend server, which is unnecessarily complex for the task at hand.

2.3 Implementation details

2.3.1 General application structure

The application core and GUI is written using Java. A simple custom-made JDBC-based solution is used to retrieve the necessary database metadata. *Pg_dump* is used to copy the Source database's schema and/or data. *Pg_restore* is used to import the data acquired from the Source into the Target.

The general application structure is as follows:

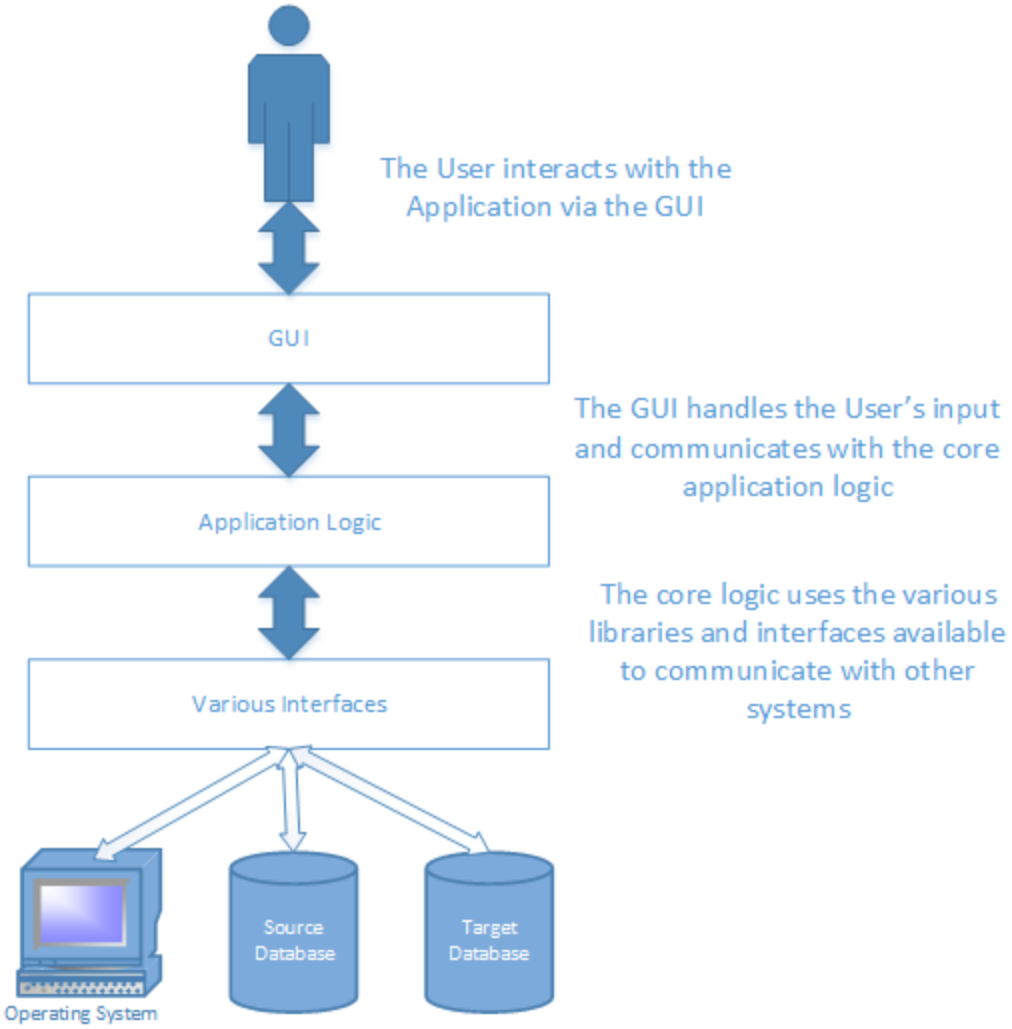

Figure 1 General application structure diagram

The application's logical structure can be described as follows:

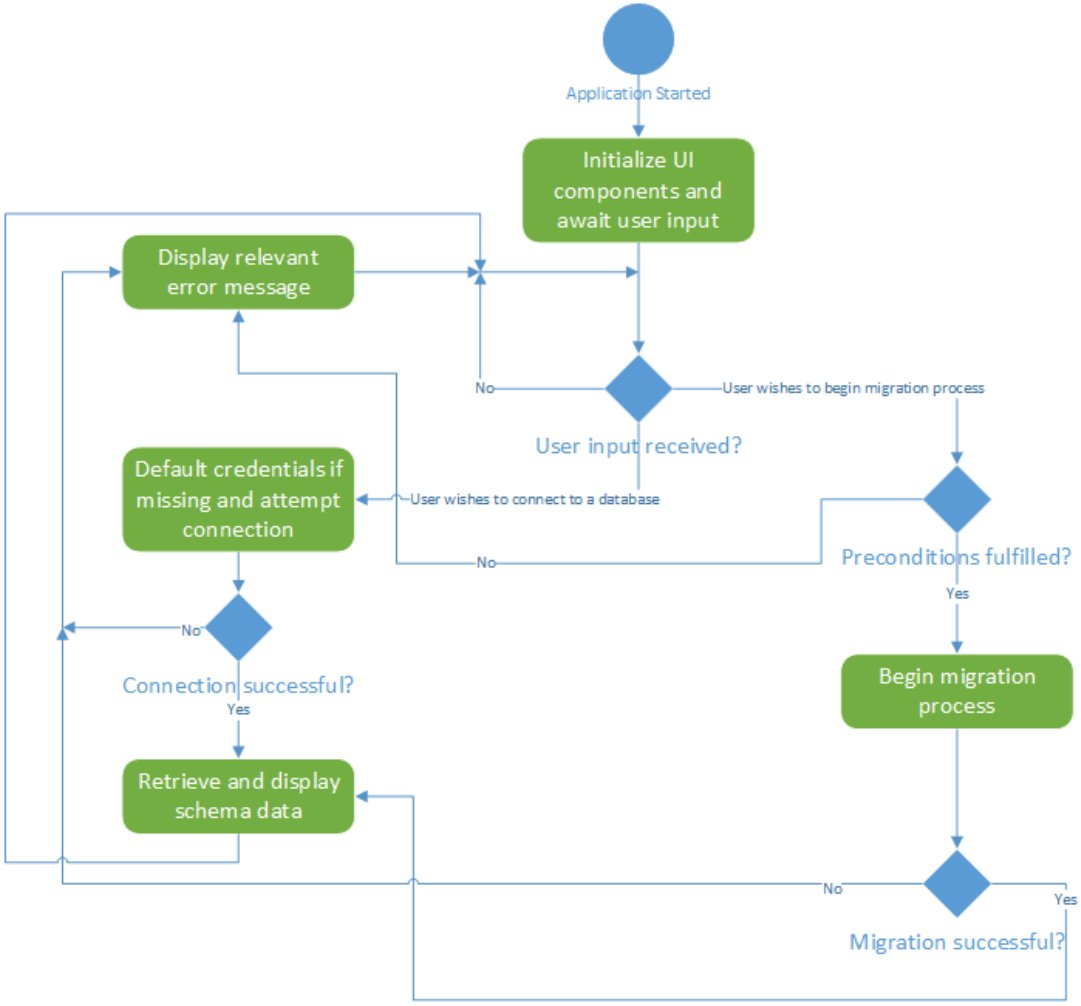

Figure 2 Application main logic loop

2.3.2 GUI Structure

The GUI has to be simple and easy to use. The following image describes the layout of the application's GUI.

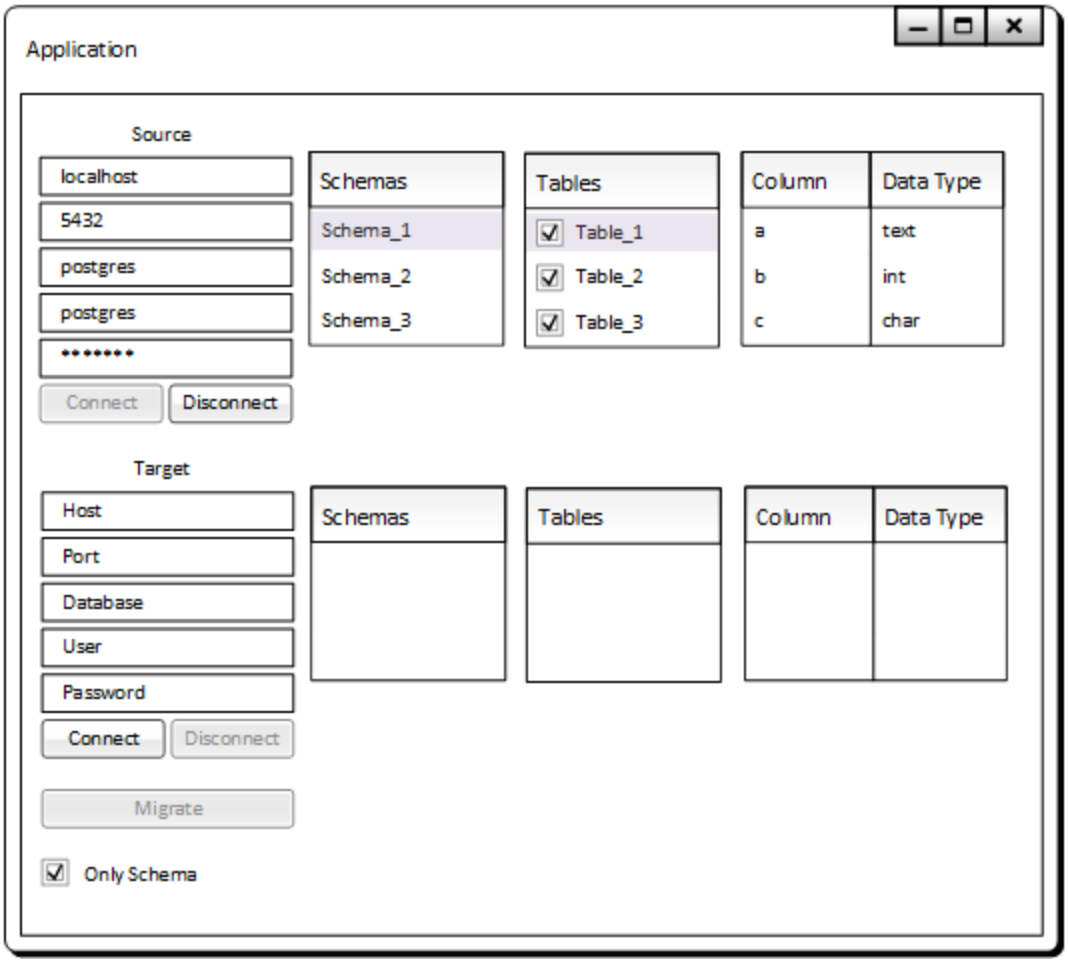

Figure 3 GUI Prototype

The GUI prototype above shows the program in a state when it is already connected to a Source database, but not yet connected to a Target database.

The GUI provides an easy way to enter the required connection credentials. Once connected, the *Connect* button is disabled, as are the credential input fields. Only the *Disconnect* button is enabled.

The tables to the left of the credentials input panel allow the user to browse the available schemas, their tables and the tables' respective columns. The user can choose to either not select any tables (which will be interpreted as the user wishing to migrate the whole

schema) or select only the tables that should be migrated. If any of the tables depend on another one, the dependencies are also automatically selected.

Once the user has connected to both databases, the *Migrate* button becomes enabled, allowing the user to perform the migration process.

2.3.3 Java classes

In this section a brief overview and description of the more important Java classes and their connections will be provided.

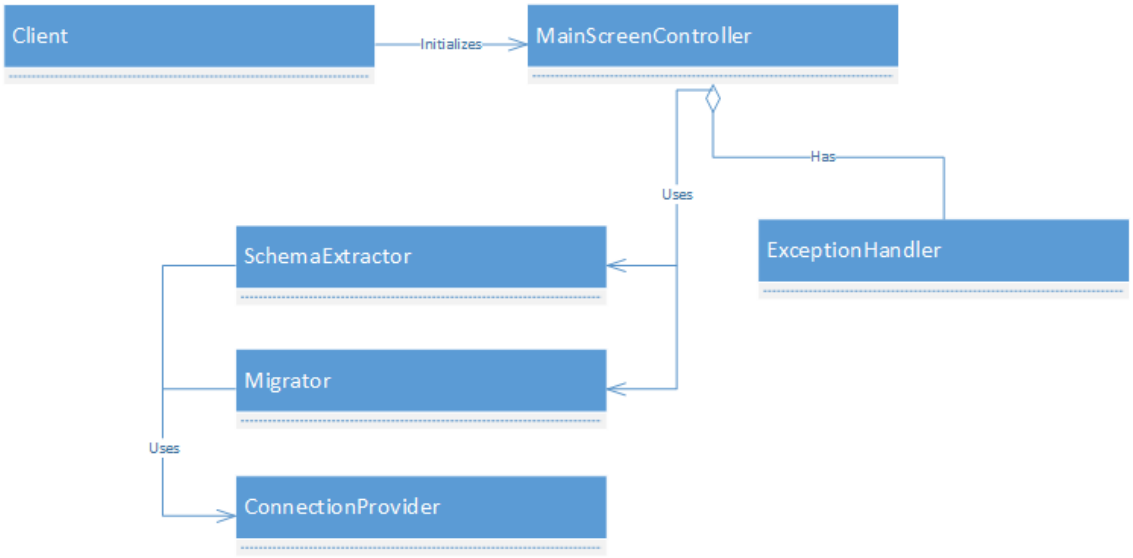

Figure 4 General Class Relationships

As can be seen on the diagram, the *Client* class is the main entry point of the program. Its *main* method is called to start the application. After this it loads the file describing the layout of the main screen. It is a JavaFX XML-based file format with an *.fxml* extension. The *MainScreenController* class is marked as being the controller responsible for this layout in the *.fxml* file. Due to this, the *Client* class also loads and instantiates the *MainScreenController* class, which then handles its own initialization accordingly.

When handling input from the user, the *MainScreenController* class can then access the *Migrator* and *SchemaExtractor* classes. These classes provide several static methods to retrieve the schema information of a database and perform the migration process. They are static because they have no state. As such, it would not make any practical sense to instantiate them. The *Migrator* and *SchemaExtractor* classes, in turn, have access to the

ConnectionProvider class, which, as the name implies, is used to retrieve and set up a JDBC connection.

In case any error occurs during execution, the *ExceptionHandler* class is used to handle them and display a relevant error message.

2.3.4 Connecting to a PostgreSQL database instance

Connecting to a database, provided a JDBC driver exists for it, is technically rather trivial. Each database vendor provides their own implementation of the JDBC connection protocol as a separate library. Once this library is loaded, the driver class needs to be registered. Usually the driver class itself provides a utility method to register it. After this it is possible to retrieve a JDBC connection via the *DriverManager* class, which is part of the Java standard library, by providing it with a connection URL, username and password.

2.3.5 Retrieving the available schema information

Creating an effective way to retrieve and browse the available schema information was a rather important piece of functionality right from the early stages of implementation. Various solutions were available, but initially schema information retrieval was performed with the help of the *SchemaCrawler* library by Sualeh Fatehi. This library provides a *SchemaCrawlerUtility* class, which takes a JDBC connection and returns a *Catalog* object, which contains all of the relevant information. Once the data is retrieved, the connection is no longer required, as the information is stored in memory.

This method has proven to be effective at first, as initial testing was performed on two databases that are located on the same machine. However, during the later stages of development it became abundantly clear that, while *SchemaCrawler* provides ample amounts of data in a rather simple to use fashion, that same quality is a double-edged sword. When retrieving schema information for a remote database, it would sometimes take several minutes to perform the retrieval, whereas the official PostgreSQL console would take mere seconds to perform the same task. The underlying cause is the same

quality that makes it great: a great amount of information that is largely independent from the underlying database vendor.

Of course, this might not be a very fair comparison, as *SchemaCrawler* retrieves all of the information at once, while the console does the same task incrementally as requested, but it was an issue either way.

In the end, it was decided that a custom-made solution has to be made, which also had to be as light-weight as possible. The solution to this problem were two SQL queries. One to retrieve a list of table names mapped to table names that they depend on. The other to select all of the schema information from a database. This information was then processed and converted to Java objects to display in the GUI. This method greatly reduced the time taken to retrieve the required info (down to a couple of seconds).

Below are the two SQL queries mentioned earlier:

```
SELECT constraints.table_schema,
       constraints.table_name AS object_name,
       constraint_column_usage.table_name AS dependency_name
FROM information_schema.table_constraints AS constraints
       JOIN information_schema.constraint_column_usage
       AS constraint_column_usage
              ON
constraint_column_usage.constraint_name = constraints.constraint_name AND
constraint_column_usage.constraint_schema = constraints.constraint_schema

WHERE constraint_type = 'FOREIGN KEY'

UNION

SELECT view_column_usage.table_schema,
       view_column_usage.view_name AS object_name,
       view_column_usage.table_name AS dependency_name
FROM information_schema.view_column_usage AS view_column_usage
WHERE view_column_usage.table_schema = view_column_usage.view_schema

ORDER BY table_schema, object_name, dependency_name;
```

Figure 5 Dependency Selection SQL Query

In the first part of the query (before the *UNION* clause), for every table or view which depends on another table, this query selects the dependent table's name and the dependency table name. It does so by joining several views in the *information_schema*

schema. In relational databases, the information schema (*information_schema*) is an ANSI-standard set of read-only views which provide information about all of the tables, views, columns, and procedures in a database. [15]

The view *table_constraints* contains all constraints belonging to tables that the current user owns or has some privilege other than SELECT on. [16] The view *constraint_column_usage* identifies all columns in the current database that are used by some constraint. [17] The query uses the first view to select all tables that have a foreign key constraint and then uses the second view to retrieve the names of the tables that contain the imported foreign key.

The second part of the query deals with views. The view *view_column_usage* identifies all columns that are used in the query expression of a view (the SELECT statement that defines the view). [18] This view is used to retrieve the view name, which depends on other tables, along with those tables' names.

```
SELECT table_schema, table_name, column_name, udt_name
       FROM information_schema.columns
              ORDER BY table_schema, table_name, column_name,
              ordinal_position;
```

Figure 6 Schema Information SQL Query

The query above deals with selecting all of the columns available for the given database. The view *columns* contains information about all table columns (or view columns) in the database. [19] As such, it is trivial to retrieve the necessary information. All of the information is directly available in the view: schema name, table name, column name and the data type name that is stored in the column.

2.3.6 Data migration

When handling schema and/or data migration, as with all kinds of information, it is extremely crucial to maintain data integrity. Otherwise the usefulness of the migration is diminished greatly, if not rendered completely useless.

Initially it was planned to handle all migration manually by generating dynamic SQL during application execution. However, it quickly became apparent that undertaking such

a task without excessive preparation and knowledge would be highly prone to errors. As such, it was decided to use tools that are built specifically for the purpose of extracting and loading the necessary information and instead implement an interface between these tools to provide seamless migration. Another key factor is the fact that using *pg_dump* on a database does not lock it, which allows other users to continue using it without any problems.

Most of the migration is performed with the help of the *pg_dump* and *pg_restore* command-line utilities. They are a standard part of every PostgreSQL installation.

The process itself can be described with the following figure:

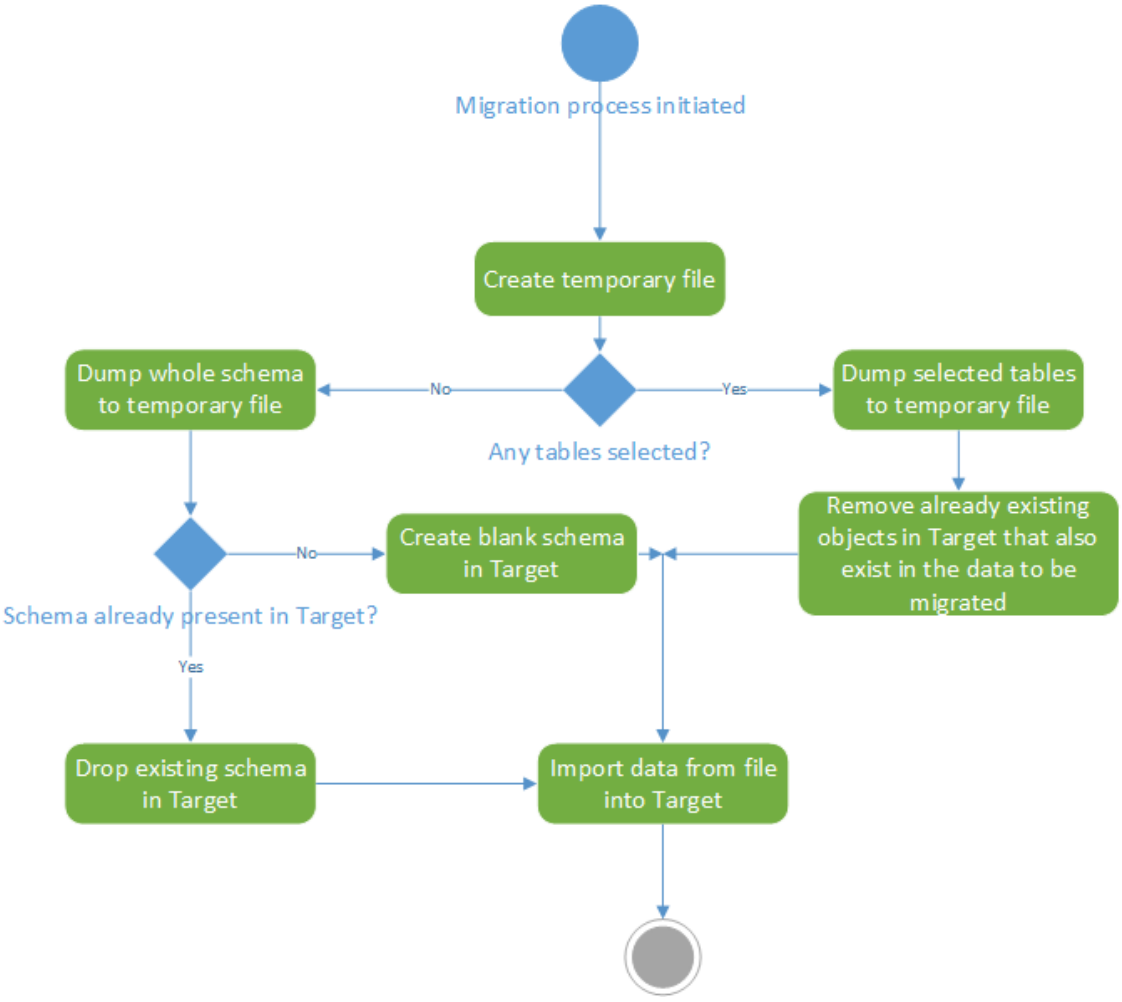

Figure 7 Migration process diagram

Based on the above diagram, one question might arise, namely, why is a temporary file used and not something along the lines of piping the output from *pg_dump* into

pg_restore? The answer is rather simple. After some testing, it became clear that it is not actually possible to pipe the output correctly either via Java or via the usual console invocation method. As such, there is a need to use an intermediate file, which is deleted immediately after migration.

2.3.7 Performance testing

An important part of migrations is the amount of time that it takes to perform a migration. Unless the speed is acceptable, the time used for migrations might be better used elsewhere.

In order to test the performance of the application, the following testing method was devised:

- A simple schema was created, resembling a real-world schema for a simple CRUD application
- Two remote PostgreSQL databases were obtained. One in the USA, one in Ireland. The schema was created in both of the databases
- Tables were filled with randomized data. The total amount of rows was 70000
- 10 migrations per database were performed to a machine in Tallinn, Estonia. The total migration time was measured using logging capabilities in Java, which also provided timestamps for the messages logged.

The testing results are given in seconds in the table below. Numbers in the topmost row represent the test sequence numbers:

Table 3 Performance testing results for 70000 rows

Location	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**
Ireland	14.45	14.01	15.29	15.78	15.17	14.30	13.91	15.79	14.90	15.38
US	33.65	34.10	33.32	33.67	33.71	33.86	32.20	31.86	34.59	33.94

Average for migrations from Ireland: 14.90 seconds.

Average for migrations from the US: 33.49 seconds

In order to determine the effects of the amount of data present in the database on migration times, the same tests were then performed again with databases containing 40000 and 10000 rows. Below are the results from those tests:

Table 4 Performance testing results for 40000 rows

Location	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**
Ireland	11.94	12.09	12.91	12.36	13.02	11.85	11.70	12.82	12.35	13.02
US	29.56	29.79	29.72	30.08	30.06	31.00	29.34	27.54	29.12	28.86

Table 5 Performance testing results for 10000 rows

Location	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**
Ireland	9.58	9.72	9.57	9.66	9.67	10.15	9.72	10.23	9.81	10.33
US	24.23	25.74	25.63	24.61	25.05	24.59	24.90	24.67	25.85	24.96

This data can then be graphed to observe the relationship between the amount of rows and migration times. First, a table has to be constructed for both locations:

Table 6 Average migration times from Ireland

Amount of rows	**Average migration time (seconds)**
10000	9.84
40000	12.41
70000	14.90

Table 7 Average migration times from USA

Amount of rows	**Average migration time (seconds)**
10000	25.02
40000	29.51
70000	33.49

The formula for determining the average migration time is as follows:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Where:

- i is the sequence number of the migration
- x_i is the migration time for the migration at sequence number i
- n is the amount of total migrations performed

The resulting graph showing migration times versus amount of rows for both locations is below:

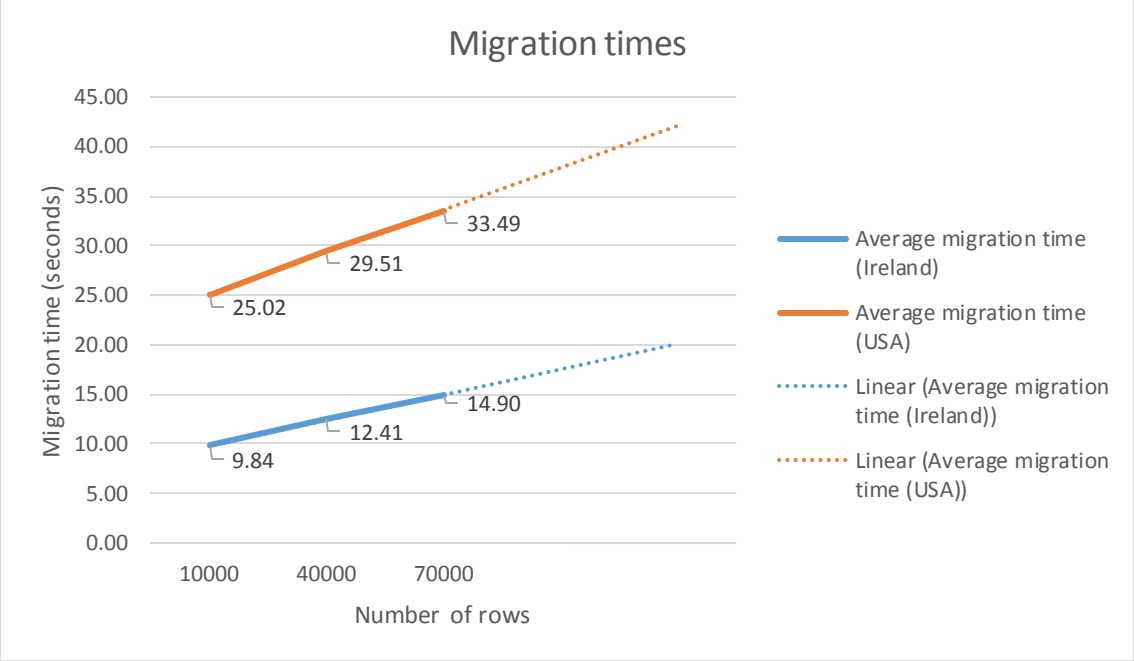

Figure 8 Migration times versus amount of rows

From the graph it is evident that the relationship appears to be rather linear.

As an example of the possible migration times, the trendline for Ireland can be used to calculate the rough migration time for an arbitrary amount of rows.

The equation for the trendline is a simple linear equation:

$$y = \alpha x + \beta$$

Where the slope of the trendline can be calculated as:

$$\alpha = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$$

And the offset is calculated via the following equation:

$$\beta = \frac{\sum y - \alpha \sum x}{n}$$

Where:

- x is the number of rows
- y is the migration time

From these equations, we can calculate the trendline equation for the Ireland migration times:

$$y = 8.43 * 10^{-5} x + 9.01$$

If the user wished to migrate a database from Ireland to Tallinn, Estonia that has 1 000 000 rows, it would take roughly 1.5 minutes to perform the migration, if the trendline equation is to be believed. The full graph can be observed below:

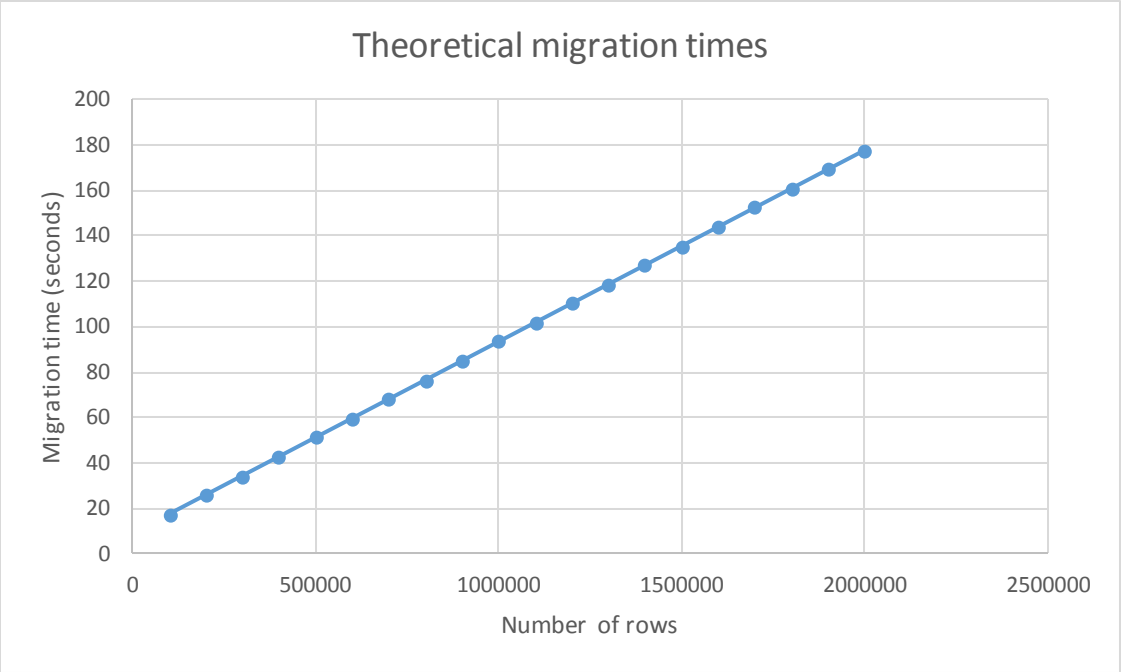

Figure 9 Theoretical migration times from Ireland to Tallinn, Estonia

The observed performance is rather impressive and should not, in theory, be a bottleneck for the users, seeing as the surrounding infrastructure of the test machines is in no way special or exceedingly powerful.

3. Discussion

This section is devoted to describing and discussing the known limitations and possibly unsolved problems.

3.1 Appending migrated data to existing data

Due to the nature of the application, it is not currently possible to append any data from a Source database to the data that already exists in a Target database. The difficulty of solving such a problem lies in the fact that real-world database structures vary greatly among themselves. As such, a solution that would be able to perform such a task has to be either extremely intelligent or extremely configurable.

A possible solution to this problem would be to create a kind of business key engine. In relational model database design, a natural key (also known as business key) is a key that is formed of attributes that already exist in the real world. For example, a US citizen's social security number could be used as a natural key. In other words, a natural key is a candidate key that has a logical relationship to the attributes within that row. A natural key is sometimes called domain key. [20] What this engine would allow is for the user to manually mark what tables represent which logical entities and what columns represent their business keys. A business key would then be used during migration to determine whether a logical entity from the Source database is already present in the Target database. In essence, this can be thought of as a unique constraint that is checked during migration.

3.2 Error recovery

Currently, only the bare minimum error management is in place. If any errors arise, the user is warned accordingly. However, no measures are in place that guarantee data integrity of the migration Target. This means that any unexpected error during the actual data or schema import can have a completely unknown effect on the migration target.

It is possible to solve this issue by introducing a target backup mechanism that would create a copy of the migration target before performing any modifications.

Another feature related to error recovery is the option to keep the intermediate migration file to avoid recurring data downloads in case of large databases.

4. Conclusion

The goal of this thesis was to document and implement an open-source schema and data migration utility for migrations between two PostgreSQL databases. The solution had to be a simple-to-use desktop application, requiring minimal configuration to work.

The goal of this thesis was successfully fulfilled. The problem at hand was analyzed, different components were used (both already available and newly created), tested and in the end, the best combination of components and design principles was chosen. The completed result allows users to easily, quickly and effectively solve the problem of obtaining a copy of a remote PostgreSQL database's schema and data. The finished solution is fully open-source and can be used on any platform that supports the Java programming language.

The whole analysis and design process was thoroughly documented and explanations were given for the design choices made. As an extra, a real-world performance test was performed, the data from which was used to create a theoretical performance prognosis for users wishing to use the tool.

However, improvements can, and most certainly should, be made. Currently only the bare minimum of error handling exists and any unforeseen error either during the export or import process could lead to data corruptions in the migration target. Extra measures have to be implemented in order to prevent any unwanted effects on the user's database, resulting in a better and more robust application.

References

[1] "ANSI. FAQ," ANSI, [Online]. Available: http://www.ansi.org/about_ansi/faqs/faqs.aspx?menuid=1. [Accessed 7 May 2016].

[2] "GNU General Public License," [Online]. Available: http://www.gnu.org/licenses/gpl-3.0.en.html. [Accessed 7 May 2016].

[3] "Wikipedia. Graphical User Interface," [Online]. Available: https://en.wikipedia.org/wiki/Graphical_user_interface. [Accessed 7 May 2016].

[4] "Oracle. Java SE Technologies - Database - JDBC," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javase/jdbc/index.html. [Accessed 7 May 2016].

[5] "PostgreSQL. About," [Online]. Available: http://www.postgresql.org/about/. [Accessed 7 May 2016].

[6] "Wikipedia. Database schema," [Online]. Available: https://en.wikipedia.org/wiki/Database_schema. [Accessed 7 May 2016].

[7] "Encyclopedia Britannica. SQL," [Online]. Available: http://www.britannica.com/technology/SQL. [Accessed 7 May 2016].

[8] "Wikipedia. View (SQL)," [Online]. Available: https://en.wikipedia.org/wiki/View_(SQL). [Accessed 7 May 2016].

[9] "Wikipedia. Deployment environment," [Online]. Available: https://en.wikipedia.org/wiki/Deployment_environment. [Accessed 9 April 2016].

[10] "MySQL Workbench: Database Migration," [Online]. Available: https://www.mysql.com/products/workbench/migrate/. [Accessed 26 April 2016].

[11] "Advanced Query Tool: Data Loader," [Online]. Available: http://www.querytool.com/tourload.html. [Accessed 26 April 2016].

[12] "SQL Server Import and Export Wizard," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ms141209.aspx. [Accessed 26 April 2016].

[13] "Oracle Data Pump," Oracle, [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14215/dp_overview.htm. [Accessed 26 April 2016].

[14] "MySQL Red Gate Comparison Bundle," Red Gate, [Online]. Available: http://www.red-gate.com/products/mysql/mysql-comparison-bundle/. [Accessed 26 April 2016].

[15] J. Melton and A. R. Simon, "19.2 Metadata, Repositories and The INFORMATION_SCHEMA," in *Understanding the New SQL: A Complete Guide*, 1993, p. 371.

[16] "PostgreSQL 9.5.2 Documentation, Chapter 34. The Information Schema, 34.50. table_constraints," [Online]. Available:

http://www.postgresql.org/docs/9.5/static/infoschema-table-constraints.html. [Accessed 28 April 2016].

[17] "PostgreSQL 9.5.2 Documentation, Chapter 34. The Information Schema, 34.17. constraint_column_usage," [Online]. Available: http://www.postgresql.org/docs/9.5/static/infoschema-constraint-column-usage.html. [Accessed 28 April 2016].

[18] "PostgreSQL 9.5.2 Documentation, Chapter 34. The Information Schema, 34.61. view_column_usage," [Online]. Available: http://www.postgresql.org/docs/9.5/static/infoschema-view-column-usage.html. [Accessed 28 April 2016].

[19] "PostgreSQL 9.5.2 Documentation, Chapter 34. The Information Schema, 34.16. columns," [Online]. Available: http://www.postgresql.org/docs/9.5/static/infoschema-columns.html. [Accessed 28 April 2016].

[20] "Wikipedia. Natural Key," [Online]. Available: https://en.wikipedia.org/wiki/Natural_key. [Accessed 1 May 2016].