EXPLORING OBJECT-RELATIONAL MAPPING (ORM) SYSTEMS AND HOW TO EFFECTIVELY PROGRAM A DATA ACCESS MODEL

Mikhail Gorodnichev[1], Marina Moseva[2], Ksenia Poly[3], Khizar Dzhabrailov [4], Rinat Gematudinov [5]

[1,2,3,4,5] Moscow Technical University of Communications and Informatics, Moscow, Russia

[1] m.g.gorodnichev@mtuci.ru

Mikhail Gorodnichev, Marina Moseva, Ksenia Poly, Khizar Dzhabrailov, Rinat Gematudinov Exploring Object-Relational Mapping (Orm) Systems And How To Effectively Program A Data Access Model-- Palarch's Journal Of Archaeology Of Egypt/Egyptogy 17(3), 615-627. ISSN 1567-214x

Keywords: Software Development, Relational Databases, Orm System, Opti-Mization Of Program Code, Object-Oriented Programming.

ABSTRACT

The most common problem when using ORM libraries is a decrease in application performance compared to access to the database by executing manually written SQL queries. Several studies by Russian and foreign authors indicate a significant drop in speed when working with databases through ORM, sometimes by 5 or more times. Since existing studies show a significant variation in the results, and there are no studies conducted using the latest version of Entity Framework 6. This study is dedicated to solving the problem of the joint use of two basic technologies for designing information systems - object-oriented programming and relational database management systems. The paper discusses two ways of interacting with an RDBMS - through ORM in the form of its specific implementation of Entity Framework 6.2, or through SQL queries written without using ORM - and assesses the effect of an additional layer of abstraction in the form of ORM on the speed of interaction with the database, and also considers ORM performance optimization methods. The paper presents the results of the optimization and final testing of the use of the ORM system, draws conclusions about the effectiveness of using such systems in the development of software applications.

INTRODUCTION

The main technology for storing and processing data today are relational DBMSs - they have proven their effectiveness and safety. However, the design of software applications primarily uses the object-oriented approach (OOP). Both approaches are widely used in modern information systems, and due to their popularity, they are of- ten used together to create complex software systems [1]. Since OOP and relational DBMSs are based on completely different principles, fundamental differences are inherent in them, and their joint use is fraught with several problems. In the early 1990s, these differences were called the "semantic gap".

This discrepancy between the two most popular paradigms of modern computer science is due to several reasons. The fact is that OOP is based on proven principles

of software development. In fact, objects (instances of classes) refer to each other and therefore form a hierarchy (graph in the mathematical sense). Relational schemes, by contrast, are tabular and based on relational algebra, which defines related heterogeneous tuples (grouping data fields in a row with different types for each field). In addition, the fundamental principles of OOP are encapsulation, abstraction, inheritance, and polymorphism, which have not been developed in the relational database model. Thus, OOP uses concepts that are poorly combined with the architecture of relational DBMSs, which are mainly presented in the form of systems that execute queries in SQL. These fundamental differences are a key issue in this paper.

To solve the above problems, the use of object-relational mapping technology (abbr. ORM), which is the object of this study, is called upon. Currently, ORM is a common tool for developing complex systems that allows you to combine an object- oriented model of data representation with a relational one, namely, to connect the RDBMS with the concepts of OOJP, creating the so-called virtual object database.

The subject of the study is one of the most popular ORM libraries called Entity Framework, originally developed by Microsoft, but since version 6.0, it is an open source library.

Since the most frequent problem when using ORM libraries is a decrease in application performance (as compared to accessing the database by executing SQL queries written by hand), the existing studies of this problem show a significant spread in the results (although everyone comes to the same conclusion that ORM to one degree or another reduces the speed of working with the database).

The purpose of this work is to demonstrate in practice the difference between the two ways of working with RDBMSs - through ORM in the form of its specific implementation of Entity Framework 6.2, or through

SQL queries written without using ORM - and to evaluate the effect of an additional abstraction layer in the form of ORM on the speed of interaction with DB, and also consider methods for optimizing ORM performance.
The objectives of this study:

1. To study the phenomenon of the semantic gap between relational DBMS models and the object-oriented paradigm, considering the relational DBMS and the reasons for their popularity, the principles of OOP, the causes and problems of the semantic gap and alternative ways to solve them.
2. To propose a solution to the problem of semantic gap through the technology of object-relational mapping (ORM), to describe the principles of this technology.
3. Consider the ways to create a data access model (EDM) in the Entity Framework, as well as the operation scheme and features of this library.

4. Design and develop test applications that interact with RDBMSs with or without ORM in order to assess the impact of using ORM on application performance.
5. Evaluate the possibility of optimizing performance when using the Entity Frame- work 6.2 ORM library.

METHODS

Reasons for the semantic gap between OOP and RDBMS

The semantic gap is a set of conceptual and technical difficulties that are often encountered when a relational DBMS (RDBMS) is served by an applications (or several applications) written in an object-oriented language or programming style, because the definitions of objects or classes must be mapped to database tables defined by a relational schema. As previously described in [2], the semantic gap is observed in the following aspects:

1. Basic principles of OOP. OOP entities are objects belonging to different classes. Classes possess inheritance properties, forming a hierarchy, which is the natural paradigm of OOP, and polymorphism, when the behavior of a descendant object can be changed when inheriting from a parent class. Also, when designing a class hierarchy, abstraction is used to simplify the structure of each class and to differentiate their area of responsibility and encapsulation to protect data from possible changes from outside the class. In contrast to the OOP principles described above, in RDBMS only two-dimensional tables are used, where the rows are records and the columns are fields that are the same for all table records, but the most important OOP principles described above are not supported.

Compliance with the data scheme. Objects are not required to follow the

"parent schema," while rows in a relational table must conform to a single schema — each row can belong to one and only one entity (table).

Normalization and associative relations. In RDBMSs, normalization principles are applied (which are usually ignored in OOP) - after the database is reduced to the first normal form to avoid data duplication, they are divided into many different tables connected by primary and secondary keys, while in OOP class fields may contain references to objects of another class and there is no need to use keys to establish communication between objects of different classes.

The methods for accessing linked data through OOP and RDBMS are fundamentally different, since in OOP, transitions from a parent to descendant objects are implemented sequentially through links, and objects are initialized as necessary, but such an algorithm is inefficient for reading information from an RDBMS, since the number of queries to the database should be minimized, and the necessary data located in different tables must be loaded at the same time by using complex SQL queries.

Access rules and the relationship between entities (fields) and actions (methods).
Declarative approach versus imperative. Semantic differences are especially evi- dent in the aspects of data manipulation.
Use of various data types.

Identity of objects (records). Objects (other than immutable) are usually considered unique; two objects that are in the same state at a given time are not considered identical. On the other hand, for a relational model, it is common practice to create globally unique keys as identifiers (although sometimes this is considered bad

practice for those entries in the database that do not directly correspond to the essence of the subject area).

Within the framework of OOP, objects have interfaces in the form of special methods ("getters" and "setters", or properties in the C # language), which are the only way to access the internal fields of the object from the outside. The relational mod- el, by contrast, uses representations to provide various data slices, and constraints to ensure data integrity.

Semantic gap issues between OOP and rdbms

Since for a software application to work with data, they need to be stored in relational DBMSs and retrieved from there, the following problems arise (partially described earlier in [2]):

Implementation of CRUD logic needs to be done for each class in the application, manually programming the four above functions to maintain the integrity of information in the database and to ensure encapsulation of classes in the application.

The data downloaded from the RDBMS in the form of tables must be converted to the corresponding application objects and then back to the tables to be stored in the RDBMS, while the relationships between the entities must be considered.

Often the object model contains more classes than the number of corresponding tables in the database, which introduces additional difficulties in converting data from objects to a tabular form.

Object-relational transformations in most cases should be bidirectional, since you can make changes to the data on either side, and these changes must be compared with the other side.

The data used by the software application can span several storage systems (DBMS), each with its own protocols and features.

Solving the problem of semantic gap through the technology of object-relational mapping (ORM)

The task is to provide work with data in the form of instances of classes, rather than data tables, and, on the contrary, convert the data of class objects into data suitable for storage in an RDBMS, and also provide an interface for CRUD operations on data. One of the modern solutions to this problem is the ORM technology, which solves the previously described problems, while saving developers from writing a large amount of monotonous code - ORM "creates an additional layer of abstraction in order to facilitate manipulation of objects and relationships of the subject area stored in the database" [3].

So, object-relational mapping (English object-relational mapping, abbreviated as O/ R mapping or ORM) in the broad sense in computer science is a method of converting data between incompatible systems for their storage and processing using object- oriented programming languages. However, in a narrower sense, which we will use in this and subsequent sections, ORM is a library of a programming language that maps objects of a relational model to objects of a programming language and vice versa. ORM library automates data conversion between.

The task of the ORM system is to implement the correspondence between the objects used in the application and related tables in the database - it provides developers with a conceptual abstraction for comparing database records with application objects. Such an abstraction significantly reduces the amount of code that developers need to write [4, 5]. At the same time, ORM automatically resolves many issues that arise when creating such a correspondence - with the help of such mapped objects, developers can access database records without worrying about the details of the queries. In fact, the developer, when designing the application and writing the code, can general- ly “forget” about the existence of a DBMS and operate solely with OOP entities, and the rest of the work of the ORM will

be taken care of.

Advantages and disadvantages of using ORM technology in various types of applications

Among the advantages of ORM distinguish (as previously described in [2]):
The presence of an explicit description of the database schema, implemented in some programming language, which is located and edited in one place;

An opportunity for an application developer to operate with the usual tools of a programming language, that is, classes (objects) and their fields and methods, rather than structures of a relational database;

The ability to isolate the program logic from the data interface, which greatly simplifies the structure and code of the application;
Ability to automatically create database queries;

No need to change the application code and SQL queries when transferring data to a DBMS of another manufacturer, since the corresponding ORM adapter is responsible for this;

No need to write SQL queries and work out a significant amount of program code (which is usually monotonous and error prone) in each case of accessing data in the program;

Support by developed ORM implementations for mapping inheritance and composition to tables;

Compared to traditional methods of interaction with RDBMSs, ORM in most cases significantly reduces the amount of code that needs to be written (often several times, see [5]).

The drawbacks of ORM tools are usually associated with a high level of abstraction that hides what happens in the executable code. In addition, a strong dependence on ORM libraries was described in [6] as the main factor in creating poorly designed databases.
Also, with all its advantages, a specific implementation of ORM technology may have several other disadvantages:
Potentially lower ORM performance compared to accessing the DBMS directly through well-written SQL code.

The appearance of hard-to-debug errors in the program in those cases where there are flaws in the implementation of ORM.

The presence of separate tables in the case of direct mapping of classes into tables and the need to display fields that store collections, since many ORMs are based on the idea that tables are entities.

If we talk about the main drawback of ORM - a potential decrease in

performance, the reason for this lies in the fact that most ORM systems support a wide range of possible scenarios for working with data that no single application can ever use in its entirety. The question of the appropriateness of using ORM, as a rule, arises in large highly loaded projects, however, it is worth considering such an important criterion as the resources and time required for independent development of the object-relational mapping layer. In the case of small projects that do not face high workloads, the use of ORM is obvious. It should also be noted that many ORM systems provide the developer with the ability to manually write SQL query code if necessary. Moreover, to optimize performance and reduce the number of DBMS calls, modern ORM systems use local cache memory [7].

Entity Framework data access model programming approaches and Entity Framework features

We also consider the key features of the Entity Framework (EF) and the reasons for choosing this ORM library to use in this paper:

1. Cross-platform.
2. Modeling. EF creates an EDM (Entity Data Model) data model based on POCO (Plain Old CLR Object) objects with the properties of acquiring and setting various data types. This model is used when querying or storing object data in the database.
3. Requests EF allows you to use LINQ (in C # or VB.NET) to retrieve data from a database.
4. Track changes and save them to the database.
5.
6. Parallelism. EF uses optimistic concurrency by default to protect changes made by another user (stream) from being overwritten from the moment data is received from the database.
7. Transactions EF performs automatic transaction management when querying or saving data, and provides options for setting up transaction management.
8. Caching EF implements the first level of caching "out of the box" - thus, a repeated request to already read data will return data from the cache instead of re-accessing the database, thereby increasing performance.
9. Setting the context. EF allows you to modify the data model using attributes for da- ta annotation or the Fluent API to override the default conventions.
10. Support for various DBMSs. EF implements layers of data providers for most popular DBMSs (such as MS SQL Server, MySQL, SQLite, PostgreSQL, Oracle, DB2, etc.).

EDM Data Model

Consider the internal structure of the Entity Framework. Its main element is the EDM data model (from the English Entity Data Model), which describes the relationship between classes in the application and tables in the database [3, p. 20]. EDM stores in memory all metadata consisting of

three blocks:

1. A conceptual model that describes application classes and the relationships be- tween them.

The storage model, which describes the related tables located in the database. When using the "code first" approach, the storage model will be generated on the basis of the conceptual model, and when using the "DB first" approach, from the target database.

Mapping, which contains the correspondence scheme between the conceptual model and the storage model, i.e., between application classes and database tables.

All this information about the database structure (Storage model), about the data model (Conceptual model) and about their mutual display is contained in XML in a file with the extension .edmx.

EF performs CRUD operations using EDM - for example, it uses EDM to implement SQL queries based on LINQ expressions by executing Select, Insert, Update, Delete commands and converting the results obtained from the database into application objects and vice versa.

Feature Services Layer

In addition to the EDM, the Entity Framework also contains a few important components called layers, which we will consider later. According to Codd's fifth rule, "a relational database management system must support at least one relational language" [3]. Entity Framework offers two ways to access the DBMS: LINQ to Entities and Entity SQL.

1. LINQ to Entities is a LINQ extension (Language-Integrated Query) for creating queries to the Conceptual model (i.e., application class objects) in C # or VB.NET.

Entity SQL, according to [7], "is a repository-independent query language like SQL. Entity SQL allows you to query the entity data, presented either as objects or in tabular form. "This language is a bit more complicated than LINQ to Entities and its consideration is beyond the scope of this work.

The Object Services Layer is the most important component of the Entity Framework, which allows the user to use the programming language (LINQ to Entities or Entity SQL) to create database queries. This layer works with objects of the application classes, synchronizing them with the data in the database tables. In this layer, actions such as fixing the current state of objects and converting the data obtained from the data- base tables as a result of the query into objects of the application classes are per- formed.

Data Provider Layer

If the developer interacts directly with the object services layer using LINQ to Entities or Entity SQL, then further work on converting class data into data suitable for storage in the DBMS is performed by data provider layers.

Initially, a LINQ to Entities or Entity SQL query is received by the Entity Client Data Provider layer. Upon receiving the request, this layer converts it to SQL and transfers it to the ADO.NET Data Provider Layer (ADO.NET Data Provider), which is designed to directly access the DBMS using ADO.NET technology. At this point, queries created on LINQ to Entities or Entity SQL must be converted to SQL queries.

Description of test data set and test algorithm

It is generally accepted that the overhead of the data provider layer (binding of input and output parameters, preparing the request for execution, etc.) is small compared to the time of processing the request on the DBMS side. Indeed, the total query execution time includes the "slow" operations of the network and, especially, disk I / O. Also, the query execution time depends on computer performance. But since all tests were performed on the same machine, the database used was located locally, and the purpose of the tests is to determine the difference in performance, and not its absolute value, the above factors should not affect the result of the study.

s,not having special knowledge of working with DBMS, while OLAP-systems (English On-line Analytical Processing) are used for analytical purposes, the data in them are used only occasionally and often in large volumes and the need to develop special software APPENDIX to access them does not arise, t. To. OLAP-systems users are generally specialized. The selected OLTP scheme is characterized by a large number of short CRUD online transactions, and the main emphasis is on very fast request processing, maintaining data integrity in systems with multiple access and efficiency, as measured by the number of transactions per second. The OLTP database contains detailed and current data, and the scheme used to store transactional data is an entity model, which meets the classical requirements of the relational model.

The structure of the database "Adventure Works 2017" imitates the subject model of the activities of a fictitious company that produces and sells bicycles, accessories and spare parts for them. The database restored from the AdventureWorks2017.bak archive published in the official repository occupies 336 MB of disk space and has a complex structure - it contains 68 tables with complex relationships, united by be- longing to 5 schemes: Human Resources (frames), Person (individuals), Production (manufacturing), Purchasing (purchases), Sales (sales). Developed test applications access 22 database tables To evaluate the performance of the Entity Framework (EF), a set of tests was per- formed that simulated working with the database during the industrial operation of

the application, which was developed for the most common operations when interacting with the DBMS. Although the subject area of the database used is not important for testing purposes, to approximate the actual operating conditions of the database, the developed testing algorithm simulates the likely daily work with the database of each of the 17 sellers (sales representatives), consisting of the following CRUD operations:

1. (READ) Table 18 downloads complete information on all seller's orders placed on a randomly selected date (for which the seller has at least one order), including re- lated data from tables 1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22 (which confirms the complex structure of the relationships in the database).
2. (CREATE) 10 new customers are created in tables 8 and 16, including entering contact details in the related tables 1, 3, 4, 7, 9.
3. (CREATE) 10 new orders are rented with today's date in tables 18 with the creation of 3 to 15 line items for each order in table 17.
4. (UPDATE) Contact details of 10 customers in tables 7 and 9 are changed.
5. (UPDATE) The delivery date for today's 10 orders in table 18 changes.
6. (DELETE) 10 of today's orders are deleted from table 18 and the related data from table 17.

Thus, diverse CRUD queries are performed, namely, reading from a large number of related tables (step 1), creating new records, including related tables (steps 2 and 3), updating data in previously existing records (step 4), and in the newly created (step 5) and deleting records (step 6). The testing scheme described above is brief and de- scribes only the main steps, excluding operations to extract related information from the database.

Test applications are designed in such a way as to separately consider the time spent on each type of CRUD operations (create, read, update, delete). At the same time, the total time of the test is also measured, the excess of which over the sum of measurements by type of operation may indicate possible overhead costs.

To implement the EDM data access model in a test application using the Entity Framework, we used automatic model generation for 22 database tables using the Code First From Existing Database approach.

RESULTS

During testing, the test suite was sequentially launched (6 steps for each of the 17 sellers), both using ORM and without it (using only SQL), 10 measurements for each application. The results for each test measurement varied within acceptable limits - the difference between the minimum and maximum time did not exceed 11% for an application with "pure" SQL and 28% for an application with ORM, so the absence of statistical outliers makes it possible to use average time values for all 10

measurements. The unaccounted time in the form of the difference between the total operating time of the application and the amount of time spent on each type of CRUD operations was no more than 0.41% for all tests, and therefore it can be neglected.

Initial testing revealed a very significant difference in the performance of Entity Framework (EF) and "pure" SQL - the latter worked on average 37 times faster (see Fig. 1). This difference was due to a much slower execution of the create, update, and delete operations, while the read operations through EF, on the contrary, were faster by an average of 1.6 times, but the share of the reading time was negligibly small relative to the total test time (0.28% when using ORM), and therefore it did not have a big impact on the final result. In addition, an application using EF consumed approximately 2 times more RAM compared to using only SQL (maximum consumption of 74 MB versus 36 MB).

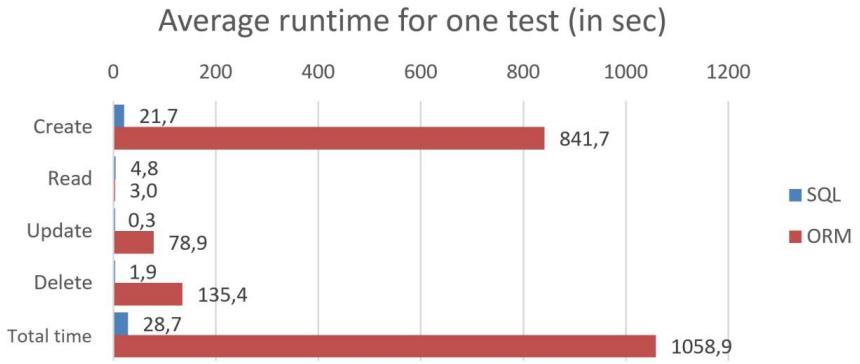

Fig. 1. Average runtime of one test by types of CRUD operations

Given such a large drop in performance when using the Entity Framework, a hypothesis was put forward about the insufficiently optimal way to use this technology and a number of techniques were applied to optimize calls to the database context from those described in [4], namely:

Using different database contexts for each testing step (except steps 5 and 6, for which the same data context was still used, because in these steps the program accesses a set of the same records). In order to use different contexts, the code of each step was wrapped in the using directive with the initialization of the context inside it, which, at the end of each step, provided an automatic call to the Dispose method, which cleans up the resources used.

Using different contexts gives us the opportunity to use the

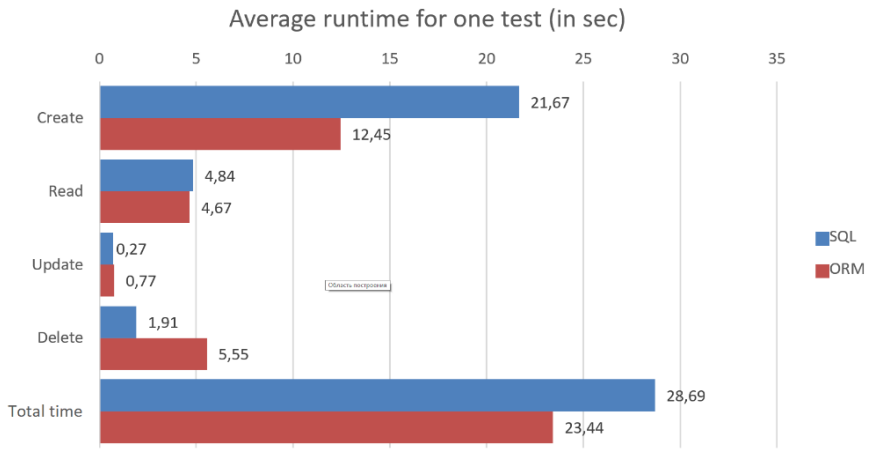

Fig. 2. Average runtime for one test after optimization

The results of the optimization and final testing allow us to conclude that the wide- spread opinion about the strong influence of the use of ORM systems on reducing the performance of object-oriented applications does not find confirmation with the com- petent use of modern ORM libraries in accordance with official documentation and recommendations of specialists with sufficient experience using ORM technologies. At the same time, ORM systems provide developers with a conceptual abstraction for comparing database records with program objects, which allows not only to create and maintain applications oriented to work with relational DBMS with less code, but also simplifies their development and support, automatically resolving many issues that arise when creating such a mapping, since developers can access database records without worrying about the details of SQL queries

CONCLUSION

In this paper, we examined the concept of object-relational mapping (ORM), the pre- requisites for its use, advantages and disadvantages, as well as a comparison with possible alternatives.

At the design level, the semantic gap between the object-oriented approach (OOP) in programming and relational DBMSs is an inevitable aspect of almost any application that interacts with data. Nevertheless, OOP in programming has long been a de facto standard, and at the same time, the prevalence of relational DBMSs, their effectiveness in querying large data arrays and other advantages make it necessary to find a compromise between the convenience of developing applications in an object- oriented style and limitations superimposed on the designed system using an RDBMS.

At the implementation level, there are several ways to implement the effective interaction of an object-oriented application with a relational database management system, which, however, are too time-consuming and often error-prone. To take ad- vantage of relational DBMSs along with the convenience of OOP, Object Relational Mapping (ORM) tools are used to provide a more productive and efficient development process.

Summing up the theoretical part of the research, we can say that ORM is a tool for solving the problem of semantic gap between relational and object data models, which forces you to describe the data structure twice - in the essence of relational DBMS (tables) and object-oriented programming (classes), introducing additional costs and difficulties in the process of developing applications that interact with RDBMS. ORM is used to simplify the process of saving objects to a relational database and retrieving them, while the ORM library itself takes care of converting data between two incompatible states.

However, in practice, the most common problem when using ORM libraries is a decrease in application performance compared to access to the database by executing SQL queries written by hand. A few studies by Russian and foreign authors indicate a significant drop in speed when working with databases through ORM, sometimes by 5 or more times. Since existing studies show a significant variation in the results, and there are no studies conducted using the latest version of Entity Framework 6.
Thus, the results of the optimization and final testing allow us to conclude that the widespread opinion about the strong influence of the use of ORM systems on reducing the performance of object-oriented applications does not find confirmation with the competent use of modern ORM libraries in accordance with official documentation and recommendations specialists with sufficient experience in using ORM technologies.

ACKNOWLEDGMENTS

REFERENCES

Ambler, S.W.: The Object-Relational Impedance Mismatch. Ambysoft Inc. (2010). http://www.agiledata.org/essays/impedanceMismatch.html, last accessed 2016/11/21.

Codd, E.F.: Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks. Research Report, IBM Research Division, RJ 599 (#12343), (1969).

Codd, E.F.: The Relational Model for Database Management. Addison-Wesley (1990).

Ritchie, C.: Relational Database Principles. Thomson Learning (2004).

Sumathi, S., Esakkirajan, S.: Fundamentals of Relational Database Management Systems (Studies in Computational Intelligence). Springer-Verlag, Berlin Heidelberd (2010).

DB-Engine Ranking. DB-Engines. https://db-engines.com/en/ranking, last accessed 2016/11/21.

Captain, F.A.: Six-Step Relational Database Design. (2013).