Pedro Santos Rodrigues

Bachelor in Computer Science

Accelerating SQL with Complex Visual Querying

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: Teresa Romão, Assistant Professor,
NOVA University of Lisbon

Co-advisers: Rui Nóbrega, Assistant Professor,
NOVA University of Lisbon

Tiago Simões, Principal Product Designer,
OutSystems

Examination Committee

Chair: Name of the committee chairperson
Rapporteurs: Name of a raporteur
Name of another raporteur
Members: Another member of the committee
Yet another member of the committee

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2020

Accelerating SQL with Complex Visual Querying

This document was created using the (pdf)LaTeX processor, based on the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Lorem ipsum.

Acknowledgements

The acknowledgements. You are free to write this section at your own will. However, usually it starts with the institutional acknowledgements (adviser, institution, grants, workmates, ...) and then comes the personal acknowledgements (friends, family, ...).

Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work.

Steve Jobs

Abstract

This dissertation addresses a usability improvement of a graphical user interface that allows query formulation without using textual query languages, such as SQL. This visual tool, called Aggregates, is provided on the OutSystems Low-Code Development Platform, to formulate data queries, through interaction and manipulation of visual components.

Since Aggregates do not support all the existing functionalities of SQL, the OutSystems Platform allows users to build queries using this textual query language. Nonetheless, by evaluating customers' SQL queries, it was revealed that a considerable subset of the queries written in SQL could have been formulated using the visual tool.

The users' interviews and the results of the SQL queries evaluation have foreseen that the cause of the reduced acceptance of the visual approach, could be the existing usability problems on the interface. Furthermore, the interface is inadequate to build more complex queries, which involve more entities and conditions.

Through an iterative design process, this dissertation includes the design, implementation, and evaluation of prototypes with different fidelity levels. The aim is to optimize the effectiveness and efficiency of the process where the user communicates to the system what data they intend to extract from the database. Moreover, the readability and comprehension improvement of the query visual representation is intended, reducing the time and the effort required to understand what data will be gathering from the database. The final goal is a functional prototype incorporated on the OutSystems Platform which accelerates the query formulation process without harming the learnability of the system.

Keywords: Visual Query Interfaces, Low-Code Development, User-Centered Design, Human-computer Interaction, Iterative Design, Database Querying

Resumo

Esta dissertação tem como objetivo melhorar a usabilidade de uma interface gráfica que permite consultar dados sem recorrer a linguagens de consulta textuais, tais como o *SQL*. A ferramenta visual abordada, denominada *Aggregates*, está inserida na Plataforma de Desenvolvimento *Low-Code OutSystems*, de modo a permitir a formulação de consultas a bases de dados, através da interação e manipulação de componentes visuais.

Tendo em conta que a interface gráfica disponibilizada não suporta todos os tipos de consultas suportadas pelo *SQL*, os utilizadores podem recorrer a esta linguagem textual para construir as suas pesquisas. No entanto, ao avaliar estas consultas criadas textualmente em *SQL*, por clientes da plataforma, percebeu-se que: um conjunto considerável de consultas foram construídas usando *SQL*, embora pudessem ter sido construídas usando a ferramenta visual disponibilizada.

Tanto as entrevistas dos utilizadores, como a análise das consultas construídas usando SQL, indicaram que a falta de aceitação do método visual de construção de consultas era causada por problemas de usabilidade na interface. Para além disso, quando as consultas de dados envolvem mais entidades e condições, a interface apresenta um pior desempenho.

Através de um processo de desenho iterativo, esta dissertação apresenta o desenho, implementação e avaliação de protótipos com diferentes níveis de fidelidade. O objetivo é otimizar a eficácia e a eficiência do processo utilizado pelo utilizador para comunicar ao sistema que dados pretende consultar da base de dados. Além disso, também se pretende melhorar a legibilidade da representação visual da consulta, de modo a diminuir o tempo e esforço necessário para compreender que dados pretendem ser extraídos da base de dados, através da respetiva consulta. O objetivo final é a criação de um protótipo funcional, incorporado na Plataforma OutSystems, que acelere o processo de criação de consultas de dados sem aumentar o nível de aprendizagem necessária para utilizar o sistema.

Palavras-chave: Interfaces Gráficas de Consulta de Dados, Desenvolvimento *Low-Code*, Desenho Centrado no Utilizador, Interação Pessoa-Máquina, Desenho Iterativo, Consulta de Bases de Dados

Contents

List of Figures

List of Tables

Acronyms

ANSI	American National Standards Institute
DBMS	Database Management System
DQL	Data Query Languages
HCI	Human-computer Interaction
IDE	Integrated Development Environment
ISO	International Organization for Standardization
IT	Information Technology
RDBMS	Relational Database Management System
SQL	Structured Query Language
UX	User Experience
VQI	Visual Query Interface
VQL	Visual Query Language
VQS	Visual Query System

Chapter 1

Introduction

Nowadays, database queries are required not only in computer systems areas but also in most sectors of professional environments or personal demands. This database information gathering claim needs should resort to actual technologies to optimize the time spent and reduce the errors of this query process since the most important is the obtaining of the intended information.

In the decades of the 1960s, Database Management Systems (DBMSs) arose, and later at 1970s new management systems that use relational models, designated as Relational Database Management Systems (RDBMSs). Moreover, the first Data Query Languagess (DQLs) appeared, like Structured Query Language (SQL) [5] which was considered by the American National Standards Institute (ANSI) and International Organization for Standardization (ISO) as the standard query language [10]. These technological evolutions have improved the effectiveness and efficiency of the querying process. However, to find more specific and complex information on databases a higher degree of DQL understanding was required. Thus, if from one side, the technological evolution and the digital transformation have optimized the data querying process, only a subset of people could use these powerful querying technologies.

VQSs, defined by Catarci, *et. al.* [3] as "systems for querying databases that use a visual representation to depict the domain of interest and express related requests", are used to mitigate some problems already referred. These systems use different visual representations and interaction strategies to make database queries, using a more intuitive visual approach, instead of using textual languages, which are more difficult to learn mainly for people without programming base knowledge.

Visual Query Systems (VQSs), defined by Catarci, *et. al.* [3] as "systems for querying databases that use a visual representation to depict the domain of interest and express related requests", are used to mitigate some problems already referred. These systems

use different visual representations and interaction strategies to make database queries, using a more intuitive visual approach, instead of using textual languages, which are more difficult to learn mainly for people without programming base knowledge. Besides, even if it is not mandatory to be considered a VQS, some systems have also data visualization features which can be useful to view the query result, or even the possibility to manage the database schema visually.

However, usually, these visual languages are associated as more useful to naive users, while textual languages are associated with more expert users. Conversely, some studies have revealed that visual languages might be convenient to the expert users too. For example, the comparison made by Catarci and Santucci [4] concludes that diagrammatic languages can reduce the error rate of the queries, in comparison with those made in a textual language by expert users. These results imply that even expert users make mistakes in simple queries (e.g., they may not remember the name of the tables or the precise syntax of some language expressions). Thus, it is important to analyze how those languages could be used to optimize the querying process not only to the users with a low experience level but also for highly experienced users.

1.1 Motivation

Low-Code Development is a recent development paradigm that seeks to reduce the time and effort spent on tasks that will not have a significant impact on the final product outcome. Just as high-level languages, APIs and third-party infrastructures allowed developers to be more productive and focus on the most valued sections of the software they produce. Low-code approaches follow this endeavor, using visual Integrated Development Environments (IDEs), connectors between components and lifecycle managers to employ an abstraction layer on the high-level languages, removing concerns of infrastructure or pattern reimplementation. In this way, one can focus on the tasks that truly accelerate the growth of the end product, achieving the desired goals with greater efficiency [60].

The goal of this dissertation is to analyze and improve the OutSystems Platform [54] data querying component to create relational database queries through drag and drop interactions and simple configurations beyond visual components and multiple drag and drop features. The platform aims to provide the application development environment where users with different development backgrounds can build, deploy and manage their applications, following good practices and using state-of-the-art technologies.

However, conversely to No-Code approaches, the development through low-code systems gives, many times, the possibility to use low-level code, written in textual languages, such as Java, .NET or SQL, in order to increase the extensibility and the power of low-code solutions [62]. In that way, an alternative to perform the requests that are not supported by the low-code visual approaches is also provided. Nonetheless, if the visual languages

of low-code platforms are more robust, responding more thoroughly to users' requirements, there is a diminished demand to resort to these textual programming languages which are high error-prone and have a worse learning curve, requiring also, on multiple situations, previous coding experience.

Furthermore, as mentioned by Amaral *et. al.* [11], web and mobile applications produced on OutSystems' technology, have proven an increase in quality. Also, it was concluded that low-code developers are 10.9 times more productive than the standard of Information Technology (IT) Industry, which does not use these rapid software solutions [19]. These results reinforce the importance of the improvement of the visual languages used on these platforms.

1.2 Problem Description

The OutSystems platform provides a Visual Query Language (VQL) that allows users to retrieve data from databases by simple processes. Besides, with this language, it is possible to perform some operations that are usually supported by textual DQLs, namely join, filter, sort, and group operations.

Currently, the OutSystems' solutions have been applied to digital transformation processes in multiple industries that deal with high quantities of data, in order to accelerate the application development processes, unlocking its value and growth.

To this extent, the already implemented querying tool is not able to deal with a set of scenarios, because it might not be accurate when the domain has a lot of tables involved or does not support essential advanced constructors. The following SQL functionalities are an example of the not supported operations: IN, NOT IN, EXISTS, NOT EXISTS, DISTINCT, UNION and the possibility to use subqueries.

Accordingly, the following questions were fundamental to analyze and explore the problem and its requirements:

- What query features are supported by the existing OutSystems visual query language?
- Why do OutSystems developers often use SQL to perform database queries?
- Why are some queries that can be built visually written through SQL instead?
- What are the main causes that users point out to use SQL?
- Who are the users more unsatisfied with the current provided visual approach to retrieve data? What are their reasons?

Under the above mentioned circumstances, the goal of this thesis is to design and evaluate a new and more powerful Visual Query Interface (VQI) to provide an improved User Experience (UX) that allows developers to formulate complex data queries intuitively and efficiently, without using SQL.

1.3 Research Questions

The main research question that is being addressed in this dissertation is:

> Can we enable OutSystems developers to easily do complex database queries without ever using SQL?

Regarding the main question and the diverse background of the system target users, it is important to research how can be developed a solution that covers the requirements of all user types with the best usability possible.

The following research questions focus on this usability trade-off which depends on the users and the system particularities:

Research Question 1: Does the current implementation have usability problems for the less experienced users?

Considering the VQI already implemented, the most complex queries have not been correctly covered by the tool. However, it is important to analyze also if novice users had similar problems or others that could have an impact on the task performing.

Research Question 2: Can experienced users take advantages in using the visual interface to build queries instead of SQL?

Since the users more experienced who know other textual query languages, such as SQL, can use SQL to perform the queries, are advantages for them in the usage of a VQI? What are the advantages and disadvantages of this type of approach for these users?

Research Question 3: Can expert users' UX be improved without reducing the system's learnability and satisfaction for less experienced users?

The usability attributes trade-off depends on the system's target users' expectations and requirements. Thus it is important to take into account if the development to improve the efficiency, effectiveness, and satisfaction of expert users does not harm the effectiveness and the learnability of the operations performed for the less experienced users.

1.4 Main Expected Contributions

This work aims to provide a set of contributions not only as scientific research but also as additional value to OutSystems. Thus, a summary of the main expected contributions for this project are presented below:

- A synthesization of the design concepts, including relevant interaction and conceptual models, usability definitions, guidelines and principles, and a description of the processes to evaluate the Human-computer Interaction (HCI) in the context of a software analysis;

- Provide a state-of-the-art about what are the most significant VQSs, presenting also a comparison of visual representation techniques and interaction strategies used, as well as other important features proper for this study;
- A description of the analysis made to identify what are the most impactful problems regarding the existing VQI, which includes user interviews and data analysis;
- Design and implementation of a new graphical user interface prototype that tries to improve the existing solution to visually build queries. This prototype expects to fix some predominant problems selected as the most relevant to solve;
- An usability evaluation of the prototype developed through the use of user tests;
- Increase the number of users that prefer using the visual query interface instead of SQL.

1.5 Document Structure

The remaining chapters of this thesis are organized as follows:

- Chapter 2 - Background: introduces some design and usability concepts to be used in this work. Besides, it is provided a context of the OutSystems Platform current progress, which explains the functionalities of the existing data querying tool;
- Chapter 3 - Related Work: analyses the users' interaction with database systems to improve the interfaces' suitability to the target users of the system. Also, approaches used by other systems to create interfaces that allow to visually build queries, are described and compared, detailing the interaction strategies used.
- Chapter 4 - Proposed Solution: explains the work made in the requirements analysis, including user's interviews and metrics gathering, and the decisions of what are the principal and impactful problems to be tackled. Moreover, the design approach and process adopted to develop the solution will be detailed.

Chapter 2

Background

This thesis aims to improve the interface that allows users to build queries in a more efficient and effective way. Therefore, HCI is a core subject of the work since such an interface can only be improved if its interaction and usability in the user perspective are studied.

Accordingly, this chapter will introduce key concepts of HCI, as well as, a brief contextualization of the OutSystems Platform that is indispensable for the comprehension and progression of this study.

2.1 Human-Computer Interaction

Although computer systems have been designed by humans, these two parts of HCI do not speak the same language. Nonetheless, these types of systems were created to support, in a transparent way, human tasks and requirements, forgiving careless mistakes [9]. Thus, HCI aims to study the relationship of users and computer systems, in the context of the users' desired tasks, in order to "unfold and reveal challenges and insights, and to instrument appropriate solutions for alleviating the current obstacles to the access and use of advanced information technologies" [25].

2.1.1 Main Concepts

The **Usability** of a system is one of the most important concepts in HCI, that can not be forgotten on the design process, since its attributes must be taken into account performing also a guidance function through all this process. This concept was standardized in ISO-9241 [45] as "extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use".

However, usability is not a single-dimensional property, being always associated with its attributes, that characterize the user accessibility when is using the system into five different points, such as referred by Nielsen [16]:

- **Learnability**: How easy is the learning process until a novice user (who has not used the system before) achieves a high-level of proficiency using the system [27]. The learnability is higher as the learning process is faster, and the user has to spend less effort to reach his goal. Also, it depends on the tutorials and training provided to users. A system that requires less training has higher learnability than a system that requires more training. The time that a novice user requires to perform some specific tasks can be used to measure learnability. Learnability can be improved using tips while a novice user explores the system doing his first tasks.

- **Efficiency**: Refers to the productivity level of a user who has already learned how to use the system. Efficiency can be measured analyzing the time that expert users spent to do specific tasks on the system. This attribute can be improved, for example, adding shortcuts to accelerate the interaction process.

- **Memorability**: Defines how easy is for a user, that was using a system before but did not use it for a time period, to do his desired tasks on the system. So it's related to how many times the user has not used the system and the time that the user needs to remember how the system works. Therefore, if a system has good memorability the user does not need much time to remember it, even if it has stopped using it for a long period of time. Memorability can be measured, for example, analyzing the interaction process of a user who has been away from the system, while he uses the system again. The use of visual components and metaphors with real-life objects helps, sometimes, the users in this process.

- **Errors**: A system not only must have a low error-rate but if an error occurs, the user should be able to recover from that. Since there are multiple types of errors with different severity levels, catastrophic errors should not occur. This attribute can be measured by the evaluation of the error-rate, taking into account the severity levels of the errors. Furthermore, if a system has errors, that can be reverted and does not have a negative impact on the final result, cannot be forgotten that these errors also harm the efficiency of the system.

- **Satisfaction**: The most subjective attribute of the usability that is related to the overall satisfaction of the user when uses the system. It could be measured by asking the users about the experience while they are using the system, always searching for subjective answers.

Nonetheless, as mentioned by Nielsen [16]: **"it is not always possible to achieve optimal scores for all usability attributes simultaneously"**. Thus, when a system is

designed it is necessary to prioritize what are the most important attributes for the users and the domain where the system will be used and applied. These trade-offs are one of the most challenging tasks of the design processes because it depends on user expectations and their backgrounds, as well as, the problem domain and what are the main focus of the system use. Accordingly, it is fundamental that the design process can focus on target users of the systems. Therefore, the main concepts, processes, and techniques for a design process centered on the users will be described.

2.1.2 User-centered Design

Before user-centered design principles and methodologies were adopted, the Waterfall model was commonly adopted as a software development process. This model comprises five sequential phases, from the requirements specification phase to the operation and maintenance phase, and has a good quality control since documentation and planning are a major concern of this methodology [2]. However, the stages of this model are not overlapping stages, so other development methodologies and philosophies arose to mitigate this problem and include the user on the design process, due to their impact on the usability of the system.

Consequently, it was necessary a new model that has the users included in the development process to verify, along with the development, if the approaches adopted are positive and what is the users' acceptability. Nielsen [15] reinforces this saying that "user interfaces should be designed iteratively in almost all cases because it is virtually impossible to design a user interface that has no usability problems from the start. Even the best usability experts cannot design perfect user interfaces in a single attempt, so a usability engineering lifecycle should be built around the concept of iteration".

The Spiral model of iterative design arose as an iteration through design, implementation and evaluation phases where the cost and accuracy increase on each iteration. The first iterations should use low-cost resources, like paper prototypes, and when the results are positive the accuracy should be incremented, changing to high-fidelity prototypes, such as computer prototypes [12].

2.1.2.1 User and Task Analysis

Regarding the concept of usability presented above and the importance of the users on the design process, it is important to define the users and their desired tasks of the system in order to find the best solution as possible to the usability attributes trade-offs. Just a good description of the users and the tasks of the system leads the designers to the best choice of what are the usability attributes most important for the system.

Accordingly, it is important to make a **User Analysis** to understand all users' characteristics that could have an impact on the acceptability of the system. The expected result of this analysis should be a set of structured information that characterize the users of

the system in terms of technological expertise, knowledge of the business domain, application experience, educational background, gender, and age, as other aspects that might be useful to comprehend, depending on the system's users and domain [8]. The more traditional process to gather this information is through questionnaires or interviews, but that can also be obtained by conducting market analyses or observational studies [16].

Furthermore, it is essential to enumerate and analyze the tasks the users should perform using the system. The **Task Analysis** process aims to aggregate information about the tasks that should be performed on the system, starting from the system's overall goals and break down these to obtain individual tasks [16]. Moreover, the goal of this analysing process is to obtain more structured information about: how the tasks are performed using the existing systems, what are the pre-conditions and the requirements of each task, why the users need to perform this tasks, and others that might be useful to characterize the tasks of the system.

The techniques used, to extract information to this analysis, aims at figuring out how the tasks should be done instead of how they would perform them. The idea is to resort to examples, as well as possible, in order to understand what type of strategies are used, what type of exceptions from their normal workflow is occurring, and other aspects that can be observed where the communication with users is on a concrete level [16]. In addition, Nielsen [16] points out that "The users' model of the task should also be identified, since it can be used as a source for metaphors for the user interface", which reinforces that these dialogues with users to obtain analysis content, can be useful also to find relevant solutions to latter design process phases.

Therefore, the outcome of this analysis should contain a list of the entire tasks that users what to perform in the system, the information that is required to complete them, the steps and the dependencies between tasks, all the outputs that must be generated, and how is the communication process between the users associated with the system's tasks [16].

2.1.2.2 Sketching and Prototyping

After the user and task analysis process, designers must start sketching and prototyping ideas and approaches, in order to think about how can they solve the problems. Nevertheless, this phase of the design process should start with sketching techniques, as these are not only a good and inexpensive starting point to communicate ideas, but also help to develop structure and enrich the reasoning, leading to the perception of other details as well as of other approaches to solving the related problems.

While sketching techniques are more plentiful, and have a low detail level, being mainly based on suggesting and exploring, rather than retrieving results, the prototyping phases, have more refinement approaches and are used to test the design choices made.

However, prototypes may have different thoroughness degrees, presenting different advantages and disadvantages. Thus, it is important to start the prototyping process with

low fidelity prototypes, as paper prototypes, since the objective of these is to evaluate the conceptual model (if the users understand the system), the functionalities presented, the navigation, the screen components distribution, and the terminology used. After the evaluation of these prototypes presents good results, high fidelity prototypes should be used, like computer prototypes. There are a set of available tools to assist in the building process of these prototypes, such as Balsamiq [29] and Mockingbird [52]. These different prototype types can detect different issues when tested with users, so it is very important to test prototypes with different granularity levels.

Furthermore, there is another relevant aspect of the prototype designing, that is the scope definition of the prototype. It is important to define what features the prototype undertakes and what is the inherent detail level. Nielsen [16] describes this as two dimensions of prototyping: horizontal prototyping and vertical prototyping, as demonstrated in Figure 2.1. A vertical prototype is characterized as a prototype to test a restricted part of the system but with real users and circumstances. A horizontal prototype is presented as suitable for test the entire system but in a less realistic approach.

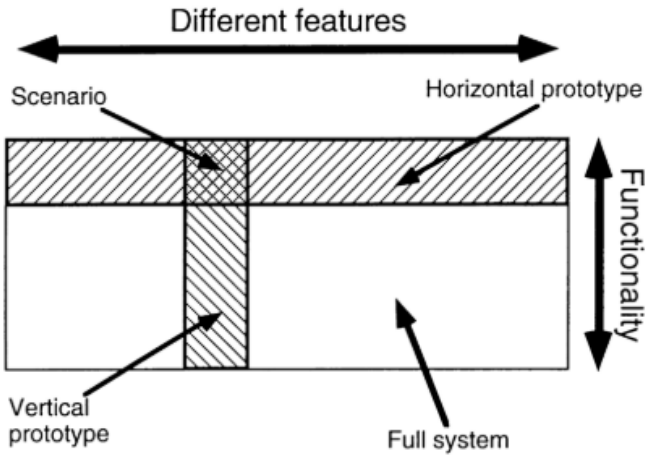


Figure 2.1: The two dimensions of prototyping: Horizontal prototyping keeps the features but eliminates depth of functionality, and vertical prototyping gives full functionality for a few features (source: Nielsen [16])

Finally, regarding the methodology about how to use the prototypes built, Dix et al. [9] refer that are three main approaches:

- **Throw-away**: After the prototype is built and tested, it is used on the final system development, but after that, the prototype is discarded and the rest of the design process continues without relying on the prototype previously built;

- **Incremental**: First, the system is separated into different parts, and each part is built, one at a time. So the prototypes are developed separately, regarding its correspondent part, and finally are combined to build the final system;

- **Evolutionary**: Contrary to the throw-away approach, the prototypes developed are used as the basis for the next iteration of the design.

2.1.2.3 Evaluation Techniques

The evaluation phases are crucial in the design process, since they allow designers to understand the systems' specific problems and the impact of the interfaces on users. So, the expected outcome of these processes is a list of usability issues ordered by priority level, referring to what usability principles and guidelines are not being accomplished and what solutions can be applied to solve the problem [16].

However, there are different approaches to evaluate interfaces. First, one important topic that distinguishes two types of techniques is who performs the evaluation. There are techniques where only the designers and specialists are involved in the process and are another type of evaluation where users participate in [9]. Thus, there are two types of evaluation: evaluation through expert analysis and evaluation through user participation.

Evaluation through expert analysis

In this type of evaluation, designers, or other specialists, evaluate the system, supporting their analysis on cognitive principles, to preview usability problems likely to occur. Moreover, this evaluation not only is cheaper because it does not involve users, as it also can be applied to any phase of the design process, from the design specification to the high fidelity prototypes [9].

Regarding the approach presented above, these two methods are one of the most used:

- **Heuristic Evaluation** [17]: The system is thoroughly analyzed in order to find problems that do not follow importantly and recognized usability heuristics. After a problem has been detected, it should be reported, not only with a description of the problem but also the indication of the heuristics that have not been accomplished, the severity level of the problem and possible solutions to solve it [16].

- **Cognitive Walkthrough** [20]: This method uses a sequence of actions as the principal resource to guide the evaluation process. For each action, the evaluator tries to understand if all steps are clear and visible, as well as, if the system gives clear feedback, confirming if the action has been completed. Usually, the main focus of this method is to analyze the learnability of the system. Mainly, to understand if the system provides a good learning mechanism through exploration, rather than using manuals, training or other types of *a priori* learning processes [9].

A study made by Desurvire *et al.* [7] concluded that Heuristic Evaluation made by specialists is better to predict some problems before the user testing process than Cognitive Walkthrough. The reason pointed out for this result is that heuristic evaluation can often help to remind the designers of problems since this method analyses more dimensions of the system than Cognitive Walkthrough.

Evaluation through user participation

Although there exist methods that do not need the users to evaluate the system usability, it is difficult to predict all the behaviors of the users when they interact with a system. Therefore, there are multiple methods to make usability tests with people that are the system's target. Some methods of user testing which can be resourceful on the context of this work, are the following, respecting the terminology used by Dix *et al.* [9]:

- **Observational Methods**: the main principle of these methods is observing users using the system to conclude important usability information about it. Usually, it is requested to the user to 'thinks aloud' since it might be possible to obtain more insights that can be useful, not only to understand why the user might have made an error, but also it can be a good strategy to find starting points for other possible solutions. Moreover, although usually, these tests have a set of predetermined tasks, since it is easier to find the user reaction to the system part the need to be tested, also can be executed tests only by evaluating the normal tasks of the users on their work;

- **Experimental Methods**: starting from a properly defined test hypothesis, it is selected a set of users to perform an experimental test to verify if the test hypothesis proves to be true or not. Thus, it is necessary to define previously all the experimental environment, which includes: the test hypothesis that wants to be verified, the users that will perform the test, and the independent and dependent variables. The dependent variables are the ones that express the result of the test in function of the independent variables, such as the task execution time or the number of the errors that occurred. The independent variables are chosen by the test designer to produce different conditions for comparison. Examples of independent variables can be the size of an interface component or the use of an interaction technique. This method is very useful to verify through a test hypothesis which of the possible solutions presented (independent variables) have a better performance for the intended application context;

- **Query Methods**: these methods focus on what the users think about the system, collecting information from interviews or questionnaires to analyze their opinion. One advantage of this method is that it might reveal issues not observed previously complained before, but contrary a lot of cases are not tested, since in the interaction field, many times, the user only finds a problem when it occurs for the first time. So it is not possible to extract concrete information from users that never have passed for this situation.

Finally, independently of the evaluation process adopted (through expert analysis or user participation), severity ratings should be attributed to the problems identified in order to define the main priorities and understand which might have a larger impact

on the system acceptability. However, although these levels should be attributed by specialists, if they use not only cognitive principles, but also use the results observed from the user testing phase to sustain the classification of the problems detected, the result can be more accurate.

Besides, the Figure 2.2, extracted from [16], displays two influential factors that should be taken into account to attribute a severity level to a problem: how many users are experiencing this problem and what is the impact level on the user.

		Proportion of users experiencing the problem	
		Few	*Many*
Impact of problem on the users who experience it	*Small*	Low severity	Medium severity
	Large	Medium severity	High severity

Figure 2.2: Severity Levels of the Problems based on their impact on the users (source: Nielsen [16])

2.1.2.4 Errors Classification

The errors that occurred when a user interacts with a system are excellent indicators for designers because the understanding of the reason that led to error situations is a good strategy to classify them. Therefore, errors can be classified as slips and mistakes, as will be presented below according to Dix *et al.* [9]:

- **Slips**: in these types of errors, the user knows how to do the intended task on the system, however, he presses a wrong button or closes one window accidentally. So, he understands the action, but a misaction does not allow that he reaches his goal;

- **Mistakes**: these errors occur when the user does not understand the system, formulating a wrong goal. An example of a mistake is when the user does not understand the action associated with an icon, performing a not intended action.

Therefore, the strategy to mitigate the problems associated with these two types of errors could be different, as mentioned by Dix *et. al.* [9]: "Slips may be corrected by, for instance, better screen design, perhaps putting more space between buttons. However, mistakes need users to have a better understanding of the systems, so it will require far more radical redesign or improved training, perhaps a totally different metaphor for use."

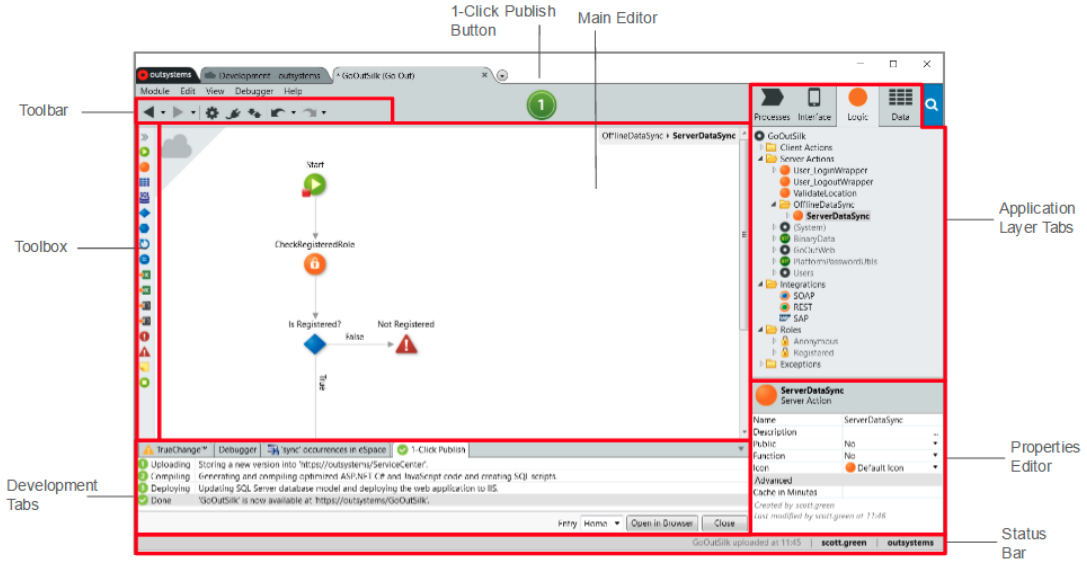


Figure 2.3: Main areas of Service Studio (source: OutSystems[55])

2.2 OutSystems Background

The OutSystems Platform has the mission of simplifying and accelerating the development and management of digital enterprise solutions, no matter the dimension and domain of the applications. It covers the entire development lifecycle which aims to promote rapid development and integration, to facilitate and speed up the deployment stages, to keep track of the status and health of the applications produced, and to expedite the management of daily operations and configurations on the final products [53].

2.2.1 Visual Development Environment

Service Studio is the low-code development environment of the platform, which allows the developers to create complete applications using visual elements to perform drag and drop actions. Figure 2.3 presents an overview of the IDE, which highlights the different areas of the workspace.

Using the widgets and icons provided in the toolbox, the main area is dedicated to designing the applications' interface and logic. So, in the main area, there are visual elements, which can be set using the properties editor, placed on the bottom right corner of the screen.

Also, it includes other sections whose main purpose is not the product development, but are related to key actions of the software development process. Therefore, on the window's top, there is a toolbar which has shortcuts to some of the most common operations, and a green circle button, denominated "1-Click Publish button", to start the automated deployment process provided. Besides, the bottom of the window is dedicated not only to the presentation of messages, errors, and warnings but also to debugging the application.

Since the development environment can be used to develop a complete full-stack application, the elements, which can be manipulated in the Service Studio, can be related to different parts of the application. The Application Layer Tabs, which contain a tree view of its elements, are the following:

In the **Processes** tab is possible to create and manage business processes of the systems through a flow that can be composed by human or automatic activities, time waits, conditional decisions and indications to execute processes. Also, this section can be used to configure the timers of the application. Then, it is possible to indicate when a timer should start, what is it period and what action should be performed when it is triggered.

The **Interface** tab can be used to manage the components related to the visual interface of the final application. It is possible to observe all applications' screens, as well as, the variables and the actions related to each one. Moreover, flows between the various screens can be also defined. If one screen or action is selected, it will be possible to add new visual components to the interface or assign new elements to the action flow in order to define all the client-side logic of the screen.

The **Logic** tab is where the core logic of the application could be defined. It includes not only the server actions of the application but also the exceptions specification, the existing user's roles and also the integration with external services. Although there is a data section on the application layer tabs, which will be described below, the actions which require data querying are managed in this section.

In the **Data** section is covered the database modeling, making possible the creation of diagrams to represent the schema of the database. Moreover, in the tree view of this tab is allowed to establish new entities and static entities in order to define the data model of the application.

2.2.2 Visual Data Querying

The main topic of this work is the improvement of the visual data querying process on the low-code development of applications, using OutSystems. Consequently, the headway of visual querying components of the platform is an essential factor to properly understand the entire project. Regarding the last version of the OutSystems platform, the actual visual data querying tool, which is the starting point of this study, will be described.

2.2.2.1 Previous Work

Since 2002, which is the release date of the first OutSystems low-code development environment, the Hub Edition 1.0, allows two manners to create database queries [61]:

- **Simple Queries**: visual query builder that allows the creation of some less complex queries, interacting with a graphical user interface;

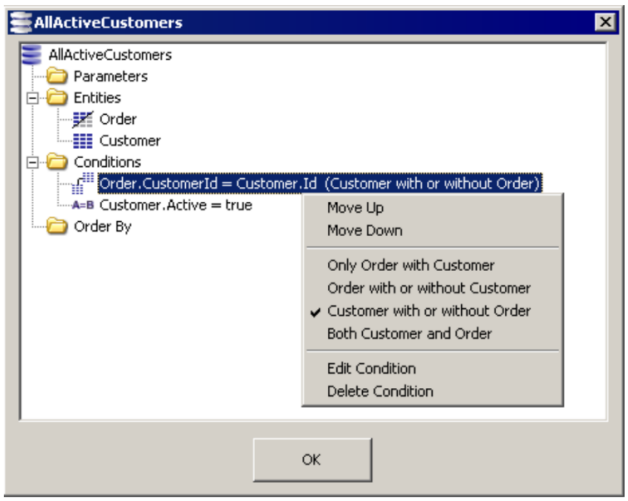


Figure 2.4: Simple Query example in Hub Edition 2.0 (source: OutSystems [56])

- **Advanced Queries**: feature to specify queries textually, using a language based on SQL, but includes some extra syntax to reference variables used of the application development;

Then, since the first versions of the IDE, it is provided two ways to build queries. The first uses the visual language, which does not need so many learning requirements but also diminishes the necessity to remember the entities' name or the language syntax. The second provided relies on SQL, which is the standardized textual query language, known for the developers' majority.

The first simple queries versions have accelerated the query building process finding automatically the relationships between the entities chosen. The main idea is that the developer only needs to select the entities intended and the respective join conditions would appear automatically in the query view interface. After this, it will be possible to change the join type, as well as, to add, edit and remove filtering or sorting conditions. Figure 2.4 presents a simple query example in Hub Edition 2.0 (2003) when the developer was changing the join type to an Outer Join.

This visual querying approach continued in the next versions, adding some minor improvements, such as the support to structures in order to store temporary information without changing the entity definition [57], and the inclusion of a properties pane to view and change the properties of all query elements in a single window [58].

However, at the launch of the OutSystems Platform 9 (2013), an entirely new way to manipulate data and express database queries has been released. These new components of the system, called **Aggregates**, have replaced Simple Queries, promoting a new interaction strategy to query databases, where the main focus is the data, instead of query design. Also, new features have been introduced to improve the expressiveness of the VQL, namely grouping functions and the ability to add calculated columns easier.

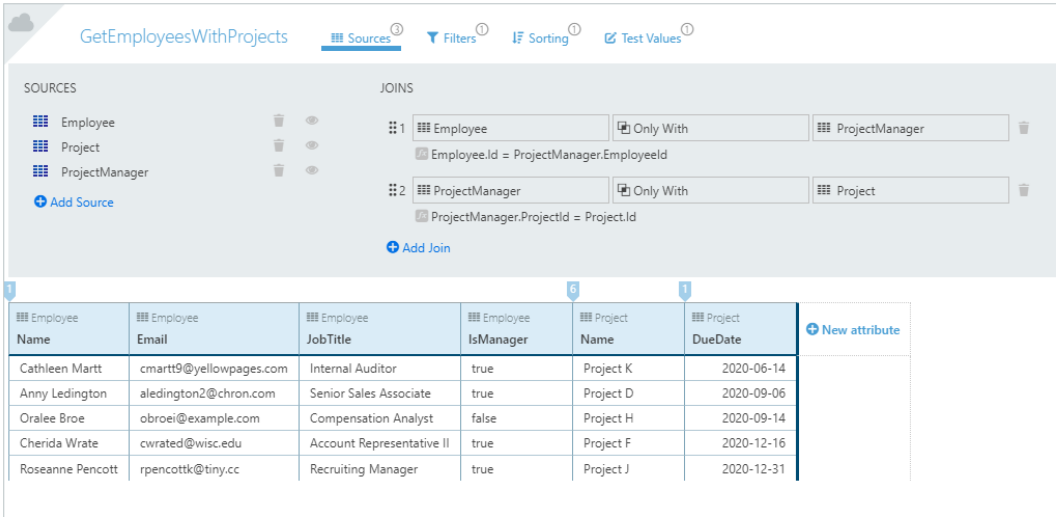


Figure 2.5: Aggregate Example

2.2.2.2 Current Progress

Since the implementation of Aggregates, the query process without textual languages is more visual and more focused on the query outcome. Aggregates can be used to query data from the server or mobile local storage, and can be created in the following ways:

- If someone is designing the user interface and wants to present some data gathered from the database (server or local storage), he can right-click on the screen where data will be displayed and select "Fetch Data from Database";
- When an action flow is designed, there is an Aggregate icon in the toolbox that can be dragged and dropped to the main area;

When the Aggregate is created, the first step is the data source selection in order to specify what entities should be included in the query. Then, the developer should click in the main area and choose the entities he wants or drag and drop the entities to add them to the query. When an entity is added, all its attributes are automatically included in Aggregate. Since an Aggregate can include one or more entities, added on the beginning or later, if it has more than one, it will be analyzed to verify if there are relationships between them. Unless there is a relationship that links them, the Aggregate will request the condition to join them. Otherwise, the entities will be included automatically using an inner join.

Hereupon, the Aggregate has already been created and its data source specified, so the visual querying process can be started, in a progressive way, seeing at the same time the query output. Figure 2.5 demonstrates an Aggregate on the referred state, to be possible to understand the structure of the interface.

First, this component of the OutSystems Platform presents two main capabilities, represented visually in two distinct areas: the query design area, and the viewer of the

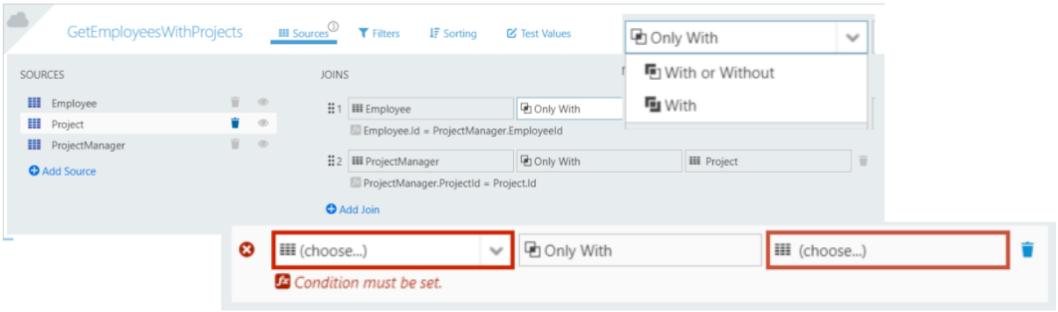


Figure 2.6: Aggregate - Defining Sources

query result. The former, located at the top of the window, is composed of a set of four tabs, where each selection action changes the grey area to the respective form-based interface. The latter is a table-based interface, which is similar to spreadsheets applications, such as Microsoft Excel [47] of Google Sheets [44] and its principal aim is to provide a direct and visual approach to show the query output.

Focusing on the query design area, the sources tab, illustrated in Figure 2.6, can be used to add, change or remove the entities of the quey, defining also, the join types between them. There are three join types available: "only with", "with or without", and "with". The respective joins in SQL are: "inner join", "left join", and "full outer join". Additionally, each one appears with a visual representation similar to Venn Diagrams [14] to be easier to identify which data is selected on each join type. Moreover, it is possible to edit the join condition textually, using the platform expression editor.

The filtering tab can be used to apply filters in the query likewise the WHERE statements in SQL. These filters can be defined through boolean conditions inserted in the expression editor. The conditions are specified textually with the assistance of some auto-completes and shortcuts available in a tree view.

The sort conditions can be added in the sorting tab choosing the "add sort"or the "add dynamic sort"options available. The difference between them is that dynamic sort relies on a variable of the system, contrary to the other that depends only on an entity attribute. To add a sort, the user has to select what entity wants to sort and specify what are the sort criteria, for example, ascending or descending. Moreover, more than one sort can be inserted and they can be ordered to establish the priorities between them.

Finally, the last tab has a different behavior when compared with the rest of the options available in this interface area. The main goal of this feature is the testing of query output when concrete values are assigned to the variables referred to in the query. Thus, it does not contribute to the query design in the same way as the other tabs, presenting only a test purpose in the context of the query result visualization.

Figure 2.7 presents a usage example of the last functionalities mentioned, to create a query to filter and sort dates, regarding the value of a variable.

On the other side, the area dedicated to viewing the query output provides also some

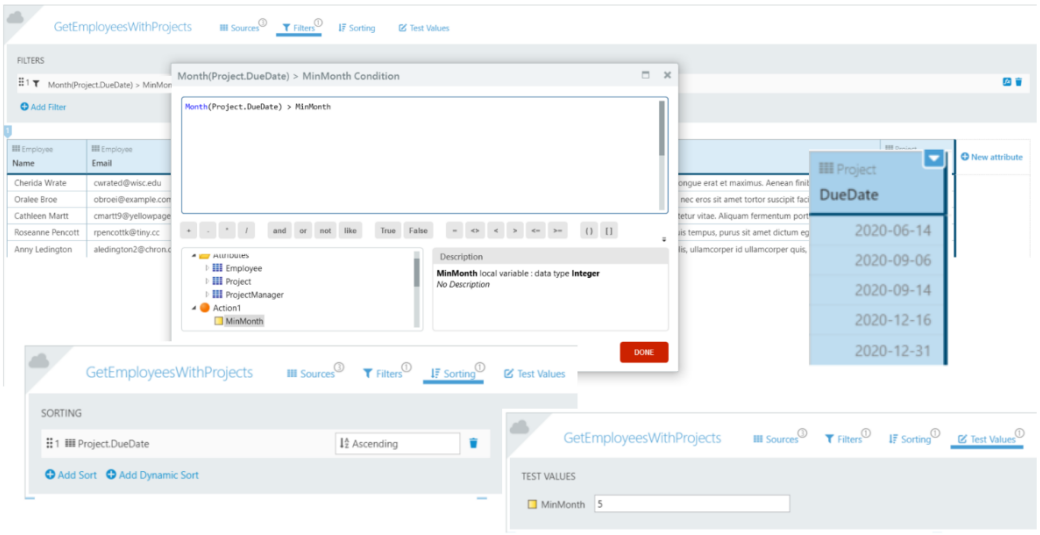


Figure 2.7: Aggregates - Filtering, Sorting and Test Values in an example of querying DueDates after a month indicated in a variable

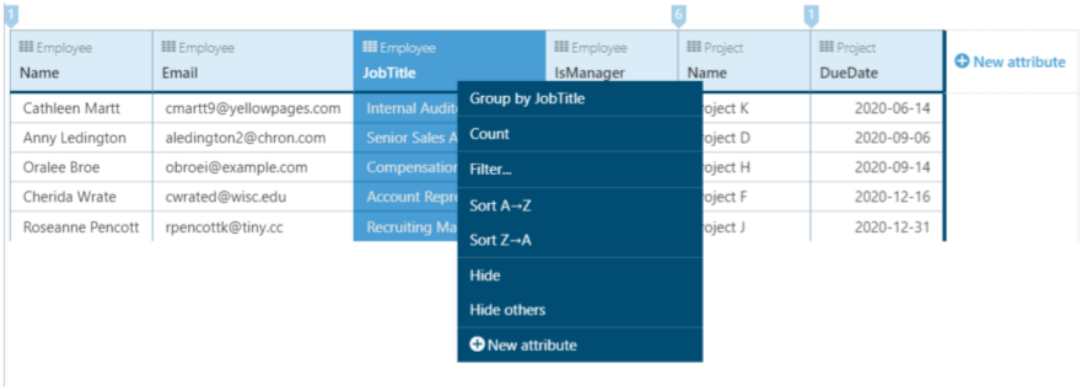


Figure 2.8: Query Design functions while interacting with Query Result

functionalities to design the query while the user is interacting and exploring the query result. Therefore, as represented by an example in Figure 2.8, the user can change the query when he performs a right-click on a column or when clicks on the new attribute. The only options in the list presented that does not change the query are the hide options since they just change the result in the presentation layer. So, if the user hides a set of columns, he will not see them in the result table, however, the query did not change. Besides, the user can add other attributes based on the existing ones, so Figure 2.9 shows an example of this functionality.

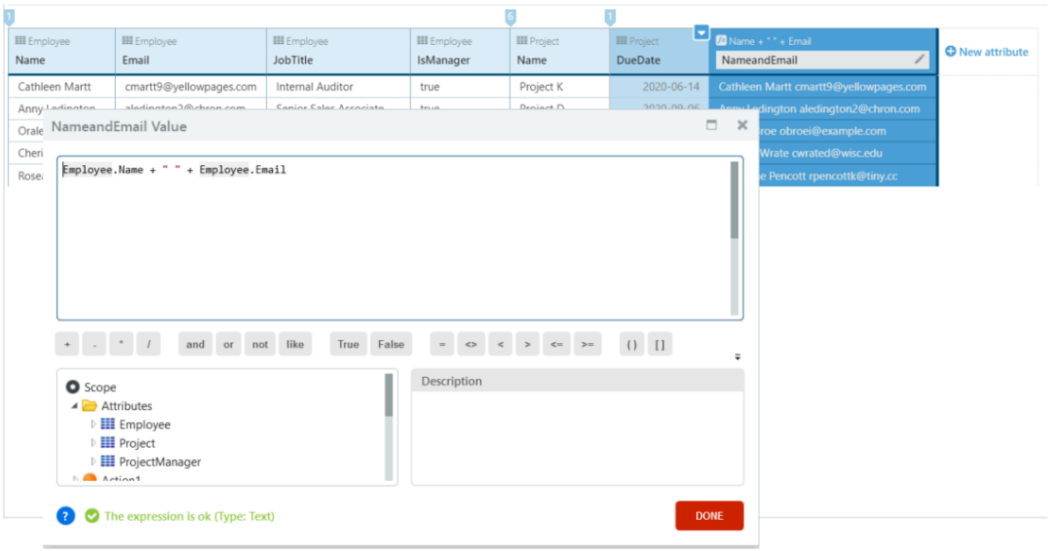


Figure 2.9: Calculated Attribute Insertion

Chapter 3

Related Work

Since the users are a fundamental component of the study, their relationship with data and what are their expectations of these query systems will be discussed. Moreover, a set of technologies and techniques will be described and compared. This information can turn to be useful to explore solution, and understand different points of view to manage problems, regarding the project scope.

3.1 Query Conceptual Models

The perception of the user's conceptual model is important to understand how the user reason while interacting with the system the perform the intended actions. A query is built to gather data. To transmit what is the intended data, the user needs to think about how it could express the data required in the query. The understanding of the user's conceptual model could be useful to remove the existing gap between what the user wants to query from the database and what system register that the user wants to retrieve.

Some studies have analyzed this reasoning process of the users when they were writing queries. Siau *et al.* [23] have referred that "The semantics communicated through the interface can be classified according to abstraction levels, such as the conceptual and logical levels". Also, there is one more level, the physical level, where is considered the system details, such as physical storage and access structures. Since the physical level is low level, usually, the conceptual and logical levels are most used. The logical level takes into account abstract structures for data and operations, and the conceptual level uses real-world objects and concepts to communicate. Through an empirical method of evaluation, the conceptual level has revealed a higher accuracy. Also, this abstraction level makes the users more confident in their answers than the physical level or logical levels.

Moreover, the time that users need to design the queries is reduced using conceptual levels [23].

Reisner [21] provided a model of query writing from the reading of the query intention in an English statement to the query writing in SQL (Figure 3.1a). After understanding what data is required, the user applies two parallel steps. In the **Template Generation** phase, the user formulates a template identifying the SQL keywords necessary, such as SELECT, FROM, and WHERE. In the other step, called **Lexical Transformation**, the user identifies the name of the tables and columns involved. Finally, the results of these two steps are combined in the last step, denominated **Insertion**, in order to produce the final query [21]. The recall of table and column names represents a significant use of long-term memory, being a concern that should be taken into account [24].

In the same field, Ogden [18] presented a three-stage cognitive model of the query process (Figure 3.1b):

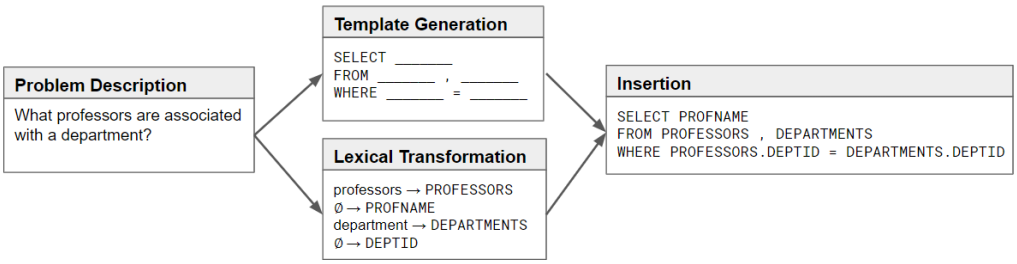
1. **Query Formulation**: according to the existing data, the user specifies, in natural language, what data is required;

2. **Query Translation**: regarding the existing data model, the operations and relations necessary are defined, in order to adapt the natural language request to the pragmatics of the intended query;

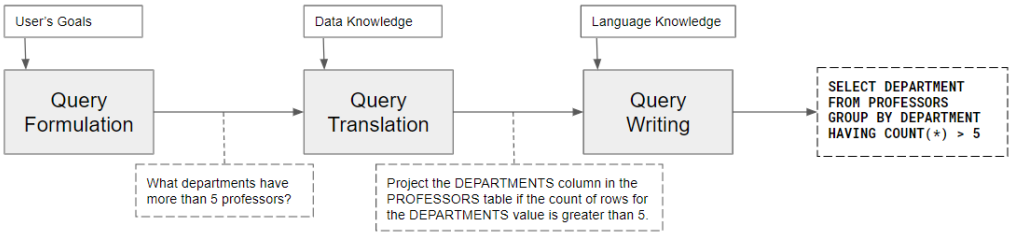
3. **Query Writing**: the information of the previous steps is used to built the query, using the syntactic and the semantics of the query language.

Comparing the two previous models, the Lexical Transformation phase is present in the Query Translation phase of the latter model, as well as, Template Generation and insertion are part of the Query Writing phase [6]. Moreover, although the query writing and comprehension are the focus of this work, it was verified in a comparison between three different models (relational model, extended-entity-relationship model, and object-oriented model) that the data model influence the query writing and comprehension [6].

The query comprehension is one of the important concerns of this work, since it is important to consider if the query representation, no matter if it is visual or textual, indicates clearly what data will be gathered. Chan *et al.* [6] have postulated that the query comprehension could be covered by the reverse of the stages included in the Ogden Model. First, the user should identify the data structure and operations, to translate them, in the next step, to the natural language. After this, the user needs to read and understand what data gathering is intended. Moreover, in this evaluation, it was concluded that although data modeling influence query writing, it does not influence the query comprehension. The explanation is provided by the authors: "Both query writing and comprehension require an understanding of the query language syntax. This is a component not needed in the data modeling task."



(a) Reisner Model (adapted: Reisner [21])



(b) Ogden Model (adapted: Ogden [18])

Figure 3.1: Models of Query Writing Process.

The experience with the data involved is another factor that influences the query comprehension. If novice users do not understand the data involved, they cannot validate if the query result is correct. This situation was analyzed by Robb *et al.* [22] that were distinguished the query process between new users and experienced users (Figure 3.2). Besides, they concluded that if the novice users were alerted to the details of the data queried, the query effectiveness will increase.

3.2 Query Formulation Problems

Since the goal of this work is the improvement of an interface that allows its user to build queries, it is important to summarize a set of significant problems that usually occur in query formulation. The problems that will be presented are related to SQL queries. However, as the visual tool of this work aims to substitute some functionalities of SQL, the comprehension of the interaction problems that exist in the textual language can be considered and mitigated in the development of the new interface.

Lu et al. [13] have evaluated the SQL usage in a diverse population, which includes people of different enterprise areas with different levels of experience in the database systems domain. The authors concluded that the comprehension of the queries is difficult, as well as the logical errors are difficult to detect. Moreover, the joins and aggregation functions are the other problems pointed out.

The query errors could be syntactic or semantic. The **Syntactic Errors** are related to the grammar rules of the language and are detected by the compiler. Therefore, the impact of these errors is reduced since the user can see that the query is incorrect through the compiler alert. The **Semantic Errors** are a major concern because they occur when

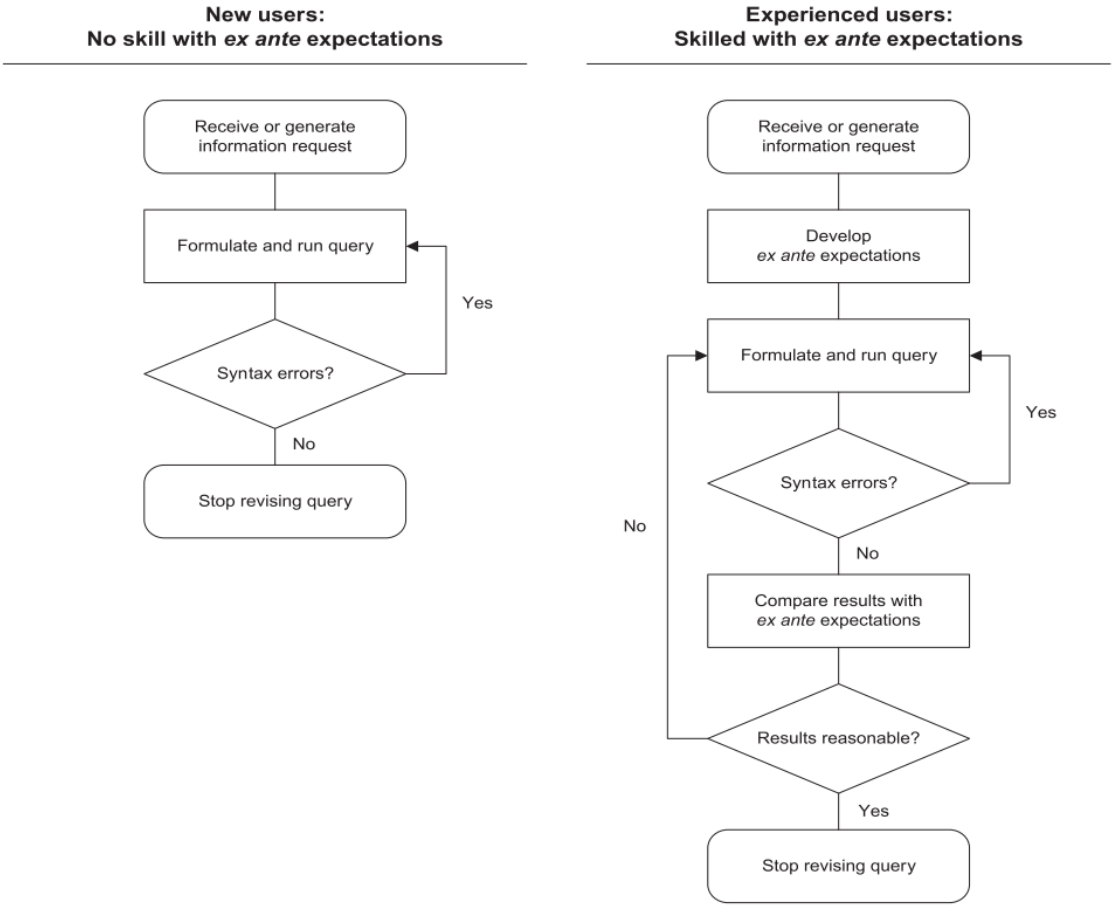


Figure 3.2: New and experienced users' query processes (source: Robb *et al.* [22]).

the returned information is not intended by the user, even if the query does not have compilation errors [24]. These errors could affect the correctness of the results.

In SQL, the join clauses are used when is necessary to merge data from different tables in one column in order to specify which data of each table will be considered. Several studies have demonstrated that the indication of the join clause is one of the most representative semantic errors [1, 13]. Smeller [24] has studied what are the cognitive causes that lead the user to forget the join clauses:

- **Working memory overload**: if the user needs to recall the table and column names, and the conditions necessary after the identification of the join's requirement, the required join clauses could be forgotten in this period;

- **Absence of the clue**: when the statement that explains what data is required do not present clues for the join necessity;

- **Procedural fixedness**: when a query that only extracts data from one table is reused for another that requires the join clauses but this join is forgotten;

- **Ignorance**: when the user does not know how to merge the tables and specify the join clauses.

The cognitive causes of the problems are important to develop interfaces that could mitigate the existing problems in the query formulation. For instance, if the join clues are provided in the interface, the user does not need to remember them. This approach, which follows one of the usability heuristics presented by Nielsen [16], minimizes the user memory load.

A study that evaluated novice programmers' semantic mistakes concluded also that omissions are the principal semantic error, mainly in the WHERE clause [1]. Besides, the authors have referred also the problem of working memory overload: "This error may occur when the capacity of a student's working memory is surpassed".

Another study has analyzed a large dataset of queries composed by university students enrolled in an introductory database course. However, this study is more extensive and includes a list of the principal errors committed by the students [26]. Continuing the focus on the semantic errors, the authors have pointed out the following error categories: inconsistent expressions, inconsistent joins, joins omission, duplicate rows, redundant column outputs [26].

3.3 Visual Query Composition

The interfaces to build queries resort to visual representations to communicate with the user. Catarci *et al.* [3] presented an interesting classification according to the visual formalism which the interface is based on:

- **Form-based**: based on forms, which can be seen as a rectangular grid of other components (subforms, groups of cells, a combination of cells, etc.) that group objects in a named collection regarding its structure. Forms and tables are similar, but contrary to the tables, forms allow nesting. Thus, forms can be seen as a generalization of tables. In this approach, the relationships can be represented among cells, cells subsets, or even the overall set, providing to the user three information levels;

- **Diagram-based**: usage of graphical representations, such as graphs, charts, and diagrams to better transmit the relationships among data. The aim is to use visual representations to help the understanding of the relationships between concepts which are represented by textual labels;

- **Icon-based**: as the opposite of the diagram-based, this type of interface tries to facilitate the understanding of the concepts instead of relationships. So, Icons are used, which are visual segmented objects to transmit a message or information, using analogies and metaphors with the real-world objects, or even conventions that are used to express no tangible objects, as computer processes;

Table 3.1: Query Language Requisites

Specification	Description	SQL Indication
Data Source	Entities and attributes which will be presented in the query	Using SELECT and FROM statements
Merge Type	Define how will be merged attributes of different entities	Using JOIN clauses
Filtering Criteria	Criteria that can be used to filter records, presenting in the result only those that fulfil a set of conditions	Using WHERE or HAVING clauses
Sorting Criteria	Define what are the criteria to sort the records of the result	Using ORDER BY
Aggregation Functons	Group a set of records by comparison or using mathematical functions	Using GROUP BY statements or SQL functions, such as MIN, MAX, COUNT, AVG and SUM
Calculated Attributes	Attributes added, based on existing ones	Using SELECT statement
Distinct Values	If only different values will be considered in the result (removing duplicated values)	Using SELECT DISTINCT statement
Unions	Combine the result of two different queries	Using UNION operator
Subqueries	Defining a query that uses other queries, for example, to filter the result	Nesting SELECT statements

- **Hybrid**: these approaches can combine the previous visual formalisms in order to select the best combination of advantages to the application usage domain.

In order to compare different interfaces, it is essential to analyze what a query language has to support to build the query. Accordingly, Table 3.1 presents the query creation required specifications, comparing them with the respective indication in SQL.

Nevertheless, there are two relevant aspects, according to the last requisites presented: the interaction process to indicate the query specifications, the overview of what data wants to be retrieved using the current query. Both are fundamental since a good visual query language aims to simplify not only, the query formulation process but also, the query readability, promoting an efficient and effective recognition of what are the desired data.

Data Source:

Chartio [31] has two components to query databases visually: using the Data Explorer [32] or using the new Visual SQL [34]. Regarding the data source specification, these two systems use different strategies to select and present the entities and attributes related

to the query. In the Data Explorer, the user can expand the items of a list of tables in a scrollable and searchable tree view, which is pinned in one side of the window, to choose the desired attributes. This system divides the attributes into two different types: Measures and Dimensions. Usually, measure refers to quantitative data and dimensions to categorical data. So, to insert the attributes in the query, users can drag and drop the required attributes to the form-based interface that contains the Measures, Dimensions, and Filters of the query (Figure 3.3a) [32].

On the other hand, the new component of Chartio, Visual SQL provides a different interface to select the data sources. Contrary to the previous approach, in this interface, there is no fixed list to choose the attributes. In this way, there is only a search text component that is activated when the user clicks on "add column" action. When this action occurs, a pop-up style component that has a list, similar to the referred above, where it is possible to preview some data entries of the table, is presented (Figure 3.3b) [34].

In the systems referred above the columns are added one by one sequentially, but other systems have different methods to select the table's columns. For example, in Tableau Prep [68] there is a checkbox list to chose the intended attributes (Figure 3.3c), and in Microsoft Power BI [48] the table is chosen using a list, and all its attributes are added automatically. Also, users can remove, the columns not desired afterwards [50, 69].

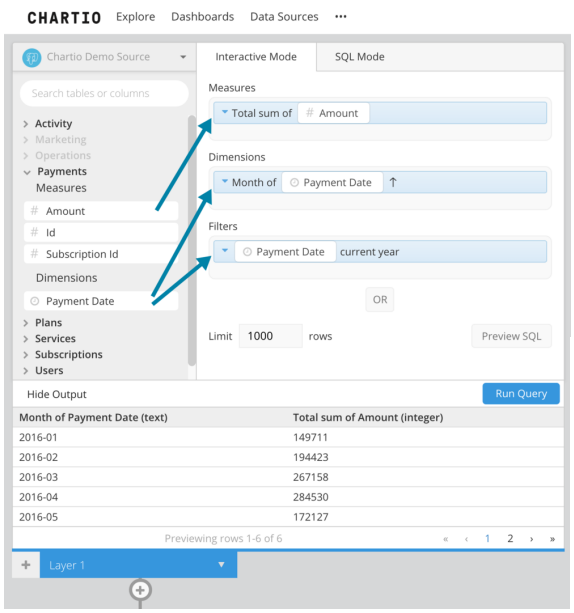
Other systems, as Devart dbForge Query Builder [40], uses a diagram-based interface to select the entities and attributes of the query. In this system, the user can drag and drop the desired tables to the diagram area, and select through checkboxes the intended attributes, that are presented in the database schema diagram (Figure 3.3d).

Merge Type:

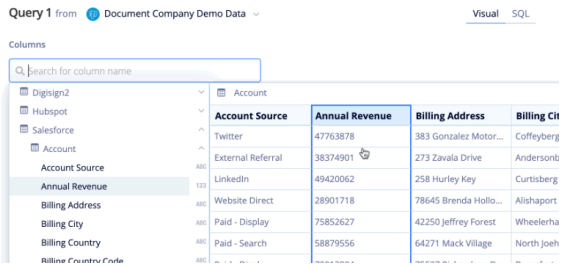
Merges are used when it is necessary to extract data from different tables, so it is necessary to establish what is the join kind to merge the data. Therefore, the interface needs to adopt an interaction and representation technique to specify it. To define a join in Devart dbForge Query Builder [40], the user can only select the attributes' checkboxes of the different tables and the system generates an inner join automatically. In this system, there are buttons on the toolbar to select all rows of one table, of another, or both, allowing to perform left, right and outer joins respectively [39].

Another approach used by some systems, such as Chartio Data Explorer [32] and Microsoft Power BI [48], is a form-based interface to insert a join. In the former, two queries can be merged clicking on a button to popup a form that can be used to select the merge type and the first columns that will be merged using dropdowns [32] (Figure 3.4a). Also, if there are null values on the merge related columns, there is an option to include or not the null values match rows [33]. Similarly, the latter provides a button to merge queries that opens a modal where the attributes that will be used on the merge (viewing also a table preview) and the join kind can be chosen, using a dropdown [50] (Figure 3.4b).

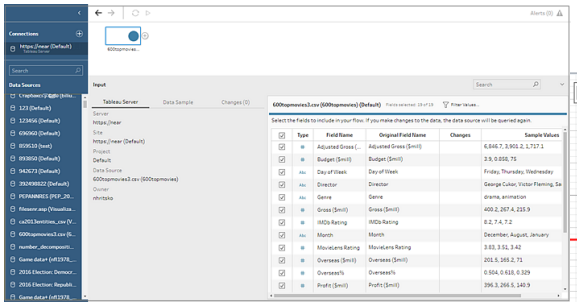
Tableau Prep [68] provides two options to start a join between two tables: clicking



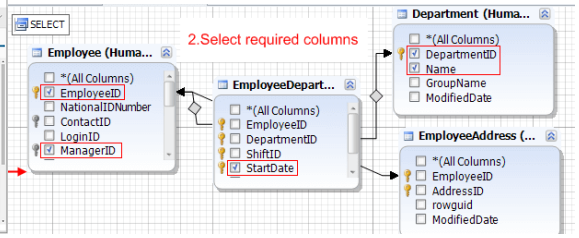
(a) Chartio Data Explorer (source: Chartio[32])



(b) Chartio Visual SQL (source: Chartio [34])



(c) Tableau Prep (source: Tableau [69])



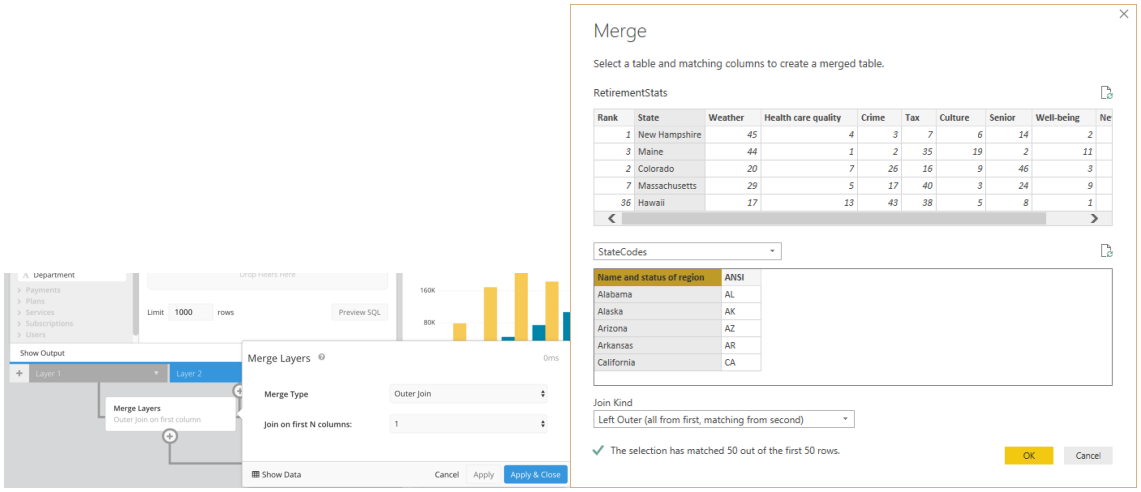
(d) Devart dbForge Query Builder (source: Devart [40])

Figure 3.3: Different approaches to select the entities of the query.

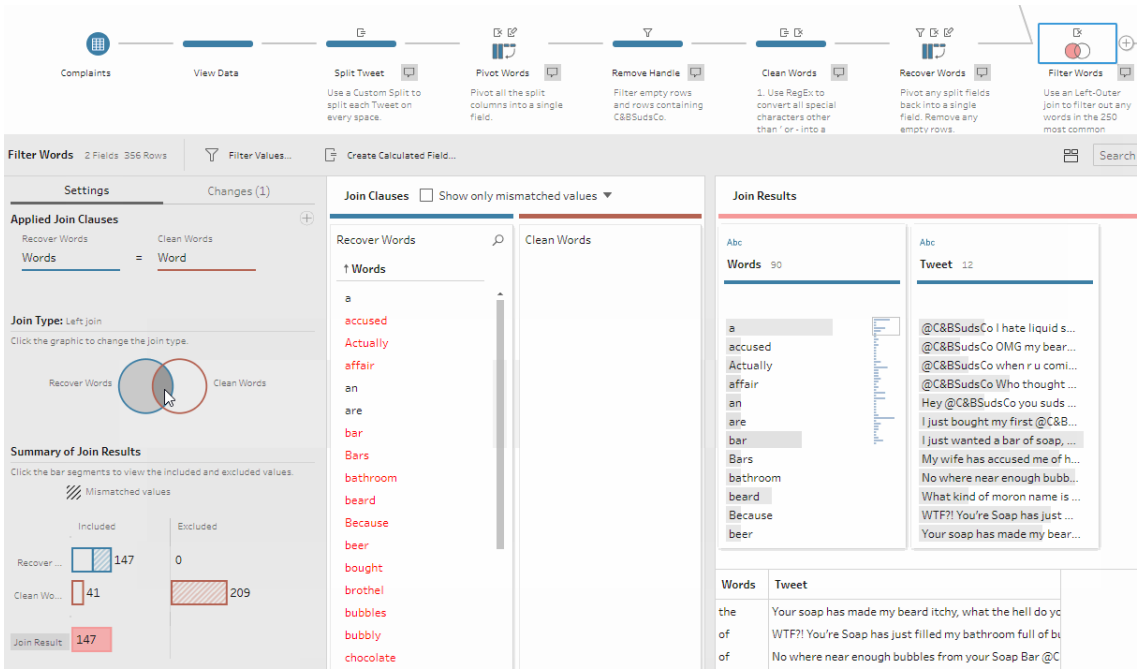
on a "add join"hover button above the table visual representation with the suggestion of related tables, or merge the visual representation of two tables using a drag and drop action. After this selection, the inner join type is selected automatically by the system according to the tables' relationship [63, 64]. However, the user can configure the join in a dedicated section (Figure 3.4c) where it is possible to define the join type using a Venn Diagram, to manage the join clauses using dropdown lists to select the fields, including also some join clause recommendations based on the database schema. Moreover, a summary of the join result that contains counters with the values included and excluded by each table, in a visual way using diagrams is provided. Finally, a list of the values included and excluded, where the red values represent the values excluded is presented, as well as a preview of the join result [64].

Filtering Criteria:

To represent and manage the filtering criteria of the queries, usually it is used text to indicate the logical conditions. However, some systems are trying to optimize the



(a) Chartio Data Explorer (source: Chartio [32]) (b) Microsoft Power BI (source: Microsoft [50])



(c) Tableau Prep (source: Tableau [28])

Figure 3.4: Data merging approaches.

usability, helping the user in the specification process through some autocompletes. These diminish the necessity to remember all the syntax and the name of the functions. Besides, other systems present the logical conditions using a more structured layout, although keeping resorting in a textual representation. An example is Devart dbForge Query Builder [40] that represents the WHERE and HAVING clauses in a tree where the user can organize the conditions into groups [37].

Furthermore, some query systems also allow the user to view the query result, providing a shortcut in the column's name to insert filters. So, the user can change the query while is viewing the results. In this way, the user can apply a filter and view its effect

almost immediately. Power BI Query Editor [48] is an example of a system that uses this approach.

Different interfaces can be used to select filters regarding the field data type. The advantages of the graphical interfaces could support the filtering criteria definition. For example, if a filter is applied to constraint dates, then a date input box with a visual calendar can be useful to simplify the date typing. In this way, some software, such as Chartio Visual SQL [34] and Chartio Data Explorer [32], not only helps in the data typing but also provide dropdown lists that contain operators that can be applied to the referred data type (e.g. less than, contains, like, etc) to helps the user in the boolean operator specification [32, 36].

Tableau Prep [68] follows this more strictly, distinguishing between data types when filtering criteria are indicated. It provides different forms depending on the data type. The main difference of this system is that not only provides a more diversity of controls, as integrating range selectors, radio button, and the option to include or exclude fields through an action accessible near its value, but also gives to the user the option to use a calculation form where the interaction style is more textually and more extensible [66].

Sorting Criteria:

Usually, in the actual VQIs, the sorting criteria could be indicated in two ways: a right-click action on the column header of the table that represents the query result, or using a form-based interface to define the sort criteria of each entity. Devart dbForge Query Editor [40] is a pure example of the first approach [41]. Chartio Data Explorer [32] adopts the second approach, providing a pop-up form to apply sorting criteria to the query. The user can use this form to select the intended attributes and the criteria to apply [30]. Moreover, Chartio Visual SQL [34], Tableau Prep [68], and Microsoft Power BI [48] combines the previous solutions with the possibility to redefine the priority of the sorting criteria, through drag-and-drop actions [36, 50, 67].

Aggregation Functions:

In order to perform aggregations functions, such as MIN, MAX, COUNT, AVG, SUM, or GROUP BY, some systems provide these functionalities through interaction with the query result table. Devart dbForge Query Editor [40] provides this option to create an aggregation function. Moreover, in this editor exists another aggregation dedicated view which contains the aggregation functions in a tree view. In this view, users can group or ungroup the elements present in the window [38]. Microsoft Power BI [48] allows also the users to add aggregations through the right-clicking on the column header, but in this case, it is open a form to enter the intended function and columns [49]. Chartio Visual SQL [34] and Chartio Data Explorer [32] presents a pop-up form where could be selected the columns and the aggregations intended [35]. Using a different interaction strategy, in Tableau Prep [68], the user drag and drop the desired columns to a specific area that is divided into two: Grouped Fields and Aggregated Fields. The first is to add the SQL corresponding GROUP BY, and the second to add the other aggregation functions, such as COUNT, MIN, MAX, AVG, and SUM.

Other Specifications:

Regarding the option to add new calculated attributes, all the systems referred above allow inserting calculated attributes to the query, excepting the Devart dbForge Query Editor which does not support [36, 49, 65].

The option to show only distinct values is provided in Tableau Prep [68], Microsoft Power BI [48], and Devart dbForge Query Editor [40], through a button or a checkbox to does not see in the result the duplicated values. However, Chartio Visual SQL [34] and Chartio Data Explorer [32] does not support a specific interaction method to specify this. Nonetheless, the distinct effect can be applied, using the group by in all the columns of the query.

Some system provides visual options to build queries that contain UNIONs. For example, Chartio Visual SQL [34] and Chartio Data Explorer [32] provides this option in the same components of the joins. In these systems, when the user chooses the join type, the union is one of the join types available, although there is a different operation in SQL. Tableau Prep [68] and Microsoft Power BI [48] present different options between the joins and unions, but the interface and the interaction strategies are similar [50, 64].

Moreover, a visual way to perform subqueries is provided by the diagrammatic-based interface of Devart dbForge Query Editor [40], using tabs to alternate between the selected query. Links are used as assistants and shortcuts to view and change between the queries [42, 43].

3.4 Discussion

The conceptual models of query writing presented in this chapter will be taken into account along the design of the solution since these are a good baseline to understand how users reason while are interacting with the system to achieve their goals. The system's tasks will be optimized according to the presented users' conceptual models presented, in order to reduce the semantic errors, as much as possible. Therefore, the most relevant semantic errors that occur using SQL were presented. In the design of the solution, these errors are part of the problems to solve using a new user interface and UX. As referred in section 3.2, the semantic errors will be the main concern since these could have a negative impact on the results. The syntactic errors also will be addressed in this work but require less study about the users' conceptual model. The major concern in this type of error is to provide the maximum feedback to the user. In that way, the user will understand the error and correct it.

Furthermore, since the design of an interface needs to tackle with the usability attributes trade-off, it is important to characterize and consider the problems of the application's target users. This characterization is essential to define the usability priorities of the interface. The population used in the studies presented in this chapter will be an excellent reference to the target user analysis. The target users of the intended system are low-code developers. Since low-code development has advantages not only for not

Table 3.2: VQSs Summary

Feature \System		Chartio Data Explorer	Chartio Visual SQL	Tableau Prep	Power BI Query Editor	Devart dbForge Query Builder
Tables and Columns	Only the required	✓	✓	✓	✗	✓
	All the attributes at once	✗	✗	✓	✓	✓
	Remove Columns	✓	✓	✓	✓	✓
Merge	Inner Join	✓	✓	✓	✓	✓
	Left Join	✓	✓	✓	✓	✓
	Right Join	✗	✓	✓	✓	✓
	Full Outer Join	✓	✓	✓	✓	✓
	Cross Join	✓	✓	Using Calculated Attributes	Using Calculated Attributes	Textually
	Null Option	✓	✓	✗	✓	✗
	Define Join Condition	✓	✓	✓	Selecting Columns Visually	Textually
Filtering Criteria		✓	✓	✓	✓	✓
Sorting Criteria		✓	✓	✓	✓	✓
Aggregation Functions		✓	✓	✓	✓	✓
Unions		✓	✓	✓	✓	✗
Calculated Attributes		✓	✓	✓	✓	✗
Distinct		✗	✗	✓	✓	✓
Subqueries		✗	✗	✗	✗	✓

programmers but to programmers with different levels of expertise, the target users of the system are wide. Thus, the studies that have evaluated a wide diversity of users, such as the evaluation of Lu *et al.* [13], could reveal important results to the work development. Moreover, the studies presented that have analyzed students of database systems [1], [26], could add more important data, since some details only occur if the queries formulated are more complex.

Finally, a set of actual graphical user interfaces used to formulate queries was presented and compared. The approaches used by other systems are a relevant object study for the design and the development of the new interface. Table 3.2 represents a summary of the database query operations supported by each one of the presented systems.

Chapter 4

Proposed Solution

After the problem contextualization, and the description of the related concepts, techniques, and studies, this chapter presents the work already prepared. Starting with the requirements analysis which includes user interviews and extraction of metrics to conclude, in quantitative manners, what type of cases could lead to SQL usage. A current progress state of the solution developed will be presented, describing the actual development state and a foreseeing schedule of the work plan for the remaining time until the final of the dissertation.

4.1 Requirements Analysis

In order to understand the functionalities and visual approaches used by Aggregates to visually build queries, the project has started with an exploration of this system (more detailed description ins section 2.2.2). Meanwhile, not only were identified the expressiveness problems of the visual language, previously mentioned in section 1.2, but also usability problems could be perceived in the performance of available actions.

After that, user interviews have been prepared to explore the limits of the tool and understand the impact of the problems on user actions. So, in the interviews, after perceiving the background of the participant, more directed questions were asked to pick up the first responses of the users about the advantages and disadvantages of the visual query tool. Next, it was asked what are the situations when SQL usage was preferred to obtain insights about the reasons not to use the visual tool to perform these queries. Also, it was required that users present examples to explain the problems they have, whenever possible, because this strategy can be useful to understand the impact level of the problem and sometimes show other problems not primarily identified. When the users did not mention anything about expressiveness problems of the visual tool, direct

questions about if they never needed to use the operations not supported by Aggregates were asked to cover the possibility of forgetting to mention that aspect.

The results reveal that the most novice users, with less than six months of experience, feels that Aggregates are easy to use and covers their necessities, referring also that were simpler to learn than SQL. The most experienced users, who use the OutSystems platform to develop applications, every day, for professional purpose, and have a technological background, reported that in the visual tool they cannot have a good understanding of the global view of the query, mainly if many tables, columns and business rules are involved. Switch between tabs in the interface to view the data sources, and the filtering and sorting criteria were other issue presented, as well as the lack of control on the query output[1]. Asked about the aggregation functions[2], they do not refer any problem with the approach interaction strategy adopted. Furthermore, other usability problems that decrease the users' satisfaction have been pointed out, like the difficulty to search for a column in the query result, when there are numerous columns.

The analysis was complemented with a metric study on queries executed on the Outsystems cloud, in order to find patterns that could justify users reasons to use SQL instead of Aggregates. The queries analyzed, which have been extracted in July 2019 from customers' projects, were built using Advanced Queries[3].

Firstly, the data set is composed of 214.400 statements, but only 60.8% were used in this study since only the queries were important for this context and not other instructions, such as inserts, updates, deletes, and transactions. Since the last set, which includes 125.613 queries, has duplicated queries, these were removed and the final queries set used to extract information comprehends 67.828 queries. The operators and clauses were figured out using a SQL Parser developed in JavaScript [46].

After the abstract syntax trees of the queries were obtained in a JSON file, these were analyzed in a program to count the operators and the clauses that were present in the queries. A first analysis can measure the operators that are not supported by the visual tool, since the textual language is the only way to perform these operations. Table 4.1 summarizes the number of queries that contain these operations, where the intersection is not null, so there are queries that have two or more of the indicated operations. Thus, each percentage represents the subset of queries that use the operation inside, no matter if they have other operations or not.

Furthermore, it was measured how many queries are performed using the textual language but could be designed using Aggregates. Table 4.2 shows the results obtained that divide the queries performed in three categories:

- Not Supported: Queries which include operations not supported by Aggregates,

[1] As referred on section 2.2.2.2, when a user adds an entity to an Aggregate, all its attributes are added automatically and if the user hides them the output of the query does not change.

[2] Funtionality added when Simple Queries have been replaced by Aggregates (Section 2.2.2.1)

[3] The option of the OutSystems Platform, invoked in section 2.2.2.1, that allows the query design in a textual way using a language based on SQL.

Table 4.1: Queries that contain operations not supported by Aggregates

	IN	NOT IN	EXIST	NOT EXIST	UNIONS	DISTINCT	SUBQUERIES	Total
Queries	7538	2697	132	118	2385	7987	6827	67828
Percentage	11.11%	3.98%	0.19%	0.17%	3.52%	11.78%	10.07%	100%

Table 4.2: Queries that could be designed using Aggregates and the queries which the tool does not support

	Not Supported	Supported (Simpler)	Supported (More Complex)	Total
Queries	28130	24026	15672	67828
Percentage	41.5%	35.4%	23.1%	100%

such as IN, NOT IN, EXIST, NOT EXIST, Unions, Distincts and Subqueries;

- Supported by Aggregates: Queries that could be designed totally using Aggregates. These are divided into two subcategories:
  - Simpler: Queries which include only operations supported by aggregates excluding the indication of sorting criteria and the use of aggregation functions (e.g. GROUP BY or SUM, AVG, MIN, MAX, COUNT);
  - More Complex: Queries that are supported by Aggregates excluding the above (simplers).

Since user interviews suggested that aggregation functions and sorting criteria were not the main problems. The queries supported by Aggregates were divided to compare the quantitative analysis with the qualitative analysis extracted in interviews. This is important because these operations can be specified using a different interaction technique where the user changes the query when he is interacting with the query result, as mentioned in section 2.2.2.2.

4.2 Scope Definition

The results of the quantitative analysis show that it is in conformity with the qualitative analysis results (user interviews). Both of the analysis, concludes that the main priority of the project should be the usability improvement of the visual querying tool interface. The usability problems not only are the main concern extracted by the users' interviews, but it also has metrics that sustain that hypothesis. In the set of queries analyzed, around 58.5% could be designed using Aggregates, is evident that the lack of support of some SQL expressions might not be the main problem.

Although the results show a small set of queries that are not supported, these expressiveness problems of the visual tool can influence users to change to the textual languages. So, as the visual approach does not support some features users may want to change to the textual language where they can perform all the queries.

Under the circumstances, the results were discussed together with the stakeholders. It was determined that the main problem was the usability of the system. Regarding the improvement of the visual interface expressiveness, it was defined that the main priorities were the IN / NOT IN and the DISTINCT operations. Thus, along with the project, the main focus will be the improvement of the usability and if it is possible conciliating to these two new features of visual expressiveness.

4.3 Proposed Implementation

According to the analysis made and the results concluded, the implementation covers an iterative design process aiming at improving the usability of Aggregates, the component of the OutSystems Platform to visually build queries. The prioritized improvements of the VQL expressiveness (IN / NOT IN and DISTINCT) will be included in the project, if possible, depending on the development progress of the usability issues. If in the next stages of development it is found that it is possible to address these expressiveness problems without impairing the development related to improving usability, these will be designed and developed, otherwise, the focus will be only the usability improvement.

As mentioned above in Section 2.2.2.2, the actual visual tool include in the interface sections to design queries visually and to view their results. Therefore, the aim is to design and evaluate how these two parts of the interface can be changed to improve the usability of the system. In a nutshell, there are two aspects that are needed to be taken into account, in order to guide the design process: the simplicity, efficiency and effectiveness to construct queries independently of how many tables or conditions, and the readability of the global query to perceive what data of the database the query will be gathering.

The design and development process shall be divided into an initial preparation and analysis phase and the iterative design process phase:

- **Preparation and analysis phase**: before the development of the prototypes, the users and the tasks of the should be analyzed. Also, it should be started sketching as a means to bring up ideas that could represent starting points to tackle the existing problems. Therefore, this preparation process before the development of the prototypes will include the following tasks:
  - **Data Extraction**: although there are results obtained in the interviews and in the quantitative analysis process, detailed in 4.1, additional data about the queries design using aggregates also will be extracted. It could be important to the process to understand the user's usage of the existing visual tool since the results obtained are only about the queries built using the textual language;
  - **User and Task Analysis**: users and their tasks will be characterized and classified regarding the concepts presented in section 2.1.2.3 This description and

classification will be used not only as a reference point throughout the design process but also to define the most important trade-offs on usability attributes;

  - **Sketching**: the phase where the first sketches of possible solutions will be designed. The most important is the initial exploration of several possibilities to tackle the problems through a low-level and fast approach. As referred in section 2.1.2.2 it is useful to discover new ideas, keeping register the first approaches to tackle the problems. At the later phases, the sketches could be useful to remember the starting point of the design process;

- **Iterative design**: this process starts with low-level prototypes and in each iteration, these are evaluated to create higher fidelity prototypes in the next iteration. However, as referred in section 2.1.2.2, there are different strategies to reuse or not the prototypes over iterations. In this project, an evolutionary strategy where the prototypes developed are used as the basis for the next design iteration will be adopted. Moreover, the design process will include three iterations:

  - **Paper Prototype (1st iteration)**: regarding the information obtained in user and task analysis, and the main ideas raised on the sketching stage, a first functional paper prototype will be built. The evaluation of the entire concept of the first interaction strategies is the main concern of this phase. Cognitive Walkthrough and Observational Methods of user testing will be the evaluation techniques used on this phase;
  - **Low-fidelity Prototype (2nd iteration)**: using the results of the previous iteration, the idea is to develop a low-fidelity computer prototype on technologies, such as Balsamiq [29] or Mockingbird [52], which although do not create native applications, allow the employment of visual components and animations more similar with the intended in the final solution. So, more aspects can be tested regarding a higher complexity and fidelity of the interface. At this stage, Heuristic Evaluation will be used by experts and more user tests will be performed. However, it is likely that not only Observational Methods are performed. Experimental Methods could be necessary to test specific hypotheses or discuss what is the better option between a set of possible implementations. Moreover, Query Methods, like questionnaires, could be valuable as well, to understand the user's satisfaction;
  - **Final Prototype (3rd iteration)**: computer prototype integrated in the OutSystems Platform. The development focuses on the presentation layer of the application due to improving the Aggregates interface and UX, using the new results obtained. The front-end of the application is developed in React [59], using the TypeScript [51] language. Accordingly, these are the technologies that will be used in the development of the final prototype. Being the last prototype and consequently the final result of this project, it will be evaluated

also by users and experts to summarize the final results of all the design and development process;

4.4 Work Plan

According to the proposed implementation approach, Figure 4.1 represents a work plan until the final of the thesis, scheduling each one of the tasks explained in the previous section throughout the remaining weeks. The tasks are assigned to its respective major phases, and the focus on the writing of the document is primarily at the end of each phase or iteration to summarize all the contents addressed during these periods, although the report can be updated every time it reveals useful.

Phases	Tasks \ Weeks	March					April				May				June					July				August				September		
		2	9	16	23	30	6	13	20	27	4	11	18	25	1	8	15	22	29	6	13	20	27	3	17	24	31	7	14	21
Analysis & Preparation	Data Extraction																													
	User & Task Analysis																													
	Sketching																													
1st Iteration	Paper Prototype Design																													
	Paper Prototype Implementation																													
	Paper Prototype Evaluation																													
2nd Iteration	Low-Fidelity Prototype Design																													
	Low-Fidelity Prototype Implementation																													
	Low-Fidelity Prototype Evaluation																													
3rd Iteration	Development Environment Set Up																													
	Final Prototype Design																													
	Final Prototype Implementation																													
	Final Prototype Evaluation																													
Report	Writing of the Document																													

Figure 4.1: Task Scheduling throughout the weeks until the final of the project.

Bibliography

[1] A. Ahadi, J. Prior, V. Behbood, and R. Lister. "Students' Semantic Mistakes in Writing Seven Different Types of SQL Queries." In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '16. Arequipa, Peru: Association for Computing Machinery, 2016, 272–277. ISBN: 9781450342315. DOI: 10.1145/2899415.2899464. URL: https://doi.org/10.1145/2899415.2899464.

[2] A. Alshamrani and A. Bahattab. "A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model." In: *International Journal of Computer Science Issues (IJCSI)* 12.1 (2015), p. 106.

[3] T. CATARCI, M. F. COSTABILE, S. LEVIALDI, and C. BATINI. "Visual Query Systems for Databases." In: *J. Vis. Lang. Comput.* 8.2 (Apr. 1997), 215–260. ISSN: 1045-926X. DOI: 10.1006/jvlc.1997.0037. URL: https://doi.org/10.1006/jvlc.1997.0037.

[4] T. Catarci and G. Santucci. "Diagrammatic Vs Textual Query Languages: A Comparative Experiment." In: *Visual Database Systems 3: Visual information management*. Ed. by S. Spaccapietra and R. Jain. Boston, MA: Springer US, 1995, pp. 69–83. ISBN: 978-0-387-34905-3. DOI: 10.1007/978-0-387-34905-3_5. URL: https://doi.org/10.1007/978-0-387-34905-3_5.

[5] D. D. Chamberlin and R. F. Boyce. "SEQUEL: A Structured English Query Language." In: *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. SIGFIDET '74. Ann Arbor, Michigan: Association for Computing Machinery, 1974, 249–264. ISBN: 9781450374156. DOI: 10.1145/800296.811515. URL: https://doi.org/10.1145/800296.811515.

[6] H. Chan, H. Teo, and X. Zeng. "An evaluation of novice end-user computing performance: Data modeling, query writing, and comprehension." In: *Journal of the American Society for Information Science and Technology* 56.8 (2005), pp. 843–853. DOI: 10.1002/asi.20178. eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/asi.20178. URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.20178.

[7] H. Desurvire, J. Kondziela, and M. E. Atwood. "What is Gained and Lost When Using Methods Other than Empirical Testing." In: *Posters and Short Talks of the 1992 SIGCHI Conference on Human Factors in Computing Systems*. CHI '92. Monterey, California: Association for Computing Machinery, 1992, 125–126. ISBN: 9781450378048. DOI: 10.1145/1125021.1125115. URL: https://doi.org/10.1145/1125021.1125115.

[8] A. Dillon and C. Watson. *User analysis in HCI: the historical lesson from individual differences research*. 1996. URL: http://hdl.handle.net/10150/105824.

[9] A. Dix, A. J. Dix, J. Finlay, G. D. Abowd, and R. Beale. *Human-computer interaction*. Pearson Education, 2003. ISBN: 978-0-13-046109-4.

[10] J. Gehrke and R. Ramakrishnan. *Database management systems*. McGraw-Hill, 2003.

[11] H. Henriques, H. Lourenço, V. Amaral, and M. Goulão. "Improving the Developer Experience with a Low-Code Process Modelling Language." In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. MODELS '18. Copenhagen, Denmark: Association for Computing Machinery, 2018, 200–210. ISBN: 9781450349499. DOI: 10.1145/3239372.3239387. URL: https://doi.org/10.1145/3239372.3239387.

[12] P. Jennifer, R. Yvonne, and S. Helen. "Interaction design: beyond human-computer interaction." In: *NY: Wiley* (2002).

[13] H. Lu, H. C. Chan, and K. K. Wei. "A Survey on Usage of SQL." In: *SIGMOD Rec.* 22.4 (Dec. 1993), 60–65. ISSN: 0163-5808. DOI: 10.1145/166635.166656. URL: https://doi.org/10.1145/166635.166656.

[14] J. V. M.A. "I. On the diagrammatic and mechanical representation of propositions and reasonings." In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 10.59 (1880), pp. 1–18. DOI: 10.1080/14786448008626877. eprint: https://doi.org/10.1080/14786448008626877. URL: https://doi.org/10.1080/14786448008626877.

[15] J. Nielsen. "Iterative user-interface design." In: *Computer* 26.11 (1993), pp. 32–41. ISSN: 1558-0814. DOI: 10.1109/2.241424.

[16] J. Nielsen. *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 0-12-518406-9.

[17] J. Nielsen and R. Molich. "Heuristic Evaluation of User Interfaces." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '90. Seattle, Washington, USA: Association for Computing Machinery, 1990, 249–256. ISBN: 0201509326. DOI: 10.1145/97243.97281. URL: https://doi.org/10.1145/97243.97281.

[18] W. C. Ogden. "IMPLICATIONS OF A COGNITIVE MODEL OF DATABASE QUERY: COMPARISON OF A NATURAL LANGUAGE, FORMAL LANGUAGE AND DIRECT MANIPULATION INTERFACE." In: *SIGCHI Bull.* 18.2 (Oct. 1986), 51–54. ISSN: 0736-6906. DOI: 10.1145/15683.1044078. URL: https://doi.org/10.1145/15683.1044078.

[19] OutSystems. *OutByNumbers - Benchmark Overview Repor*. Tech. rep. 2013.

[20] P. G. Polson, C. Lewis, J. Rieman, and C. Wharton. "Cognitive walkthroughs: a method for theory-based evaluation of user interfaces." In: *International Journal of Man-Machine Studies* 36.5 (1992), pp. 741 –773. ISSN: 0020-7373. DOI: https://doi.org/10.1016/0020-7373(92)90039-N. URL: http://www.sciencedirect.com/science/article/pii/002073739290039N.

[21] P. Reisner. "Human Factors Studies of Database Query Languages: A Survey and Assessment." In: *ACM Comput. Surv.* 13.1 (Mar. 1981), 13–31. ISSN: 0360-0300. DOI: 10.1145/356835.356837. URL: https://doi.org/10.1145/356835.356837.

[22] D. A. Robb, P. L. Bowen, A. F. Borthick, and F. H. Rohde. "Improving New Users' Query Performance: Deterring Premature Stopping of Query Revision with Information for Forming Ex Ante Expectations." In: *J. Data and Information Quality* 3.4 (Sept. 2012). ISSN: 1936-1955. DOI: 10.1145/2348828.2348829. URL: https://doi.org/10.1145/2348828.2348829.

[23] K. L. Siau, Hock Chuan Chan, and Kwok Kee Wei. "Effects of query complexity and learning on novice user query performance with conceptual and logical database interfaces." In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 34.2 (2004), pp. 276–281. ISSN: 1558-2426. DOI: 10.1109/TSMCA.2003.820581.

[24] J. B. Smelcer. "User errors in database query composition." In: *International Journal of Human-Computer Studies* 42.4 (1995), pp. 353 –381. ISSN: 1071-5819. DOI: https://doi.org/10.1006/ijhc.1995.1017. URL: http://www.sciencedirect.com/science/article/pii/S1071581985710178.

[25] C. Stephanidis. "User interfaces for all: New perspectives into human-computer interaction." In: *User Interfaces for All-Concepts, Methods, and Tools* 1 (2001), pp. 3–17.

[26] T. Taipalus, M. Siponen, and T. Vartiainen. "Errors and Complications in SQL Query Formulation." In: *ACM Trans. Comput. Educ.* 18.3 (Aug. 2018). DOI: 10.1145/3231712. URL: https://doi.org/10.1145/3231712.

[27] M. Unsöld. "Measuring Learnability in Human-Computer Interaction." Master's thesis. 2018.

Webography

[28] C. Alchin, J. Martin, J. Allenby, and T. Prowse. *2019: Week 9 Solution*. URL: https://preppindata.blogspot.com/2019/04/2019-week-9-solution.html (visited on 02/19/2020).

[29] Balsamiq. *Balsamiq*. URL: https://balsamiq.com/ (visited on 01/29/2020).

[30] Chartio. *Advanced Sorting*. URL: https://chartio.com/help/data-pipeline/advanced-sorting/ (visited on 02/10/2020).

[31] Chartio. *Chartio*. URL: https://chartio.com/ (visited on 02/07/2020).

[32] Chartio. *Chartio Data Explorer | Documentation*. URL: https://chartio.com/docs/data-explorer/ (visited on 02/07/2020).

[33] Chartio. *Chartio FAQs: Joining Data Across Databases*. URL: https://chartio.com/docs/visual-sql/actions/ (visited on 02/10/2020).

[34] Chartio. *Chartio Visual SQL (beta) | Documentation*. URL: https://chartio.com/docs/visual-sql/ (visited on 02/07/2020).

[35] Chartio. *Data Pipeline Steps*. URL: https://chartio.com/docs/data-pipeline/basic/steps/ (visited on 02/10/2020).

[36] Chartio. *Visual SQL Actions | Chartio Documentation*. URL: https://chartio.com/docs/visual-sql/actions/ (visited on 02/10/2020).

[37] Devart. *Building WHERE or HAVING Clause*. URL: https://docs.devart.com/querybuilder-for-sql-server/building-queries-with-query-builder/building-where-and-having-clause.html (visited on 02/10/2020).

[38] Devart. *Grouping Data In Grid*. URL: https://docs.devart.com/querybuilder-for-sql-server/working-with-data-in-data-editor/grouping-data-in-grid.html (visited on 02/10/2020).

[39] Devart. *Making Joins Between Tables*. URL: https://docs.devart.com/querybuilder-for-sql-server/building-queries-with-query-builder/making-joins-between-tables.html (visited on 02/10/2020).

[40] Devart. *Query Builder Tool in dbForge Studio for SQL Server*. URL: https://www.devart.com/dbforge/sql/studio/query-builder.html (visited on 02/07/2020).

[41] Devart. *Sorting Data*. URL: https://docs.devart.com/querybuilder-for-sql-server/working-with-data-in-data-editor/sorting-data-in-grid.html (visited on 02/10/2020).

[42] Devart. *Subqueries in From Clauses*. URL: https://docs.devart.com/querybuilder-for-sql-server/building-queries-with-query-builder/subqueries-other-clauses.html (visited on 02/10/2020).

[43] Devart. *Subqueries Overview*. URL: https://docs.devart.com/querybuilder-for-sql-server/building-queries-with-query-builder/subqueries-overview.html (visited on 02/10/2020).

[44] Google. *Google Sheets*. URL: https://www.google.com/sheets/about/ (visited on 02/05/2020).

[45] ISO. *ISO 9241-11:2018(en) Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. 2018. URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en (visited on 01/27/2020).

[46] JavaScriptor. *js-SQL-parser*. URL: https://github.com/JavaScriptor/js-SQL-parser (visited on 02/10/2020).

[47] Microsoft. *Microsoft Excel*. URL: https://products.office.com/en/excel (visited on 02/05/2020).

[48] Microsoft. *Microsoft Power BI*. URL: https://powerbi.microsoft.com/en-us/ (visited on 02/07/2020).

[49] Microsoft. *Perform common query tasks in Power BI Desktop*. URL: https://docs.microsoft.com/en-us/power-bi/desktop-common-query-tasks#group-rows (visited on 02/10/2020).

[50] Microsoft. *Tutorial: Shape and combine data in Power BI Desktop*. URL: https://docs.microsoft.com/en-us/power-bi/desktop-shape-and-combine-data (visited on 02/07/2020).

[51] Microsoft. *TypeScript - JavaScript that scales*. URL: https://www.typescriptlang.org/ (visited on 02/20/2020).

[52] Mockingbird. *Mockingbird*. URL: https://gomockingbird.com/home (visited on 01/29/2020).

[53] OutSystems. *Evaluation Guide (Developing with OutSystems*. URL: https://www.outsystems.com/evaluation-guide/developing-with-outsystems/ (visited on 02/01/2020).

[54] OutSystems. *Low-Code Development Platform for Enterprise Applications*. URL: https://www.outsystems.com/platform/ (visited on 01/31/2020).

[55] OutSystems. *Service Studio Overview - OutSystems 11 Documentation*. URL: https://success.outsystems.com/Documentation/11/Getting_Started/Service_Studio_Overview (visited on 02/02/2020).

[56] OutSystems. *What's new in OutSystems Hub Edition 2.0*. 2003. URL: https://drive.google.com/file/d/1wkKESumKhJbwK6aoeW2DGU4-TMq-snOp/view (visited on 02/03/2020).

[57] OutSystems. *What's new in OutSystems Hub Edition 2.2*. 2004. URL: https://drive.google.com/file/d/0B7C37RyL27_oNHF4UmFRX3pFMlpMTVlCWnV5dzJCcmhRS2tj/view (visited on 02/03/2020).

[58] OutSystems. *Agile Platform What's New in Version 5.0*. 2009. URL: https://www.outsystems.com/home/document-download/542/31/0/0 (visited on 02/03/2020).

[59] React. *React - A JavaScript library for building user interfaces*. URL: https://reactjs.org/ (visited on 02/20/2020).

[60] M. Revell. *What Is Low-Code?* 2020. URL: https://www.outsystems.com/blog/what-is-low-code.html (visited on 01/24/2020).

[61] T. Simões. *What's (Not) New in OutSystems: A Product Timeline*. 2018. URL: https://www.outsystems.com/blog/posts/not-new-product-timeline/ (visited on 02/03/2020).

[62] C. Souther. *Low-Code vs. No-Code: What's the Real Difference*. 2019. URL: https://www.outsystems.com/blog/posts/low-code-vs-no-code/ (visited on 01/25/2020).

[63] Tableau. *Add More Data in the Input Step - Tableau*. URL: https://help.tableau.com/current/prep/en-us/prep_add_input_data.htm (visited on 02/10/2020).

[64] Tableau. *Aggregate, Join, or Union Data - Tableau*. URL: https://help.tableau.com/current/prep/en-us/prep_combine.htm (visited on 02/10/2020).

[65] Tableau. *Create a Simple Calculated Field*. URL: https://help.tableau.com/current/pro/desktop/en-us/calculations_calculatedfields_formulas.htm (visited on 02/10/2020).

[66] Tableau. *Filter Your Data - Tableau*. URL: https://help.tableau.com/current/prep/en-us/prep_filter.htm (visited on 02/10/2020).

[67] Tableau. *Sorting Data*. URL: https://help.tableau.com/current/reader/desktop/en-us/reader_sort.htm (visited on 02/10/2020).

[68] Tableau. *Tableau Prep*. URL: https://www.tableau.com/products/prep (visited on 02/07/2020).

[69] Tableau. *What's New in Tableau Prep Builder*. URL: https://help.tableau.com/current/prep/en-us/prep_whatsnew.htm (visited on 02/07/2020).