JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

Data Migration between Heterogeneous Relational Databases – Oracle, MySQL, PostgreSQL and Microsoft SQL Server

M. Elamparithi,
Associate Professor, Kamalam College of Arts and Science, Bharathiar University, Tamilnadu, India.
E-Mail: profelamparithi@gmail.com

ABSTRACT
All of a source database application's components are changed into their equivalents in a target database environment throughout the migration of database applications procedure. In order to do this, the source schema must be converted into the target one, the source data must be transformed into the target database format, application programmes must be migrated to the new, non-relational DBMS, and queries and update operations must be translated into their equivalents in the target platform. However, software engineering is required to convert application programmes and queries. Consequently, it is believed that database migration will involve both data conversion and schema translation.

1. INTRODUCTION

By putting the techniques described in Chapter 3 into practise, a prototype for data migration between relational databases is created. The algorithms were created using the C++ programming language, the Code Blocks Integrated Development Environment (IDE), and the SQLAPI++ middleware on a computer running Windows XP Professional Service Pack 3 and equipped with a Pentium IV 3.2 GHz CPU and 2048 MB of RAM. The input RDBs to be migrated are stored in an RDBMS, such as Oracle 11g, MySQL 5.5, PostgreSQL 9.2, and Microsoft SQL Server 2012, which has been connected via the SQLAPI.lib. The most used programming languages nowadays include Java, C#, C++, and Visual Studio. Each has its own unique qualities. Despite the fact that alternative programming languages might be used with more work, C++ and the Code:: Blocks IDE were selected for the creation of the migration process. This is due to Code:: Blocks, an open-source IDE that allows software reuse and is platform neutral. It enables effective algorithm encoding and is cross-platform compatible with Linux, Mac, and Windows operating systems. Additionally, it supports multiple compilers and does not include any proprietary libraries or interpreted languages. Additionally, SQLAPI++ middleware libraries offer complete access to a specific RDB's metadata and, using its metadata APIs, may easily get a description of the database's tables and constraints in the form of 'ResultSet' objects from data dictionaries.

2 CONTEMPORARY RELATIONAL DATABASES AND MIDDLEWARE

The properties of information storage and retrieval are described by a structured collection of data called a relational database. In other words, a layer known as a database server or a relational database management system sits between the users of the system and the physical database (DBMS). The database is utilised by the application system and is managed by a DBMS. There are numerous modern RDBMSs, including SQL Server 2012, Oracle 11g, MySQL 5.5, and PostgreSQL 9.2. Database systems provide the benefits of speed, accuracy, and accessibility over strong systems, which allow for the storage of enormous volumes of data.

2.1 Oracle 11g

Relational database management system (RDBMS) Oracle Database, sometimes known as Oracle RDBMS or just Oracle, is a product of Oracle Corporation. It includes data storage and at least one instance of the application. Tablespaces and data files are used by the Oracle RDBMS to store data logically and physically, respectively. At the physical level, data files are made up of one or more data blocks, with different data files having different block sizes. Data dictionaries, indexes, and clusters are all characteristics of Oracle. Versions Following the introduction of 10g, grid computing capabilities were made available, allowing instance applications to utilise the CPU resources of another grid node.

2.2 MySQL 5.5

More than 10 million people have installed MySQL, a free, open-source, multithreaded, and multi-user SQL database management system. The fundamental application functions as a server that grants multiple users access to certain databases. In addition to extensions, cross-platform support, stored procedures, triggers, cursors, updatable views, and X/Open XA distributed transaction processing support, MySQL provides a sizable chunk of ANSI SQL 99. Additionally, it offers replication with one master per slave or many slaves per master, an embedded database library, an independent storage engine, SSL support, a two-phase commit engine, and ACID compliance utilising InnoDB cluster engines.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

2.3 PostgreSQL 9.2

The sole object-relational database in the group is PostgreSQL, which supports table and data type inheritance. Additionally, user-defined string to date conversion is only available in only one Open-Source Database Management System (OSDMS). Compared to other OSDMS, it offers more sophisticated indexing methods. It provides sub-selects, sequences, collections, and cascading update/delete. It is the only one that offers cross-database linkages for using a script operating in a different schema to access data in a database. Through ODBC, JDBC, or one of the programming languages (C, C++, Java, Perl, Python, PHP, Ruby, and R), one can access the database.

2.4 Microsoft SQL Server 2012

A relational database management system (RDBMS) created by Microsoft is called SQL Server. Transact-SQL, an implementation of the ANSI/ISO standard SQL used by both Microsoft and Sybase, is its main query language. Atomic, consistent, isolated, and durable transactions are supported by Microsoft SQL Server. Database mirroring and clustering are supported. For read-only indexes that categorise data, it offers column store indexes, simplifying huge data warehouse queries. It creates reports from business intelligence that are combined. It offers distributed replay from a production-based server and improved auditing.

2.5 SQLAPI++ Library

A C++ library called SQLAPI++ is used to access various SQL databases (Oracle, SQL Server, DB2, Sybase, Informix, InterBase, SQLBase, MySQL, PostgreSQL, SQLite, SQL Anywhere and ODBC). Applications created using the SQLAPI++ library run quickly and effectively because it employs the native Application Programming Interfaces (APIs) of the target DBMS. Additionally, the solution offers a low-level interface that gives access to database-specific functionality to developers. The SQLAPI++ library serves as middleware and provides database portability by encapsulating a vendor's API.

3 SYSTEM ARCHITECTURE

The prototype that has been created to demonstrate DMRDBs is introduced in this part. SNRConstructor, Schema Convertor, and Data Translator are the prototype's three core modules. The prototype architecture and the primary information flow between its modules and components are shown in Figure1. The user only interacts with the system to perform the following actions: 1) choose the source RDB to be migrated, providing the necessary username, password, database string, and host string; 2) choose the type of target database to be generated, deciding whether to produce a schema only or a complete database; and 3) view the target RDB using SQL DDL and DML queries.

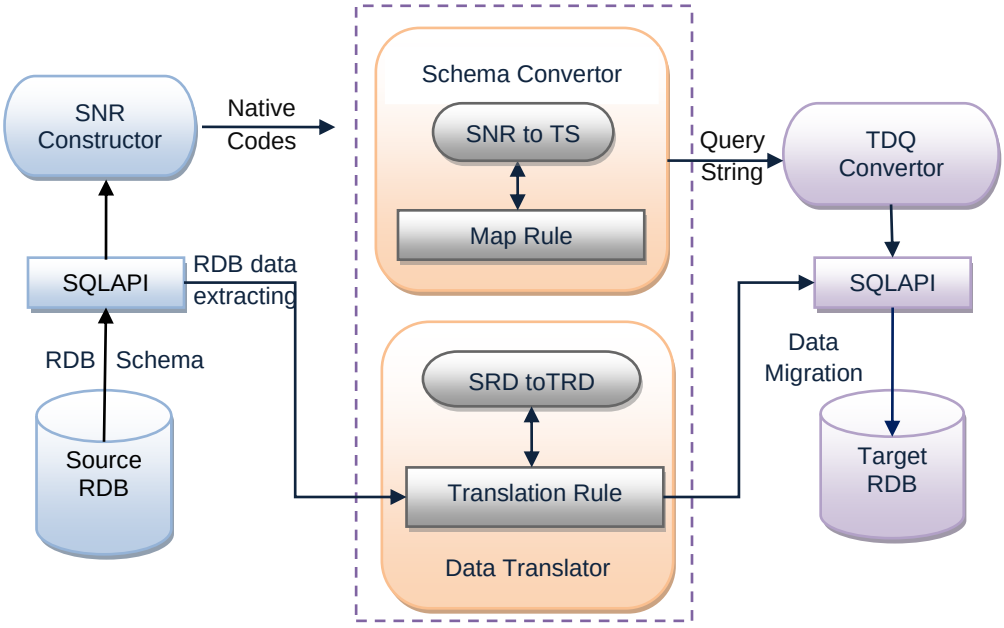

Figure 1: The overall architectural design

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

3.1 SNR (Schema Node Representation) Constructor

The SNR Constructor's purpose is to gather metadata about an existing RDB, producing a linked list structure as the **ConstructSNR** class. The core C++ class ConstructSNR, which yields the address of the linked list holding metadata about existing RDB, is how the algorithm is implemented. The functions called from the main class of the C++ code are used to create objects of the ConstructSNR class.

```
struct Node
{
    char *attrName, attrKey,attrIsNull;
    int attrType,NodeCount;
    long int attrSize,attrNativeType;
};
public class constructSNR
{
    // Constructor
    public constructSNR()
    {
        ...
        // linked list creation for SNR Construction
        for (int i=1;i<=RDBCmd.FieldCount();i++)
        {
            newnode = new Node;
            strcpy(newnode->attrName,RDBCmd[i].Name());
            newnode->attrType=RDBCmd[i].FieldType();
            newnode->attrNativeType=RDBCmd[i].FieldNativeType();
            newnode->attrSize=RDBCmd[i].FieldSize();
            strcpy(newnode->attrIsNull, RDBCmd[i].IsNull());
            if (RDBCmd[i].IsNull == 'N')
                    strcpy(newnode->attrKey, RDBCmd[i].FieldKey());
            newnode->link=NULL;
            NodeCount++;
        ...
        }
    }
};
```

Figure 2: A fragment of C++ code for the ConstructSNR class

A portion of the ConstructSNR class's constructor's C++ code is shown in Figure 2. This function Object() {[native code]} communicates with the RDBMS and gets details about the table in the RDB that has been chosen for migration. The linked list is created based on this data. The source table of the original RDB is used to get the key restrictions and each attribute's values. Column name, data type, length, precision, scale, null/not null, and especially the native value of the attributes are among the attributes' informational components. The name of the key table, the key column, the native value, and the name of the key constraint are all included in the description of each column of a primary key, foreign key, exported key, and unique key. The functions of SQLAPI++ previously discussed, such as Name(), FieldType(), FieldNativeType(), FieldSize(), IsNull(), and FieldKey, are used to get this information (). When all of this data has been retrieved from a single RDB table and is placed in a linked list, the schema convertor receives it.

3.2 The Schema Convertor

The Schema Convertor is responsible for translating SNR from the source RDB enriched into the corresponding target schemas. The C++ code of Schema Convertor is performed by implementing the algorithms and the mapping rules designed to produce the target schemas from the schema convertor, described in the class SchemaConversion Schema convertor implemented as a function createTargetSchema() as shown in **Figure** 3, where its schema conversion mapping rules are encoded. Figure 3 shows a fragment of C++ code of Schema Convertor.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

```
public class SchemaConversion
{    ....
     public void createTargetSchema()
     {    ....
          int i=0;
          S=HeadNode_of_SourceRDB;
          T=TargetNode_of_TargetRDB;
          while(i<NodeCount)
          {    //Mapping Rule for character data type
               if(S->attrNativeType==T->NativeType(CharacterDataType))
               {
                    strcat(qryString,S->attrName);
                    strcat(qryString," varchar(");
                    strcat(qryString,StringFromCLSID(S->attrSize,Buffer,10));
                    strcat(qryString,")");
               }
               //Mapping Rule for integer data type
               if(S->attrNativeType==T->NativeType(Integer))
               {
                    strcat(qryString,S->attrName);
                    strcat(qryString," integer(");
                    strcat(qryString,")");
               }
               ....
          }
          qryString[strlen(qryString)-1]=' ';
          strcat(qryString,")");
          RDBCmd.setCommandText(qryString);
          RDBCmd.Execute(qryString);
     }
};
```

Figure 3: C++ Code for Schema Convertor

This function is in charge of carrying out a certain mapping task, such as mapping attributes and the types of those attributes and converting relationships. The function goes through a loop for each attribute and maps them one by one. The data type of the source attribute is converted into target schema if its **attrNativeType**is mapped with target schema, then new data type is added to the query string. Finally, the new query string is generated which is passed to the TDQ constructor.

3.3 TDQ (Target Data Query) Convertor

The TDQ convertor takes the query generated by the Schema Convertor. This TDQ convertor is to generate a query for target data from existing RDB schema into the target schema. The C++ code of TDQ converter is performed by implementation the algorithm described in the class **constructTDQ**. The function TDQuery() shows to create query string for target data with suitable data types.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

```
public class constructTDQ
{       ....
        public void TDQuery()
        {
            ....
            strcpy(qryString,"insert into");
            strcat(qryString,sourceRDB_TableName);
            strcat(qryString,"values(");
            for (int i=0; i<RDBCmd.FieldCount();i++)
            {
                StringFromCLSID(S->attrSize,Buffer,10);
                strcat(qryString,":");
                strcat(qryString,Buffer);
                if (i!=RDBCmd.FieldCount()-1)
                strcat(qryString,")");
            }
            ....
        }
};
```

Figure 4: C++ Code for TDQ Convertor

3.4 The Data Translator

In order to convert existing RDB data into the format specified by the destination schemas, the Data Translator uses the TDQuery that was generated by the TDQuery to access the RDBMS records (tuples). The C++ code of data translator is as shown in Figure 4. Data stored as tuples in an RDB (source) are translated into target RDB. First, the TDQuery string is passed into command object of dataTranslation class. To execute the command, bind input variables by assigning values to dataTranslation object. The following chapter 4 details how experimental results and the research work were conducted to evaluate DMBRDBs. The experimental results are presented and compared with the selected databases for validation.

4 EXPERIMENTAL RESULTS WITH PERFORMANCE ANALYSIS

The prototype DMBRDBs is implemented with inputs of most preferred contemporary relational databases such as MySQL 5.5, Oracle 11g, PostgreSQL 9.2 and SQL Server 2012. Performance has been analysed on this anyone RDB as source and the remaining RDB as targets. In total, four combinations of performance analysis have been done. Analysis based on migration time (in seconds) and memory utilisation (in MB) takes to complete the migration process. The execution times are obtained and then tabulated for the further processing. The performance analysis done on all mentioned contemporary RDBs. This experiment result shows the performance on migrating 10 to 100000 rows of data from source RDB to target and tabulated migration time and memory utilisation.

However, the procedure of 100000 rows of data, migration takes time consuming, memory utilisation.Experiment results are varied based on the size of the tuple. If the quantity of the data is too large, this leads to time consuming and memory utilising size. For the experiment, the input consists five attributes (ID, NAME, AGE, ADDRESS and SALARY) on ‘Customers ‘relation. This ‘customers’ is the input migration tuple from source RDB to target. Rows of ‘Customers’ tuples are migrated from source to target RDB. The subsequent sections are described time and memory utilisation analysis of input RDBs.

4.1 MySQL 5.5 to other Target RDBs

The performance of proposed prototype is evaluated with experiments in the above-mentioned experimental environment, taken MySQL 5.5 as source RDB and Oracle 11g, PostgreSQL 9.2 and SQL Server 2012 as target RDBs.

Table 1 and Figure 5 show Execution time (in seconds) performance values of data migration from MySQL 5.5 to other RDBs. On migrating 10 rows of data, no noticeable time difference is found between MySQL to other targets RDBs.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

However, on migrating 100000 rows of data, a significant increase occurs in execution time. To sum up, MySQL as source RDB, Oracle 11g takes shorter execution time, and PostgreSQL takes longer time significantly. As a result, migration to Oracle is to be considered as the top option.

Table 1: Execution time performance of MySQL 5.5 as Source RDB

MySQL 5.5	**Migration Time (in Seconds)**		
Rows of data	**Oracle 11g**	**PostgreSQL 9.2**	**SQLServer 2012**
10	0.001	0.015	0.001
100	0.016	0.063	0.031
1000	0.156	0.625	0.375
10000	2.421	7.375	4.937
100000	20.094	69.641	44.157

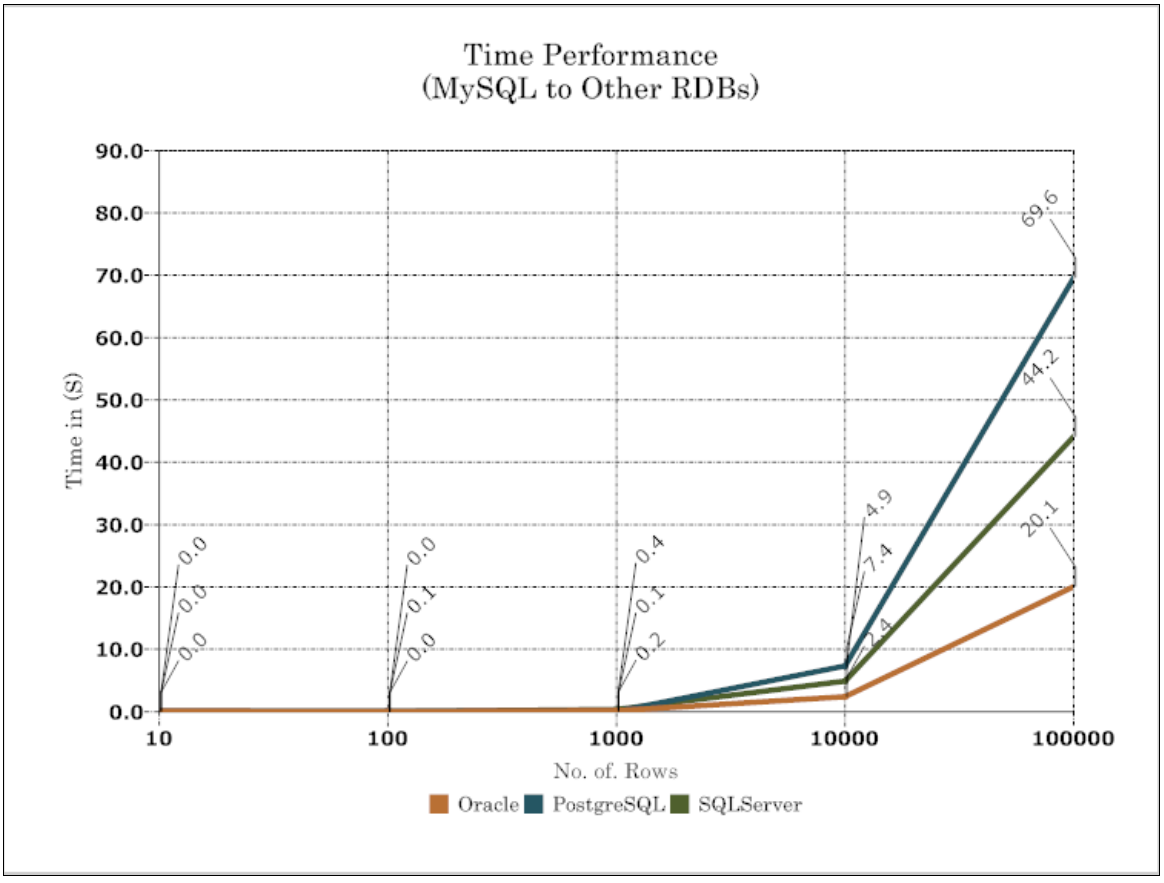

Figure 5: Execution time performance of MySQL 5.5 as Source RDB

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

Table 2: Memory utilization performance of MySQL 5.5 as Source RDB

MySQL 5.5	Memory Utilization (in MB)		
Rows of data	**ORACLE 11g**	**PostgreSQL 9.2**	**SQLServer 2012**
10	0.03	0.01	0.01
100	0.03	0.01	0.01
1000	0.03	0.06	0.03
10000	0.43	1.38	0.44
100000	3.54	6.14	3.92

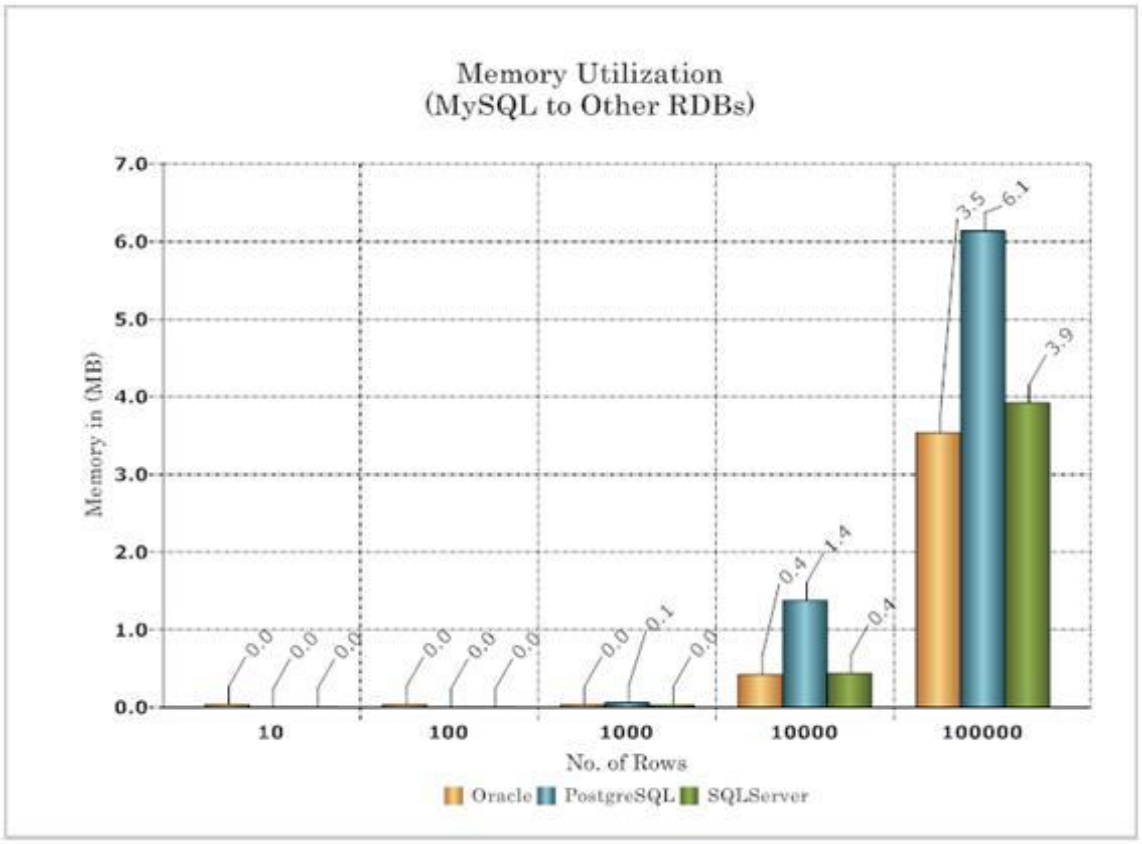

Figure 6: Memory utilization performance of MySQL 5.5 as Source RDB

According to Table 2 and Figure 6, data migration from MySQL 5.5 to PostgreSQL 9.2, the performance of memory utilisation is significantly higher than Oracle 11g and SQL Server 2012. Less migration time (values are shown in Table 1 and Figure 5) is found during migrating from MySQL 5.5 to Oracle 11g. This is because significantly low memory utilisation of MySQL 5.5 compared with PostgreSQL 9.2 and SQL Server 2012. It is evident from the results that the performance 'execution time' and 'memory utilisation of data migration of MySQL 5.5 to Oracle 11g is faster than PostgreSQL 9.2 and SQL Server 2012.

4.2 Oracle 11g to other Target RDBs

For the next combination, Oracle 11g as source RDB and MySQL 5.5, PostgreSQL 9.2 and SQL Server 2012 as target RDBs. Execution time performance of data migration from Oracle 11g to other RDBs is shown in Table 3 and chart representation illustrated in Figure 7. Table 3 and Figure 7 show Execution time (in seconds) performance values of data

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452
migration from Oracle 11gto other RDBs. On migrating 10 rows of data, no noticeable time difference is found between Oracle 11g to other targets RDBs. However, on migrating 100000 rows of data, a significant increase occurs in execution time. To sum up, Oracle as source RDB, SQL Server takes shorter execution time, and MySQL takes longer time significantly. As a result, migration to SQL Server is to be considered as the top option.

Table 3: Execution time performance of Oracle 11g as Source RDB

Oracle11g	**Migration Time (in Seconds)**		
Rows of data	**MySQL 5.5**	**PostgreSQL 9.2**	**SQLServer 2012**
10	0.234	0.016	0.016
100	2.813	0.094	0.062
1000	25.029	1.01	0.594
10000	245.079	10.145	7.407
100000	2445.579	91.187	62.687

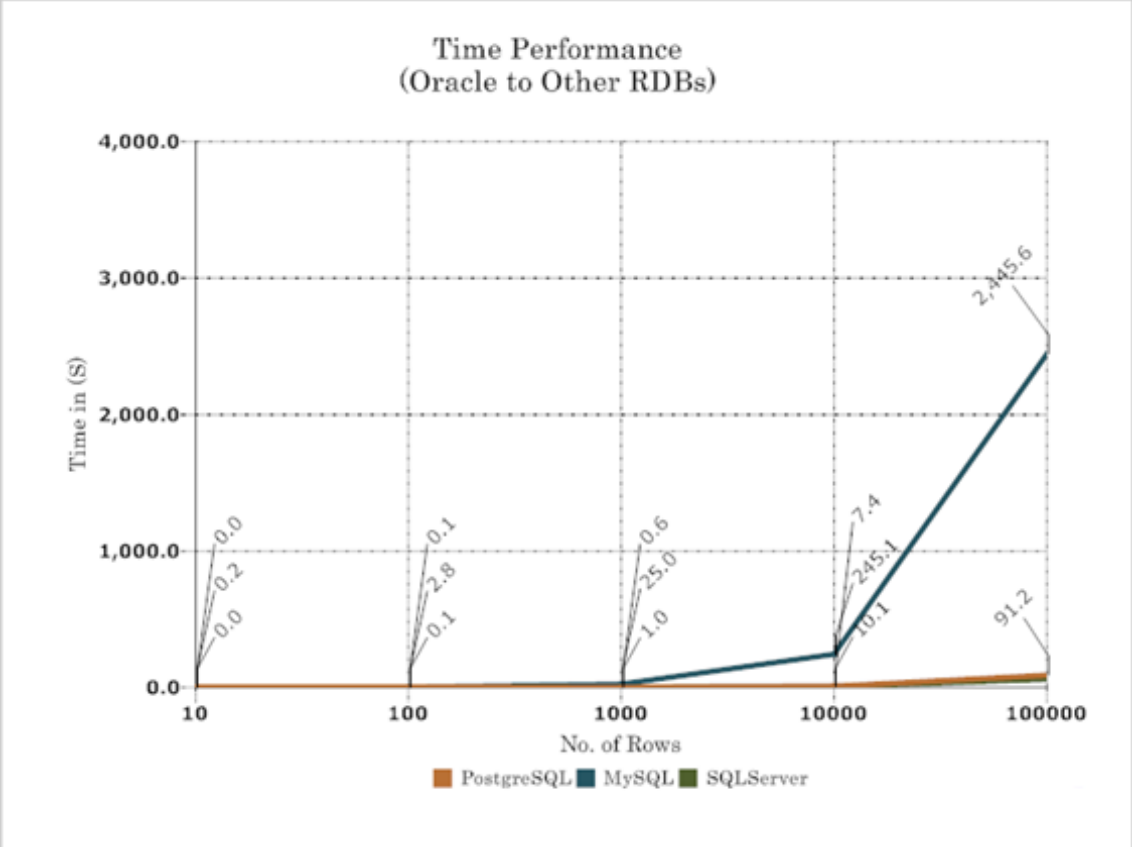

Figure 7: Execution time performance of Oracle 11g as Source RDB

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

Table 4: Memory utilization performance of Oracle 11g as Source RDB

Oracle11g	Memory Utilisation (in MB)		
Rows of data	**MySQL 5.5**	**PostgreSQL 9.2**	**SQLServer 2012**
10	0.02	0.01	0.01
100	0.2	0.01	0.01
1000	0.08	0.06	0.03
10000	1.52	1.38	0.44
100000	6.52	6.14	3.92

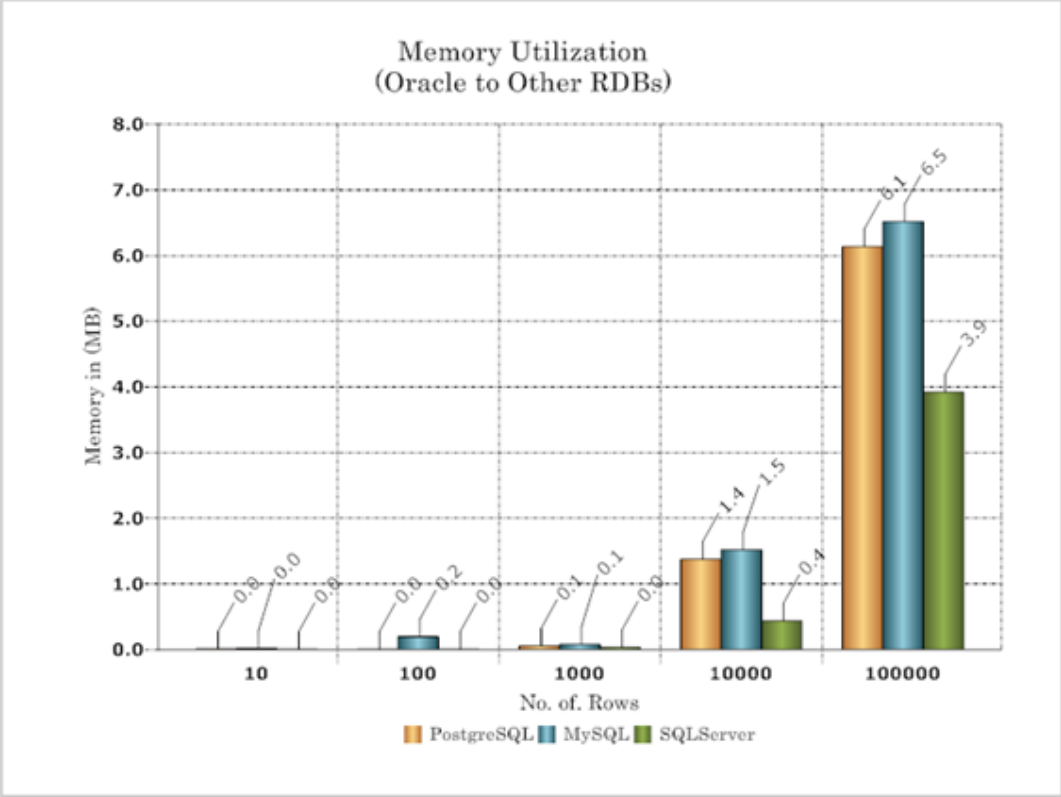

Figure 8: Memory utilization performance of Oracle 11g as Source RDB

As can be seen in Table 4 and Figure 8, data migration from Oracle11g to MySQL 5.5, the performance of memory utilisation is significantly higher than PostgreSQL 9.2 and SQL Server 2012. Less migration time (values are shown in Table 3 and Figure 7) is found during migrating from Oracle 11g to SQL Server 2012. This is because significantly low memory utilisation of Oracle 11g compared with PostgreSQL 9.2 and MySQL 5.5. It is evident from the results that the performance 'execution time' and 'memory utilisation of data migration of Oracle 11g to SQL Server 2012 is faster than PostgreSQL 9.2 and MySQL 5.5.

4.3 PostgreSQL 9.2 to other Target RDBs

For the next combination, PostgreSQL 9.2 as source RDB and MySQL 5.5, Oracle 11g and SQL Server 2012 as target RDBs. Table 5 and Figure 9 show Execution time (in seconds) performance values of data migration from

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452
PostgreSQL9.2 to other RDBs. On migrating 10 rows of data, no noticeable time difference is found between PostgreSQL to other targets RDBs. However, on migrating 100000 rows of data, a significant increase occurs in execution time. To sum up, PostgreSQL as source RDB, Oracle 11g takes shorter execution time, and MySQL takes much longer time. As a result, migration to Oracle is to be considered as the top option.

Table 5: Execution time performance of PostgreSQL 9.2 as Source RDB

PostgreSQL 9.2	**Migration Time (in Seconds)**		
Rows of data	**MySQL 5.5**	**Oracle 11g**	**SQLServer 2012**
10	0.25	0.016	0.078
100	2.5	0.047	0.031
1000	22.288	0.156	0.391
10000	279.203	2.203	4.765
100000	3078.766	22.844	43.703

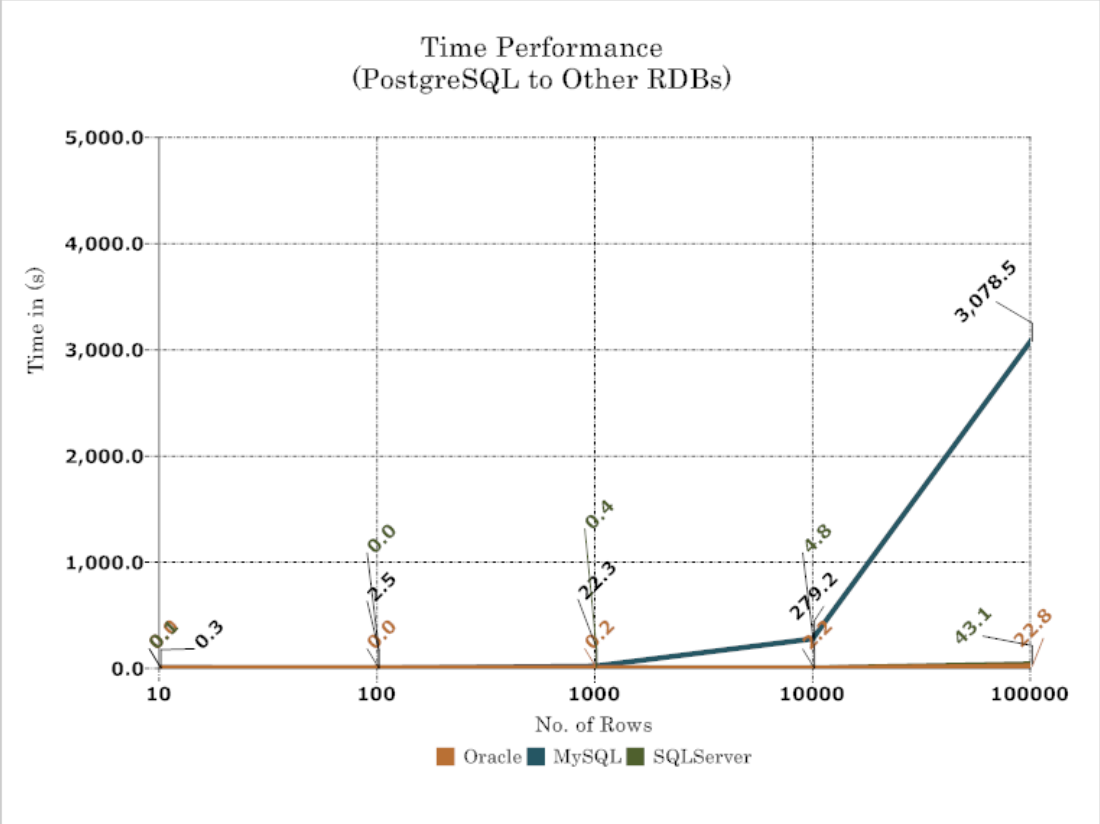

Figure 9: Execution time performance of PostgreSQL 9.2 as Source RDB

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

Table 6: Memory utilization performance of PostgreSQL 9.2 as Source RDB

PostgreSQL 9.2	**Memory Utilization (in MB)**		
Rows of data	**MySQL 5.5**	**Oracle 11g**	**SQLServer 2012**
10	0.02	0.03	0.01
100	0.2	0.03	0.01
1000	0.08	0.03	0.03
10000	1.52	0.43	0.44
100000	6.52	3.54	3.92

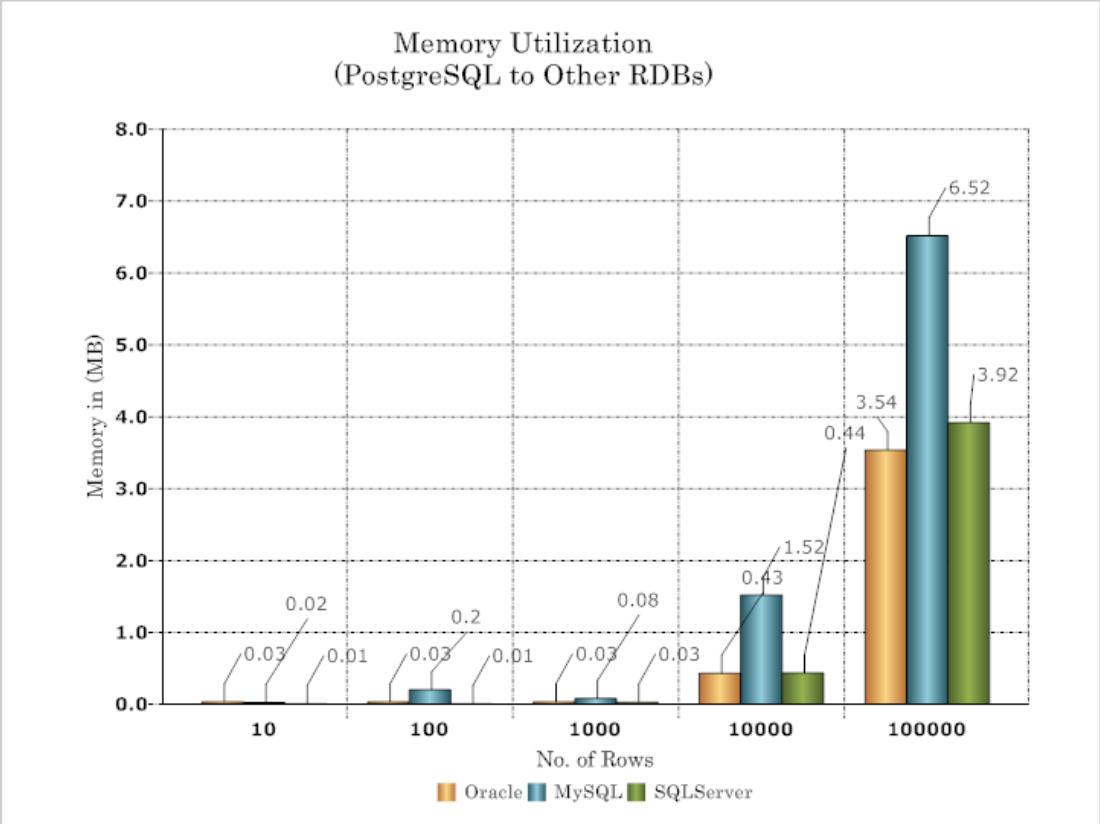

Figure 10: Memory utilization performance of PostgreSQL 9.2 as Source RDB

As can be seen in Table 6 and Figure 10, data migration from PostgreSQL 9.2 to MySQL 5.5, the performance of memory utilisation is significantly higher than Oracle 11g and SQL Server 2012. Less migration time (values are shown in Table 5 and Figure 9) is found during migrating from PostgreSQL 9.2 to Oracle 11g. This is because significantly low memory utilisation of PostgreSQL 9.2 compared with MySQL 5.5 and SQL Server2012. It is evident from the results that the performance 'execution time' and 'memory utilisation of data migration of PostgreSQL 9.2 to Oracle 11g is faster than SQL Server 2012 and MySQL 5.5.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

4.4 SQL Server 2012 to other Target RDBs

For the last combination, SQL Server 2012 as source RDB and MySQL 5.5, PostgreSQL 9.2 and Oracle 11g as target RDBs. Table 7 and Figure 11 show Execution time (in seconds) performance values of data migration from SQL Server 2012 to other RDBs. On migrating 10 rows of data, no noticeable time difference is found between SQL Server to other targets RDBs. However, on migrating 100000 rows of data, a significant increase occurs in execution time. To sum up, SQL Server as source RDB, Oracle 11g takes shorter execution time, and PostgreSQL takes longer time significantly. As a result, migration to Oracle is to be considered as the top option.

Table 7: Execution time performance of SQL Server 2012 as Source RDB

SQL Server 2012	**Migration Time (in Seconds)**		
Rows of data	**MySQL 5.5**	**Oracle 11g**	**PostgreSQL 9.2**
10	0.500	0.001	0.001
100	2.765	0.016	0.078
1000	25.063	0.156	0.64
10000	280.968	2.359	7.563
100000	3082.812	20.828	70.125

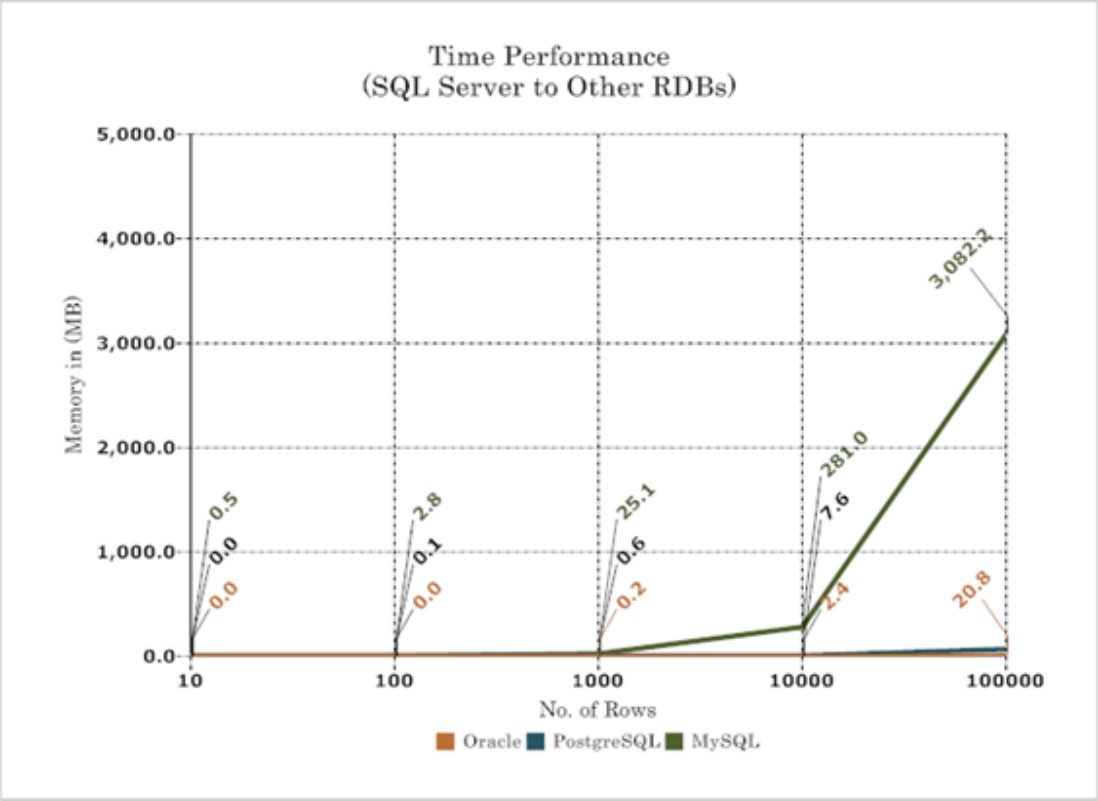

Figure 11: Execution time performance of SQL Server 2012 as Source RDB

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

Table 8: Memory utilization performance of SQL Server 2012 as Source RDB

SQL Server 2012	**Memory Utilisation (in MB)**		
Rows of data	**MySQL 5.5**	**Oracle 11g**	**PostgreSQL 9.2**
10	0.02	0.03	0.01
100	0.2	0.03	0.01
1000	0.08	0.03	0.06
10000	1.52	0.43	1.38
100000	6.52	3.54	6.14

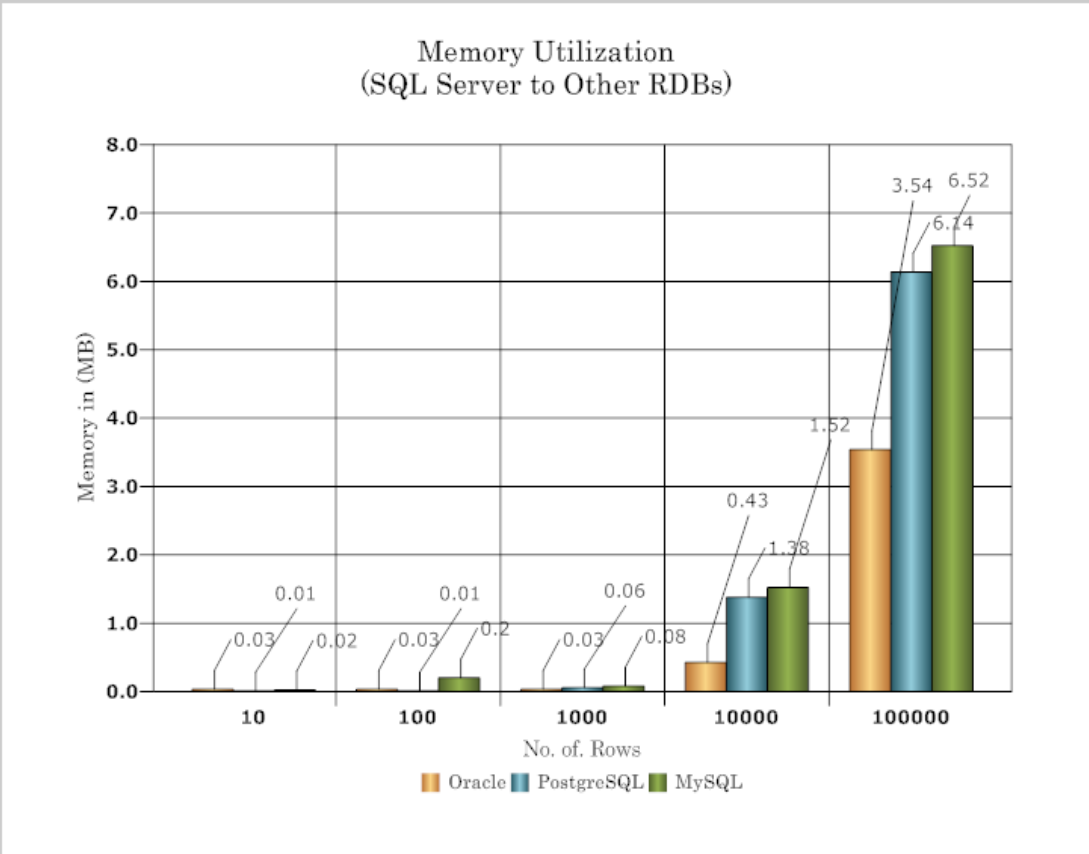

Figure 12: Memory utilization performance of SQL Server 2012 as Source RDB

As can be seen in Table 8 and Figure 12, data migration from SQL Server 2012 to MySQL 5.5, execution time is significantly higher than PostgreSQL 9.2 and Oracle 11g. Less migration time (values are shown in Table 7 and Figure 11) is found during migrating forms Server 2012to Oracle 11g. This is because significantly low memory utilisation of SQL Server 2012 compared to MySQL 5.5 and PostgreSQL 9.2. It is evident from the results that the performance 'execution time' and 'memory utilisation of data migration of SQL Server 2012to Oracle 11g is faster than PostgreSQL 9.2 and MySQL 5.5.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.637-653
https://publishoa.com
ISSN: 1309-3452

5. CONCLUSION

In Relational Database Migration, the performance of execution time is directly proportional to the performance of memory utilisation. If memory utilisation is significantly low, the performance of execution time also significantly low. In DMBRDBs, DDL queries are performed at tuple level operations. The execution time of DDL queries is different from one RDB to another. For instance, MySQL RDB had been taken 19 milliseconds (Bassil, 2012) to perform '*select * from customers;*', but Oracle 11g and SQL Server had taken 23 and 19 milliseconds for the same query. Memory utilisation of the same query, MySQL 5.5 had utilised 3MB, but Oracle 11g utilised 11MB.If MySQL 5.5 as source RDB, best migration is to Oracle 11g.Perhaps, Oracle 11g as source RDB, best migration is to SQL Server 2012. In fact, these DDL queries have a different level of complexity for different heterogeneous RDBs. The investigation of this result is to be done in future research work because different RDB had heterogeneous schema structures and different data types. Therefore, deep study and research work are to be done in future work.

References

Abdelsalam Maatuk, A. A., & Rossiter, N. (2008). An Integrated Approach to Relational Database Migration. In *in Int. Conf. on Information and Communication Technologies. (IC-ICT 2008)*, (pp. 1-6). Northumbria Research Link.

Alhajj, R. (2003). Extracting the extended entity-relationship model from a legacy relational database. *Information Systems*, *28*(6), pp. 597-618.

Alhajj, R., & Polat, F. (2001). Reengineering relational databases to object-oriented: Constructing the class hierarchy and migrating the data. In *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on* (pp. 335-344). IEEE.

Bassil, Y. (2012). A comparative study on the performance of the Top DBMS systems. *arXiv preprint arXiv:1205.2889*.

Behm, A., Geppert, A., & Dittrich, K. R. (1997). *On the migration of relational schemas and data to object-oriented database systems* (pp. 13-33). Universität Zürich. Institut für Informatik.

Bisbal, J., Lawless, D., Wu, B., & Grimson, J. (1999). Legacy information systems: Issues and directions. *IEEE software*, *16*(5), (pp. 103-105).

Brodie, M. L., & Stonebraker, M. Migrating legacy systems: gateways, interfaces & the incremental approach. 1995.

Bychkov, Y., & Jahnke, J. H. (2001). Interactive migration of legacy databases to net-centric technologies. In *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on* (pp. 328-334). IEEE.

Chen, P. P. S. (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, *1*(1), (pp. 9-36.)

Chiang, R. H., Barron, T. M., & Storey, V. C. (1994). Reverse engineering of relational databases: Extraction of an EER model from a relational database. *Data & Knowledge Engineering*, *12*(2), (pp. 107-142).

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), (pp. 377-387).

Codd, E. F. (1990). *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc..

Crowe, M. K. (1993). Object systems over relational databases. *Information and Software Technology*, *35*(8), (pp. 449-461).

Curé, O., & Squelbut, R. (2005, December). A database trigger strategy to maintain knowledge bases developed via data migration. In *Portuguese Conference on Artificial Intelligence* (pp. 206-217). Springer Berlin Heidelberg.

Davis, K. H., & Alken, P. H. (2000). Data reverse engineering: A historical survey. In *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on* (pp. 70-78). IEEE.

Devarakonda, R. S. (2001). Object-relational database systems—the road ahead. *Crossroads*, *7*(3), (pp. 15-18).

Draheim, D., Horn, M., & Schulz, I. (2004, June). The schema evolution and data migration framework of the environmental mass database IMIS. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on* (pp. 341-344). IEEE.

Elamparithi, M. (2010, December). Database Migration Tool (DMT)-accomplishments & future directions. In *Communication and Computational Intelligence (INCOCCI), 2010 International Conference on* (pp. 481-485). IEEE.

Elamparithi, M., & Anuratha, V. (2015). A Review on Database Migration Strategies, Techniques and Tools. *World Journal of Computer Applications*, *3*(3), (pp. 41-48).

Elmasri, R. and Navathe, S. B. (2006). Fundamentals of database systems (5th Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Fahrner, C., & Vossen, G. (1995, December). Transforming relational database schemas into object-oriented schemas according to ODMG-93. In *International Conference on Deductive and Object-Oriented Databases* (pp. 429-446). Springer Berlin Heidelberg.

Fishman, D. H., Beech, D., Cate, H. P., Chow, E. C., Connors, T., Davis, J. W., & Mahbod, B. (1994, July). Iris: an object-oriented database management system. In *Readings in database systems (2nd ed.)* (pp. 827-837). Morgan Kaufmann Publishers Inc..

Fong, J., & Cheung, S. K. (2005). Translating relational schema into XML schema definition with data semantic preservation and XSD graph. *Information and software technology*, *47*(7), (pp. 437-462).

from http://www.InfoWorld.comSEI (2004), Software Engineering Institute Glossary. [online] http://www.sei.cmu.edu/str/indexes/glossary/.

Graham, I. S., & Graham, A. (1995). *Migrating to object technology*. Addison-Wesley Longman Publishing Co., Inc..

Hainaut, J. L. (1991, November). Database Reverse Engineering: Models, Techniques, and Strategies. In *ER* (pp. 729-741).

Hardwick, M., & Spooner, D. L. (1989). The ROSE data manager: using object technology to support interactive engineering applications. *IEEE transactions on knowledge and data engineering*, *1*(2), (pp. 285-289).

Hohenstein, U., & Körner, C. (1997). A graphical tool for specifying semantic enrichment of relational databases. In *Database Applications Semantics* (pp. 389-420). Springer US.

Hohenstein, U., & Plesser, V. (1996). Semantic enrichment: A first step to provide database interoperability. In *Workshop Föderierte Datenbanken, Magdeburg* (pp. 3-17).

Horstmann, J. (2005). Migration to open source databases. *Computation and Information Structures (CIS)*, (pp. 12-13).

Housel, B. C., Lum, V. Y., & Shu, N. (1974, May). Architecture to an interactive migration system (AIMS). In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control* (pp. 157-169). ACM.

Hudicka, J. (2000). The complete data migration methodology. *Dulcian Inc., June*.

Kim, W. (1991). *Introduction to object-oriented databases*. MIT press.

Leavitt, N. (2000). Whatever happened to object-oriented databases?. *IEEE Computer*, *33*(8), (pp. 16-19).

Lim, W. M., & Harrison, J. (1996, July). An Integrated Database Reengineering Architecture-A Generic Approach. In *Australian Software Engineering Conference, 1996., Proceedings of 1996* (pp. 146-154). IEEE.

Maatuk, A. (2009). *Migrating relational databases into object-based and XML databases* (Doctoral dissertation, Northumbria University).

Maatuk, A., Ali, A., & Rossiter, N. (2010). Converting relational databases into object relational databases. *Journal of Object Technology*, *9*(2), (pp. 145-161).

Marks, R. M., & Sterritt, R. (2013). A metadata driven approach to performing complex heterogeneous database schema migrations. *Innovations in Systems and Software Engineering*, *9*(3), (pp. 179-190).

McBrien, P., & Poulovassilis, A. (2002, May). Schema evolution in heterogeneous database architectures, a schema transformation approach. In *International Conference on Advanced Information Systems Engineering* (pp. 484-499). Springer Berlin Heidelberg.

Meier, A. (1995, September). Providing Database Migration Tools-A Practicioner's Approach. In *Proceedings of the 21th International Conference on Very Large Data Bases* (pp. 635-641). Morgan Kaufmann Publishers Inc..

Meier, A., Dippold, R., Mercerat, J., Muriset, A., Untersinger, J. C., Eckerlin, R., & Ferrara, F. (1994). Hierarchical to relational database migration. *IEEE Software*, *11*(3), (pp. 21-27).

Monk, S., Mariani, J. A., Elgalal, B., & Campbell, H. (1996). Migration from relational to object-oriented databases. *Information and Software Technology*, *38*(7), (pp. 467-475).

Moriarty, T., & Hellwege, S. (1998). Data migration. *Database Programming & Design*, (pp. 11-14).

Parent, C., & Spaccapietra, S. (2000). Database integration: the key to data interoperability.*Advances in Object-Oriented Data Modeling*, The MIT Press, (pp. 221-254).

Park, J., & Ram, S. (2004). Information systems interoperability: What lies beneath?. *ACM Transactions on Information Systems (TOIS)*, *22*(4), (pp. 595-632).

Premerlani, W. J., & Blaha, M. R. (1993, May). An approach for reverse engineering of relational databases. In *Reverse Engineering, 1993., Proceedings of Working Conference on* (pp. 151-160). IEEE.

Ramanathan, S., & Hodges, J. (1996). Reverse engineering relational schemas to object-oriented schemas. *Techical Report No. MSU-960701, July*, *1*.

Rankins, R., Bertucci, P., Gallelli, C., & Silverstein, A. T. (2010). *Microsoft SQL server 2008 R2 unleashed*. Pearson Education.

Schwartz, E. (2005). Expanding Legacy Apps. *InfoWorld*. Retrieved November 15, 2006,

Sockut, G. H., & Goldberg, R. P. (1979). Database reorganization-principles and practice. *ACM Computing Surveys (CSUR)*, *11*(4), (pp. 371-395).

Sousa, P., Pedro-de-Jesus, L., Pereira, G., & e Abreu, F. B. (1999). Clustering relations into abstract er schemas for database reverse engineering. In *Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on* (pp. 169-176). IEEE.

Stonebraker, M., & Moore, D. (1995). *Object Relational DBMSs: The Next Great Wave*. Morgan Kaufmann Publishers Inc..

Thalheim, B., & Wang, Q. (2013). Data migration: A theoretical perspective. *Data & Knowledge Engineering*, *87*, (pp. 260-278).

Walek, B., & Klimes, C. (2012). A methodology for data migration between different database management systems. *International Jouranal Comput Inf Eng*, *6*, (pp. 85-90).

Wang, C., Lo, A., Alhajj, R., & Barker, K. (2005, April). Novel approach for reengineering relational databases into XML. In *21st International Conference on Data Engineering Workshops (ICDEW'05)* (pp. 1284-1284). IEEE.

Wang, G., Jia, Z., & Xue, M. (2014). Data migration model and algorithm between heterogeneous databases based on web service. Journal of Networks, 9(11), (pp. 3127-3134).

Wei, B., & Chen, T. X. (2012). Criteria for evaluating general database migration tools.[online] Research Report, RTI Press Publication No. OP-0009-1210. Research Triangle Park, NC:RTI Press. http://www.rti.org/pubs/op-0009-1210-chen.pdf (Accessed 28 August 2015)

Weiderman, N. H., Bergey, J. K., Smith, D. B., & Tilley, S. R. (1997). *Approaches to Legacy System Evolution* (No. CMU/SEI-97-TR-014). CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Wilkinson, K., Lyngbaek, P., & Hasan, W. (1990). The Iris architecture and implementation. *IEEE Transactions on Knowledge and Data Engineering*, *2*(1), (pp. 63-75).

Zhang, X., & Fong, J. (2000). Translating update operations from relational to object-oriented databases. *Information and Software Technology*, *42*(3), (pp. 197-210).

Zhang, Z., Wang, W., & Zhang, R. (2011, August). Research of the Distributed Heterogeneous Database Conversion Mechanism. In *International Conference on Computer Science, Environment, Ecoinformatics, and Education* (pp. 59-64). Springer Berlin Heidelberg.