# Advanced Migration of Schema and Data across Multiple Databases

D.M.W.E. Dissanayake

139163B

Faculty of Information Technology

University of Moratuwa

May 2017

# Advanced Migration of Schema and Data across Multiple Databases

D.M.W.E. Dissanayake

139163B

Thesis submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Degree of Master of Science in Information Technology.

May 2017

## Declaration

We declare that this thesis is our own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Name of Student Signature of Student

Date:

Supervised by

Name of Supervisor Signature of Supervisor

Date:

# Dedication

I dedicate this thesis to my dear parents for being great role models and encouraging me every day to achieve my goals in life.

# Acknowledgements

I would like to thank my thesis supervisor Mr. S. C. Premaratne of the Faculty of Information Technology at University of Moratuwa, for the guidance and support he provided throughout the whole research.

# Abstract

Migration of existing databases is an unavoidable task in modern technical and business environment. Most of the time, the DBAs are forced to work with a tight budget to choose the ideal software to accomplish the migration requirement of their organization and complete the process within a limited time period.

When considering the commercial and freely available data migration tools we found a number of disadvantages. Some of the tools are very expensive. Therefore the management of small scale companies may not approve purchasing such software unless they and are convinced it's a necessary expenditure as well as a good investment for the future. Even though they offer many features, some of the tools are suitable only for cloud based migrations. Some of the tools can only perform the migration process, inwards. Some tools are not available as standalone programs, so the customer has to purchase the whole software package which the required tool is included. Some tools don't provide GUI for the users. Therefore the user must have good programming knowledge in order to work with it.

In order to overcome these issues, I. M. Wijewardana presented a solution which offers an open and extensible migration process with attractive and user-friendly interfaces. However there are some important features yet to be implemented, including the features which have been suggested as future work by Wijewardana and colleagues.

With this research, we have extended the work of I.M. Wijewardana and implemented the following features. Handling the effects of primary key change of a table in between two data migration sessions. Informing the relevant details to the user or take action based on user preferences in an event of existing data replacement. Incremental update of data: Facility to update tables column-wise as well as row-wise, without causing any loss of data. Handling changes when the number of columns in a table (in source database / target database) gets increased / decreased. Handling changes when the constraints of table columns get changed. Facility to migrate the parent tables of a table which were initially selected to migrate based on relationships between the tables. Handling table and column name changes, column mapping, data type mapping. Migrating data based on different criteria (Data selection via SQL queries). Rollback database to a stable / pre-migration state in an event of unsuccessful data migration

# Contents

## List of Figures

## List of Tables

# Chapter 1

# Introduction

## 1.1 Prolegomena

Many well-known organizations have legacy information systems that are expensive to maintain and difficult to modify. These legacy systems can damage an organization's competitiveness, reputation and its viability. As time passes these become major issues and difficult to solve. Best way of overcoming these issues is to upgrade the existing system or transfer to a completely new system. Migration of existing databases is an unavoidable task in such a situation. A business merger is another instance where data migration becomes an unavoidable task (when the parallel systems in the two companies need to be merged into one).

However, this could become a tedious task, when a database is used not just as data storage, but also to represent business logic in the form of stored procedures and triggers. Therefore, close attention must be paid when performing the data migration. If the target database does not support the features available in the source database, then the appropriate changes should be made by the database migration application to overcome that issue and make sure the data migration process completes successfully.

Database migration by conventional methods is resource demanding, which requires allocating large teams and spending long hours in manual, tedious migrations with unpredictable migration results. But in some cases, seriousness of the data migration process and the duration of the data conversion process are underestimated by the top management. Therefore the DBAs have to work with a tight budget and complete the process within a limited time period. This puts a lot of pressure on DBAs as choosing wrong data migration tool could bring up to unexpected errors during migration process, which in turn could lead to loss of valuable business data.

Currently there are many products available for data migration (commercial products as well as free tools). Researchers also have proposed many approaches and developed several solutions in order to overcome these issues. After careful consideration, we concluded that none of the research solutions and currently available products that we had reviewed could meet our requirements without significant customization. Therefore, we decided to build a new database migration tool to perform our task.

## 1.2 Background and Motivation

The choice of suitable tools should be a part of any data migration plan. But in most cases, the managers don't pay much attention to tools designated to support data transfer. Therefore, the only tools used for data migration are the ones which the company already owns. These tools may not be designed specifically for data migration, and they're often made to support a combination of functions on a general level.

When using the right tool, it should examine elements of source database before migration. Then it should form the new target database, create tables if required, create indexes and then migrate data to the target database. If there are potential problems in data type's incompatibility, relations between source and target databases, the tool should flag them for attention.

We have checked the popular commercial and freely available data migration tools in order to find a suitable tool for data migration across industry leading database engines like MySQL, MS SQL Server, Oracle, PostgreSQL and etc. Because even though these database systems are similar to each other in some ways, their support for data types, metadata organization and internal data manipulation capabilities are different from each other.

## 1.3 Problem Definition

Migration and transformation of schema and data across multiple databases have been a research challenge.

## 1.4 Aim and Objectives

We aim to provide a reliable and user friendly application with advanced features, in order to assist Database Administrators with the process of migrating schema and data across multiple databases.

We wish to accomplish that by extending the work of I.M. Wijewardana and developing a solution which offers the following features.

- Handling the effects of primary key change of a table in between two data migration sessions (Eg: Source table PK is a single column, but target table has a composite PK)
- Inform the relevant details to the user / take action based on user preferences in an event of existing data replacement
- Incremental update of data: Facility to update tables column-wise as well as row-wise, without causing any loss of data.

- Handling changes when the number of columns in a table (in source database / target database) gets increased / decreased.
- Handling changes when the constraints of table columns get changed.
- Facility to migrate the parent tables of a table which were initially selected to migrate based on relationships between the tables.
- Handling table and column name changes, column mapping, data type mapping.
- Migrate data based on different criteria (Data selection via SQL queries).
- Rollback database to a stable / pre-migration state in an event of unsuccessful data migration

### 1.5 Structure of the Thesis

The rest of the thesis is structured as follow. The chapter 2 is on literature review of database migration tools. The chapter 3 presents technology adopted towards an automated database migration tool for migrating and transforming schema and data across multiple databases. The chapter 4 provides the overall picture of our novel approach to an automated solution for migrating and transforming schema and data across multiple databases. The chapter 5 discusses the design of the solution. The chapter 6 is about the implementation of the solution. The chapter 7 reports on the evaluation of the proposed solution. The chapter 8 concludes the thesis with a note on further work.

### 1.6 Summary

This chapter gave an overview of the automated solution for database migration. A brief introduction was given about database migration and this chapter provides a discussion of some of the background and motivation for this study. We defined the problem definition and the hypothesis for this thesis. This chapter mentioned about aim and objective of this research work. Next chapter shows the current developments of database migration area.

# Chapter 2

# Current Developments in Database Migration

## 2.1 Introduction

The choice of suitable tools should be a part of any data migration plan. But in most cases, the managers don't pay much attention to tools designated to support data transfer. Therefore, the only tools used for data migration are the ones which the company already owns. These tools may not be designed specifically for data migration, and they're often made to support a combination of functions on a general level.

When using the right tool, it should examine elements of source database before migration. Then it should form the new target database, create tables if required, create indexes and then migrate data to the target database. If there are potential problems in data type incompatibility, relations between source and target databases, the tool should flag them for attention.

We have read and reviewed work of other research teams to study current developments in data migration. The following sections describe how they carried out researches on data migration. Some of these papers focus on presenting a novel/revised approach for migration of data. Some of these papers focus on technical details and developing a practical solution. Some of these papers discuss issues that the developers have to face at each stage and instructions for solving/avoiding them. Some of these papers discuss the criteria to consider when developing a solution.

## 2.2 A Methodology For Data Migration Between Different Database Management Systems - By Bogdan Walek, Cyril Klimes

Data migration requires the proper coordination and management, because it is necessary transfer data and their structure to the new system correctly [1].
Walek and Klimes state that tools which are currently available for data migration between different relational database management systems have several disadvantages. They state that some tools have problems with migration of foreign keys which are a part of the relations between database tables. Another disadvantage is the impossibility of modifying or extending of existing tools. It is also impossible to change parameters and features of target database tables and their attributes. (E.g.: cannot change data types of the target database tables attributes)

By presenting this paper, Walek and Klimes aim to achieve two goals. Their first goal is to present a methodology for data migration between of RDBMS, which should help to reduce or eliminate disadvantages of the existing tools. The main steps of the methodology are as follows. Specification of the source and target RDBMS, loading the logical structure of the source database, proposal of suitable data types by expert system, selection of suitable data types, generating SQL dump file for creating the target database Their second goal is to present an expert system which can automatically map field data types of the target database tables with field data types of the source database tables.
This expert system contains a knowledge base that is composed of IF-THEN rules. Based on the input data, it suggests appropriate data types of columns of database tables. These researchers conclude their paper with presenting results of migrating data from a MySQL database to an Oracle database, based on their new methodology.

The scope of Walek and Klimes's research does not include maintaining source data integrity or validating data before migration. To address this matter, Balushi and colleagues present a paper with details of a framework for data migration between various types of RDBMS.

### 2.3 A Framework For Data Migration Between Various Types of Relational Database Management Systems - By Ahlam Mohammad Al Balushi

Data migration, although not a new problem, is never a simple and a straight-forward task (Morris, 2006; Lin, 2008).
Balushi and colleagues state that built-in data transfer utility offered by most RDBMS operate based on two crucial assumptions, which is a drawback. First, content of the source data is correct, especially with entity integrity, domain integrity and referential integrity [2]. Second, the target database has an identical structure to that of the source database.

Aim of this paper is to propose an improved framework for migrating database tables and their data between various types of RDBMS. The proposed framework is designed based on one of the existing approach which is expert system for data migration between different Database Management Systems [3]. By presenting this improved framework, authors expect to overcome the drawback mentioned earlier and solve gaps that exist in the expert system approach.

The proposed framework consists of three main steps which are divided into sub steps. First step is data acquisition. It includes specifying, identifying the source and target relational database systems and extracting or retrieving Relational Schema Representation (RSR) of the source Relational Database Second step is data preparation. It includes detecting and removing missing data, inconsistent and redundant data, etc from source data. In addition, data quality problems are classified into single-source and multi-source problems at this step. Third main step is data loading, which includes generating an SQL text file that contains data and physical model of the target database.

This paper focuses on solving part of single-source problems which arise at the data preparation stage. Single-source problems include lack of integrity constraints (schema level) and missing values, redundancy and wrong data (instance level).
Balushi and colleagues expect to solve the issues at instance level by carrying out Sorted Neighborhood Algorithm several times independently [4], on data source. Authors further state that this algorithm solves problem of redundant data efficiently by removing missing values or empty records and removing wrong data value through business rules.

Authors expect to solve the issues at schema level by taking each table in the source database and checking their primary keys and foreign keys to determine whether entity integrity and referential integrity are properly maintained. Details of primary keys and foreign keys were gathered earlier from metadata. If a relationship between tables is missing, this framework will allow the user to create the missing relationship first and then insert the records to relevant tables.

Balushi and colleagues conclude this paper with the suggestion that the proposed framework can be improved by enhancing its ability to migrate more than one RDBMS. They further suggest this can be achieved by focusing more on data cleaning issues of multi-sources in order to improve data quality.

### 2.4 Transform! Patterns for Data Migration – By Andreas Ruping

Because of the one-time nature, data migration effort and complexity is usually underestimated (Shepard, 2004; Russom, 2006).

Authors of this paper aim to give the readers a reasonable idea of what needs to be done when migrating data and how it should be done. They also aim to give a realistic feel for the underlying complexity in data migration.

There are several issues associated with any data migration project. Most common issues include the following: The legacy data might be complex and difficult to understand, the legacy data might be of poor quality, the amount of data can be rather large, the target data model might still be subject to change. (Morris 2006, Matthes Schulz 2011, Matthes Schulz Haller 2011, Fowler 2008, Keller 2000).

In this paper, Ruping and colleagues present 8 scenarios which address the issues mentioned above. Each of these scenarios contains 8 sections: context, problem, forces, solution, example, benefits, and liabilities. They demonstrate techniques and strategies that help to meet the typical requirements of a data migration project.

Six of the scenarios focus on issues which could arise during the design and development stages of a data migration application. The other two scenarios focus more on the data migration process. The scenarios are as follows. Making legacy data available to the new system. Preventing the migration process from unexpected failure, facilitating the analysis of problems that may occur during the transformation of possibly large amounts of data, preventing the new application from being swamped with useless data right from the start, catching errors in data transformation process, avoiding problems with the processing of mass data during the execution of your data migration process, avoiding unacceptable down times of your application during data migration process, avoiding trouble when the new application is launched.

## 2.5 A Metadata Driven Approach to Performing Complex Heterogeneous Database Schema migrations – By Robert M. Marks, Roy Sterritt

Software updates often involve data migration, especially when converting legacy software implemented to interface with outdated relational database management systems or other non-relational database electronic files [5].

Currently, the most popular way of executing a database upgrade is to run SQL scripts. This approach includes drawbacks such as having to execute thousands of SQL statements, having to support different migrate versions, having to support multiple database vendors These activities raise the likelihood of users making errors or scripts becoming out of sync.
Marks and Sterritt present a tool which auto generates most of the simple tasks (tasks which be achieved using a single SQL statement) and some of the more complex tasks. With this research, they expect to overcome the issues mentioned above. Adding, deleting and renaming an existing table, adding, deleting and renaming a column are considered as simple tasks. Manipulating data in place, handling column type changes, updating foreign keys, manipulating large objects and merging and splitting tables are considered as complex tasks.
The database migration tool (i.e.: Cutover Tool) presented in this paper is a metadata based Java application. It uses the JDOM library [6] for its XML parsing/creation. Its architecture is split roughly into three stages: cutover schema generation, manual updates, database upgrade.

At the cutover schema generation stage; first, it takes two database connections of the source and target database and then produces a basic cutover XML file specific to the database upgrade. At the manual updates stage; it allows the user to manually edit the XML file created earlier, and add complex operations which cannot be generated automatically. Then it inspects the XML file and makes sure that the generated schema is correct. Finally, it takes the edited XML file as input and executes it against the target database at the database upgrade stage. Authors state that this tool can be used only after the source database update is complete, which is considered a limitation.
They further state that work has already begun in developing a client–server/peer-to-peer application which continuously runs in the background (a monitoring agent). The main job of the tool will be to look for changes in the development/source database and to append these changes into a meta-data file.

This file will be constantly validated against a target database using the existing Cutover Tool, essentially creating self-migration and self-upgrades functionality into the system.
Marks and Sterritt conclude their paper stating that they expect the future research will focus on developing a fully autonomic, self-monitoring, self-adjusting and even self-healing data migration tool.

### 2.6 QuickMig : Automatic Schema Matching for Data Migration Projects
**- By Christian Drumm, Matthias Schmitt, Hong-Hai Do, Erhard Rahm**

This paper states that data migration requires solving two difficult tasks: matching schemas to identify similar or semantically related elements between the source and target systems, mapping discovery to determine mapping expressions which are capable of transforming instance data from the source format to the target format.

In this paper, Drumm and colleagues propose a new and integrated approach (i.e.: QuickMig) for schema matching and mapping discovery to support migration and transformation of data between heterogeneous sources. They further state that compared to previous work [7, 8, and 9], their approach exhibits the following improvements: Novel use of sample instances, New instance-based matchers, Comprehensive set of mapping categories, Enhanced mapping reuse, Schema reduction based on domain knowledge and Real-world evaluation.
Drumm and colleagues state that QuickMig is a further development of the schema matching tool COMA++ [10, 11]. QuickMig extends COMA++ by implementing three new instance-based matching algorithms, namely the Equality, the Split-Concat and the Ontology-based matcher, and by improving the reuse matcher.

The proposed migration system consists of 5 steps. First step is Answering a Questionnaire. In this step, a person with some knowledge of the capabilities of the source system will answer a questionnaire, to provide information about the source system as much as possible. Second step is Injection of Sample Instances. In this step, instances in the target system are manually created in the source system by a user. These sample instances are used by the instance-based matching algorithms in order to determine correspondences between the source and the target schemas.

In the third step, the source schemas and the corresponding sample instances are imported into the QuickMig system. The fourth step is Matcher Execution. In this step, the schema matching algorithms will be executed and a mapping proposal will be determined automatically using different matching algorithms. Developers can review and correct the mapping proposal in the final step. Then real mapping code is generated and stored in a mapping repository for later execution or reuse.

In a latter section they present the results of experimental evaluations carried out using real SAP schemas. According to those results, QuickMig was able to identify the correct mapping categories with an average precision of 0.97.

Drumm and colleagues conclude the paper with stating their future plan to prototypically integrate the QuickMig approach into SAP data migration tools and applying it in further scenarios.

## 2.7 Criteria for Evaluating General Database Migration Tools

**- By Bin Wei, Tennyson X. Chen**

Choosing the right DMT (Data Migration Tool) can be vital to the fate of a software project that might directly contribute to the success of a business operation. Yet there are few guidelines in how to evaluate the usefulness and effectiveness of a general DMT. With this paper, Wei and colleagues discuss five criteria in detail that can serve as standards for current and future DMT products.

By presenting this paper Wei and colleagues expect to achieve two goals. First, assisting software development project managers to evaluate general DMTs and make informed decisions when facing data conversion tasks. Second goal is, providing guidelines to software developers on design and implementation considerations for future DMT products.

Authors of this paper state that the first criterion is the Types of databases that a DMT is able to support. Second criterion is User interface configurability, maintainability, and reusability. This includes checking whether a DMT can perform the following tasks. Selecting specific tables and columns to transfer, Adding, changing, or removing column names, types, or other properties, Adding, changing, or removing constraints like primary key, foreign keys, and other properties. Adding, changing, or removing view, functions, or other utilities in the destination database. Third criterion is Support for data integrity. This means that a good DMT should support entity integrity, referential integrity and domain integrity.

Fourth criterion is customization adaptability of the DMT. Authors state that a good DMT allows users to change data in the source database and make necessary adjustments in the destination database. A good DMT allows users to write customized code to be incorporated into the data transferring process to implement special business rules when writing the data into the destination database. A good DMT allows users to analyze dependencies among tables in the source and destination databases and arrange the data transfer in the correct sequence.

Fifth criterion is data correctness verification. To check data correctness in columns with “Numeric” data type, researchers have compared the maximum, minimal, average, and summary values between the two databases. For columns with “String” data type, they have compared the string length and checksum values between the two databases. For columns with “Date/Time” data type, they have converted the values into the numeric representation and have compared the databases with the method applied on Numeric columns.

In a latter section, authors of this paper mention that migration performance and cost is another important criterion which should be considered when evaluating data migration tools. Wei and colleagues state that the complexity of database migration varies from project to project. Therefore, depending on the data conversion task, database administrators may be interested in different features of a DMT, and they may not consider each criterion with equal weight. This paper also includes details of the real life projects they worked on, requirements of each project and how the suitable tools were selected. They present a comparison of popular DMTs, on how their ability to handle the criteria discussed above. Wei and the research team state that a complex ETL system may go beyond what these criteria can evaluate (eg: complex data like image, audio, and video files).

They conclude the paper with the suggestion that future work should focus on how to evaluate the migration of complex data, as these areas will help developing a complete set of data migration evaluation standards.

### 2.8 Migrate and Transfer Schema and Data across Multiple Databases

**- By I. M. Wijewardana**

This research offers a Database Migration Tool that supports migration of database schema and data across industry leading databases such as Oracle, MS SQL Server, MySQL, and PostgreSQL [12].

They have discussed advantages and drawbacks of popular data migration tools including FlySpeed, ESF Database Migration Toolkit, SwisSQL, MySQL Workbench, Microsoft SQL Server Migration Assistant (SSMA) and Oracle SQL Developer. One of the products is very expensive. Owners of another product have decided to discontinue any development and investments on their product. Some of these products have limited functionality. (i.e.: They can only migrate data between 2 or 3 database engines out of Oracle, MS SQL Server, MySQL, and PostgreSQL)

Wijewardana and colleagues expect to overcome the issues in currently available data migration tools and provide a user friendly solution with more features. Main objective of their research is to present a solution which offers a flexible, open and extensible migration process. The research team also claims that their solution is capable of reorganizing and transforming schema and data with ease and it can rapidly migrate data across multiple databases ensuring data integrity with no loss of data.

The solution is developed on top of Java technologies such as NetBeans Platform APIs and Java Database Connectivity API. A user is presented with a wizard consist a series of visual panels. A visual panel represents a step in the migration process (i.e.: Creating/opening a project, selecting source database, selecting target database, selecting objects to copy, database mapping, table mapping, data migration summary). Each of these visual panels allows users to select different options and move forward until the process is complete. Users are allowed to go back and forth between visual panels and change previously selected options.

Authors state that the software solution can be tested with respect to different aspects such as functionality, reliability, usability, efficiency, maintainability and portability. This research team has tested their solution with Squish Test Automation Suite [13].

Wijewardana and colleagues conclude their paper with the following suggestions as future work.

Migrating data based on simple as well as complex criteria (output generated by running SQL queries). When migrating a table, its dependent tables should also be migrated based on its relationship. Handling table name changes, column name changes, data type mapping, etc. Migrating only the columns specified by user. Migrating stored procedures, functions, triggers and table indexes. Taking backups of tables in target database before migration. Appending new data into the table without corrupting the existing data.

### 2.9 Problem definition - Research Question

The above study shows that there are numerous limitations in current methods/applications of migrating schema and data across multiple databases. Based on the above, the research problem is defined as unavailability of an application which can successfully migrate schema and data across multiple databases. Solution for this problem is developing an application which can successfully migrate schema and data across multiple databases. No adequate studies have been done in migrating schema and data across Oracle, MS SQL Server, MySQL and PostgreSQL databases. We intend to solve the problem using Core Java APIs, Java Database Connectivity API and NetBeans Platform APIs with the above mentioned databases.

### 2.10 Summary

This chapter described how other researchers carried out their researches on data migration. Some of these papers focus on presenting a novel/revised approach for migration of data. Some of these papers focus on technical details and developing a practical solution. Some of these papers discuss issues that the developers have to face at each stage and instructions for solving/avoiding them. Some of these papers discuss the criteria to consider when developing a solution. The next chapter presents technology adopted to solve the research problem

# Chapter 3

# Technology Adopted for Database Migration

## 3.1 Introduction

Chapter 2 presented the current developments for giving an automation solution for migrating and transforming schema and data across multiple databases. This chapter presents the technologies to develop the database migration tool with industry leading databases such as Oracle, SQL Server, MySQL and PostgreSQL. The chapter highlighted the technologies that we are going to adopt to develop the solution.

## 3.2 NetBeans Platform

We used the NetBeans IDE 8.1, NetBeans platform with Java 8 for developing the front end of our software application. We chose NetBeans Platform due to the wide range of out-of-the-box components it can offer to the developers. The main reusable features and components comprising the NetBeans Platform are outlined below.

### 3.2.1 NetBeans Platform: Module System

The modular nature of a NetBeans Platform application gives developer the power to meet complex requirements by combining several small, simple, and easily tested modules encapsulating coarsely-grained application features. Powerful versioning support helps give confidence that modules will work together, while strict control over the public APIs modules expose will help developer create a more flexible application that's easier to maintain.
Since an application can use standard NetBeans Platform modules or OSGi bundles, developer are able to integrate third-party modules or develop his own [14].

### 3.2.2 NetBeans Platform: Lifecycle Management

NetBeans runtime container provides lifecycle services to Java desktop applications.
NetBeans runtime container understands how to compose NetBeans modules into a single Java desktop application. There is no need to write a main method for an application because the NetBeans Platform already contains one. Also, support is provided for persisting user settings across restart of the application, such as, by default, the size and positions of the windows in the application.

### 3.2.3 NetBeans Platform: Plugability, Service Infrastructure, and File System

End users of the application benefit from pluggable applications because these enable them to install modules into their running applications. NetBeans modules can be installed, uninstalled, activated, and deactivated at runtime, thanks to the runtime container.

The NetBeans Platform provides an infrastructure for registering and retrieving service implementations, enabling to minimize direct dependencies between individual modules and enabling a loosely coupled architecture (high cohesion and low coupling). The NetBeans Platform provides a virtual file system, which is a hierarchical registry for storing user settings, comparable to the Windows Registry on Microsoft Windows systems.

It also includes a unified API providing stream-oriented access to flat and hierarchical structures, such as disk-based files on local or remote servers, memory-based files, and even XML documents.

Window System, Standardized UI Toolkit, and Advanced Data-Oriented Components Most serious applications need more than one window. Coding good interaction between multiple windows is not a trivial task. NetBeans window system lets users maximize/minimize, dock/undock, and drag-and-drop windows, without providing any code at all.

Swing and JavaFX are the standard UI toolkits on the Java desktop and can be used throughout the NetBeans Platform. Related benefits include the ability to change the look and feel easily via Look and Feel support in Swing and CSS integration in JavaFX, as well as the portability of GUI components across all operating systems and the easy incorporation of many free and commercial third-party Swing and JavaFX components.

With NetBeans Platform, developers are not constrained by one of the typical pain points in Swing, JTree model is completely different to the JList model, even though they present the same data. Switching between them means rewriting the model. The NetBeans Nodes API provides a generic model for presenting data. NetBeans Explorer & Property Sheet API provides several advanced Swing components for displaying nodes.

In addition to a window system, NetBeans Platform provides many other UI-related components, such as a property sheet, a palette, and complex Swing components for presenting data, a Plug-in Manager, and an Output window.

### 3.3 MySQL Workbench

MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. MySQL Workbench is available on Windows, Linux and Mac OS X [15].

It enables model-driven database design, which is the most efficient methodology for creating valid and well-performing databases, while providing the flexibility to respond to evolving business requirements. Model and Schema Validation utilities enforce best practice standards for data modeling. It also enforces MySQL-specific physical design standards so no mistakes are made when building new ER diagrams or generating physical MySQL databases.

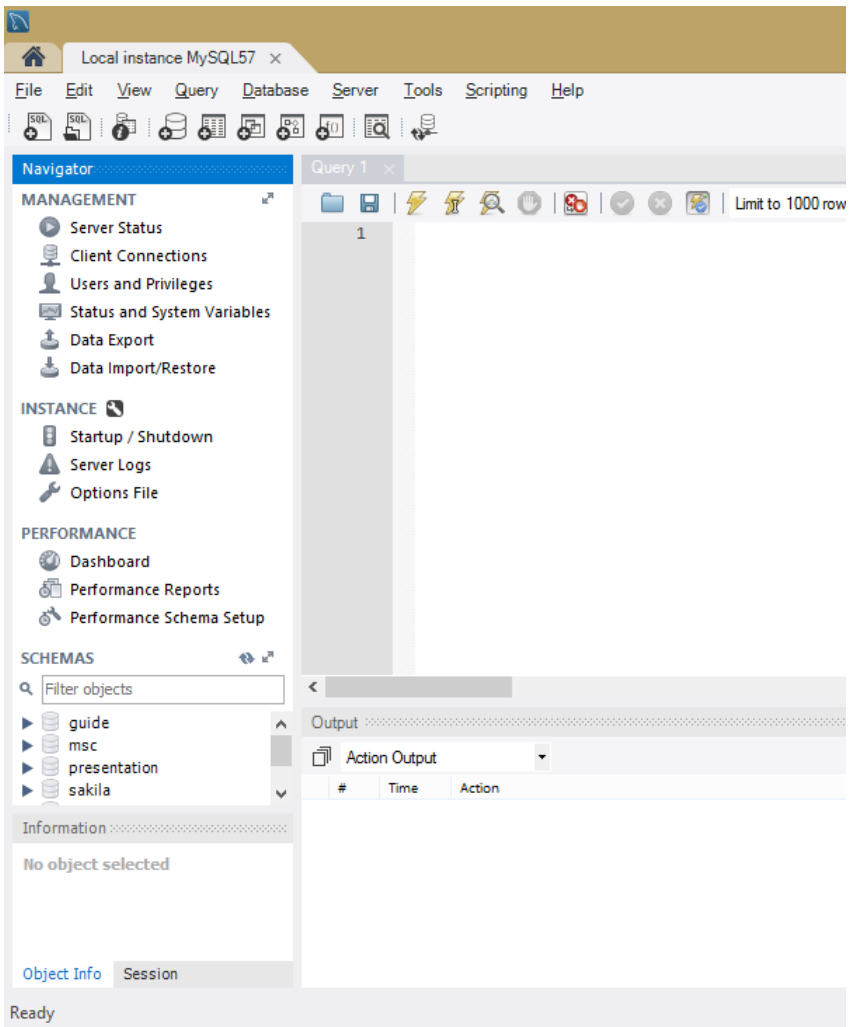


Figure 3.1: MySQL Workbench main screen window

### 3.4 pgAdmin

pgAdmin is a popular and feature rich Open Source administration and development platform for PostgreSQL databases. The application may be used on Linux, FreeBSD, Solaris, Mac OSX and Windows platforms to manage PostgreSQL 7.3 and above. PgAdmin is designed to answer the needs of all users, from writing simple SQL queries to developing complex databases. The graphical interface supports all PostgreSQL features and makes administration easy. The application also includes a syntax highlighting SQL editor, a server-side code editor, an SQL/batch/shell job scheduling agent, support for the Slony-I replication engine and much more. Server connection may be made using TCP/IP, and may be SSL encrypted for security. No additional drivers are required to communicate with the database server [16]. pgAdmin is developed by a community of PostgreSQL experts around the world and is available in more than a dozen languages. It is Free Software released under the PostgreSQL License.

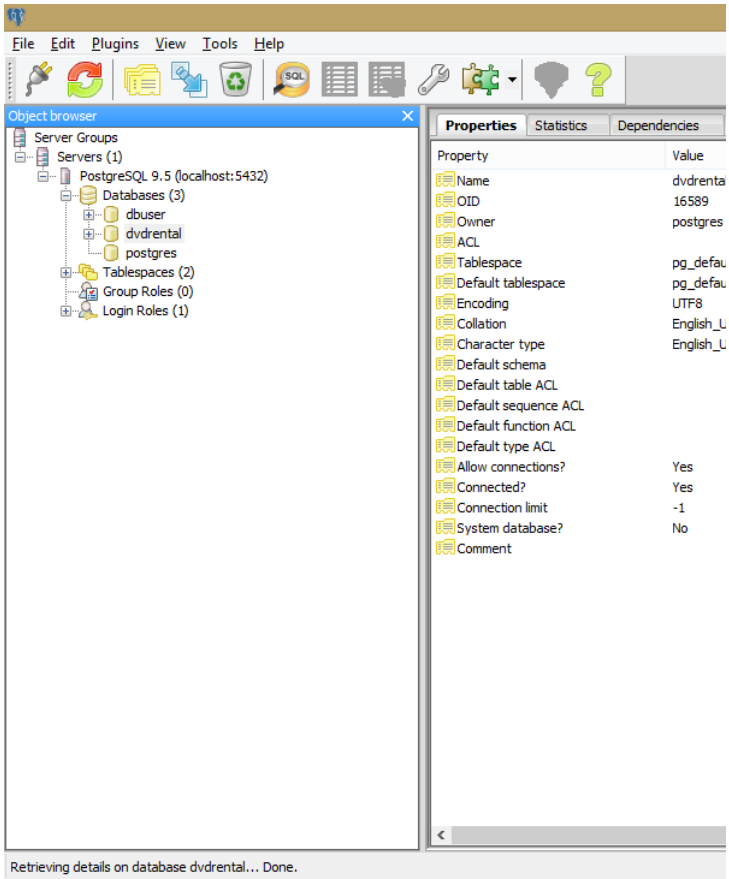


Figure 3.2: pgAdmin III for PostgreSQL

### 3.5 SQL Server Management Studio

SQL Server Management Studio (SSMS) is an integrated environment for accessing, configuring, managing, administering, and developing all components of SQL Server. SSMS combines a broad group of graphical tools with a number of rich script editors to provide access to SQL Server to developers and administrators of all skill levels [17].

SSMS combines the features of Enterprise Manager, Query Analyzer, and Analysis Manager, included in previous releases of SQL Server, into a single environment. In addition, SSMS works with all components of SQL Server such as Reporting Services and Integration Services. Developers get a familiar experience, and database administrators get a single comprehensive utility that combines easy-to-use graphical tools with rich scripting capabilities.

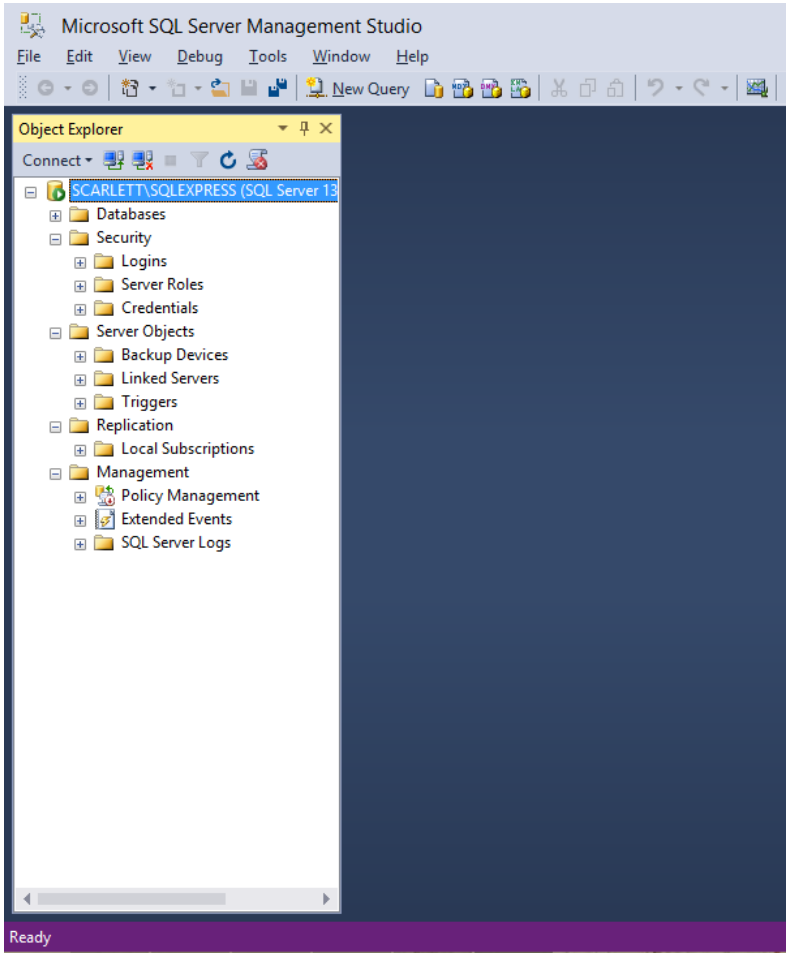


Figure 3.3: Microsoft SQL Server Management Studio

### 3.6 Oracle SQL Developer

Oracle SQL Developer(“Oracle SQL Developer,” 2016) is a free integrated development environment that simplifies the development and management of Oracle Database in both traditional and Cloud deployments. SQL Developer offers complete end-to-end development of the PL/SQL applications, a worksheet for running queries and scripts, a DBA console for managing the database, a reports interface, a complete data modeling solution, and a migration platform for moving a third party databases to Oracle.

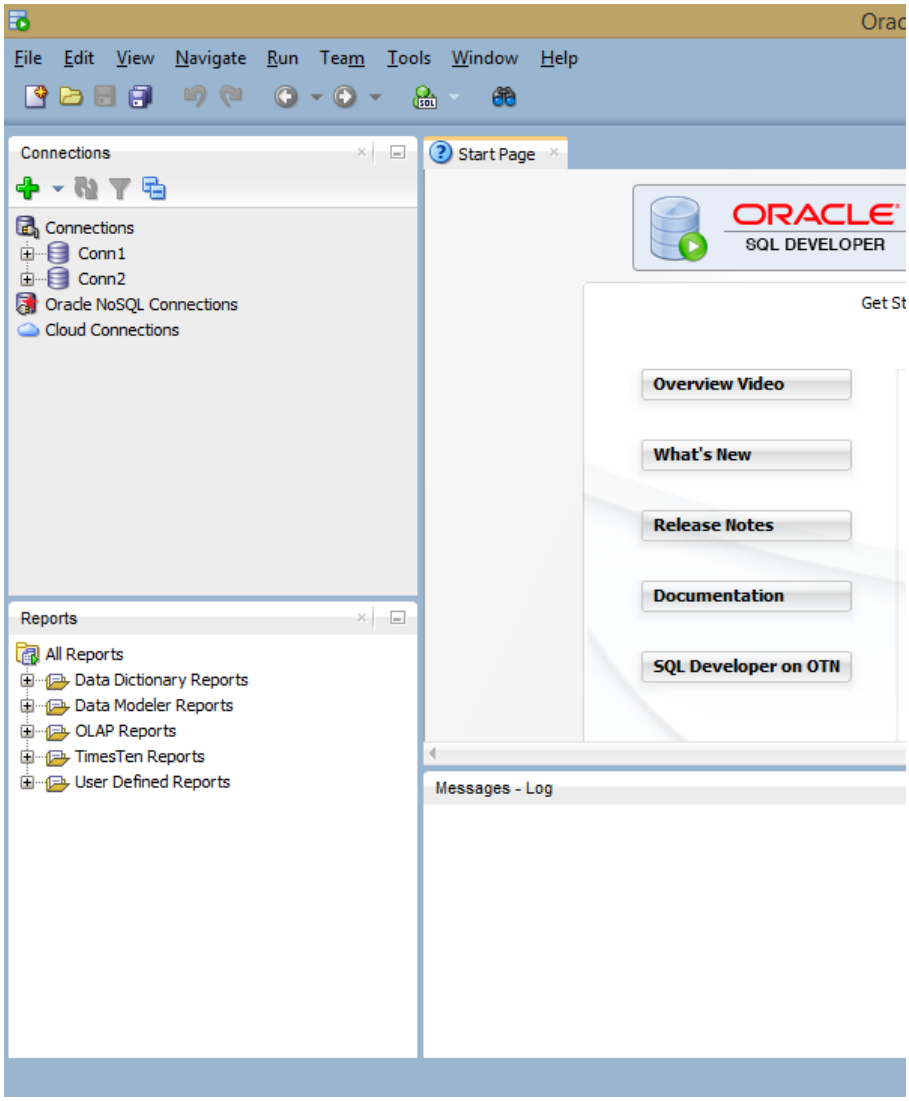


Figure 3.4: SQL Developer main window

### 3.7 The Java Database Connectivity (JDBC)

The Java Database Connectivity(“JDBC Overview,” 2016) 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. The JDBC API provides a call level API for SQL based database access.

JDBC technology allows to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data. With a JDBC technology enabled driver, developers can connect all corporate data even in a heterogeneous environment.

The JDBC API provides metadata access that enables the development of sophisticated applications that need to understand the underlying facilities and capabilities of a specific database connection.

JDBC technology exploits the advantages of Internet-standard URLs to identify database connections. The JDBC API includes an even better way to identify and connect to a data source, using a DataSource object that makes code even more portable and easier to maintain. DataSource objects can provide connection pooling and distributed transactions, essential for enterprise database computing. This functionality is provided transparently to the developer.

The JDBC API is available anywhere that the platform is. This means that applications can truly write database applications once and access data anywhere. The JDBC API is included in both, the Java Platform, Standard Edition and the Java Platform, Enterprise Edition, providing server side functionality for industrial strength scalability.

### 3.8 Summary

This chapter presented technology adopted to develop an automated solution for database migration. The Application has been fully developed with Java technologies such as NetBeans Platform APIs and Java Database Connectivity API. There are other technologies to support for the development such as MySQL Workbench, pgAdmin, SQL Server Management Studio and Oracle SQL Developer. Next chapter shows the approach that how we are going to apply these technologies to develop the solution.

# Chapter 4

# A Novel Approach to Database Migration

## 4.1 Introduction

Chapter 3 presented the technologies used for developing an advanced solution for migrating schema and data across multiple databases. This chapter begins with highlighting the features that distinguish our novel approach from the existing approaches of database migration. Next, this chapter presents the approach we have taken when developing a solution to migrate data across industry's most popular databases such as Oracle, SQL Server, MySQL and PostgreSQL under several headings namely input output, processes, users, and assumptions.

## 4.2 Features of the previous approach:

The previous approach was developed using NetBeans wizard architecture and it was user friendly. It provided several visual panels to the user for completing tasks such as selecting source and target databases, selecting tables to migrate, mapping data types, mapping table columns, and executing data migration. Before executing the migration activity, user had to decide whether to replace existing tables or skip migration of existing tables.

## 4.3 Features of the new approach:

Our solution is also developed using NetBeans wizard architecture. In addition to the features available in the previous approach, our solution includes the following features.

Identifying primary keys, foreign keys and other constraints of selected tables.

Handling the effects of primary key change of a table between two data migration sessions.

Incremental update of data without causing any loss of data.

Handling changes when the number of columns in a table gets increased / decreased.

Making required modifications when the constraints of table columns get changed.

Migrating parent tables of a selected table (based on relationships between the tables).

Migrating data based on different criteria (data selection via SQL queries).

Taking precautions to avoid creating orphan records when migrating tables.

Allowing user to rollback database to a pre-migration state, before committing changes (if required).

### 4.4 Inputs:

In each data migration session, user is provided with a wizard consist of multiple visual panels. Each of these visual panels takes different types of input from users. Input provided at one visual panel determines what should be presented at the next visual panel.

In the first visual panel, user is presented with source database systems (Eg: Oracle, MySQL) available. User can select one of them as the source database system. This is the first input accepted from user. Depending on the database system selected by the user, visual panel will load and display default values for the database host name, port number to connect, default username and password to connect to database. If the user wishes to change these values, they will be accepted as another input from user. After connecting to the database system s/he can select a particular database to read data from. This is another input. In the same manner, user can provide input for selecting a target database. Then s/he can select which tables to migrate. User can also specify whether to migrate only data, only structure or migrate both structure and data. This is another input to the system. In the next visual panel user can map columns of source tables with columns of target tables. This is another input to the system.

In the visual panel for selecting data migration criteria, user can select which records (rows) to migrate by running SQL query. This is another input to the system. User can provide more input by opening the "preferences" window from the main menu. (Eg: Drop existing tables, replace existing data, skip migrating existing records, etc).

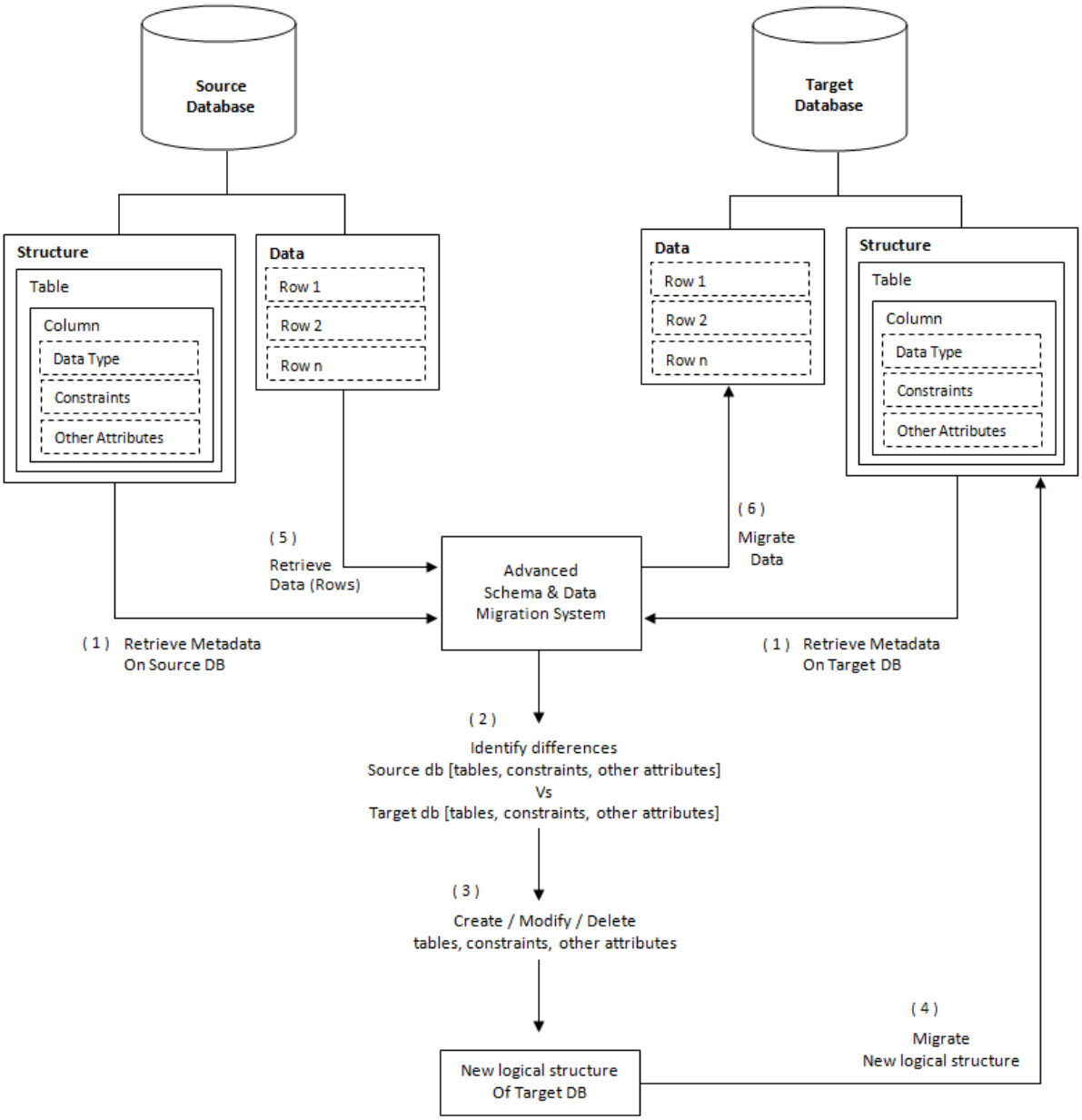


Figure 4.1: Methodology of Advanced Schema and Data Migration System

### 4.5 Outputs:

After a successful migration session, user will find that the selected tables have been migrated to target database, relationships between parent and child tables have been created, constraints of source table columns have been applied to the matching target table columns, and selected records (rows) have been migrated to tables at target database.

In the “review migration progress” visual panel, user is presented with a list of database savepoints created during each major activity. Therefore, if an activity is completed unsuccessfully (e.g: due to an unexpected error), then the user can rollback the target database to a stable state they were in before migration.

The system provides a detailed report at the end of each migration session regardless of how it ended (i.e. successful completion / unsuccessful completion)

* End of migration session:

User doesn't cancel it halfway through the migration session, and continues until the end.

### 4.6 Process:

In order to cater to various requirements of different users, it is essential to have separate interfaces. NetBeans supports creation of dynamic wizards with a number of wizard panels using Wizard classes of the NetBeans Dialogs API. Sequence of the wizard panels depends on the input provided by the user.

#### 4.6.1 Schema and Data Migration Process

User can select a Schema and Data Migration session from the main menu of the system. In this process, first the system connects to the source and target database systems using the values provided by the user (i.e. database host name, port number to connect, default username and password to connect to database). Then it retrieves metadata from source and target databases on tables which have been selected to migrate. It checks whether the selected source table structure is compatible with the mapped target table structure. System does this by comparing table names, column names, column data types, constraints (i.e. primary key, foreign key, unique, not null) on each column and other attributes (i.e. column value - auto increment). If a selected table has foreign keys, then the structures of its parent tables (in source and target databases) will be checked for incompatibilities.

If there are any incompatibilities in structures of source and target tables, it will be informed to the user and a visual panel will be presented to resolve issues. Using that visual panel, user can create a new logical structure for the target table without much effort.

Then the system checks whether the user wishes to migrate only a specific set of records (rows). If the user wishes to do so, s/he can run an SQL query on a selected table, using another visual panel and view the results quickly. If the user wishes to migrate all the records of a table, then s/he can skip this step. Then the user can go to the next visual panel and start migrating data.

First, the structures of all the selected tables will be migrated. Then the relationships between tables will be created. Then the constraints on columns will be applied.

After that, the records will be migrated. When migrating records user's preferences will be taken into consideration (Eg: Drop existing tables, replace existing data, etc).Information messages about each major action (i.e. creating database savepoints, creating tables, etc), will be displayed to the user during data migration. In the next visual panel: "review migration progress, user can choose to rollback the database to a particular savepoint and /or commit changes. The system provides a detailed report at the end of each migration session regardless of how it ended (i.e. successful completion / unsuccessful completion).

### 4.6.2 Incremental Data Updates: Column-wise

User can select an incremental data update session from the main menu of the system. User can update each column of existing rows (i.e. a row is a combination of one or more columns) by selecting column-wise update option. In order to do a column-wise update, target table and its parent tables must exist at target database. User must map source table and parent tables with its counterpart tables at the target database. Data types, constraints and other attributes of mapped columns should be identical (Column names can be different).

In this process; no new tables will be created. Columns will not be added or deleted. This process is only used for migrating data to existing columns.

### 4.6.3 Incremental Data Updates: Row-wise

If the user wishes to add new records to a table, s/he can do so by select an incremental data update session from the main menu of the system. In row-wise update; in addition to having identical data types, constraints and other attributes; total number of columns (not just the selected columns) in source and target tables should also be identical

In all three processes mentioned above, if migrating a particular value to a column is going to violate UNIQUE constraint, that record will not be migrated. When deleting a record from one table, relevant records will be deleted from its child tables where ON DELETE CASCADE is declared

### 4.7 Assumptions

This system relies on the following assumptions. Both; the source and target database are offline, during the data migration process. Data integrity (entity, referential, domain) is maintained in source table. A foreign key always references only the primary key of another table. There are no orphan records in source data (For each foreign key value in a child table, there's a matching primary key value in its parent table). Databases are not set to auto-commit changes

### 4.8 Users:

People choose to migrate databases due to a number of reasons. Growing amount of data, and growing number of end users are two main reasons to migrate to another database system. This can happen after a business merger. If the current database system can support only a limited in the number of concurrent users, then the DBAs will inform their supervisors that it’s time to migrate to an advanced DBMS. Needing more security measures than the current DBMS can provide, is another reason to migrate to another database system. Some companies may wish to migrate to a free DBMS due to budget cuts. People who have the above mentioned issues at their workplace and wish to migrate from/to Oracle, MS SQL Server, MySQL, PostgreSQL can become users of the Advanced Schema and Data Migration System.

### 4.9 Summary

This chapter presented our novel approach to develop an automated solution for database migration. it pointed out how the novel approach offers an efficient and accurate solution for database migration across multiple databases. The NetBeans Platform provides various APIs for creating dialogs and wizards mentioned earlier in this chapter. Next chapter shows the design of the novel approach presented here.

# Chapter 5

# Solution Design

## 5.1 Introduction

Chapter 4 presented the approach to develop an automated solution for migrating schema and data across multiple databases migration. This chapter elaborates the approach and describes the architecture of the solution. NetBeans Platform Wizard Architecture and Java Database Connectivity are the main foundation for this application. NetBeans Platform Wizard Architecture facilitates to design a wizard programmatically, that takes user through the migration process in a clearly defined step by step approach and the user interface provided by this wizard is much more appealing and is easy to use. This is a complete Database Migration Tool that helps for migrating and transferring database schemas and data across leading databases such as Oracle, Microsoft SQL Server, PostgreSQL, and MySQL using Java Database Connectivity (JDBC).

## 5.2 Interaction Between Objects

In each data migration session, user is provided with a wizard consist of multiple visual panels. The main visual panels include selecting source database, selecting target database, selecting tables, mapping columns of source and target tables, resolving issues, managing data migration criteria, executing updates, reviewing migration progress.

The following sections describe how the objects interact with each other in each scenario.

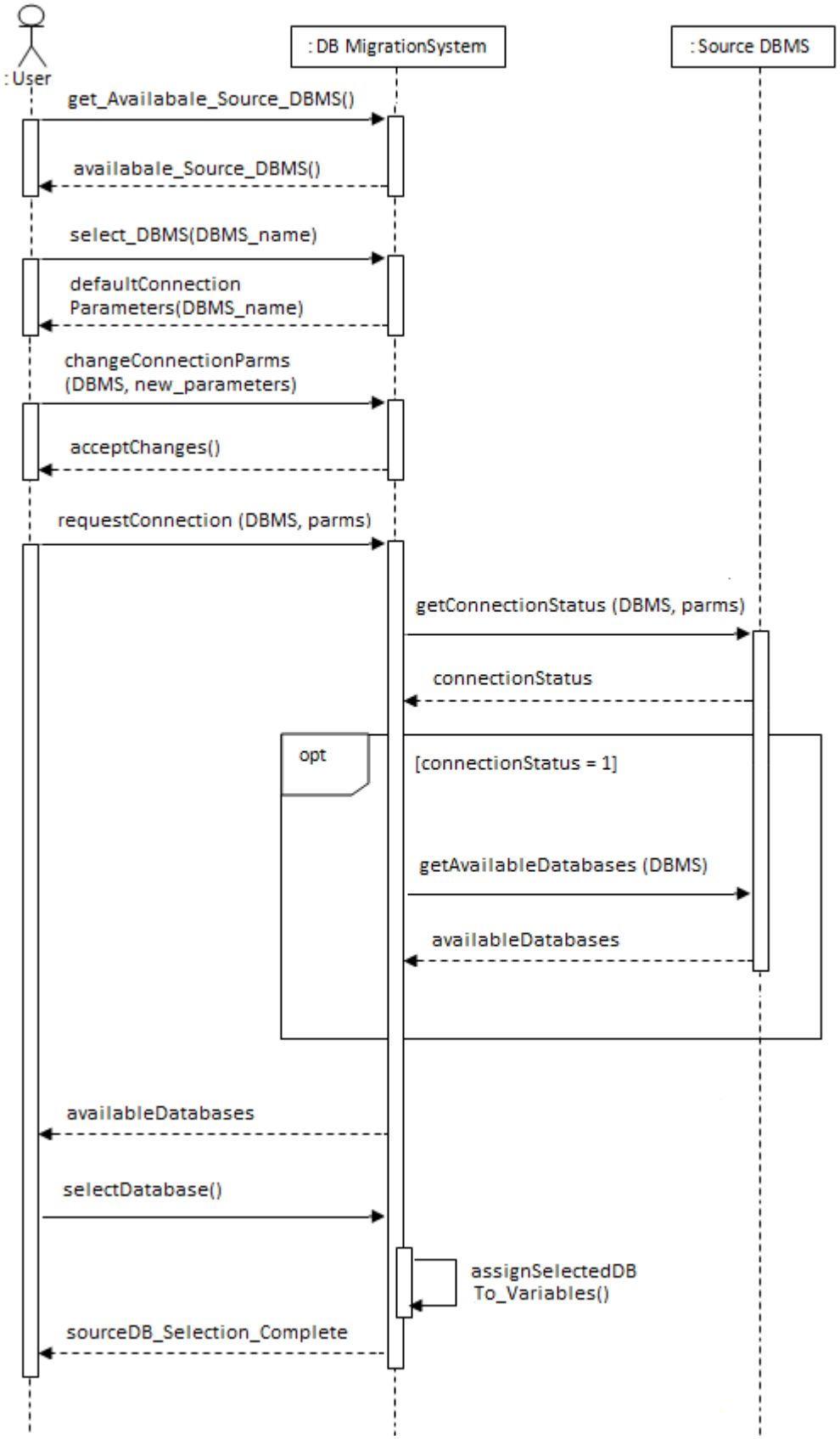


Figure 5.1: Sequence Diagram - Selecting source database

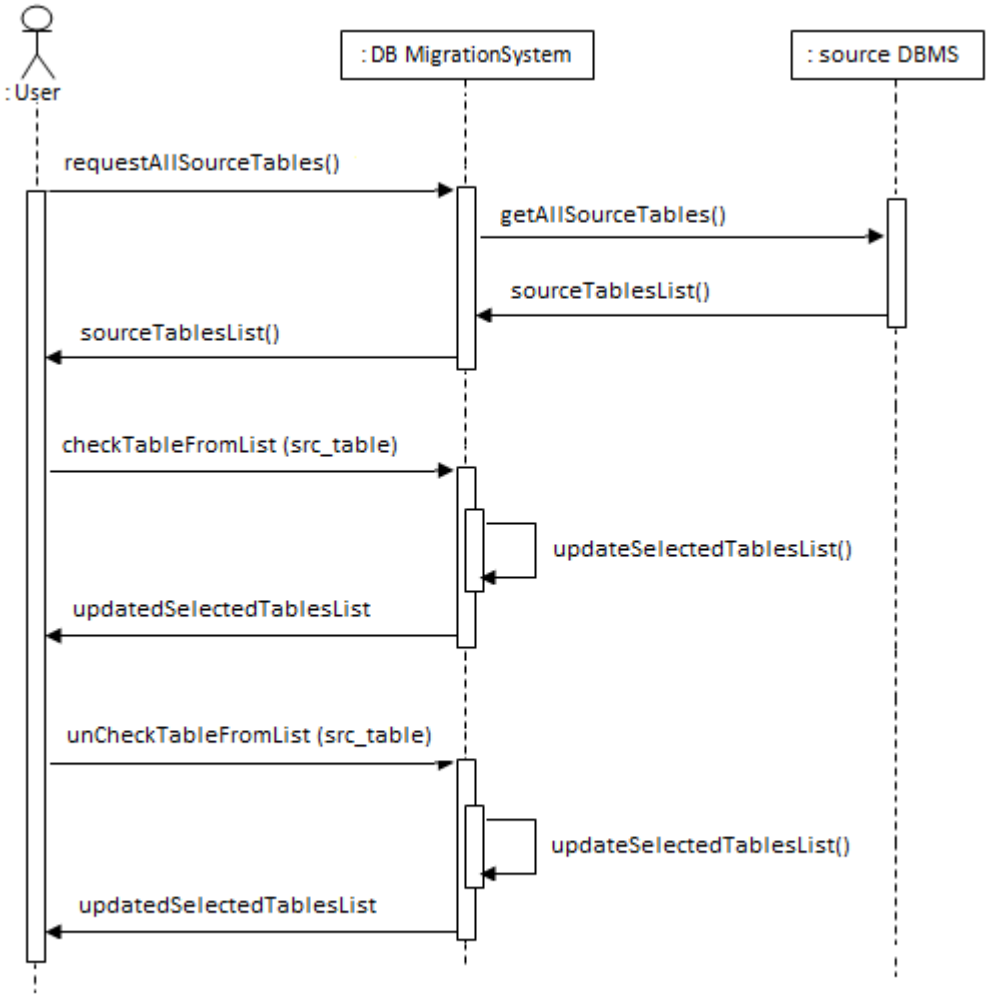


Figure 5.2: Sequence Diagram - Selecting tables

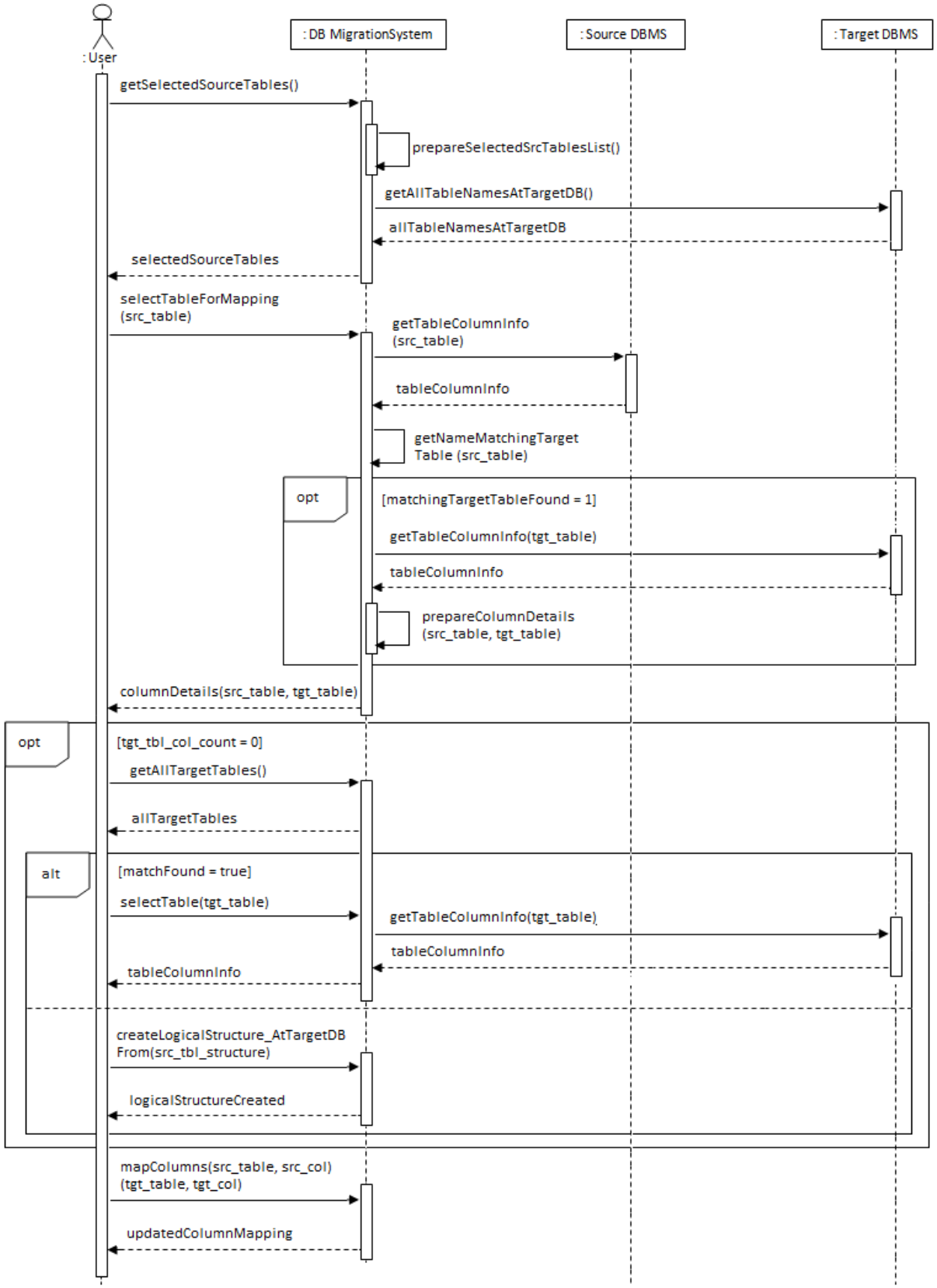


Figure 5.3: Sequence Diagram - Mapping columns of source and target tables

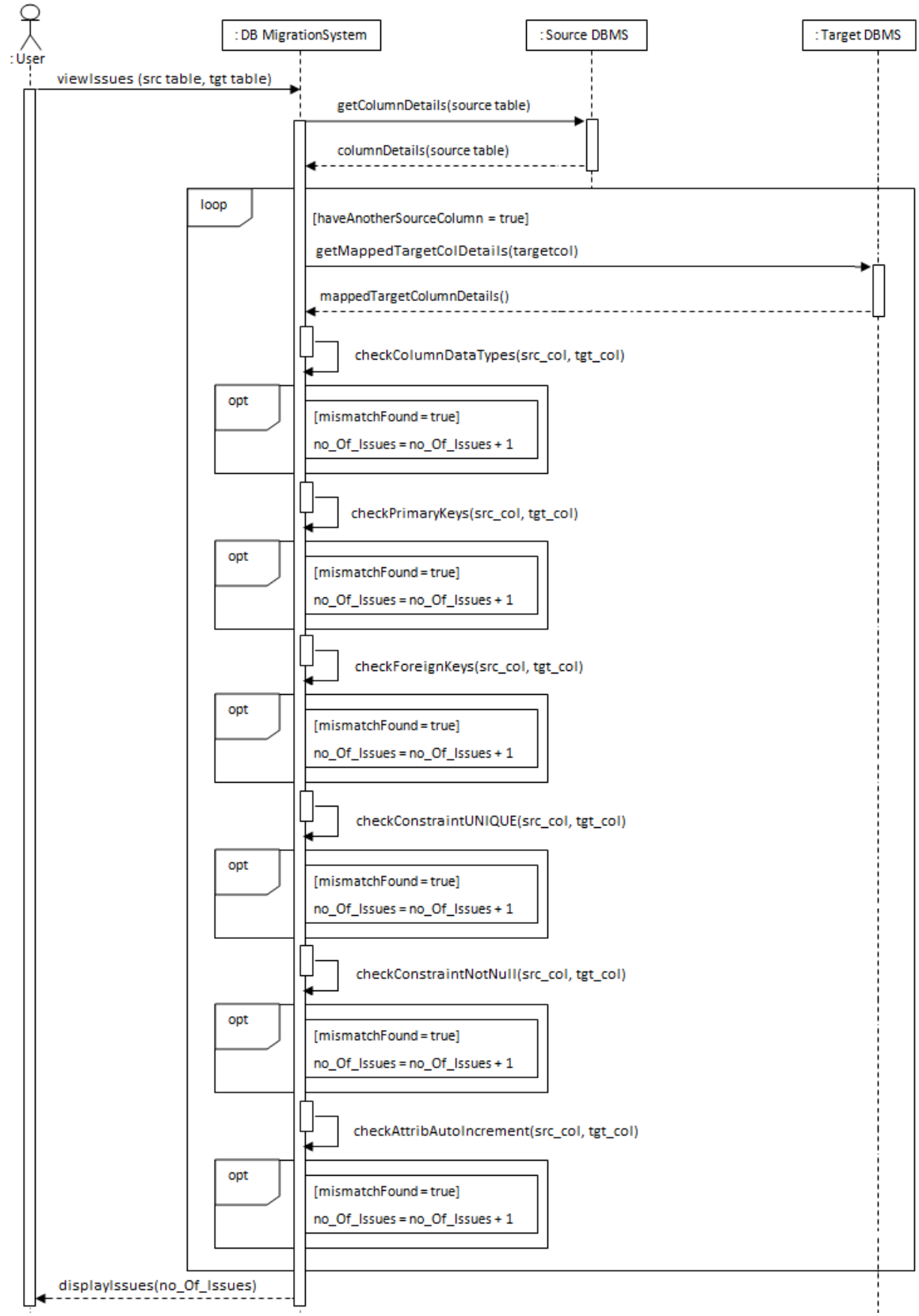


Figure 5.4: Sequence Diagram - Resolving issues

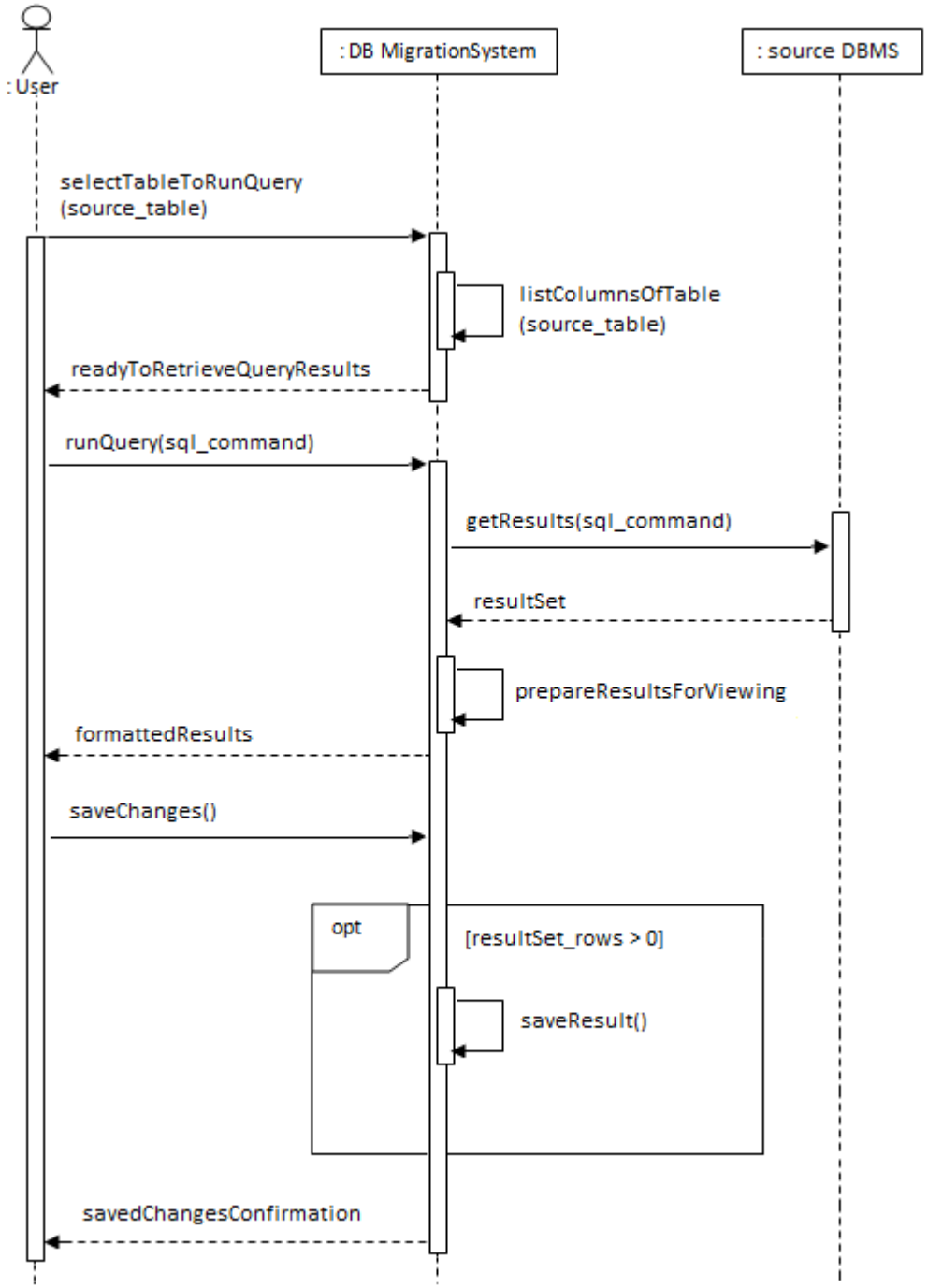


Figure 5.5: Sequence Diagram - Managing data migration criteria

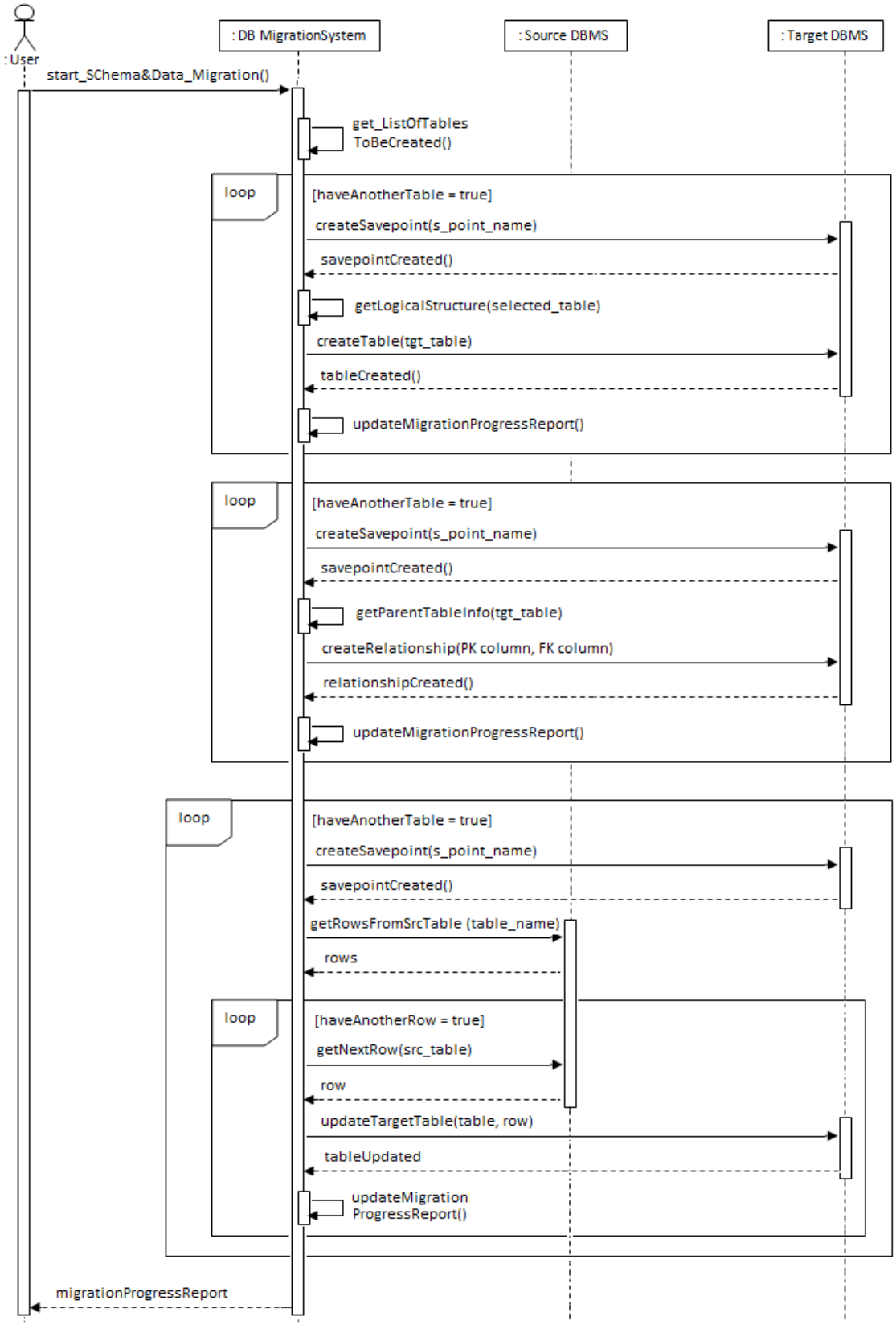


Figure 5.6: Sequence Diagram - Executing updates

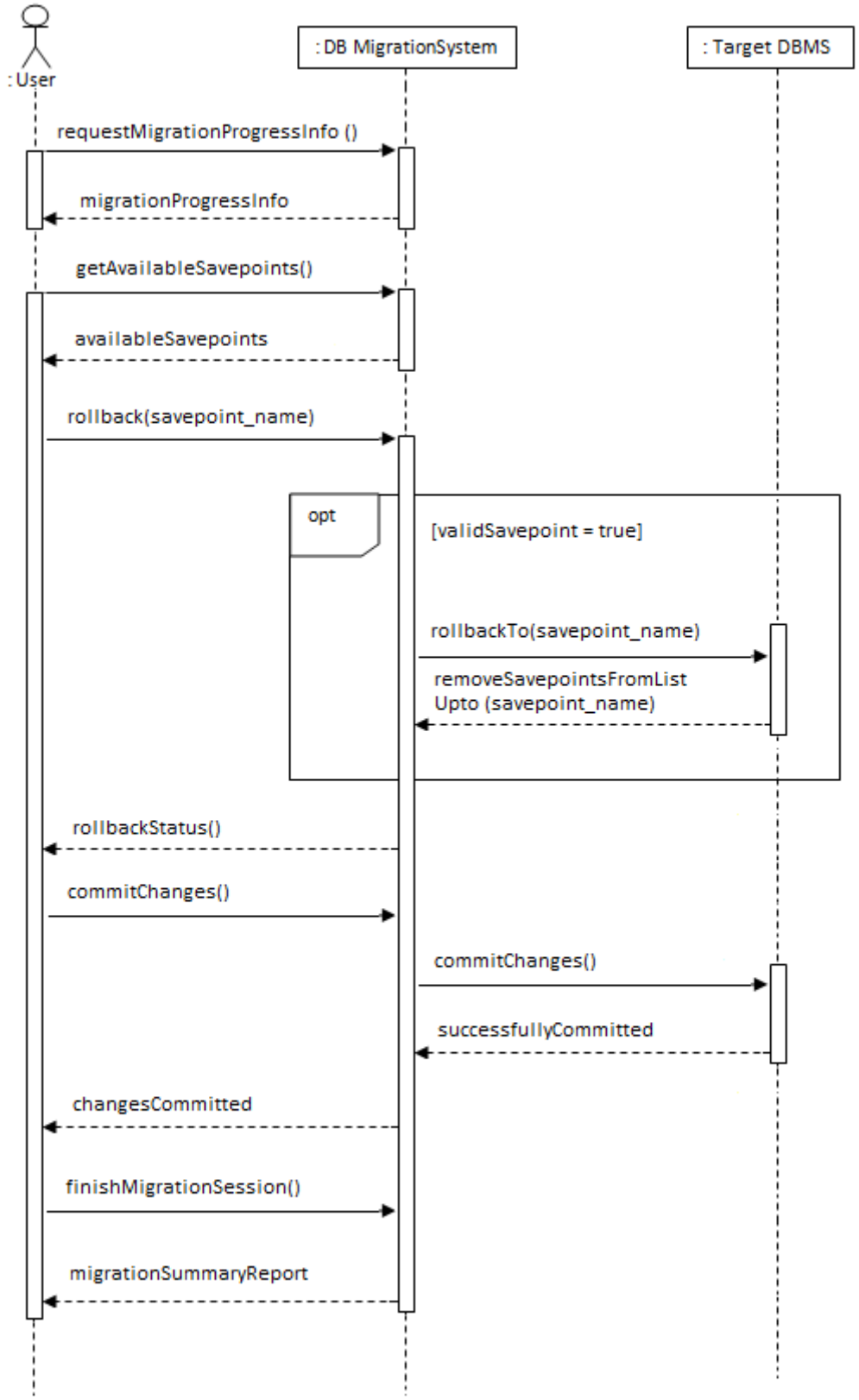


Figure 5.7: Sequence Diagram - Reviewing migration progress

### 5.3 Database Connectivity Architecture

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to multiple databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple databases. Figure 5.8 is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the application.

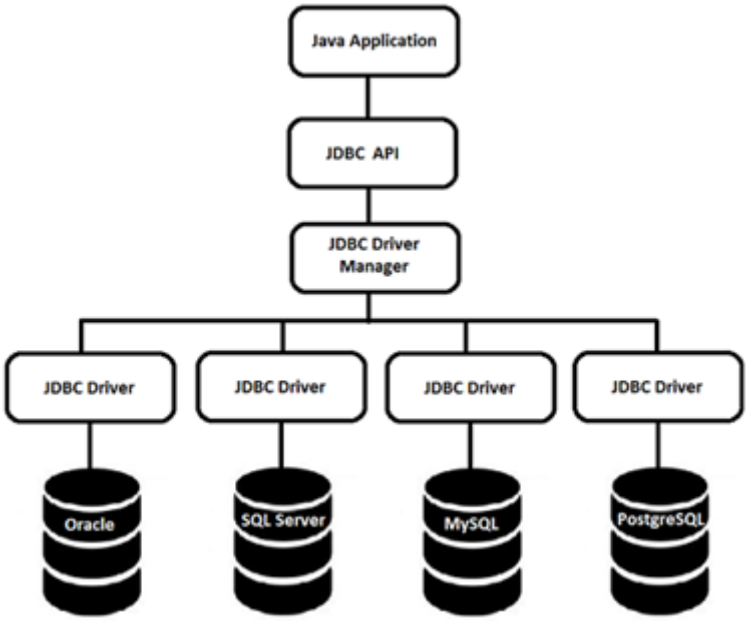


Figure 5.8: Database connectivity architecture

In the solution, we have mostly used the following interfaces and classes in JDBC API:

DriverManager: This class manages a list of database drivers. Matches connection requests from the application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.

Driver: This interface handles the communications with the database server. We will interact directly with Driver objects very rarely. Instead of that, we use DriverManager objects, which manage objects of this type. It also abstracts the details associated with working with Driver objects.

Connection: This interface with all methods for connecting a database. The connection object represents communication context which means all communication with database is through connection object only.

Statement: We use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

ResultSet: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow us to move through its data.

SQLException: This class handles any errors that occur in a database application.

### 5.4 Summary

This chapter presented design architecture of the automated solution for database migration. It showed architecture of the solution which is going to be implemented in the development stage. This design offers a well-defined and user-friendly solution for database migration across multiple databases. NetBeans Platform provides a professional APIs for the frontend and Java Database Connectivity defines interfaces and classes to communicate with the backend databases. Next chapter shows the implantation of the solution.

# Chapter 6

# Implementation of the solution

### 6.1 Introduction

Top level design is based on two major components, the first one is NetBeans platform wizard architecture for frontend and the second one is Java Database Connectivity architecture for backend. This chapter describes the implementation of the automated solution for database migration. In that sense this chapter is about how the system is implemented. The solution is platform independent, which means it can run on Windows, Linux, Mac OS and etc. It has been developed on top of Java 8 platform. Let's discuss the implementation of the automated solution for schema and data migration across multiple databases.

In order to obtain information which are required to transfer data from source database to target database, we are using mainly two interfaces.

### 6.2 Interface DatabaseMetaData

This interface has methods including [18]

getDatabaseProductVersion() - *Retrieves the version number of this database product.*

getDriverName() - *Retrieves the name of this JDBC driver.*

getExportedKeys(String catalog, String schema, String table)

*Retrieves a description of the foreign key columns that reference the given table's primary key columns.*

getPrimaryKeys(String catalog, String schema, String table)

*Retrieves a description of the given table's primary key columns.*

getSchemas() - *Retrieves the schema names available in this database.*

### 6.3 Interface ResultSetMetaData

This interface has methods including [19]

getColumnName(int column) - *Get the designated column's name.*

getColumnCount() - *Returns the number of columns in this ResultSet object.*

getColumnType(int column) - *Retrieves the designated column's SQL type.*

getTableName(int column) - *Gets the designated column's table name.*

getSchemaName(int column) - *Get the designated column's table's schema.*

Through these interfaces, we can obtain metadata about the database which the Java application is connected to. For instance, you can see database product name and version, database driver version, list of tables are defined in the database and details of columns of each table, whether specific features are supported etc.

### 6.4 Implementation

The following images display how the solution is implemented according to the design discussed in the previous chapter

User can map each column data type in source database system with column data types used in the target database system. User can do this by selecting the “Data type mapping” from the main menu.

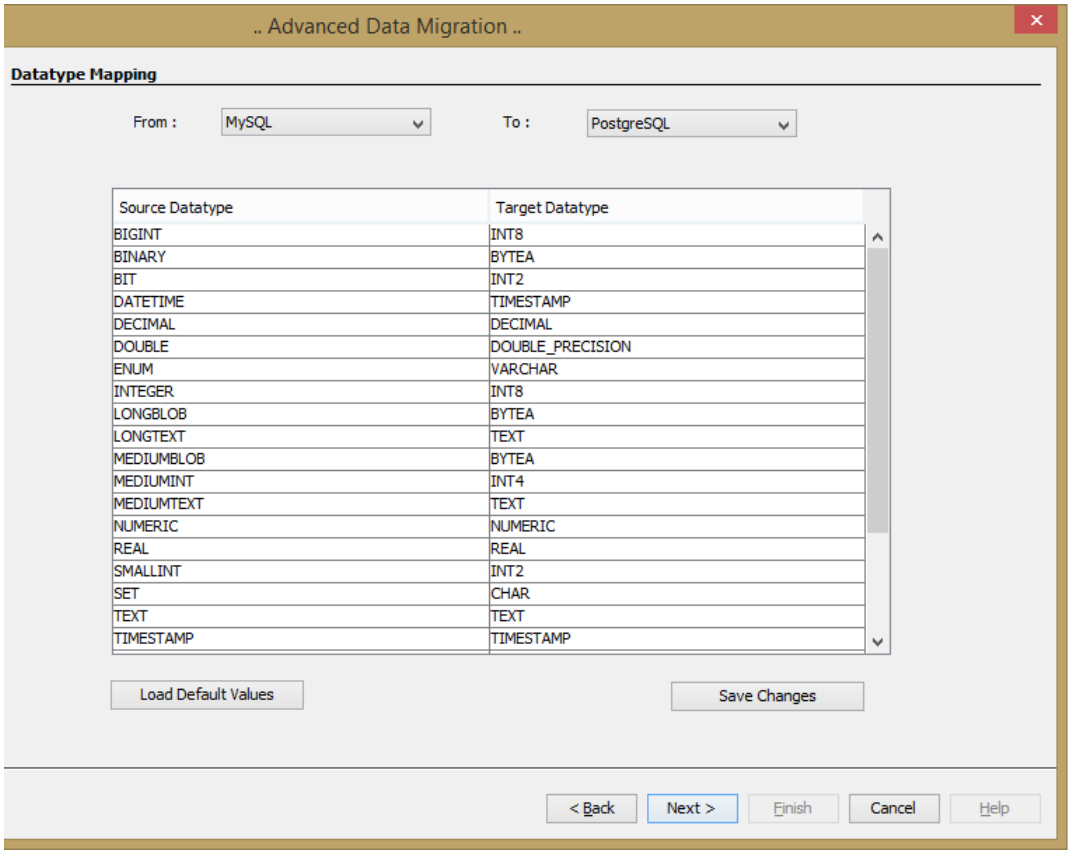


Figure 6.1: Data type mapping

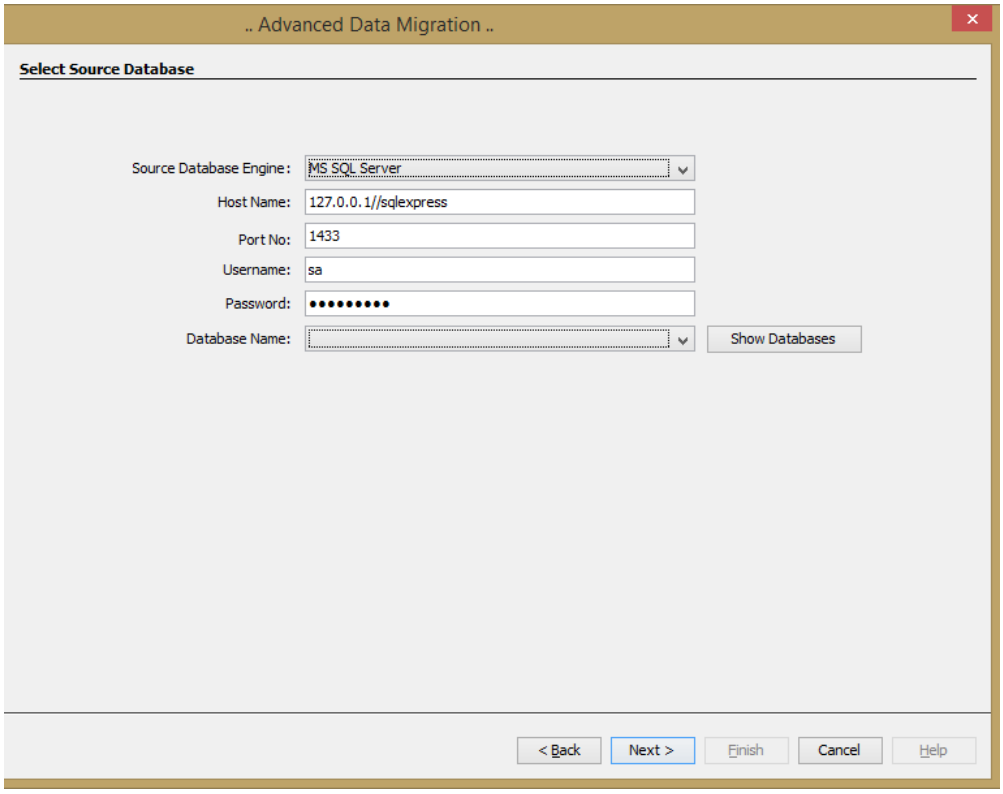


Figure 6.2: Selecting source database system

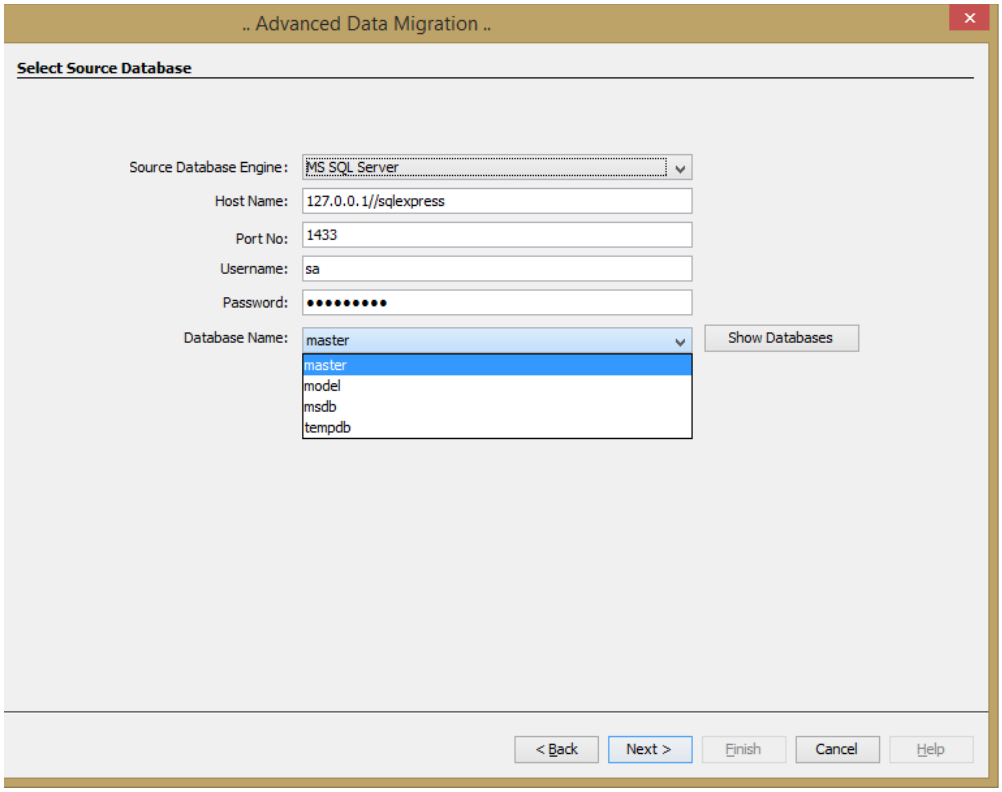


Figure 6.3: Selecting a particular database in source database system

User can view the all the source databases by clicking the “Show Databases” button.

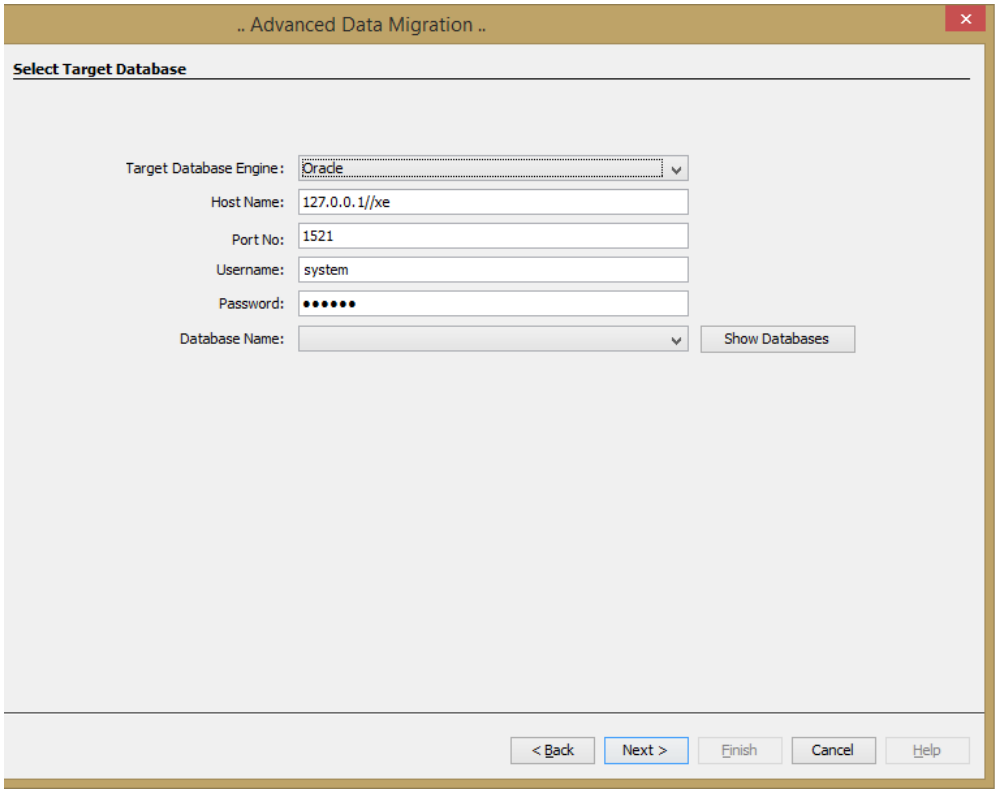


Figure 6.4: Selecting target database system

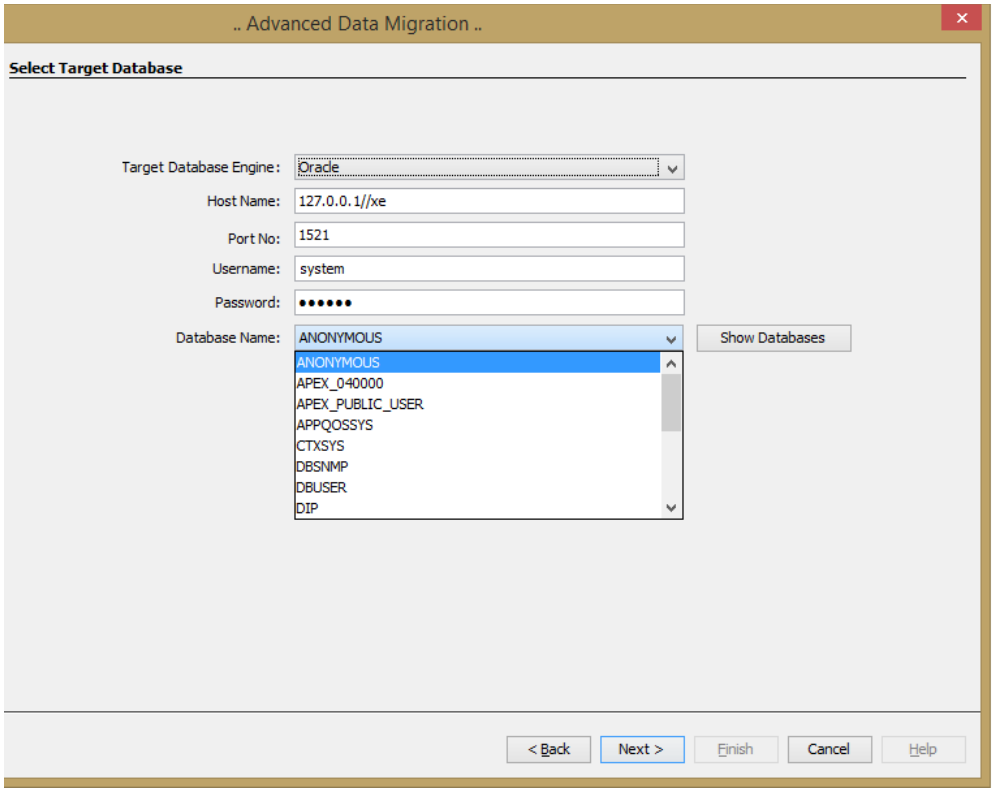


Figure 6.5: Selecting a particular database in target database system

User can view the all the target databases by clicking the “Show Databases” button.

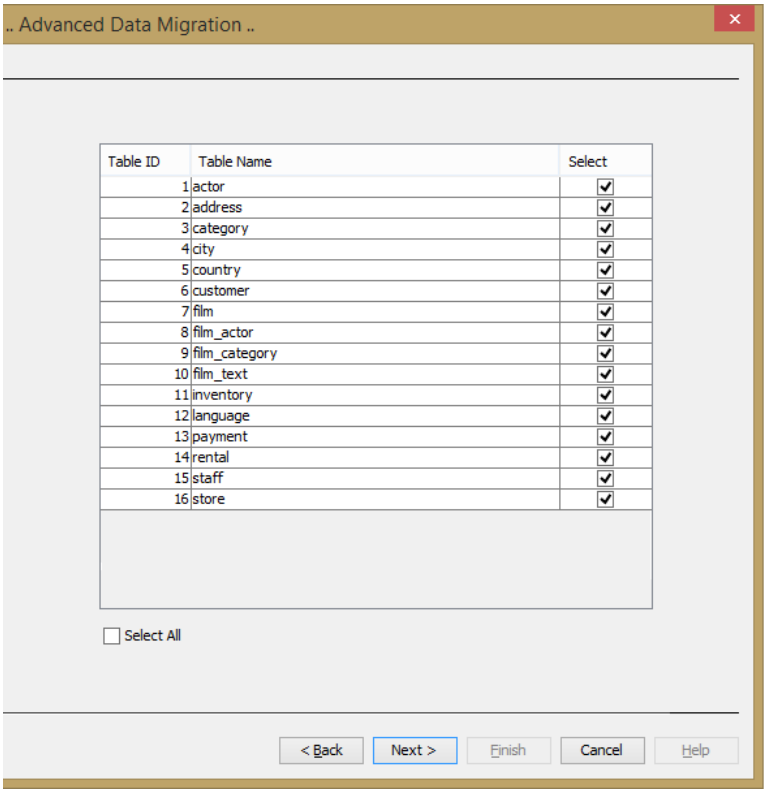


Figure 6.6: Selecting tables

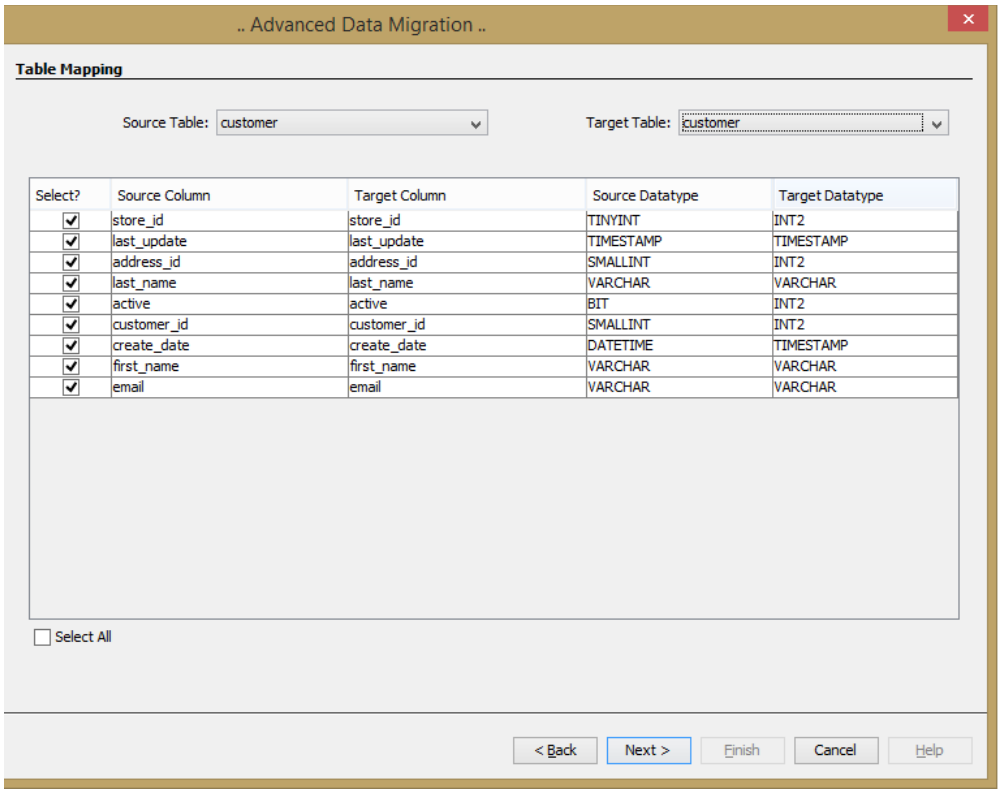


Figure 6.7: Mapping columns of source and target tables

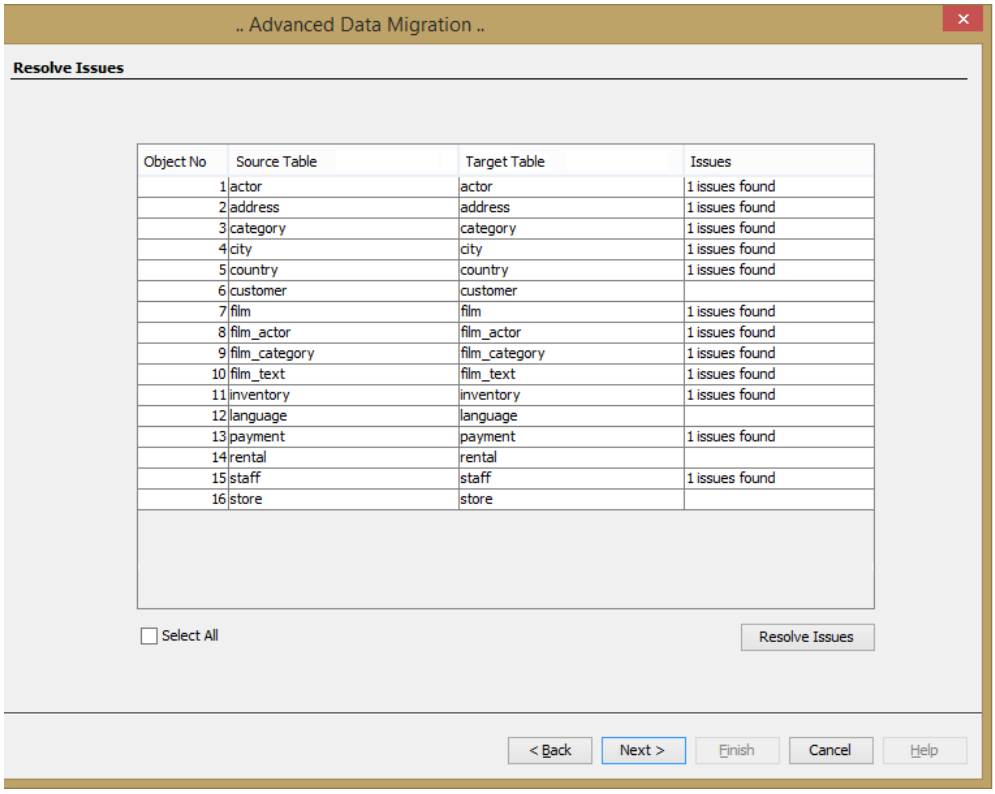


Figure 6.8: Resolving issues : Part 1

Figure 6.9: Resolving issues : Part 2

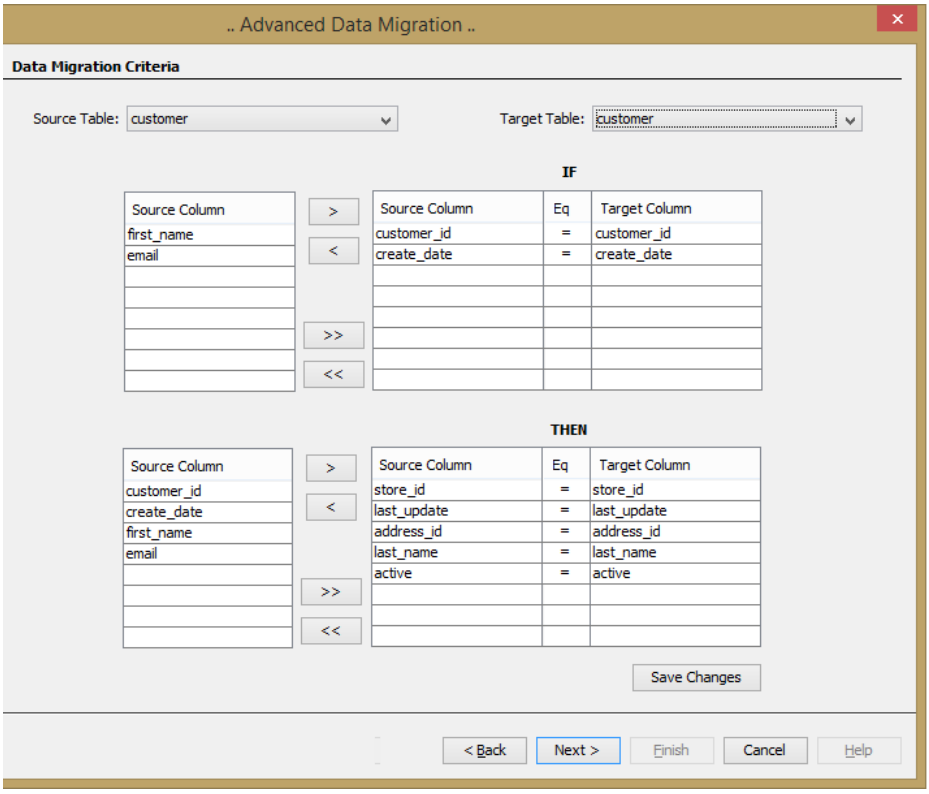


Figure 6.10: Managing data migration criteria : Part 1

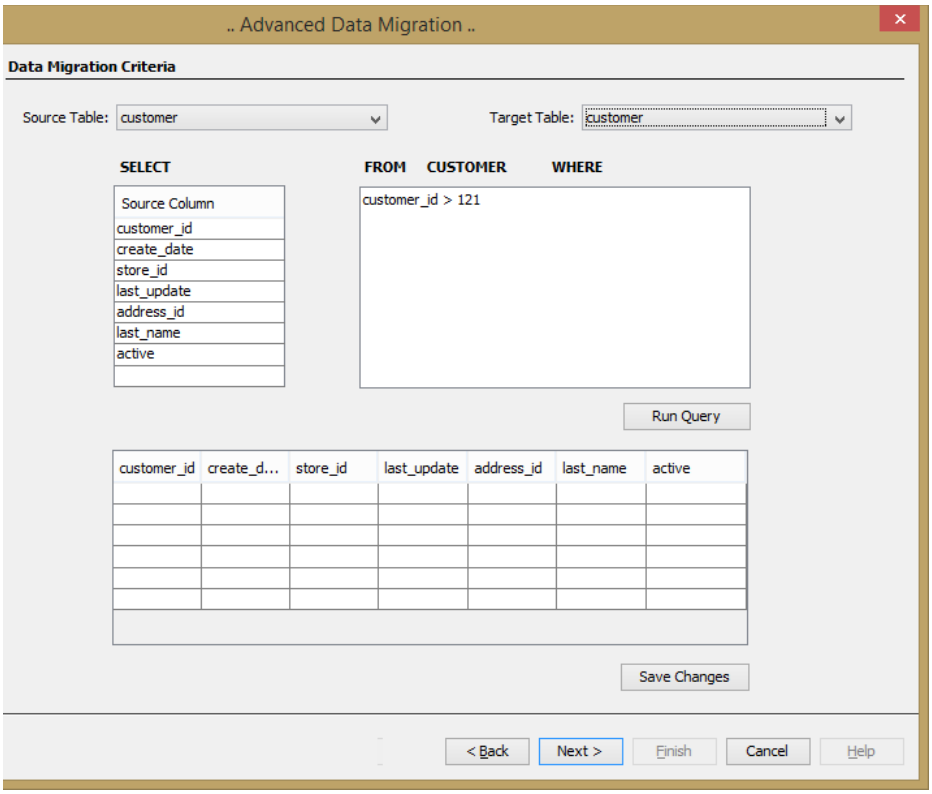


Figure 6.11: Managing data migration criteria : Part 2

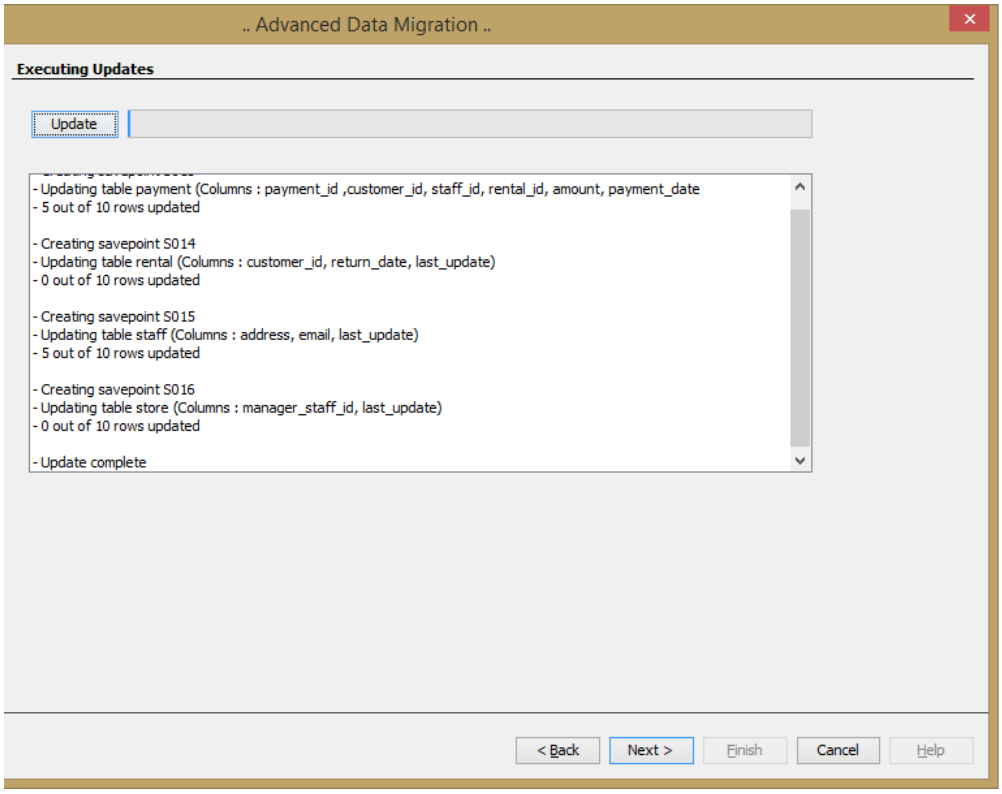


Figure 6.12: Executing updates

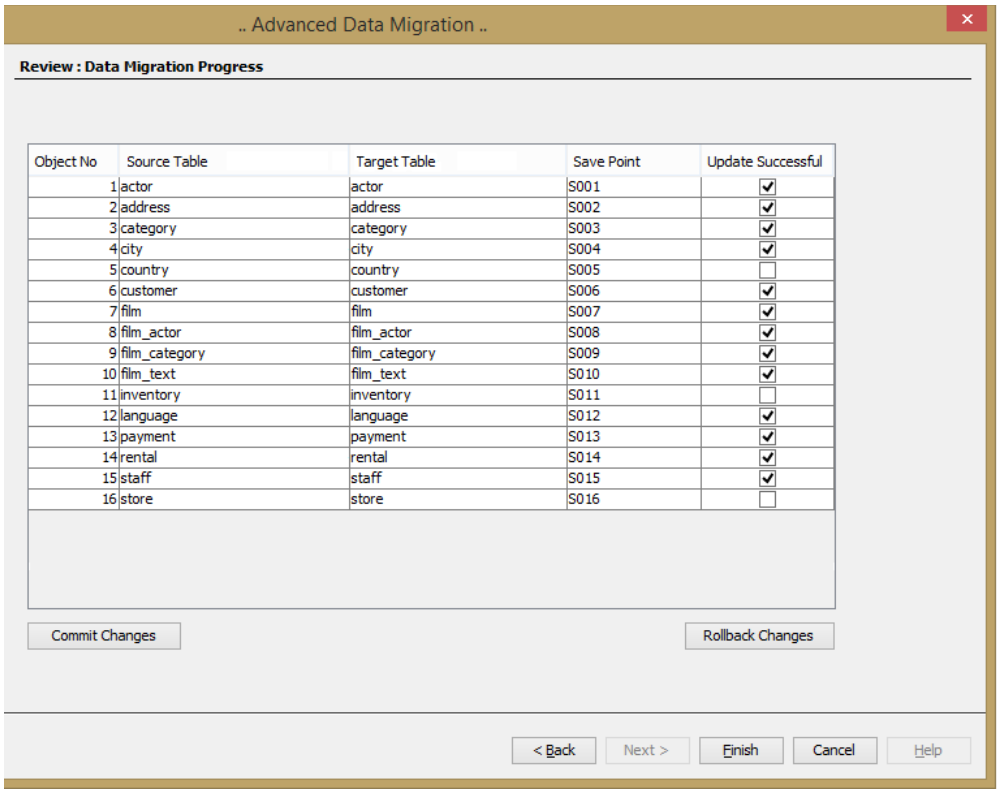


Figure 6.13: Reviewing migration progress

### 6.5 Summary

This chapter fully describes the implementation details of database migration across multiple databases. The solution offers a complete database migration that helps the migration of database schema and data across leading databases such as Oracle, MS SQL Server, PostgreSQL, and MySQL. It offers an open, user-friendly, and extensible migration process ensuring reliability and data integrity. The solution is a big step forward, as it simplifies the process of migration, with much better user interface, compared to its other products. Next chapter explains the evaluation of the product that we have already developed.

# Chapter 7

# Product Evaluation

## 7.1 Introduction

This chapter begins by highlighting the advantages of the features included in our new approach. Then it presents a comparison of features available in the old approach and the features available in the new approach. Next, this chapter describes the validation methods we have used, and how the system is handling issues which can rise during program execution. In addition, this chapter describes possible scenarios and the actions taken by the system in each situation.

## 7.2 Advantages of the features in our new approach

**Selecting parent tables of selected tables (if required).**

If the selected table has a column with a foreign key, then its parent table will also be selected for migration. Therefore, when migrating records, the system can check whether the referenced key value is available in the parent table. This is essential to maintain the integrity of records in target tables.

**Detect and display issues of migrating the selected tables.**

This feature helps the user to decide which actions to take in order to resolve issues. Each issue is given a number and description by the system. The system also provides instructions on resolving issues based on the issue number.

**Alter tables / change constraints and attributes of columns**

In the previous system, when migrating records to an existing table, user had only two options.

1. Drop the existing target table, create it as a new table and then migrate all records from the source table. If the user selected this option, then the old system will remove all the existing records from the target table.
2. Skip migrating records to that table. If the user selected this option, then the old system will not migrate new records to the target table.

But the new system is capable of altering the structure of an existing table (if required), and then migrate new records, without deleting the existing records.

**Identify and display primary keys, foreign keys and other constraints and attributes of selected tables.**

User can view the structures of source and target tables on a single visual panel. S/he can compare source table structure with new target table structure. S/he can also compare current target table structure with new target table structure. This feature highlights the differences in structures and helps the user to make the required changes.

**Handling changes made to a primary key/foreign keys of a table.**

In order to change the primary key/foreign keys of a table, user has to just check/uncheck the relevant checkbox. Then the system validates the selected action and applies the required changes to appropriate tables.

**Handling changes made to a constraint/attribute of a column.**

In order to change a constraint/attribute of a column (eg: Unique, Not null, auto-increment), user has to just check/uncheck the relevant checkbox. Then the system validates the selected action and applies the required changes to appropriate columns.

**Incremental update of data.**

This feature allows users to migrate new records to a table, without deleting the existing records.

**Migrate data based on different criteria (data selection via SQL queries).**

This feature allows users to filter records on a particular condition and view results. Then the user can migrate only the records retrieved as the result.

**Taking precautions to avoid creating orphan records.**

This feature includes running a series of validations during data migration. This is essential to maintain the integrity of records in target tables.

**Creating savepoints / Rollback the target database.**

The system creates a savepoint before executing each the following actions.

Creating initial structure of a table (with primary key), Applying foreign keys, Migrating records.

When the migration process is complete, the system displays the list of savepoints created during the migration process. If the user wishes to rollback the target database, s/he can do so by selecting a savepoint from that list.

**Commit changes made to the target database.**

Auto-commit is disabled, so that the user can commit changes only after carrying out all the sub tasks (eg: migrating records, rollback actions).

### 7.3 Comparison of features available in old approach and new approach

| Feature | Available in Old approach? | Available in New approach? |
|---|---|---|
| Guiding the user step by step using a wizard | Yes | Yes |
| Selecting source and target tables | Yes | Yes |
| Selecting tables for migration | Yes | Yes |
| Selecting parent tables of selected tables (if required) | | Yes |
| Detect and display issues of migrating the selected tables | | Yes |
| Map columns of selected tables | Yes | Yes |
| Map source and target data types | Yes | |
| Create, drop tables | Yes | Yes |
| Alter tables / change constraints, attributes of columns | | Yes |
| Identify and display primary keys, foreign keys and other attributes of selected tables | | Yes |
| Handling changes made to a primary key / foreign keys | | Yes |
| Handling changes made to an attribute of a column | | Yes |
| Incremental update of data | | Yes |
| Migrate data based on different criteria (via SQL queries) | | Yes |
| Taking precautions to avoid creating orphan records | | Yes |
| Display progress of migration tasks | Yes | Yes |
| Creating savepoints / Rollback the target database | | Yes |
| Commit changes made to the target database | | Yes |

Table 7.1: Comparison of features available in old approach and new approach

### 7.4 Actions taken by the system in various difficult scenarios

#### 7.4.1 Changing a column name in target table (which currently has records in it)

- Check whether the changed column is a FK (references columns from its parent tables)

  If so, take one of the following 2 actions

  - Rename column. Include the "REFERENCES" phrase in ALTER statement
  - Do not change column name

- Check whether the changed column is a PK, and it's not referred by child tables

  - Rename column. Include the "PRIMARY KEY" phrase in ALTER statement

- Check whether the changed column is a PK, and it's referred by child tables

  If so, take one of the following 2 actions

  - Rename column. Change the "REFERENCES" phrase of those child tables
  - Do not change column name

- Check whether there's another column with the same name in that table

  If so,

  - Delete the old column (column_1) with the same name.
    Then change the other column (column_2) name
  - Rename the old column (column_1) with a different name.
    Then change the other column (column_2) name
  - Select a different name for that column (column_2)
  - Do not change column (column_2) name

#### 7.4.2 Changing a column data type in target table (which currently has records in it)

- Check whether the changed column is a FK (references column in its parent table)

  If so, check whether the new data type is compatible with column data type in parent table

  If compatible - Change column data type of target table

  If not compatible - Do not change column data type

- Check whether the changed column is a PK, and it's not referred by child tables

  - Change column data type. Include the "PRIMARY KEY" phrase in ALTER statement

- Check whether the changed column is a PK, and it's referred by child tables

  If so, check whether the new data type is compatible with column data type in child tables

If Compatible;

- Change column data type of target table. Change column data type of child tables

If not compatible

- Do not change column data type

- Check the previous data type and records for compatibility with new data type

If they are not compatible,

- Inform about it to the user and ask him/her how to proceed
- Clear the whole column

(Only if that column can have NULL values, and not unique)

- Clear the whole column

Renumber them starting with 1 / *2001 if that column data type is numeric

- Clear the whole column

Fill them with a particular value if that column data type is char/date

- Do not change the column data type

### 7.4.3 Adding a new column to a target table (which currently has records in it)

- Fill the new field of each old record with a particular value
- Fill the new field of each old record with an incrementing value (if it's numeric)
- Leave the field value blank/empty
- Check constraints: Add/change PK, FK, UNIQUE, NOT NULL, CHECK, Auto inc

### 7.4.4 Removing a column to a target table (which currently has records in it)

- Check whether those columns are referred by child tables
- Check constraints: Removing/changing PK, FK
- Remove the relationships associated with that column, before removing the column

### 7.4.5 Adding a foreign key to column in target table (which currently has records in it)

- Check whether all the required parent tables are available

(currently available at target or listed in the migrating objects list)

- Check whether all the required fields are available in those tables

(column names, data types match with the required columns in parent / child tables)

- Check whether the referencing column is the PK of parent table

### 7.4.6 Removing a foreign key from a target table (which currently has records in it)

Inform the user that migrating the new structure is going to remove a particular relationship, and get his/her confirmation on whether to proceed with that action.

- If it's a single column foreign key, then remove the foreign key
- If it's a composite FK, and the user wants to remove all the fields of the composite key
    - Then inform that to the user. Remove the foreign key
- If it's a composite FK, and the user wants to remove only one field from the composite key,
    - Then inform that to the user. Do not remove the foreign key

### 7.4.7 Adding a primary key constraint to a column in target table (which currently has records in it)

PKs are UNIQUE and NOT NULL.

- Check whether that column has duplicates or null values

    If there are any duplicates, (If it's a composite PK with 3 columns, then consider values of those 3 columns combined, when searching for duplicates)

    Inform about it to the user and ask him/her how to proceed

    - Remove the duplicate records
    - Remove only duplicate values and renumber them starting with *1001 / *2001 (from the first duplicate record) if that column data type is numeric
    - Clear the whole column and renumber them starting with 1 / *2001
    - Do not add PK to that column

### 7.4.8 Removing a primary key constraint from a column in target table (which currently has records in it)

- Check whether that PK is referred by child tables.

    PKs are UNIQUE and NOT NULL.

    Removing PK constraint means that column can have duplicates and null values.

    Then the child tables which reference that column could get incorrect values.

    Therefore, inform about it to the user and ask him/her how to proceed

    - Remove FKs (from child tables) which refer to this PK column first.

      Then remove the PK constraint.
    - Do not remove PK constraint

### 7.4.9 Changing a primary key (Removing old PK and Adding new PK)

- When a composite PK becomes a single column PK, data duplication could occur

    Eg: Composite PK: car_manufacturer, car_model

        (Toyota-Corolla, Toyota -Prius, Toyota-Camry)

    Single column PK : car_manufacturer (Toyota, Toyota, Toyota)

- When changing a single column PK to another single column PK,

    but the new PK column is not available in the table at target

- When changing a single column PK to a composite PK,

    but one of the columns is not available in the table at target

Other Constraints: UNIQUE, NOT NULL

* 1001: When the biggest value in that field is 1000, then start the next value with 1001
* 2001: A number decided by user

## 7.5 Summary

This chapter described the logic behind the valuation methods implemented in the system. Testing is running a system in order to identify any gaps, errors, bugs or missing requirements with respect to the actual requirements. We have tested the system for various possible errors. All the tests are done manually. Next chapter explains the conclusion and the further works of the product.

# Chapter 8

# Conclusion

## 8.1 Introduction

This chapter illustrates the results which are generated from the solution and the further improvements can be done to the solution. As an example, by using this solution, we can simply migrate schema and data of a MySQL database to PostgreSQL or MS SQL Server or Oracle databases with the integrity constrains. The integrity constraints are mainly primary keys and foreign keys. The generated outputs are shown in the results section of this chapter. List of additional new features has been identified and these new features are listed in the further work section in this chapter.

## 8.2 Results

Figure 8.1 shows "EMP" table in Oracle "DBUSER" database which has 14 tables with data. EMP table has 8 columns. Primary keys and the two foreign keys of the table are also created successfully.

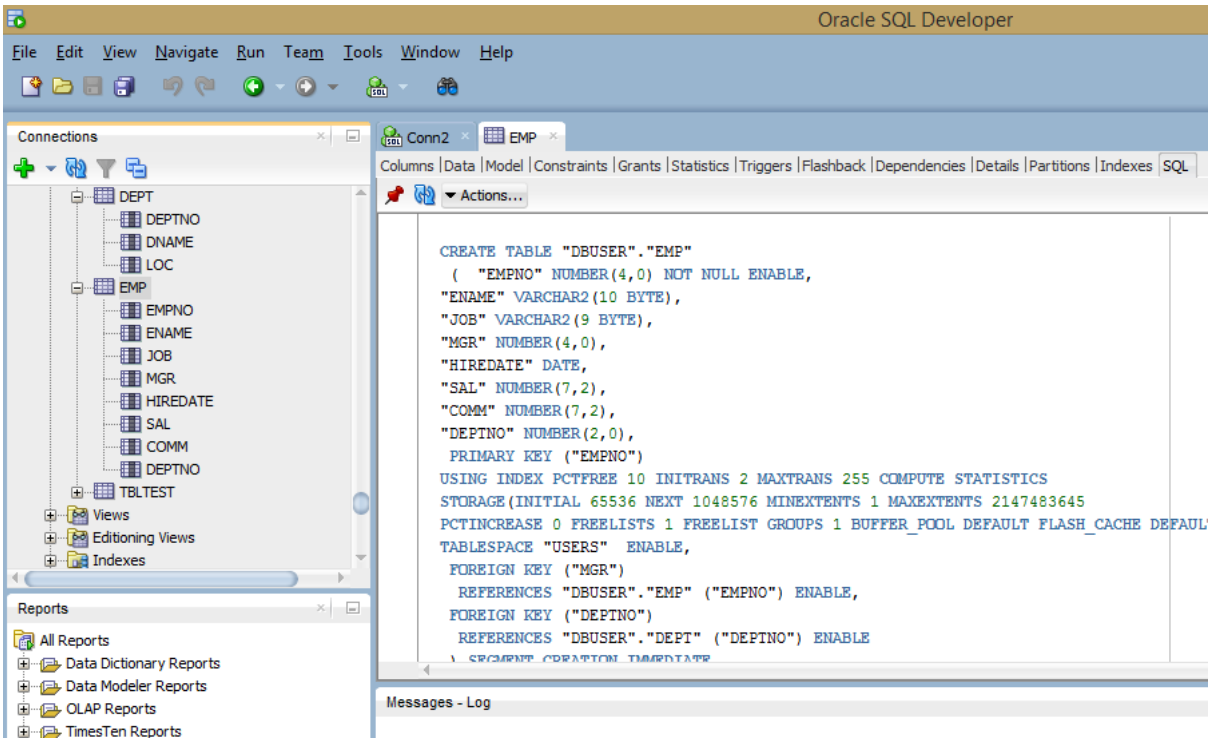


Figure 8.1: Results: Oracle Database

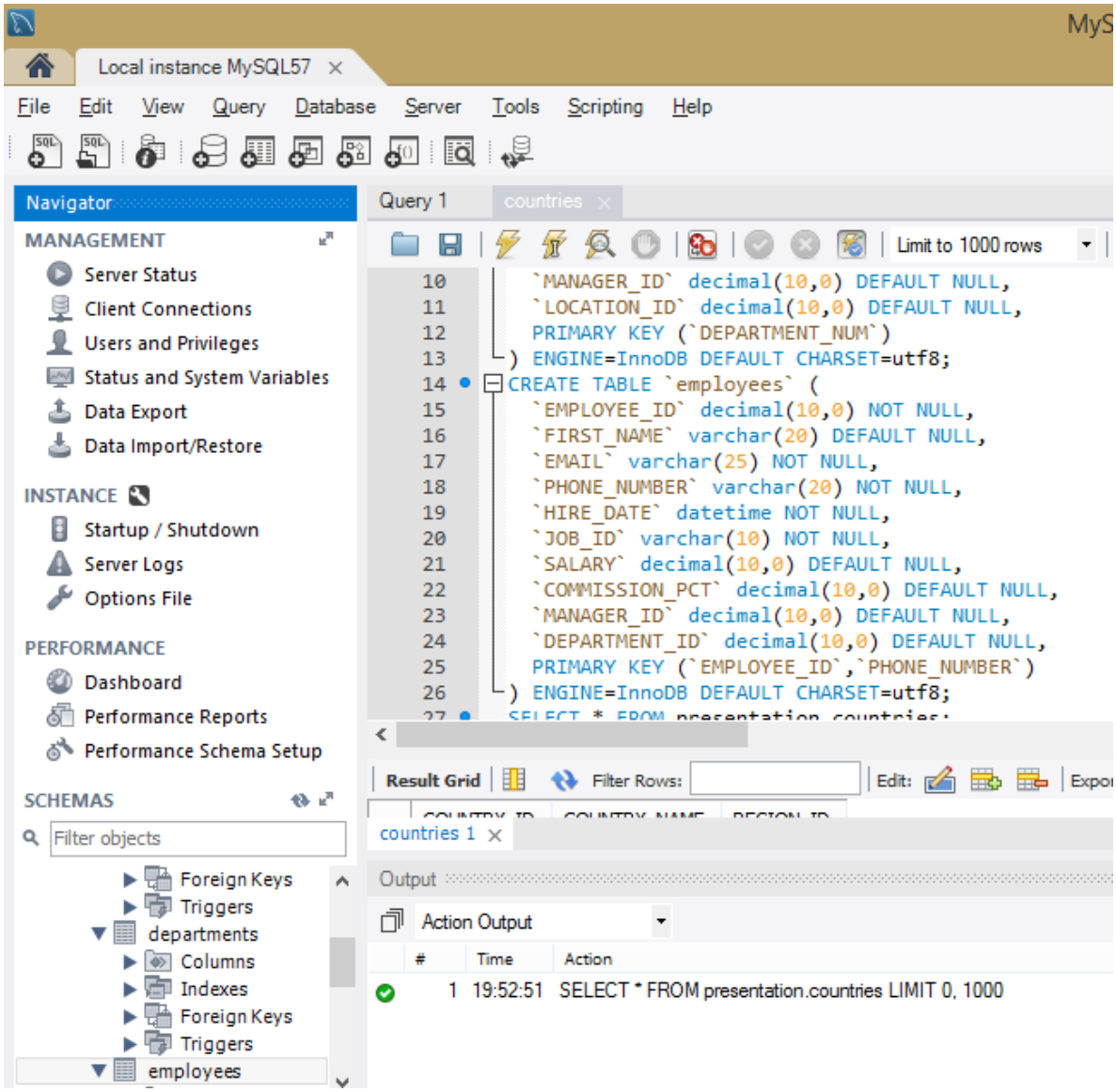


Figure 8.2: Results: MySQL database

Figure 8.2 shows “employees” table in MySQL “presentation” database which has 14 tables with data. Employees table has 10 columns. Its primary key is a combination of two columns. Results show that composite primary key and all the other columns of the table are created successfully.

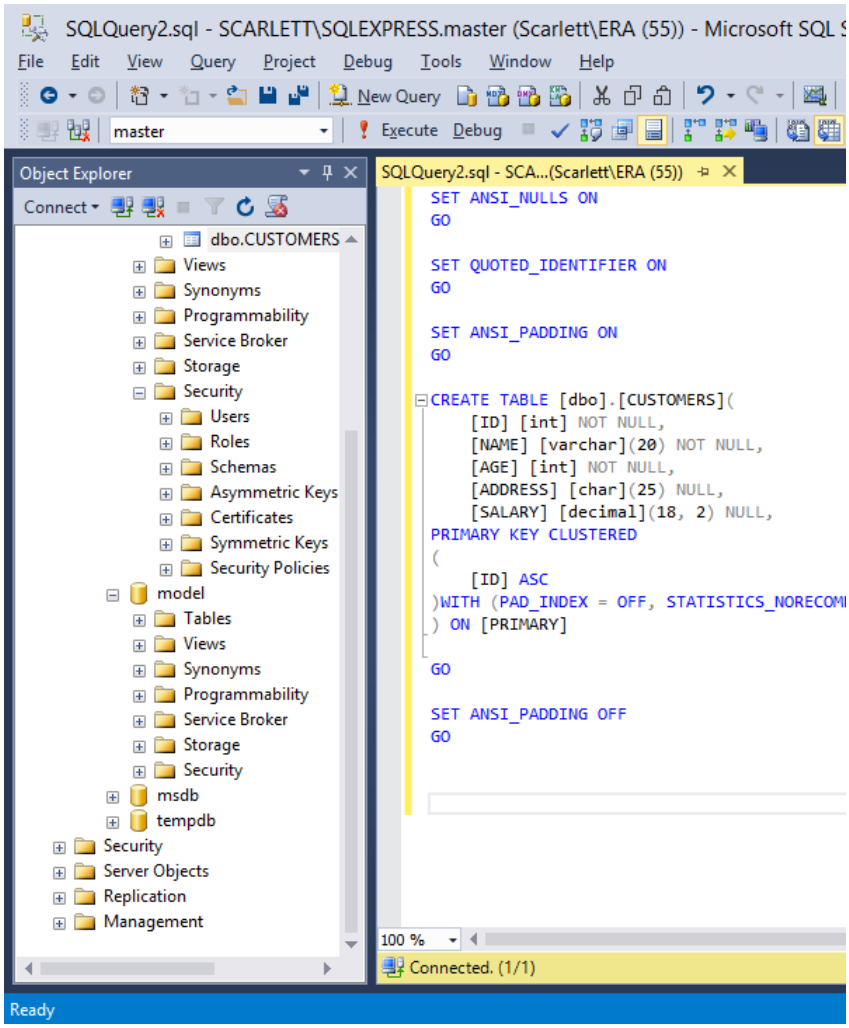


Figure 8.3: Results: MS SQL Server database

Figure 8.3 shows “CUSTOMERS” table in MS SQL Server “master” database which has 14 tables with data. EMP table has 5 columns. Results show that primary key, columns with Not Null constraint, and all the other columns of the table are created successfully.

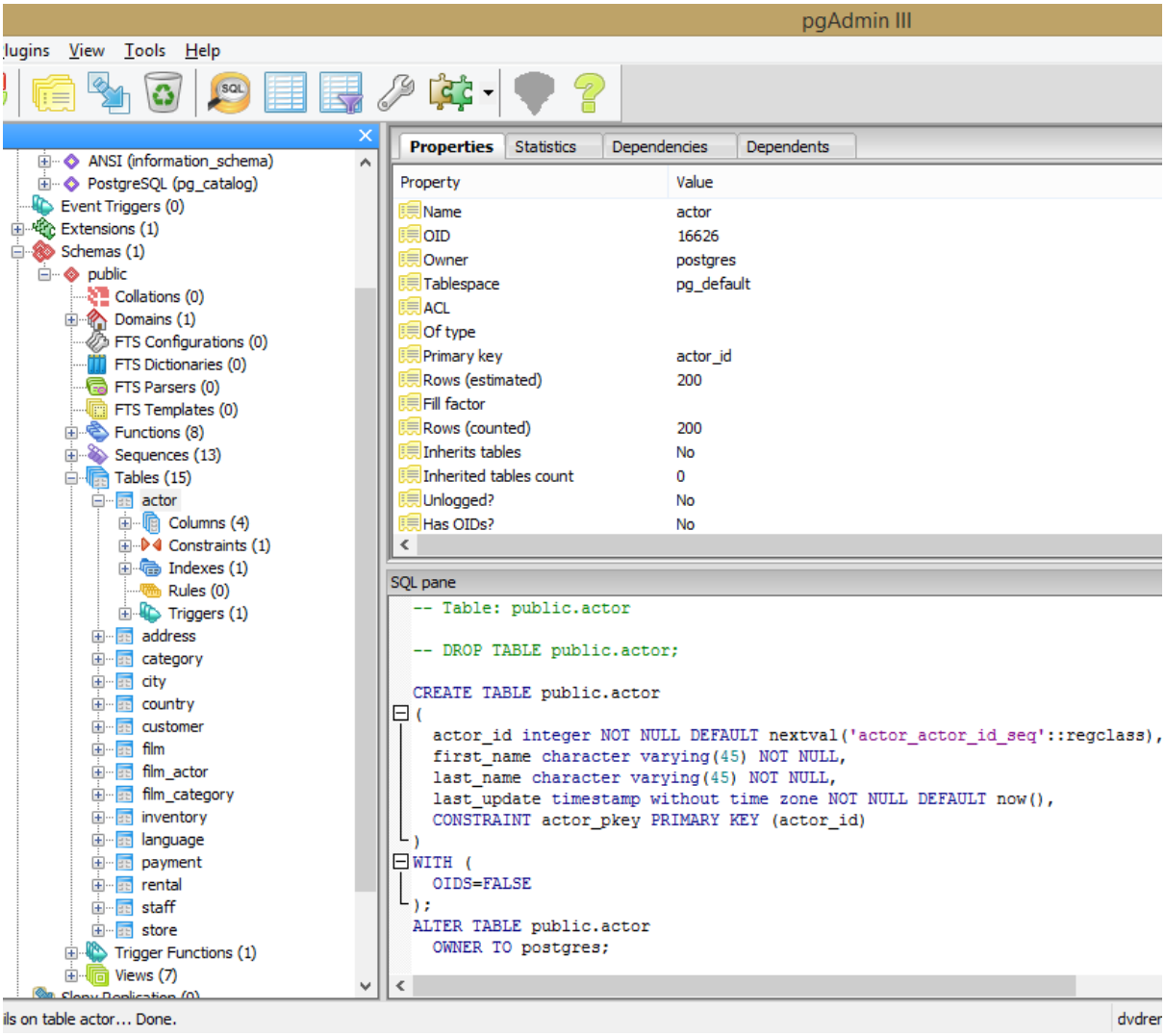


Figure 8.4: Results: PostgreSQL database

Figure 8.4 shows “actor” table in PostgreSQL “dvdrental” database which has 15 tables with data. EMP table has 4 columns. Results show that primary key, columns with Not Null constraint, and all the other columns of the table are created successfully.

### 8.3 Further Work

Further works of this application will focus on improving quality of source data before data migration. This includes checking source data for entity integrity, referential integrity, and domain integrity. Orphan records, missing data, inconsistent and redundant data will also be removed from source tables, before data migration. Values in columns which have UNIQUE, NOT NULL, DEFAULT constraints, will be checked to make sure those values are not violating the constraints.

Further works will also include; migrating triggers, stored procedures as well as migrating constraints such as DEFAULT and CHECK.

Extending the work to facilitate migration of data from multiple source databases into one target database simultaneously, will also be considered in future.

### 8.4 Summary

This chapter described the results and future work of the data migration solution. The solution offers a complete Database Migration Tool that supports the migration of database schema and data across industry leading databases such as Oracle, MS SQL Server, MySQL, and PostgreSQL. It offers a free, user friendly, and extensible migration process ensuring reliability and data integrity. The solution offers a suite of automated database migration tools, which enables migration of complex database schema and enterprise data from one database to another. It automates up to 90% of the manual tasks of database migration. There are more new features that we have identified and these identified features will be implemented as future work.

# Chapter 9

# References


[1] J. Morris, *Practical data migration*. The British Computer Society, Chippenham, 2009, ch.1.

[2] B. Walek and C. Klimes, *A methodology for Data Migration between Different Database Management Systems*, International Journal of Computer and Information Engineering, p. 6, 2012.

[3] B. Wei and T. X. Chen, *Criteria for Evaluating General Database Migration Tools*, October 2012. [Online]. Available: http://dx.doi.org/10.3768/rtipress.2012.op.0009.1210. [Accessed 25 Oct 2016].

[4] L. He, Z. Zhang, Y. Tan and M. Liao, *An Efficient Data Cleaning Algorithm Based on Attributes Selection*, 2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), pp. 375 - 379, December 2011.

[5] B. Wei, T. X. Chen, *Criteria for Evaluating General Database Migration Tools*, International Journal of Computer and Information Engineering, p. 6, 2012.

[6] JDOM (2011) (Online) http://www.jdom.org. Accessed 7 Nov 2011

[7] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, P. Domingos, *Discovering complex semantic matches between database schema*s. In Proc. of the ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD), 2004.

[8] H. H. Do. *Schema Matching and Mapping-based Data Integration*, Verlag Dr. M¨uller (VDM), 2006. ISBN 3-86550-997-5.

[9] H. H. Do, E. Rahm, *COMA - a system for flexible combination of schema matching approaches*. In Proc. 28th Intl. Conf. on Very Large Data Bases (VLDB), 2002.

[10] L. Haas, M. Hernandez, H. Ho, L. Popa, and M. Roth. Clio grows up: From research prototype to industrial tool. In Proc. ACM SIGMOD Intl. Conf. Management of Data, pages 805–810, 2005.

[11] L. Xu, D. Embley, *Discovering direct and indirect matches for schema elements*. In Proc. 8. Intl. Conf. Database Systems for Advanced Applications (DASFAA), 2003.

[12] I. M. Wijewardana, *Migrate and Transfer Schema and Data across Multiple Databases*, [Online].Available: http://dl.lib.mrt.ac.lk/browse?value=Wijewardana%2C+I.M.&type=author

[13] https://www.froglogic.com/squish/

[14] https://netbeans.org/features/platform/features.html

[15] https://www.mysql.com/products/workbench/

[16] https://www.pgadmin.org/features.php

[17] https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms

[18] https://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html

[19] https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSetMetaData.html