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

October, 2020

Accelerating SQL with Complex Visual Querying

To my dear grandparents Júlio and Fátima.

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

1

Introduction

Nowadays, database queries are required not only in computer systems areas but also in most sectors of professional environments or personal demands. The 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 to obtain the intended information.

In the decades of the 1960s, Database Management Systems (DBMSs) arose, and later in 1970s new management systems that use relational models, designated as Relational Database Management Systems (RDBMSs). Besides, the first Data Query Languagess (DQLs) appeared, such as 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]. Even though these new technologies provide a structured way to access databases, knowledge of relational logic and DQL was mandatory to fetch data from relational databases. Thereby, only a subset of people could use these powerful querying technologies.

VQSs were 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". These systems use different visual representations and interaction strategies to build database queries. This visual approach could improve the user's learning curve and reduce the mandatory previous knowledge of a DQL, which are more difficult to learn mainly for people without programming base knowledge. Furthermore, some visual interface mechanisms such as automatisms, accelerators, or feedback messages, can be explored in order to accelerate the querying formulation process and reduce the errors that users can make while they are building queries.

In that way, those visual systems are not only useful to users not familiarized with DQLs. Conversely, some studies have revealed that visual languages might be convenient

to the expert users too. For instance, the comparison made by Catarci and Santucci [4] concludes that diagrammatic languages can reduce the error rate of the queries, in comparison with the ones that were build using textual languages, even when they were formulated by expert users. These results have demonstrated 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). Therefore, the Visual Query Interfaces (VQIs) should be built strategically in order to take advantage of their peculiarities that could optimize the querying process not only to the users with a low database querying 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 would not have a significant impact on the final product outcome. As high-level languages, APIs and third-party infrastructures have allowed developers to be more productive and focus on the most valued sections of the software they produce. Low-code approaches have followed 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 that way, developers could focus on tasks that truly accelerate the growth of the end product, achieving the desired goals with greater efficiency [68].

The OutSystems provides a cloud solution of low-code development, which allows developers to build and deploy enterprise-grade applications though visual interactions optimizing the time, effort, and previous knowledge necessary. Then, the OutSystems Platform aims to provide an application development environment that could be used by users with different backgrounds to build, deploy and manage their applications, using good practices and state-of-the-art technologies, even if they do not need to concern about that. Therefore, the vision and potential of the OutSystems Platform are similar to the VQIs under-mentioned above since both intend to facilitate, accelerate, and optimize processes through visual interaction.

In spite of development in OutSystems is based principally on visual languages, there is 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 solutions. In that way, there are alternatives to performing operations not supported by the low-code visual approaches. This is the difference between Low-Code and No-Code paradigms since in No-Code is not possible to use low-level code [70]. 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.* [12], 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 [20]. These results reinforce the importance of the improvement of the visual languages used on these platforms.

Following that vision, the principal motivation is the existence of a platform component that accelerates the query building process and reduces the errors that could occur throughout, keeping the experience simple and understandable by all users.

1.2   Problem Description

The OutSystems Platform [61] provides a Visual Query Language (VQL) that allows users to query data from databases through visual interactions. Using that interface, it is possible to perform some operations that are usually supported by textual DQLs, namely join, filter, sort, and aggregation operations.

Although the existing interface turns the process of query formulation more simple and intuitive, it does not support all SQL functionalities. Due to that lack of expressiveness, the platform allows its users to formulate queries using SQL. However, as *Catarci et al.* referred [4], visual languages could give advantages in query formulation for all users, including the ones that are proficient in SQL. Thus, it is important to provide a powerful and consistent interface in order to give users the possibility to accelerate the formulation process, reducing also the rate of errors that may arise.

The principal purpose of this visual interface is to provide a more visual and dynamic tool that accelerates the query formulation process and turns it easier and less error-prone. So, the existing interface should be a useful and efficient tool for developers due to the potential of that visual approaches above-mentioned. However, OutSystems knew there were developers that were not using the visual querying interface, maintaining their preference for SQL. Therefore, it was necessary to verify the main causes that lead users to not use that visual tool.

At the beginning of this dissertation, the lack of functionalities (e.g., IN, NOT IN, EXISTS, NOT EXISTS, DISTINCT, UNION and the possibility to use subqueries) was indicated as a significant factor for users to use SQL. Nevertheless, the first interface explorations revealed impactful usability problems. That is an important point since it could considerably harm users' query formulation process, reducing the value proposition of the system which intends to accelerate the querying process, keeping it more effective and less error-prone. Moreover, there were metric results and user interviews that confirmed the presumption concluded after interface exploration, highlighting the user experience of the interface as the core subject to research and improve.

Beyond the motivation of turning the query building process faster, effective, and less error-prone, the following questions would guide the problem definition process in order

to clearly understand the existing problems of the interface:

- Why do OutSystems developers often use SQL to formulate database queries?

- 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 the design, implementation. and evaluation of a new and more powerful VQI to provide an improved User Experience (UX) that:

- Accelerates the query formulation process;

- Improves the readability of queries (i.e., turns easier to understand what data will be fetched from database);

- Maintains the interface simple and intuitive for all users even the ones that do not know SQL or OutSystems, reducing also the existing learnability curve, whenever possible.

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 advantage 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 - **??**: 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.

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" [26].

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 [47] 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 [17]:

- **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 [28]. 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 [17]: **"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 [16] 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 [13].

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 [17].

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 [17]. 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 [17]. In addition, Nielsen [17] 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 [17].

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 [30] and Mockingbird [54]. 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 [17] 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.
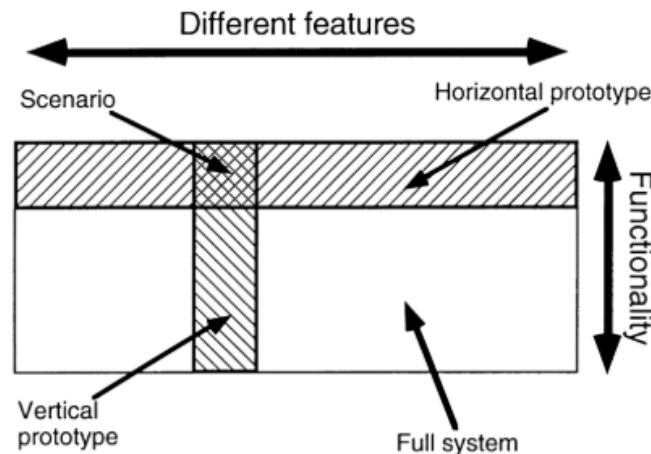
Figure 2.1: The two dimensions of prototyping: Horizontal prototyping keeps the features but eliminates depth of functionality, and vertical prototyping gives full functionality for a few features (source: Nielsen [17])

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 [17].

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** [18]: 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 [17].

- **Cognitive Walkthrough** [21]: 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 [17], 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.

Impact of problem on the users who experience it		Proportion of users experiencing the problem	
		Few	*Many*
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 [17])

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[63])

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 [60].

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.

15

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 [69]:

- **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 [64])

- **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 [65], and the inclusion of a properties pane to view and change the properties of all query elements in a single window [66].

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.

17
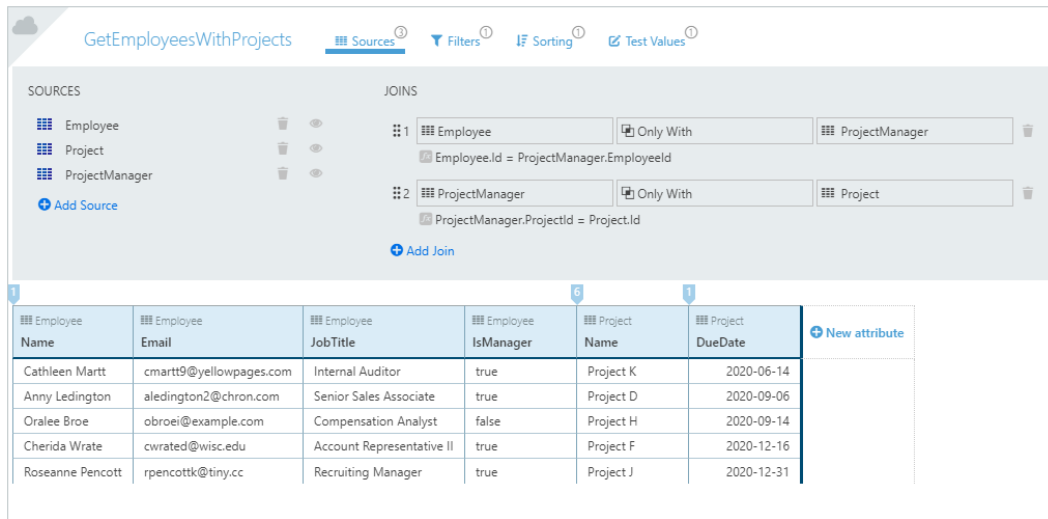
Figure 2.5: Aggregate Example

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 [49] of Google Sheets [45] 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 query, 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 [15] 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.
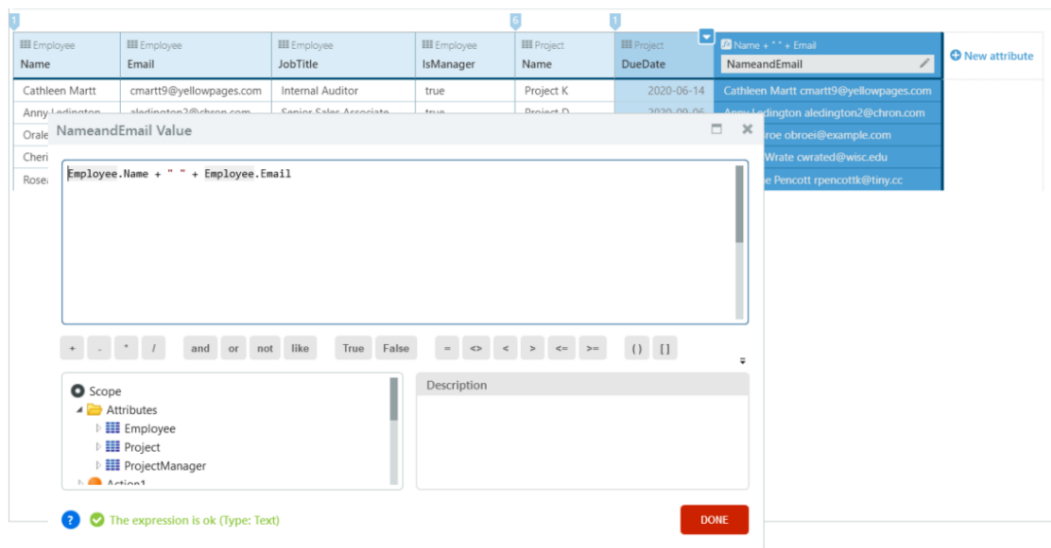
Figure 2.9: Calculated Attribute Insertion

21

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.* [24] 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 [24].

Reisner [22] 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 [22]. The recall of table and column names represents a significant use of long-term memory, being a concern that should be taken into account [25].

In the same field, Ogden [19] 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."

Template Generation

```
SELECT _____
FROM _____ , _____
WHERE _____ = _____
```

Problem Description

What professors are associated with a department?

Lexical Transformation

professors → PROFESSORS
∅ → PROFNAME
department → DEPARTMENTS
∅ → DEPTID

Insertion

```
SELECT PROFNAME
FROM PROFESSORS , DEPARTMENTS
WHERE PROFESSORS.DEPTID = DEPARTMENTS.DEPTID
```

(a) Reisner Model (adapted: Reisner [22])

User's Goals   Data Knowledge   Language Knowledge

Query Formulation → Query Translation → Query Writing →

```
SELECT DEPARTMENT
FROM PROFESSORS
GROUP BY DEPARTMENT
HAVING COUNT(*) > 5
```

What departments have more than 5 professors?

Project the DEPARTMENTS column in the PROFESSORS table if the count of rows for the DEPARTMENTS value is greater than 5.
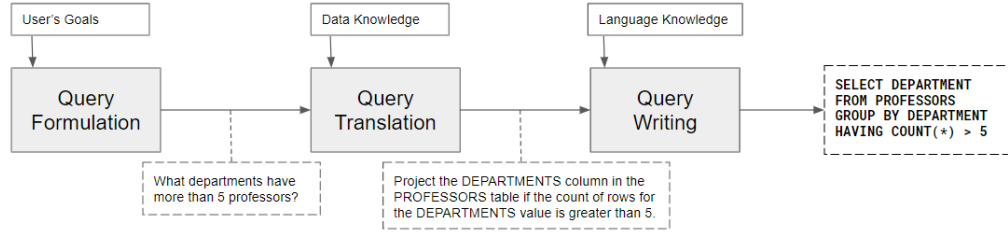
(b) Ogden Model (adapted: Ogden [19])

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.* [23] 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. [14] 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

25

Figure 3.2: New and experienced users' query processes (source: Robb *et al.* [23]).

the returned information is not intended by the user, even if the query does not have compilation errors [25]. 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, 14]. Smeller [25] 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 [17], 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 [27]. 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 [27].

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 [32] has two components to query databases visually: using the Data Explorer [33] or using the new Visual SQL [35]. 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) [33].

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) [35].

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 [77] there is a checkbox list to chose the intended attributes (Figure 3.3c), and in Microsoft Power BI [50] the table is chosen using a list, and all its attributes are added automatically. Also, users can remove, the columns not desired afterwards [52, 78].

Other systems, as Devart dbForge Query Builder [41], 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 [41], 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 [40].

Another approach used by some systems, such as Chartio Data Explorer [33] and Microsoft Power BI [50], 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 [33] (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 [34]. 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 [52] (Figure 3.4b).

Tableau Prep [77] provides two options to start a join between two tables: clicking

(a) Chartio Data Explorer (source: Chartio[33])   (b) Chartio Visual SQL (source: Chartio [35])

(c) Tableau Prep (source: Tableau [78])   (d) Devart dbForge Query Builder (source: Devart [41])
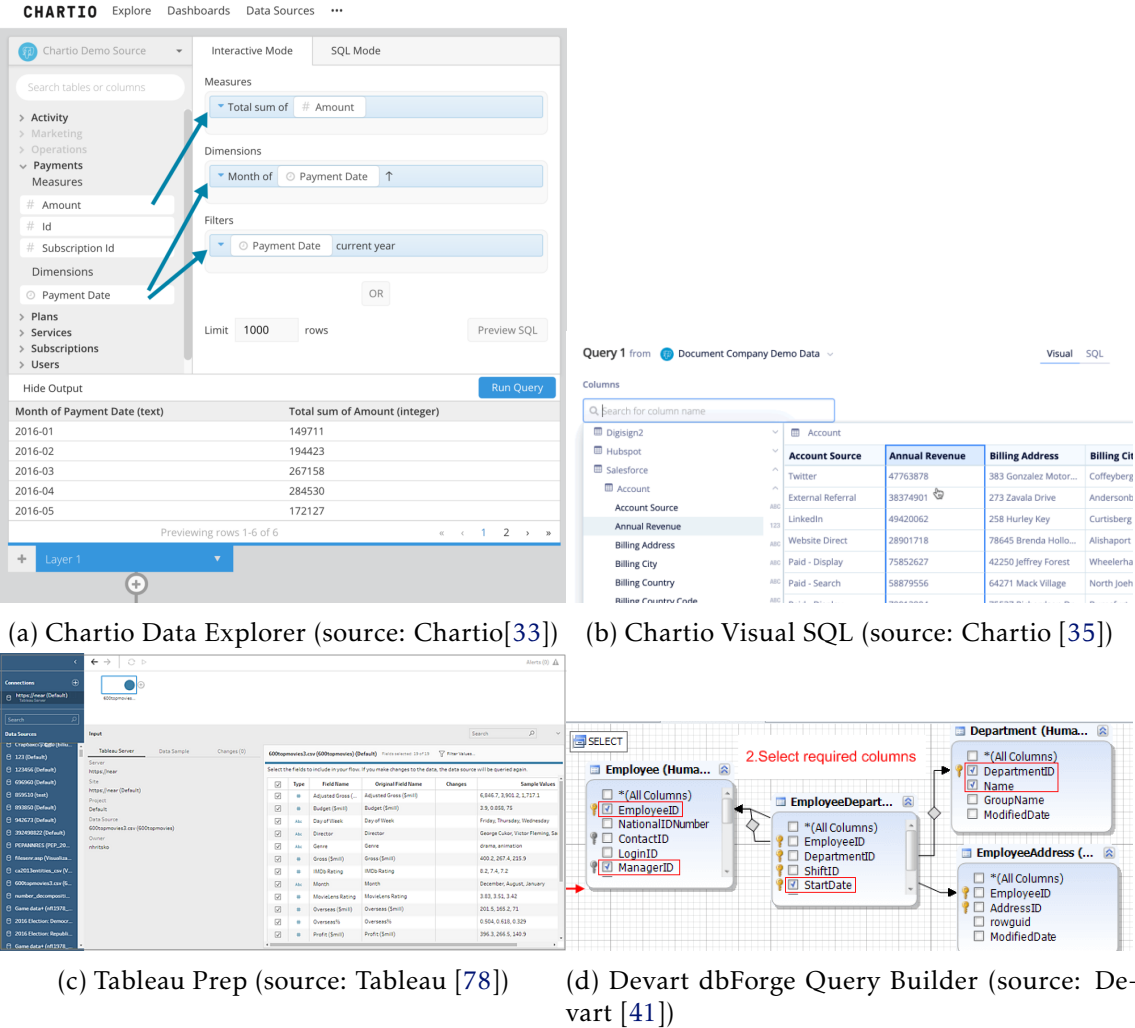
Figure 3.3: Different approaches to select the entities of the query.

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 [72, 73]. 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 [73].

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 [33]) (b) Microsoft Power BI (source: Microsoft [52])

(c) Tableau Prep (source: Tableau [29])

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 [41] that represents the WHERE and HAVING clauses in a tree where the user can organize the conditions into groups [38].

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 [50] 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 [35] and Chartio Data Explorer [33], 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 [33, 37].

Tableau Prep [77] 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 [75].

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 [41] is a pure example of the first approach [42]. Chartio Data Explorer [33] 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 [31]. Moreover, Chartio Visual SQL [35], Tableau Prep [77], and Microsoft Power BI [50] combines the previous solutions with the possibility to redefine the priority of the sorting criteria, through drag-and-drop actions [37, 52, 76].

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 [41] 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 [39]. Microsoft Power BI [50] 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 [51]. Chartio Visual SQL [35] and Chartio Data Explorer [33] presents a pop-up form where could be selected the columns and the aggregations intended [36]. Using a different interaction strategy, in Tableau Prep [77], 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 [37, 51, 74].

The option to show only distinct values is provided in Tableau Prep [77], Microsoft Power BI [50], and Devart dbForge Query Editor [41], through a button or a checkbox to does not see in the result the duplicated values. However, Chartio Visual SQL [35] and Chartio Data Explorer [33] 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 [35] and Chartio Data Explorer [33] 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 [77] and Microsoft Power BI [50] present different options between the joins and unions, but the interface and the interaction strategies are similar [52, 73].

Moreover, a visual way to perform subqueries is provided by the diagrammatic-based interface of Devart dbForge Query Editor [41], using tabs to alternate between the selected query. Links are used as assistants and shortcuts to view and change between the queries [43, 44].

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.* [14], could reveal important results to the work development. Moreover, the studies presented that have analyzed students of database systems [1], [27], 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.

Requirements and Analysis

As previously stated, the main goal of this dissertation is the development of an efficient and effective visual interface for query formulation. The design process phase is the first phase of the solution development, where all the existing relevant problems are combined and took into account in order to draft some possible solutions based on the problem requirements. For this to happen, this chapter describes the processes used to structure the problems of the current visual interface. Furthermore, target users of the system are presented and categorized, accordingly with their background, expectations, and necessities.

4.1 Problem Definition

The problem definition not only was the starting point of the design process but it also represented the source of the planed solution. As the actual visual interface has not been having the expected acceptability due to its interaction problems, these are problems that have an impactful role in the solution building process. In that way, multiple approaches were used to collect interaction problems, understanding for each of them what are the tasks involved, and the most harmed set of users.

The following sections describe the implemented strategies to define the problem as completely as possible. The result of those analyses was combined in a table that characterizes each problem identified of the existing interface. This table is presented in Taxonomy of Problems - Existing Interface and describes all the problems identified. For each problem, it is detailed the interface components involved and the Nielsen Heuristics [55] affected. Moreover, the issues are classified according to the artifact and task attributes of a Framework adapted from Usability-ODC Framework [11], as well as it contains information regarding OutSystems Community[62] posts and likes related to

each problem.

Hereinafter, the approaches and strategies used to collect and organize data regarding the existing problems of the interface will be presented. As usability issues depend on interaction with users, it was important to analyze the problems holistically, considering, in each problem, the impact caused on each type of user.

4.1.1 Analysis

The analysis of the query formulation interface through self-exploration was the first method used to comprehend the existing problems. The process started with the visualization of two OutSystems tutorials [58, 59] about visual data querying. The tutorials included some hands-on exploration that provided an initial contextualization of the visual query builder and its functionalities. After that, other scenarios of query formulation were explored to comprehend the system barriers and difficulties.

As pointed out in 1.2, the lack of some advanced functionalities were observed during that exploration process. Nevertheless, other problems, which have a negative effect on user experience and task efficiency and effectiveness, were also identified.

First of all, it was detected that some functionalities were hidden, damaging the learnability of the system, not fulfilling the "Recognition rather than recall"principle of the Nielsen Heuristics [55]. Notwithstanding that the learnability issue mainly affects the novice users, there are peculiarities of this system and its environment that aggravate this problem. Since SQL formulation is an alternative approach to build queries, if SQL users do not find the intended functionalities in the visual query builder due to its hiddenness, they could use SQL to perform their tasks, avoiding the use of the visual system.

The hidden functionalities are not only advanced features of query formulation. For instance, there is no visible option in the existing interface to add an aggregation function, such as Group By, SUM, MIN, MAX, AVERAGE, or COUNT. These functions are accessible for an exclusive interaction path which requires a right-click on the column header of the attribute where the user intends to apply the aggregation function. Figure 4.1 illustrates an example of the application of an aggregation function. Other options are not always hidden but are not prepared to keep visible due to interface components modification. For example, despite the option to add calculated attributes is visible (after all columns of the query result table), if the query result has several columns it is necessary to scroll horizontally until the end to find that option.

Furthermore, the interface has not revealed to be flexible, efficient, and effective for professional users' purposes. Obstacles have been identified as preventing users from accelerating their query formulation process and causing increases in the queries' error rates. It is presented some examples of these aspects which were identified through the interface's analysis, hindering query formulation and query comprehension, and reducing the most valued proposition of the data querying visual approach:

- Search engines: there is no option to efficiently search for an attribute neither to

Figure 4.1: Hidden option to add an aggregation function, since it is enclosed in a right-click on the query result table column header.
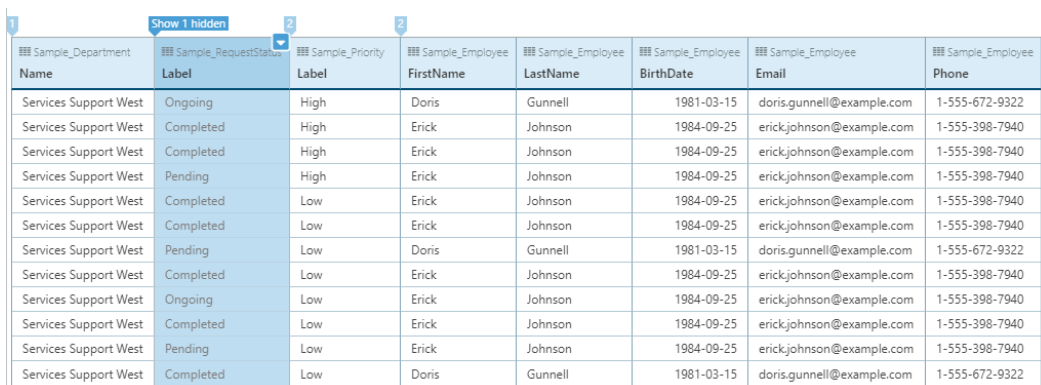
look for some data in the query result or to apply some aggregation function (as illustrated in Figure 4.1). Moreover, there is no general search in all queries, which could be useful when users need to find if some entity or variable is present in the query.

- Hidden columns: in the existing interface, primary and foreign keys are automatically hidden in the query result table since are considered as not relevant data for the query output. That strategy has been implemented since the first release of the interface to provide the most clear output for users with lack of technical background, assuming that they could not understand the meaning of those attributes. Since those attributes may contain relevant information, it is possible to unhide them. However, it must be highlighted that there is no efficient way to expand all hidden columns at once, difficulting significantly the comprehension and formulation of all queries who do not have a reduced number of entities and attributes. Figure 4.2 illustrates how the hidden attributes are presented in the interface.

- Filter edition: the adaptability of existing query filters is possible using the expression editor modal, as demonstrated in Figure 4.3. This modal is useful because it provides some guidance in the expression formulation. Users, can select the intended language expressions or the entities, attributes, or variables that need without having to recall those names. However, if users do not need that assistance, the interaction strategy implemented could led to time and visual context wastefulness every time that a modal is opened. It should be noted that the modal is definitely a good feature but it should not be the unique way to edit filters since it can increase the time spent to built queries unnecessarily. Moreover, there is no way to copy and past filter or to read effectively the content since there is no color highlighting in filters expressions.

37

- Accelerators feedback: this visual query interface has multiple accelerators that turn the query building process faster. Nevertheless, sometimes the automatic mechanisms are silent and users may not comprehend they were applied. For example, when two entities are added and are related, the system automatically joins those entities. However, there is no highlight or other visual interaction mechanism that provides feedback to the user and shows him what action were applied in the aggregation output.

- Inconsistency problems: some functionalities were accessible through a determined context and option but other alternative options do not have the same behavior. For instance, in some use cases it is not possible to add a new entity to the query using drag and drop but if the user click in another add button the entity is added.

- Multiple actions at once: sometimes it could be useful to do several actions at once in order to reduce the time spent to build the query. For instance, it should be possible to add multiple entities at once instead of adding one at a time;

As mentioned, the list of all problems identified is presented in Taxonomy of Problems - Existing Interface.

In a nutshell, the first analysis and exploration of the interface lead to conclude that the interface is useful for users without technical background and has implemented good strategies that could optimize the query formulation process even more than SQL if those strategies would be reconsidered and redesigned. Moreover, it was concluded that it is necessary to provide search engines and other accelerators to enhance the efficiency value proposition of the system. The value of the system increases if users could build queries in that system as fast as SQL or even more.

Sample_Department Name	Sample_RequestStatus Label	Sample_Priority Label	Sample_Employee FirstName	Sample_Employee LastName	Sample_Employee BirthDate	Sample_Employee Email	Sample_Employee Phone
Services Support West	Ongoing	High	Doris	Gunnell	1981-03-15	doris.gunnell@example.com	1-555-672-9322
Services Support West	Completed	High	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Completed	High	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Pending	High	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Completed	Low	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Completed	Low	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Pending	Low	Doris	Gunnell	1981-03-15	doris.gunnell@example.com	1-555-672-9322
Services Support West	Completed	Low	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Ongoing	Low	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Completed	Low	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Pending	Low	Erick	Johnson	1984-09-25	erick.johnson@example.com	1-555-398-7940
Services Support West	Completed	Low	Doris	Gunnell	1981-03-15	doris.gunnell@example.com	1-555-672-9322

Figure 4.2: Hidden attributes - primary and foreign keys are hidden by default.

4.1.2 User Interviews

After the analysis and study of the existing interface, there was a necessity to realize how are the usage of the visual query system for users. There was a demanding to comprehend
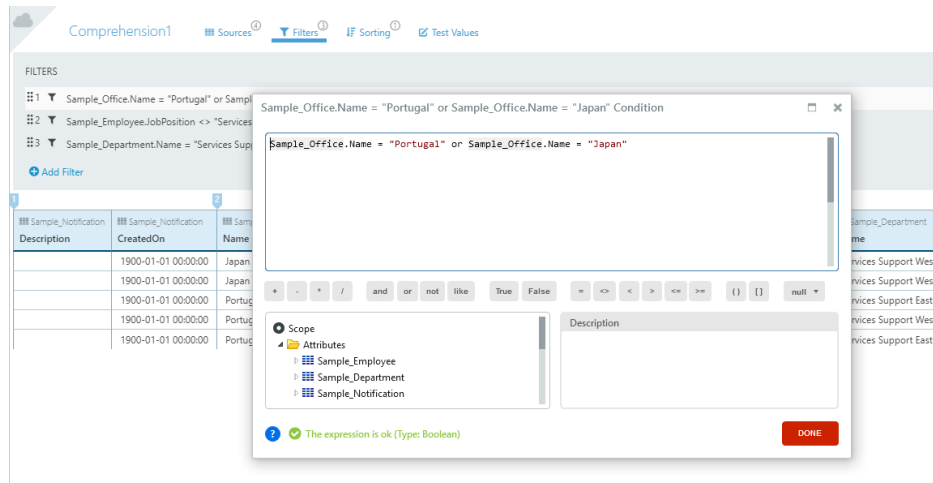
Figure 4.3: Filter edition modal - example of a filter edition after selection of the intended filter (the first one in that case).

what are the most impacting problems of the interface for users' tasks, what are the first users' reactions when asked about the utility of that interface, and if it could be applied, what are the reasons to use SQL instead of OutSystems' visual query builder. Asking those general questions to users was the technique used to understand what is their in-depth and sincere opinion about the system utility and its most impacting problems.

Therefore, ten user interviews were performed to explore the tool's limits and understand their impact on user actions. The interviews started with some brief questions to perceive the background of the participant. After that, more directed questions were asked in order to understand the users' opinions about the advantages and disadvantages of the visual query tool. Besides, it was asked in which situations users prefer to use SQL instead of the visual tool. The answer reveled to be important since it provided information about the main reasons to resort on SQL alternative and also the main causes which led the user to stop using the visual query builder interface. Participants identified and demonstrated in-loco examples of limitations of the visual query system, highlighting causes and the impact of the problems. Also, users revealed other problems which have not been already identified.

Furthermore, the set of interviewed users, answered about which advanced functionalities would benefit further adoption and some usability problems previously identified. The purpose of these last questions was to ensure that the user answered truthfully about the aspects that most affect him negatively while using the visual query system. By letting him explain all the details about his vison, on further adjustments to the system, is possible to establish aspects of common groud between the conclusions of the primary analysis and the user reasoning about major limitations.

The results revealed that the most novice users, the users with less than six months of experience, consider Aggregates simple to use and stated that it covered their necessities, referring that it is more simple to learn than SQL. The most experienced users, who use

the OutSystems platform to develop applications, every day, for professional purposes, and have a technical technological background, reported that in the visual tool they cannot have a clear query understanding at a first glance. Moreover, they referred that they work with queries that contain several tables, attributes, and business rules. In those cases, they have considered that it was difficult to formulate queries using the visual querying interface. Besides, being required to switch between tabs in the interface to view the data sources, and the filters and sorts criteria, other issues were presented such as the lack of control on the query output[1]. When asked about the aggregation functions[2], they do not refer any problem with the approach interaction strategy adopted, but they have emphasized that it is very difficult to find the intended column that need, since there is no search or navigation engine. Other usability problems that decreases the users' satisfaction, such as difficulty to search in all query or to copy and paste query components, were also pointed out.

In conclusion, the results of the problem definition phase indicated that the interface is simple to use but presents a considerable set of limitations in its components, mainly in the domain of professional purpose tasks. In that way, the suggestions to create an improved overview of the query, which would allow a faster comprehension of the query, as well as the implementation of interface accelerators to make the query editor powerful such as search engines and several access alternatives to the same functionality were the most stated aspects. Regardless of the suggestions, the user experience of the interface should be kept simple in order to continue to be easy to be learned and used by users without a technical technological background.

4.1.3 Data Analysis

Even though the analysis made and the first user interviews have indicated user experience problems of the interface as the factor that has been impacting the user acceptance of the visual query builder, a quantitative analysis was performed to perceive what are the operations most used by developers when they formulate queries textually. Thereby, 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]. The data set used is composed of 214.400 statements. However, only 60.8% were used in this study since only the queries are important for the results and not other SQL statements, such as inserts, updates, deletes, and transactions. Nevertheless, that set of 125.613 queries has duplicated results, so that these ones were

---

[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%

removed resulting in a final data set of 67.828 queries. The operators and clauses were identified using a SQL Parser developed in JavaScript [48]. After obtaining the abstract syntax trees of the queries in a JSON file, that data was analyzed in a program to count the operators and the clauses that were present in the queries.

Firstly, it was measured the percentage of SQL queries that contained operations not supported by the visual tool. Table 4.1 summarizes the number and the percentage of those queries containing not supported operations. It is important to refer that the intersection of the subsets is not null, so there are queries that have two or more of the indicated operations. Nevertheless, it can be observed that most of the operations have no significant representation in the results. For example, the DISTINCT operation was the not supported operation with the highest percentage, included in 11.78% of the queries analyzed.

Secondly, it was measured how many queries were bluit using the textual language but could be designed using the visual query builder. Table 4.2 shows the results obtained separating the queries performed in three categories:

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

User interviews suggested that aggregation functions and sorting criteria were not the main problems. Accordingly, the queries supported by Aggregates were divided into two groups in order to compare the quantitative analysis with the qualitative analysis extracted in interviews. This was considered an important element since these operations could be obtained 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.

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. Under the circumstances, the results were discussed together with the stakeholders. It was determined that the main problem was the usability of the system.

41

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%

In conclusion, the results of the quantitative analysis have confirmed the assumptions pointed out after the analysis and exploration of the interface and the first user interviews. All the studies made have concluded that the main priority of the project should be the usability improvement of the visual querying tool interface. Thereby, there are metrics that sustain the conclusions made in the problem definition phase.

4.1.4 Community Ideas

Since OutSystems has a wide worldwide Community of developers, the users can use the OutSystems Community Website [62] to express their problems or difficulties or even to communicate new ideas for the product. Thought that website, it was possible to extract information in order to align the final solution with the ideas and problems they shared. Accordingly, all 306 posts of category "Aggregates & Queries"were analyzed in order to extract useful information for the design phase of this dissertation.

First of all, it has been taken into account for each post if the topic is related to the lack of some functionalities or related to some problems in the interface. That approach leads to perceive that the predominance of the posts were about usability problems of the interface, whereas there are a reduce quantity of posts about functionalities not supported.

Therefore, the users' problems and suggestions regarding usability or enhancements of the interface were processed, transforming that information to a relevant input to the next design phases. All problems and suggestions identified were detailed in Taxonomy of Problems - Existing Interface, followed with some examples of the problems and suggestions indicated in the OutSystems Community:

- Search engines: users requested in multiple posts alternatives to search for an entity, attribute, or filter inside the query. Not only they refer that the readability in the interface is difficult primarily due to the non-existing color highlight in the text but also there is no possibility to search for the intended fields. They have also referred that this feature was extremely important for their work since they need to build queries with a large set of entities, attributes, and conditions;

- Filters Edition: it was pointed out in several posts suggestions to improve the interface in order to support filters comments with color highlighting, since they could be useful to explain the intention of the filter, or even the possibility to disable filters instead of deleting them.

- Result Count: it was referred that there is no visible count of how many rows the query result has;

- Accelerators and Utilities: it has been suggested to add different accelerators in order to accelerate and streamline the query formulation process as well as different ways to present the query structure since, in the user's point of view, the query readability, provided with the existing interface, should be improved.

In summary, beyond the necessity of new advanced functionalities, the developers on the OutSystems Community have indicated different situations where the interface turns out not to be as powerful as it could be, since there is a lack of utilities or accelerators that could assist users keeping their task on track.

4.1.5 Current Implementation Evaluation

In order to analyze users' behavior while they are interacting with the visual query interface, testing scenarios were prepared to evaluate usability attributes of the interface.[4] The main purpose of this evaluation was the analysis of the current effectiveness, efficiency, and learnability of the system to compare them with the proposed solution.

During the tests, users have mentioned other details of the interface that could be a useful input for the next design iterations. In that way, that feedback was also taken into account and registered as all the other issues identified. Thereby, Taxonomy of Problems - Existing Interface includes also all problems identified during the user testing process of the existing visual query interface.

4.2 Target Users

After obtaining a detailed and wide view of the existing problems, the following step was the exploration of how those problems can be solved. As the main goal is the development of an improved interface that gives to the users a solution to manage data queries efficiently, and effectively through intuitive interaction strategies, users are a crucial factor that must be taken into account throughout the entire solution development. Each user has his peculiarities, then the user experience of the solution should be adapted as much as possible to the target users of the query formulation system.

Considering that users will only use the visual query system if they use the OutSystems Platform, that low-code development context where the query system is inserted cannot be dissociated from the user analysis. Thereby, the user analysis process started by a study to categorize the profile of OutSystems Platform users according to the specificities of the data querying domain.

Since the low-code development paradigm has integrated more people who do not have the strict software engineer profile into software development tasks, the users of

---

[4]This evaluation process is described in sections 5.2 and **??**.

the OutSystems Platform do not have the same backgrounds, requirements, and expectations. If, on the one hand, there are OutSystems Developers that have Computer Science academic backgrounds or similar, and experience working with low-level programing language, on the other hand, there are business experts or specialized in other engineering fields that are also developing in OutSystems. In that way, the provided query building experience should be a hybrid approach that covers the traditional software developers demands without turning the development and the language intuitive for people that are not familiarized with classical development patterns, terminologies, or processes.

4.2.1 Requirements and Expectations

As the set of users that may use the visual query system is so broad and heterogeneous, it was necessary to analyze and register the aspects of the user profile which could branch out their needs and expectations in different directions. In this sense, the following three points of the users' background were considered the ones that could accurately cluster the users' expectations and demands, considering the usage context of the interface, which combines software development, low-code development, and relational databases querying domains:

Software Development Background: The previous knowledge and experiences of users in the software development scope can lead significantly to their expectations while they are interacting with a graphical user interface. Being the object of study an interface that allows users to formulate queries, the factor mentioned becomes even more relevant. In software development, most users often associate database querying to languages that are considered a standard to perform those tasks, such as SQL. In that way, most users end up unconsciously relating the visual query building process to the languages they are familiar with. Therefore, if the visual query interface will be used by users that are familiar with technical languages, it is important to not forget to apply a language that could invoke similar principles and reasonings.

OutSystems Development Experience: The user experience on low-code development, in particular using the OutSystems Platform, has also an impact on how the user will experience the visual data querying tool, even if the user does not build queries regularly through the Platform. As a complete solution for low-code development, the Service Studio has a consistent design across its sections. Consequently, a user familiarized with the Platform could have more facility to find options and understand interface language than a user that is using the Platform for the first time, since there are multiple design patterns and built-in behaviors across all the product.

Nevertheless, users who have considerable experience using the existing solution of OutSystems to build queries could be highly adapted to the existing design. Therefore, this fact should be taken into account throughout the solution design process in order to realize how modifications could impact regular users of the systems. Even knowing that frequent users could be skeptical of changes, it is necessary to properly assess whether the

changes only require adaptation or if users could reject them. Therefore, it is important to improve the user experience of the existing interface without removing all its most representative features, in order to keep its singularity.

Data Tools Expertise: Besides textual DQLs, other tools allow users to manage and visualize data through graphical user interfaces. For instance, spreadsheet applications such as Microsoft Excel [49] and Google Sheets [45] presents an intuitive interface where users can manage, organize, and edit data. Using these interfaces, users do not need to know how relational databases work since data is presented in tables that could be manipulated directly under simple controls. For this reason, the language used in those systems is frequently less technical, in such a way that is important to consider that these users are expecting direct manipulation of data and simple language to query and manage data.

4.2.2 User Groups

Being the relational database knowledge a crucial point to frame users' mental model of query formulation, it is important to split up the users who perceive relational databases to the other ones. On the other hand, low-code programming experience can affect how users interact with a visual programming system, thus it is important to study users with experience in low-code development through a different perspective. Considering the aspects mentioned, three user groups were created in order to cluster users that have similar profiles. Thereby, each user could integrate one of the following groups according to their characteristics:

Table 4.3: User Groups

Software Developer	Software Developer or Engineer who has a solid previous knowledge of programming and databases. These users are familiarized with textual programming languages, such as C#, SQL, and others. In that way, they cannot abstract their previous knowledge in such a way that communication with these users should be more technical and specific. Finally, this user group is not an expert in low-code development. However, as they have solid programming, logic, and database knowledge, they could develop some applications using low-code if necessary.
OutSystems Developer	Independently of their background, these users are experienced in OutSystems. That way, they are proficient and faster in low-code development, regardless of their experience in other traditional software development paradigms. This is your principal characteristic that should be taken into account.
Citizen Developer	Users who do not have an extensive programming or software development background. Even though they are not software developers, they could develop some simple apps using low-code due to its simplicity. As some of these users may not know how relational databases work, they may use other applications, such as Microsoft Excel, Google Sheets, Salesforce, and others to manage data.

METHODOLOGIES

Taking into account the existing usability problems and the target users' requirements and expectations, a methodology is essential to design, implement, and evaluate solutions throughout the development process. Therefore, this chapter presents the approached phases and techniques across the solution conception, from the first sketches to the final evaluations.

5.1 Iterative Design

Before starting the solution building process, it was planned the phases that will conduct the development to the final solution of this dissertation. Accordingly, it is presented the design strategy adopted, which will be further detailed in Design and Implementation. This methodology uses an iterative design strategy in order to keep a user-centric design, which prioritizes the users' needs according to the mentioned in 2.1.2. Regarding the prototyping method, the evolutionary prototyping principle (described in 2.1.2.2) was applied, where the last prototype is used as a baseline to develop the prototype of the next iteration.

Sketching: The design process started with an initial sketching phase, where the first solution ideas were explored and crafted. This is a favorable technique to contemplate how new ideas could be integrated into the existing interface. The most important aspect was to think about system transversal changes and not about particular details of specific components, since these details could be refined later. The outcome of this phase should set a more concrete idea in what can be inserted in the prototype, even if it is necessary to think more about how to implement it later.

Keeping in mind these concrete ideas explored, it was possible to start to build the

prototypes that want to be tested by users. The first decision taken was how many prototypes should be built taking into consideration the resources and time available. As mentioned in 2.1.2.2, the first prototypes should be low-fidelity prototypes and iteration by iteration this level should increase in order to refine details in the interface. In that way, it was decided to built two prototypes:

- **Paper Prototype:** Simple low-fidelity prototype implemented in paper using ruler, square, and writing materials. By this approach, it was possible to implement the first ideas faster and with a low-risk. The main concern of this prototype is the design of the major interface changes, not only in terms of layout but also the changes that might affect users' mental model. In that way, it was possible to evaluate early if the design choices applied should continue to the next iterations or if they should be redesigned.

- **Service Studio Prototype:** This is the final prototype of this dissertation, that was developed using C#, Typescript [53], and React [67], and integrated in the new Design of Service Studio. Through this prototype, the solutions implemented were validated and compared with the previous existing implementation in order to validate if the usability of the system proposed has improved.

In each one of the two above-mentioned prototypes, it was included the following phases: design, implementation, and evaluation. The Design phase was where it was thought how the solution ideas could be applied. The Implementation phase refers to the concrete prototype development process. Finally, in the Evaluation phase the prototypes were tested by final users, according to the testing approach explained below in 5.2 and 5.3.

Regarding evaluation, it was necessary to establish how many users should be tested in each one of the evaluating phases. Nevertheless, these two prototypes were not the only ones tested by final users, since the existing implementation was already tested in order to evaluate the current problems of the system. The data of that analysis is also an opportunity to has a baseline of the development starting point across the solution building. Thereby, the number of users tested in the first prototype is also an important factor.

Nielsen performed some studies to quantify how many users should be testing in a usability study, concluding that 5 users are sufficient for qualitative studies because it is almost possible to get close to the user testing's maximum benefit-cost ratio - "Testing with 5 people lets you find almost as many usability problems as you'd find using many more test participants"[56] [57]. However, to perform quantitative analysis it is necessary to get at least 20 users in order to get statistical relevance [57].

According to the studies mentioned, at least 5 users of each user group (described in 4.2.) should be tested since it would be performed a qualitative analysis to validate how users react to the changes applied and what could be improved in the next phase.

Table 5.1: Number of users tested by each user group and by each solution evaluated

	Previous Implementation	Paper Prototype	Service Studio Prototype	Total
OutSystems Developer	10	5	10	25
Software Developer	10	5	10	25
Citizen Developer	10	5	10	25
Total	**30**	**15**	**30**	**75**

Nevertheless, some statistical results should be also retrieved in order to compare in other aspects if the new solution provides improved usability than the existing with the previous one. Hence, it was been tested more users in the previous implementation and in the final prototype in order to obtain also quantitative comparisons. In that way, Table 5.1 shows how many users were tested by each user group and by each solution evaluated.

5.2 Testing Scenarios

Given the wide scope of the usability problems identified, there was a necessity to plan a testing approach that could evaluate the most important aspects of the user-interface communication in a short period. Otherwise, each user test would have a longer duration which would not be affordable.

The first point defined was the type of testing scenarios that would be proposed. For that decision, it was taken into account what were the specific aspects that should be improved in the interface usability. As mentioned, not only the optimization of the efficiency, effectiveness, learnability, and user satisfaction of the entire query formulation process was the goal, but also the improvement of the query comprehension. In such a way that it was important to evaluate if users could understand the query purpose (i.e., what data intends to be fetched from the database) as well as the time they required to realize that.

Considering those evaluation requirements, two types of testing scenarios were prepared: scenarios where users explore an existing query built through the visual querying tool and try to realize what is its purpose, and the other ones where users try to formulate it on their own. Through that approach, it was possible to analyze the usability of the interfaces tested for both points of view: comprehension and formulation.

Nevertheless, the complexity of the queries presented acts as a crucial factor when the scenarios were thought out. For example, an interface could be useful and pleasant to use in simple use cases but it could not keep that quality in more complex queries. Accordingly, there was listed the requirements that were considered relevant to be covered by user testing scenarios:

- **Query Comprehension:** Relevant aspects which should be present in the queries and consequently in the interface to evaluate as extensive as possible queries' comprehension:

- **Interface Elements Exploration:** Include use cases that contain the majority of the query components supported by the system: database entities and joins of different types, filtering and sorting criteria applied to different data types, and other query components added throughout the query formulation process, such as Group Bys, Aggregation Functions (SUM, MIN, MAX, AVG, COUNT) or Calculated Attributes [1];

- **Joins Representation:** Representation of different joins in order to analyze if users could successfully identify them. First of all, there was the concert to consider in scenarios joins of different types, such as inner joins or left joins. In addition, the natural joins (i.e., joins where the unique foreign key that references the other table is equals to the other table primary key) were not the only considered. Besides these joins, some queries that contain joins with more advanced conditions were also covered. For instance, when the join between two tables could be made using different foreign keys, as they are multiple relationships between both tables, or when the join condition contains logical operators.

- **Query Formulation:** At the same time, the following aspects were considered essential to be approached in user testing scenarios from a query formulation point of view:

  - **Add Data Sources and Joins:** Identify the options chosen to add query sources and analyze what are the users' reactions to the system automatisms to simplify the joins specification;

  - **Edit Query Filters:** Evaluate if the query filters edition is intuitive and what barriers could exist in the interfaces regarding this aspect;

  - **Insert Calculated Attributes, Group Bys, and Aggregation Functions:** Check if users could understand the situations they need to use each on of the referred functionalities and if they can discover how to apply them without difficulty.

  - **Use Hidden Columns:** Verify the difficulty felt by users when they need to apply actions in attributes that are not always visible in the interface, either because they are hidden, or because they are not visible due to the lack of available space in the interface (scroll required).

The aspects mentioned were considered the most relevant to analyze because they allow widespread use of the tool but also cover cases identified as critical in terms of usability, according to the aspects detailed in 4.1.

Nevertheless, the data model could have also a significant impact on user testing results. For instance, if a user does not understand a data model, he could not realize the purpose of a query, even if the query is presented in a simple and readable manner. In

---

[1] These operations were presented in 2.2.2.2

that way, the selection of the data model used for user testing was performed attending to the following points:

- **Simple Business Domain:** If the purpose of the database is to store and manage data that has a simple and practical day-to-day application, it would be simple to understand the data model regardless of the user's background;

- **Different data types:** A data model that contains a variety of data types it would be useful to analyze if the design approaches chosen could work for all types;

- **Multiple relationships between two entities:** The interface aims to accelerate and simplify the querying process not only for simple cases. For this reason, it is important to perceive how the interface could support users in cases that there are more than one relationship between two entities. Moreover, SQL does not have any particular syntax that helps users in these cases, then there is an improvement opportunity here.
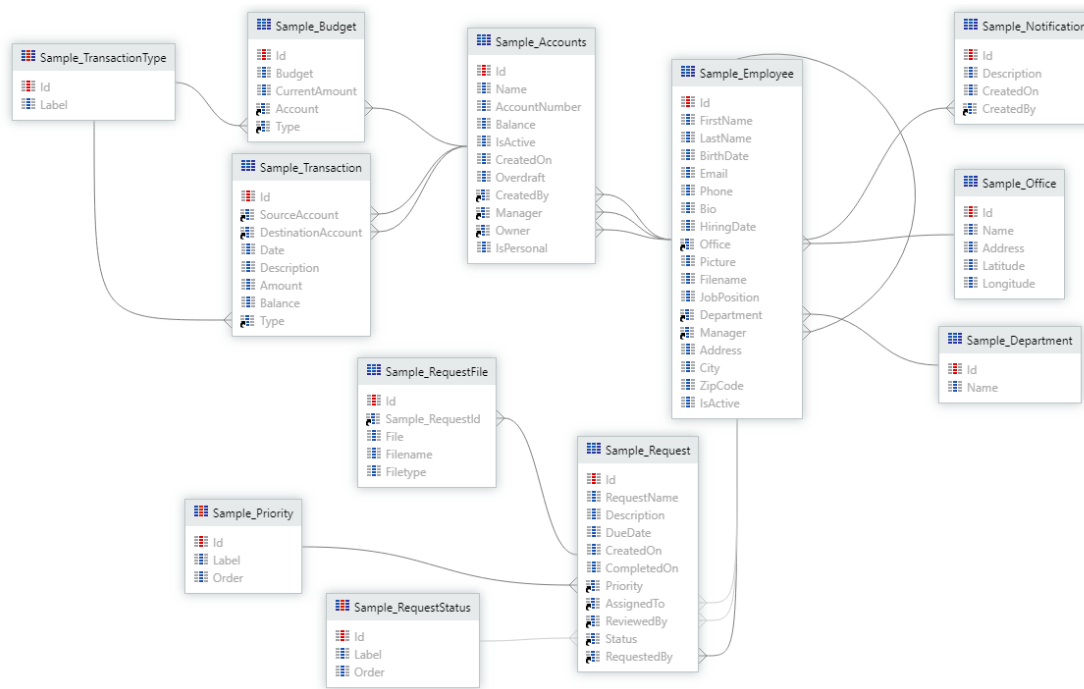

Figure 5.1: Data Model used for User Testing

Accordingly, 5.1 illustrates the data model of the database adopted to perform all usability tests.

After choosing the data model used as support for the usability tests, the list of test scenarios was elaborated. There were designed three different types of scenarios:

- **Query Comprehension:** The user explores a visual query already built and tries to indicate what are the query components presented as well as the data that would be fetched from the database through that query;

51

- **Query Modification:** After a query comprehension example, the user tries to apply some modification on the existing query previously explored;

- **Query Formulation:** Given a natural language statement that explains what data is intended to be fetched from the database, the user tries to retrieve them formulating a new visual query from scratch.

Through that approach, it was possible to keep the focus on different aspects according to the scenario used. On the one hand, the comprehension scenarios give the focus to the query readability, which promote a global exploration of the query components and gave the opportunity to understand if users clearly identified the purpose of each interface section. On the other hand, modification and formulation scenarios were used to understand if users could built queries through the interface presented.

Nevertheless, there was several results to be obtained from the user testing scenarios and it was difficult reach an approach to test all aspects related to the existing problems in a short period. Since the testing requirements presented above must be integrated in the scenarios, a strategy was used to distribute the different requirements among the different scenarios in order to evaluate them all in a fluid and natural way for the user.

Therefore, Table 5.2 was used to assign organize the complexity of each scenario and check what aspects would e tested in each one.

Table 5.2 clarifications regarding the factors integrated in testing scenarios:

- **Simple Joins:** Joins that are automatically generated by the system without requiring human intervention;

- **Complex Joins:** Joins that need to be partially configured manually (e.g., to specify the foreign key used to merge both tables or to change the join condition);

- **Left Join with Null:** Left join between entities A and B, to consider only the entities A that are not related to B (e.g., the employees who have not created any notification.);

- **Group by (without reference):** The system has an automatism that generates automatically a name to a new attribute grouped by. However, this name is not self-explanatory and there was no reference to the source of its attribute. That way, it is important to highlight this case;

- **Aggregation Functions:** The aggregation functions supported by aggregates are Max, Min, Average, Sum, and Count. As the representation in the interface as well as the insertion method is similar, only a few of these were used;

All scenarios are detailed in User Testing Scenarios, including for each scenario, the description of the testing scenario proposed and the main points taken into account throughout the tests performed. Due to the complexity of the exercise proposed, the

Table 5.2: Distribution of the relevant testing aspects among the testing scenarios designed.

Scenarios		C1	M1	C2	C3	F1	C4	M4
Relevant for Comprehension and Formulation	Number of Entities	4	4	6	3	3	5	7
	Simple Joins	2	2	4	2	1	3	3
	Complex Joins	-	-	1	-	1	1	3
	Left Join with Null	1	1	-	-	-	-	-
	Filters	3	2	3	-	-	1	2
	Group Filters	-	-	-	-	1	-	-
	Sorting (Text)	1	1	-	-	-	1	1
	Sorting (Number)	-	-	1	1	-	-	-
	Sorting (Date)	-	-	-	-	-	1	1
	Group By	-	-	1	2	4	-	-
	Group By (without reference)	-	-	-	3	-	-	-
	Max	-	-	-	-	-	-	-
	Min	-	-	-	-	-	-	-
	Average	-	-	-	-	1	-	-
	Sum	-	-	1	-	-	-	-
	Count	-	-	-	1	-	-	-
Relevant for Query Formulation or Modification	Use of not visible columns					X		
	Use of hidden columns					X		
	Insert Calculated Attribute		X					
	Add Aggregation Function					X		
	Add not automatic join					X		X
	Add same entity twice (alias)							X
	Edit some filters		X					

last two scenarios (Comprehension 4 and Modification 4) were not tested with Citizen Developers.

5.3 Evaluation Method

Having all testing scenarios established, it was necessary to plan how to identify the user, evaluate its interaction with the system, and collect qualitative and quantitative results throughout the usability tests.

First of all, users filled a survey in order to perceive what are their background regarding software development, relational databases, and data management and visualization tools. The answers were used to realize what are their user profiles according to the user

groups defined in 4.2.2.

After perceiving what are the most appropriate user group for the user in question, the data model is presented focusing the most relevant aspects for the tests. In that way, it is explained how the entities of the model are related with each other and the attributes most used in scenarios are highlighted.

As soon as they confirm that they got an overview of the existing entities, the testing process starts. However, following users' reasoning and opinions while they interact with the interface was not considered sufficient to collect all results necessary.

Therefore, an evaluation methodology was designed to collect the data required using the same approach for all tested interfaces.

For each user tested, a table, as the example presented in Figure 5.2, was figured out in order to register how much time they required to execute each scenario and how was their effectiveness in the goal achievement. The classification of the effectiveness was made according to Table 5.3.

N	Interviwee	User Group	Scenario	Start Time	End Time	Duration	Total Time	Final Answer	Completely Right?	General Observations
26		Software Developer	Comprehension1	0:05:03	0:07:06	0:02:03	0:19:25	Despite the user didn't identify correctly the merge applied between employees and notifications, he didn't identify all sources and joins as well as sorting criteria. He only gave focus to the filters that were easy to read in his opinion and tried to use that to predict what query is.		He started to analyze the first query looking for relevant data in the query result table, as he didn't find out the top tabs. After he has discovered the tabs he used that to understand the query. However, his focus has kept on the filters because the sources view was overloaded with information that is difficult to read. Furthermore, he had difficulty identifying what are the aggregated attributes presented in the table as well as their source. In terms of formulation, didn't find a new attribute by himself. He needed a clue to search on the table and then he found it. This scenario has been observed again when the user is looking for how to apply an aggregation function or group bys. Add source or add filters were the options tried to add these attributes. Also, he was revealed frustrated when he needs to find attributes in the table, principally when they are hidden.
			Modification1	0:10:03	0:13:45	0:03:42				
			Comprehension2	0:14:25	0:15:25	0:01:00		Even though he referred the foreign keys, he didn't understand what are the bold attributes (attributes grouped by).		
			Comprehension3	0:18:06	0:19:10	0:01:04				
			Formulation1	0:21:21	0:28:38	0:07:17				
			Comprehension4	0:29:20	0:30:45	0:01:25		He didn't refer the foreign key between employee and request (assigned to)		
			Modification4	0:34:17	0:37:11	0:02:54		He didn't manage to conclude this scenario because he didn't understand that need to add allias. As he clicked on "New Join" and not "Add Source", the alias weren't added automatically.		

Figure 5.2: Example a general annotation registered after a usability test.

In addition, a text highlighting the most relevant usability test topics, including users' opinions, reactions, expectations and suggestions has been included for each user. Here is registered the most qualitative result of the usability tests, which is not only important for the design of the next iterations but also to assess user satisfaction.

Furthermore, other important details more specific of each scenario were reported. In the comprehension scenarios, it was assesed the components of the query identified and the means used to perceive it. Regarding the formulation scenarios, the options used to build the query were identified. Even so, it was always recorded when the user manifested difficulty or when exploring other options not supported.

Finally, to gain a more extensive notion and some quantitative metrics about what users felt while they interacted with the interfaces, they were asked to fill in a System Usability Scale (SUS) [71] at the final of the usability test.

This was the process used to test all interfaces addressed in this dissertation: the existing interface of Aggregates, the paper prototype, and the final prototype which was integrated into Service Studio (the Visual IDE of the OutSystems Platform). The next

Table 5.3: Description of the effectiveness states considered.

Legend	Comprehension	Modification	Formulation
Achieved	When the user mentioned all components of the queries (sources, joins with foreign keys, filters, sorting, aggregation functions). The only thing that would not be considered is if the user didn't refer with or without joins when they were put automatically.	All modifications	Completely Right
Partially Achieved	If the user didn't refer to the foreign key between two tables where there is more than one foreign key between two tables. The another possibility is if the user did not specify only the sorting criteria.	Forgot to remove some filter	Group data using the wrong identifier
Not Achieved	Anything else	Anything else	Anything else

chapter will describe the design and implementation process of each prototype built, but also how they were tested using the methodology referred.

6

D ESIGN AND I MPLEMENTATION

The design process adopted was an iterative design approach to build iteratively the solution according to the users' acceptance and feedback retrieved by users test in each iteration. Throughout this chapter, it is detailed all design and implementation decisions considered in order to reach the final prototype.

6.1 Sketching

Before starting the development of the interface prototypes which were tested with users, it was performed some sketches in order to organize and explore ideas to tackle the existing usability problems. The most important to reach in the final of this phase was not a functional or complete interface, but some practical ideas that could be applied in the following prototypes.

Considering the extensive list of problems identified, prioritization was an important aspect taken into account throughout all design phases. The first problem explored was the difficulty to comprehend what is the database query purpose.

In the existing interface, users do not have a unique and clear view that facilitates the comprehension of what data could be fetched from the database through the query presented. As illustrated in Figure 6.1, the user can only see some components of the query at a time, due to the usage of tabs to individualize each type of query components. In addition, as can be observed in Figure 6.1a, when users open the query, only the query output preview was shown. That was considered a problem in the analysis of the interface for two different points of view:

- When the existing interface was tested with users who do not usually use the Plat-
  form, they do not easily find out the existence of the tabs and tried to perceive
  the query purpose only observing the data preview. Besides not being an effective

(a) Starting point (only the result is visible)

(b) Sources Tab

(c) Filters Tab

(d) Sorting Tab

Figure 6.1: Example of a database query representation through the existing visual interface.

understanding technique, it can be even more difficult when the output contains several columns;

- The users accustomed to this data tool revealed that is cumbersome to open and navigate between tabs in order to understand the query.

Therefore, the designing of a new general layout where it is possible to view the most important query components at once was considered a major requirement. A solid improvement regarding that, could optimize not only the time and effort required to comprehend queries but also to formulate them, since all information is more visible and accessible.

There was also a concern to build a layout that does not compromise the system usability for more complex cases, such as queries that contain a relevant number of entities or conditions.

Accordingly, some wireframes were sketched in order to percept how the query components could be jointly combined in a unique view. Figure 6.2 illustrates the two options elected from multiple approaches explored.

In both options, there are two principal areas in the interface as it was before: the query editor area and the preview of the query result. However, it was explored two new manners to display all information of editors jointly with the query result preview. In

(a) Option A   (b) Option B

Figure 6.2: Interface layout sketches.

Option A, the sub-editors remain in the top area and the query result preview below. The Option B illustrates another possibility sketched where all editors are presented on the left side of the screen and the visualization of the results are presented on the right side.

Regardless of the option, the layouts presented have in a single view the most important aspects to understand what query is built.

Beyond the disposition of the different interface views and components, other features were sketched in order to think more about it and structure concrete ideas to implement it.

The existing interface has no mechanism that allows the user to easily find the data of an entity or attribute. The attributes included in the query were not represented in any region of the interface beyond the query result table. Thereby, the unique way to find out the attribute is looking for in each table header, using the horizontal scroll, until the intended attribute is found. That process is cumbersome and slow, so that there was sketched an alternative that includes the attributes in the sources view. Consequently, users can click on attributes and the attribute will be highlighted automatically in the query result preview, as illustrated in Figure 6.3. Also, the idea of a search engine was considered, so that users can search for an entity or attribute faster.

Furthermore, different approaches more compact and functional were explored to display the joins used in the query. Figure 6.4 shows the idea sketched to represent joins in three different lists: a simple list of all join operations inside the query, and two other ones aggregated by the entities involved by the join kind.

Lastly, it was sketched an idea to accelerate the searching process of a specific element inside the query. Therefore, the sketch represented in Figure 6.5 demonstrates an idea taken into account to find all references of an entity. This general search could allow users to find entities, joins, filters, or other query elements faster.

Although the sketched ideas may not be applied to the final prototype since they depend on the whole iterative design process that started afterward, they were important
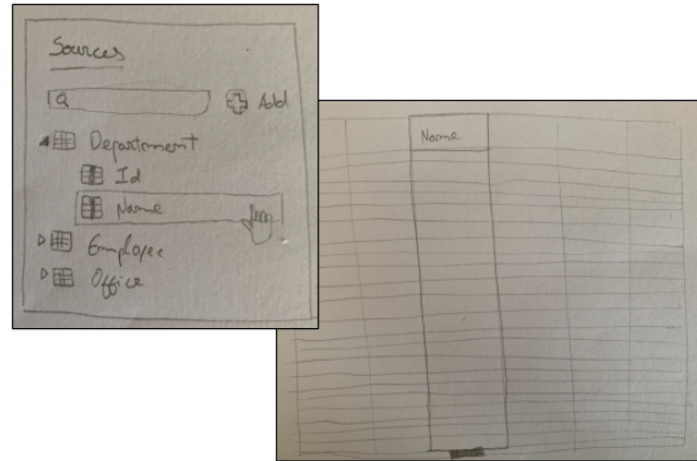
Figure 6.3: Searching for an attribute data on the query output preview.
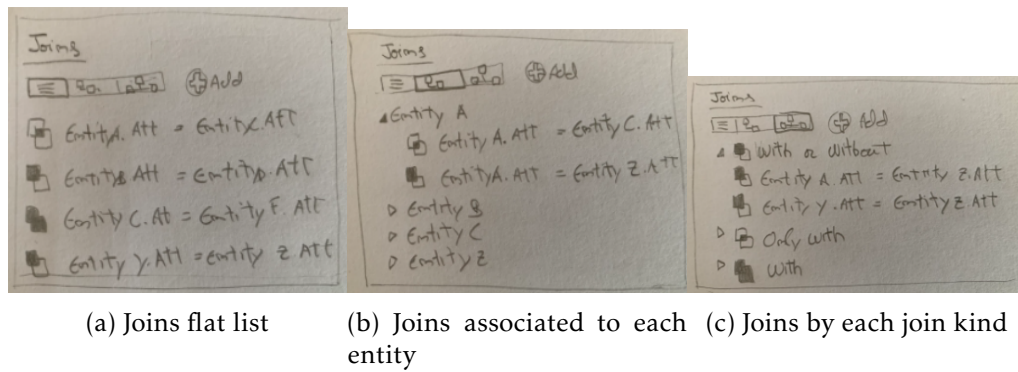
(a) Joins flat list   (b) Joins associated to each entity   (c) Joins by each join kind

Figure 6.4: Sketches elaborated to explore other approaches to represent the join operations present in the query.

to move the focus from problem definition to solution design.

6.2 Paper Prototype

The interactive design process of the solution started in its entirety with the first prototype developed: the paper prototype. The main goal of this phase is the building of a functional prototype implemented in paper using ruler, square, and writing materials, in order to create faster and with a low-risk level a prototype that could be testes by users.

However, due to the COVID-19 pandemic, the prototype was adapted since it was not possible to test the prototype in person. Accordingly, the prototype was scanned and the interactions were configured using the digital product design platform InVision [46].

6.2.1 Design

The building process of this low-fidelity prototype started with a design phase where brainstorming took place in order to establish the design priorities for the paper prototype
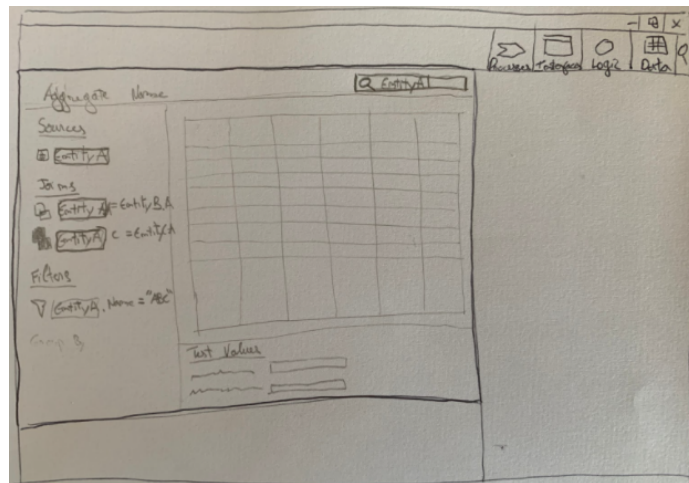
Figure 6.5: Sketch of a general search to allow users to find query elements represented in the interface.

as well as concrete ideas to apply the solutions thought.

Sub-editors arrangement:

Taking into account the sketches mentioned in the last section, the first question evaluated was which rearrangement of the sub-editors and the result preview views would be implemented in the next phase. In order to take action in this regard, there was a point in the visual query builder background considered. As mentioned in 2.2.2.1, this querying interface was completely redesigned seven years ago, which had a negative repercussion on users since they were accustomed to the previous interface. Therefore, there was a concern to change the interface without removing the main points that characterize and identifies it. In such a way that users would not feel using a completely new interface but an improved version of the existing one.

Comparing the two options sketched (Figure 6.2) with the existing interface (Figure 6.1), the Option A is the most similar because the editors area keeps on the top region of the interface and the query result data below. For this reason, this arrangement would be considered in the next phase.

Having chosen where would take place the edition area of the interface, the second aspect taken into consideration was how to organize its sub-editors. This decision was taken bearing in mind what each query aspect represents in the user's mental model when they formulate queries as well as the physical space of the interface they could occupy.

In this regard, it was thought about what are the order of the query elements that arise in user thinking. This reasoning was performed taking into consideration the formulation using SQL since one of the main goals is the improvement of the interface experience for users that are proficient in SQL. According to the conceptual models presented in 3.1, the first aspect the user thinks, after understanding what data is required, is how to translate them to the query language. In SQL, the core statements are presented in the following order:

1. **SELECT:** Indicates which attributes will be selected to be present in the query output table;

2. **FROM:** Sets out which entities are used to query data. Thereby, even it is necessary to merge tables, the join operations are specified through that statement;

3. **WHERE:** Contains the boolean conditions that would filter the results.

4. **ORDER BY:** Specify the criteria to order the data gathered.

In this visual query building tool users do not select attributes because there is an optimizing background task that will inspect where the query is used and only select the attributes that will be used. Therefore, in this system, the information specified through the SELECT statement will not be specified by the user.

However, the three other aspects of the query are clearly specified by users in three different interface areas: Sources, Filters, and Sorting respectively. Accordingly, the arrangement chosen to display these three sub-editors follows this logic. The query edition area were divided into two columns. The left side was elected to represent the sources of the query and the right side lists the filters and sorting criteria.

New approach to represent sources and joins:

Nevertheless, the way entities and joins were presented in the interface, in the previous sources tab, required to be redesigned from scratch. As can be observed in Figure 6.1b, a simple list of the entities used was presented in the left side of the editor area and the joins used to merge them in its right side.

Even though some designs regarding that were elaborated in the sketching phase (Figure 6.4), it was concluded that the main issues remain:

- Difficulty to comprehend faster and with a low-effort what entities were integrated into the query. Being a visual interface, the comprehension of the entities used as source to formulate the query should be easier to understand. However, the potential of the visual interface has not been leveraged, mainly due to the following aspects:

Therefore, the way entities and joins were listed in the tab sources of the previous interface had to be rethought since it was intended to create a simpler, intuitive, and compact viewing area.

6.2.2   Implementation

6.2.3   Evaluation

6.3   Service Studio Implementation

6.3.1   Design

6.3.2   Implementation

6.3.3   Evaluation

CONCLUSIONS AND FUTURE WORK

7.1 Results Analysis

7.2 Future Work

7.3 Conclusion

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] R. Geng, M. Chen, and J. Tian. "In-process Usability Problem Classification, Analysis and Improvement." In: *2014 14th International Conference on Quality Software*. 2014, pp. 240–245.

[12] 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.

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

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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).

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

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

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

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

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

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

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

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

[38] 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).

[39] 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).

[40] 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).

[41] 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).

[42] 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).

[43] 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).

[44] 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).

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

[46] *InVision | Digital product design, workflow and colaboration*. URL: https://www.invisionapp.com/ (visited on 10/21/2020).

[47] 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).

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

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

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

[51] 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).

[52] 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).

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

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

[55] J. Nielsen. *10 Usability Heuristics for User Interface Design*. 1994. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (visited on 05/05/2020).

[56] J. Nielsen. *Why You Only Need to Test with 5 Users*. 2000. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ (visited on 10/14/2020).

[57] J. Nielsen. *How Many Test Users in a Usability Study?* 2012. URL: https://www.nngroup.com/articles/how-many-test-users/ (visited on 10/14/2020).

[58] OutSystems. *Advanced Aggregates Course - Training - OutSystems*. URL: https://www.outsystems.com/learn/courses/132/advanced-aggregates/?LearningPathId=18 (visited on 10/30/2019).

[59] OutSystems. *Aggregates 101 Course - Training - OutSystems*. URL: https://www.outsystems.com/learn/courses/126/aggregates-101/?LearningPathId=18 (visited on 10/28/2019).

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

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

[62] OutSystems. *OutSystems Community*. URL: https://www.outsystems.com/community/ (visited on 05/06/2020).

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

[64] 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).

[65] OutSystems. *What's new in OutSystems Hub Edition 2.2*. 2004. URL: https://drive.google.com/file/d/0B7C37RyL27_oNHF4UmFRX3pFM1pMTV1CWnV5dzJCcmhRS2tj/view (visited on 02/03/2020).

[66] 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).

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

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

[69] 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).

[70] 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).

[71] *System Usability Scale (SUS)*. URL: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html (visited on 10/20/2020).

[72] 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).

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

[74] 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).

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

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

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

[78] 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).

Taxonomy of Problems - Existing Interface

The visual interface to formulate queries that was implemented before this dissertation had several usability problems which led users to prefer to use other DQLs such as SQL. In order to comprehend the existing problems of the visual interface or the characteristics that hamper users to extract advantages of that visual approach to query databases, there were performed the following studies:

- **Study and Analysis:** This was the first approach used to explore the existing problems of the interface. The process started with the visualization of two OutSystems tutorials [58, 59] about visual data querying. This was the first experience using the interface and there were pointed out some problems. As this was the first experience the were found principally issues regarding hidden operations and behaviors that were not clear for users who didn't use to the platform. After that tutorials, there has been made some more explorations using practical examples where there were found other problems related to efficiency and effectiveness of use;

- **User Interviews:** There were performed some dialogues with a reduced set of novice and expert users to comprehend what are the main issues pointed out. In the case of expert users, the causes to use SQL instead of the Visual Interface were registered as well as a set of other UX issues. Regarding users who did not have relevant experience either in SQL or OutSystems, it was registered the functionalities more difficult to learn or understand;

- **Community Ideas:** Since OutSystems contains a wide and worldwide Community where users could contribute with suggestions or express their problems or difficulties, all the ideas of the category "Aggregates and Queries"were explored to understand the problems presented. Moreover, the number of likes of each post was registered in order to know what are the problems which had more user reactions;

- **User Testing:** During the user tests of the existing implementation, there were pointed out some issues by users. Also, other problems were found through the interpretation of user-interface interaction.

Furthermore, each issue registered was characterized according to the Nielsen Heuristics [55] and the artifact and task attributes of a Framework adapted from Usability-ODC Framework [11]. Besides, for each issue it was checked if the action most hampered is the query formulation or the query comprehension as well as what are the interface components affected between the ones presented below:

- Actions and Nodes (Method/Function where the Visual Query was added)

- General Layout (The arrangement of the main interface components on the main window)

- Sources

- Joins

- Filters

- Sorting

- Aggregation Functions

- Calculated Attributes

- Static Entities

- Query Result Table

- Test Values

In that way, it is simple to categorize and prioritize the problems detected. Besides, as mentioned above, if the problem were referred in the OutSystems Community, there were included the respecting posts and their number of likes. Table X represents the result of all problems detected and categorized under the parameters mentioned:

Table A.1: Taxonomy of Aggregates' Problems - Existing Interface (Last community posts update: May 06, 2020)

Taxonomy of Aggregates' Problems		
ID	**Issue**	**Description**
1	**Entity misadded (automatic joins deletion)**	**When the user adds an entity that trigger automatically a new join, if this entity is removed, the entity could also be removed.**
Figured Out	Community Ideas	
Interface Components Related	Sources and Joins	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of Information/Results	
Task Category	Task-facilitation	
Task Subcategory	User Error Tolerance	
Nielsen Heuristics	User Control and Freedom	
Community Posts		
Post		Likes
Changes to behaviour of adding Entities to an Aggregate[1]		25
2	**Select multiple sources at once**	**There are some use cases where this is already possible. However, it should be possible in any use case where the user can add multiple entities.**
Figured Out	Study and Analysis, User Interviews, Community Ideas, and User Testing	
Interface Components Related	Sources	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Direct Manipulation	
Continued on next page		

---

[1] https://www.outsystems.com/ideas/2890/changes-to-behaviour-of-adding-entities-to-an-aggregate

Continuation of Table A.1		
ID	**Issue**	**Description**
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use, and Consistency and standards	
Community Posts		
Post		Likes
Changes to behaviour of adding Entities to an Aggregate[2]		25
Allow multiple entity selection on first screen of new aggregate[3]		6
Replace data with multiple entities[4]		1
3	**Add the same entity more than once**	**Allow drag/dropping the same entity more than once (It's possible using Select Source pop-up but it's not possible using Drag and Drop.**
Figured Out	Study and Analysis, and Community Ideas	
Interface Components Related	Sources	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Direct Manipulation	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use, and User control and freedom	
Community Posts		
Post		Likes
Changes to behaviour of adding Entities to an Aggregate[5]		25
Continued on next page		

---

[2] https://www.outsystems.com/ideas/2890/changes-to-behaviour-of-adding-entities-to-an-aggregate

[3] https://www.outsystems.com/ideas/5417/allow-multiple-entity-selection-on-first-screen-of-new-aggregate

[4] https://www.outsystems.com/ideas/7568/replace-data-with-multiple-entities

[5] https://www.outsystems.com/ideas/2890/changes-to-behaviour-of-adding-entities-to-an-aggregate

Continuation of Table A.1		
ID	**Issue**	**Description**
4	**Add automatically a static entity even if it is not mandatory**	**"It's very annoying that every time I drag to an aggregate an entity with a foreign key to a static entity (SE) - even if it's not mandatory - that static entity is dragged along with it."**
Figured Out	Community Ideas	
Interface Components Related	Sources, and Static Entities	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Direct Manipulation	
Task Category	Task-facilitation	
Task Subcategory	Task/function automation	
Nielsen Heuristics	Consistency and standards	
Community Posts		
Changes to behaviour of adding Entities to an Aggregate[6]		25
5	**Join tables with more than one identifier of the same table (other table)**	**When an entity has several identifiers of same other entity, do the join like currently but with a warning or ask which field to do the join.**
Figured Out	Study and Analysis, Community Ideas, and User Testing	
Interface Components Related	Joins	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	No-message feedback	
Continued on next page		

---

[6]https://www.outsystems.com/ideas/2890/changes-to-behaviour-of-adding-entities-to-an-aggregate

ID	Issue	Description
Continuation of Table A.1		
Task Category	Task-facilitation	
Task Subcategory	Keeping the user task on track	
Nielsen Heuristics	User control and freedom, and Visibility of System Status	
Community Posts		
Post		Likes
Changes to behaviour of adding Entities to an Aggregate[7]		25
Join tables with more than one identifier of same other table[8]		5
6	**Move multiple aggregate columns at the same time**	**Actually it is possible to reorder one column. Extend this behavior.**
Figured Out	Community Ideas, and User Testing	
Interface Components Related	Query Result Table	
Main Action Hampered	Query Formulation and Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Direct Manipulation	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Flexibility and efficiency of use	
Community Posts		
Post		Likes
Move multiple aggregate columns at the same time[9]		5
7	**Search and focus for particular columns of the query result**	**There should be an intuitive solution to navigate through the query result columns. That solution should include search fields and vertical scroll lists to improve the user navigation and control.**
Continued on next page		

---

[7]https://www.outsystems.com/ideas/2890/changes-to-behaviour-of-adding-entities-to-an-aggregate

[8]https://www.outsystems.com/ideas/2885/aggregate-join-tables-with-more-than-one-identifier-of-same-othe...

[9]https://www.outsystems.com/ideas/6590/move-multiple-aggregate-columns-at-the-same-time

Continuation of Table A.1		
ID	**Issue**	**Description**
Figured Out	Study Analysis, User Interviews, Community Ideas, and User Testing	
Interface Components Related	Sources and Query Result Table	
Main Action Hampered	Query Formulation and Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Direct Manipulation	
Task Category	Task-mapping	
Task Subcategory	Navigation	
Nielsen Heuristics	Flexibility and efficiency of use, and Visibility of System Status	
Community Posts		
Post	Likes	
Improve Aggregates to allow entity/attribute filtering[10]	18	
Aggregates improvement - Visual Filters[11]	32	
Ability to search attributes in aggregates while grouping[12]	5	
Search columns in Aggregate[13]	0	
Focus Aggregate Column[14]	2	
Aggregate Group By Tab[15]	19	
8	**Expand hidden columns.**	
Figured Out	Analysis and Study, User Interviews, Community Ideas, and User Testing	
Interface Components Related	Query Result Table	
Main Action Hampered	Query Formulation and Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-mapping	
Continued on next page		

---

[10] https://www.outsystems.com/ideas/5720/service-studio-improve-aggregates-to-allow-entity-attribute-filte
[11] https://www.outsystems.com/ideas/5955/aggregates-improvement-visual-filters
[12] https://www.outsystems.com/ideas/5904/ability-to-search-attributes-in-aggregates-while-grouping
[13] https://www.outsystems.com/ideas/6826/search-columns-in-aggregate
[14] https://www.outsystems.com/ideas/6537/focus-aggregate-column
[15] https://www.outsystems.com/ideas/7322/aggregate-group-by-tab

Continuation of Table A.1		
ID	**Issue**	**Description**
Task Subcategory	Functionality	
Nielsen Heuristics	Flexibility and efficiency of use, and Consistency and standards	
Community Posts		
Post		Likes
Expand hidden columns[16]		8
9	**Hide multiple columns with right-click**	**Actually users can use table of query results to hidden some columns. In addition, they can right-click in an attribute column and select hide. However, if they select multiple columns and right-click in one of them, only one will be selected.**
Figured Out	Analysis and Study, Community Ideas, and User Testing	
Interface Components Related	Query Result Table	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-mapping	
Task Subcategory	Interaction	
Nielsen Heuristics	Consistency and standards	
Community Posts		
Post		Likes
Ability to select multiple columns in an aggregate for hiding them[17]		1
10	**If an attribute is lengthy the user cannot see the entry.**	
Figured Out	Community Ideas	
Continued on next page		

---

[16]https://www.outsystems.com/ideas/5054/aggregate-expand-hidden-columns
[17]https://www.outsystems.com/ideas/7102/ability-to-select-multiple-columns-in-an-aggregate-for-hiding-th

Continuation of Table A.1		
ID	**Issue**	**Description**
Interface Components Related	Query Result Table	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-mapping	
Task Subcategory	Interaction	
Nielsen Heuristics	User control and freedom, and Visibility of System Status	
Community Posts		
Post		Likes
Adjust the columnsize in aggregates query data result[18]		2
Improve cell options for aggregate tables[19]		5
11	**Search and navigate between values**	**Actually users can only select columns. They cannot select rows or cells.**
Figured Out	Community Ideas	
Interface Components Related	Query Result Table	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Direct Manipulation	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Flexibility and efficiency of use	
Community Posts		
Post		Likes
Position cursor on desired field within an aggregate[20]		1
Improve cell options for aggregate tables[21]		5
Continued on next page		

---

[18]https://www.outsystems.com/ideas/7027/adjust-the-columnsize-in-aggregates-query-data-result
[19]https://www.outsystems.com/ideas/6009/service-studio-improve-cell-options-for-aggregate-tables
[20]https://www.outsystems.com/ideas/7290/position-cursor-on-desired-field-within-an-aggregate
[21]https://www.outsystems.com/ideas/6009/service-studio-improve-cell-options-for-aggregate-tables

ID	Issue	Description
Continuation of Table A.1		
Copy a value from the Aggregate preview data[22]		27
In an aggregate you can't copy (Ctrl-C) values[23]		3
12	**It's not possible to see all the results (disable remaining results).**	
Figured Out	Community Ideas	
Interface Components Related	Query Result Table	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Visibility of System Status, and Match between system and the real world	
Community Posts		
Post		Likes
In aggregates I should have an option to see all results in DEV mode[24]		18
Don't always truncate aggregate results[25]		1
Ability to switch off 'remaining results truncated'[26]		2
13	**Provide more shortcuts.**	
Figured Out	Community Ideas	
Interface Components Related	General Layout, Sources, Joins, Filters, Sorting, Aggregation Functions, Calculated Attributes, Query Result Table, and Test Values	
Main Action Hampered	Query Formulation and Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Keyboard press	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Continued on next page		

---

[22]https://www.outsystems.com/ideas/2828/copy-a-value-from-the-aggregate-preview-data

[23]https://www.outsystems.com/ideas/7015/in-an-aggregate-you-cant-copy-ctrl-c-values

[24]https://www.outsystems.com/ideas/3052/in-aggregates-i-should-have-an-option-to-see-all-results-in-dev-

[25]https://www.outsystems.com/ideas/8097/dont-always-truncate-aggregate-results

[26]https://www.outsystems.com/ideas/6824/ability-to-switch-off-remaining-results-truncated

Continuation of Table A.1		
ID	**Issue**	**Description**
Nielsen Heuristics	Flexibility and efficiency of use	
Community Posts		
Post		Likes
Aggregate Editor - Shortcut Keys - Move Column Left and Right[27]		5
14	**Present number of results (rows) of the query result.**	
Figured Out	User Interviews and Community Tests	
Interface Components Related	Query Result Tables	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Visibility of System Status	
Community Posts		
Post		Likes
Record count in Aggregates[28]		7
Display number of rows returned for an aggregate[29]		2
15	**Invert members of join operation**	**As we only have left join (with or without) it would be interesting one button that changes the entity in the left side of the join with the one that is in the right side.**
Figured Out	Community Ideas and User Testing	
Interface Components Related	Joins	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Continued on next page		

---

[27] https://www.outsystems.com/ideas/3322/aggregate-editor-shortcut-keys-move-column-left-and-right

[28] https://www.outsystems.com/ideas/1780/record-count-in-aggregates

[29] https://www.outsystems.com/ideas/6825/display-number-of-rows-returned-for-an-aggregate

Continuation of Table A.1		
ID	**Issue**	**Description**
Artifact Subcategory	Mouse click	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use	
Community Posts		
Post		Likes
Aggregate: swap join entities[30]		9
16	**Improve joins readability**	**Change the visual representation of joins, because could be difficult to users to understand what are the tables joined in the query, principally if there is some nesting.**
Figured Out	Community Ideas and User Testing	
Interface Components Related	Joins	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Object appearance	
Task Category	Task-mapping	
Task Subcategory	Interaction	
Nielsen Heuristics	Visibility of System Status	
Community Posts		
Post		Likes
Aggregates: make them smarter (nested joins)[31]		14
Continued on next page		

---

[30] https://www.outsystems.com/ideas/1870/aggregate-swap-join-entities
[31] https://www.outsystems.com/ideas/1846/aggregates-make-them-smarter-nested-joins

Continuation of Table A.1		
ID	**Issue**	**Description**
17	**Delete last joins automatically added if a user want to remove last entity added**	**When an entity is added, the system analyzes the relationships with other tables. In some cases the system put automatically other entities related with the added. But if the user removes the entity that he had added, the joins automatically added, remain in the aggregate.**
Figured Out	Community Ideas	
Interface Components Related	Sources and Joins	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Object appearance	
Task Category	Task-facilitation	
Task Subcategory	Keeping the user task on track	
Nielsen Heuristics	Consistency and standards, and Match between system and the real world	
Community Posts		
Post		Likes
Delete joins automatically when we delete an entity[32]		22
18	**Copy and past filters**	**Could be more efficient to user, to generate a similar filter.**
Figured Out	Community Ideas and User Testing	
Interface Components Related	Filters	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Continued on next page		

---

[32]https://www.outsystems.com/ideas/8104/aggregates-delete-joins-automatically-when-we-delete-an-entity

ID	Issue	Description
Continuation of Table A.1		
ID	**Issue**	**Description**
Artifact Subcategory	Keyboard press	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use, and Consistency and standards	
Community Posts		
Post		Likes
Ability to copy and paste Filters in Aggregates (including in bulk)[33]		2
19	**Copy and past joins**	**Could be more efficient to user, to generate a similar join.**
Figured Out	Analysis and Study	
Interface Components Related	Joins	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Keyboard press	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use, and Consistency and standards	
20	**Comments in Filters**	**Improve the highlighting of comments. It doesn't matter if the commentls will be showed separated or if the comments would have a different color. Just examples...**
Figured Out	Community Ideas	
Interface Components Related	Filters	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Continued on next page		

---

[33]https://www.outsystems.com/ideas/6627/ability-to-copy-and-paste-filters-in-aggregates-including-in-bull

Continuation of Table A.1		
ID	**Issue**	**Description**
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-facilitation	
Task Subcategory	Keeping the user task on track	
Nielsen Heuristics	Match between system and the real world	
Community Posts		
Post		Likes
Comments and Activate/Deactivate on Aggregates (Filters, Joins)[34]		13
Comments on aggregate filter[35]		6
Comments inside Aggregates[36]		7
21	**Disable filters**	**Option to disable filter's effect.**
Figured Out	Community Ideas	
Interface Components Related	Filters	
Main Action Hampered	Query Formulation and Query COmprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Mouse click	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	User control and freedom	
Community Posts		
Post		Likes
Comments and Activate/Deactivate on Aggregates (Filters, Joins)[37]		13
Aggregate Filter option[38]		9
Temporarily disable aggregate Filters[39]		2
Allow to disable (not remove) aggregate filters[40]		5
Continued on next page		

[34] https://www.outsystems.com/ideas/2058/comments-and-activate-deactivate-on-aggregates-filters-joins
[35] https://www.outsystems.com/ideas/7664/comments-on-aggregate-filter
[36] https://www.outsystems.com/ideas/2891/comments-inside-aggregates
[37] https://www.outsystems.com/ideas/2058/comments-and-activate-deactivate-on-aggregates-filters-joins
[38] https://www.outsystems.com/ideas/1765/aggregate-filter-option
[39] https://www.outsystems.com/ideas/7170/temporarily-disable-aggregate-filters
[40] https://www.outsystems.com/ideas/7565/allow-to-disable-not-remove-aggregate-filters

Continuation of Table A.1		
ID	**Issue**	**Description**
22	**Turn filter edition more accessible**	**Change aggregate filter without openning aggregate.**
Figured Out	Community Ideas	
Interface Components Related	Filters	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use	
Community Posts		
Post		Likes
Change Aggregate filter without opening Aggregate[41]		23
23	**Change filter without necessity to open the modal of expression editor**	**Turn the insertion and edition process more faster.**
Figured Out	Analysis and Study	
Interface Components Related	Filters	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Direct Manipulation	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use	
24	**Increase the readability of filters (text highlighting) withou open expression editor modal.**	
Figured Out	Analysis and Study	
Interface Components Related	Filters	
Main Action Hampered	Query Comprehension	
Continued on next page		

---

[41] https://www.outsystems.com/ideas/3309/change-aggregate-filter-without-opening-aggregate

90

	Continuation of Table A.1	
ID	**Issue**	**Description**
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-facilitation	
Task Subcategory	Keeping the user task on track	
Nielsen Heuristics	Match between system and the real world	
25	**Remove aggregate filter if rule is empty.**	
Figured Out	Analysis and Study, Community Ideas, and User Testing	
Interface Components Related	Filters	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-facilitation	
Task Subcategory	Task/function automation	
Nielsen Heuristics	Consistency and standards	
Community Posts		
Post		Likes
Remove Aggregate filter if rule is empty[42]		10
26	**Bin button to delete filter is not visible.**	
Figured Out	Community Ideas	
Interface Components Related	Filters	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Screen layout	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	User control and freedom, and Recognition rather than recall	
Community Posts		
Post		Likes
	Continued on next page	

---

[42] https://www.outsystems.com/ideas/5056/remove-aggregate-filter-if-rule-is-empty

ID	Issue	Description
Continuation of Table A.1		
Don't make me scroll to delete filters[43]		8
Move delete filter button to be near the actual filter in an aggregate[44]		3
27	**Group by attributes' names are not perceptive**	**For example if two different entities have attributes called "Name"and a user group by this two attribuytes, they will be presented as "Name1"and "Name2". Moreover, there is not any reference to attribute's entity.**
Figured Out	Analysis and Study, Community Ideas, and User Testing	
Interface Components Related	Aggregation Functions	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Language	
Artifact Subcategory	Nameling/labeling	
Task Category	Task-mapping	
Task Subcategory	Interaction	
Nielsen Heuristics	Visibility of system status, and Error prevention	
Community Posts		
Post	Likes	
Set attribute name when 'Group by Id' to Entity-NameId[45]		4
28	**View area of group bys.**	**There is no list view with all aggregated attributes, in order to improve the efficiency of query comprehension.**
Continued on next page		

---

[43] https://www.outsystems.com/ideas/5686/dont-make-me-scroll-to-delete-filters

[44] https://www.outsystems.com/ideas/7145/move-delete-filter-button-to-be-near-the-actual-filter-in-an-agg

[45] https://www.outsystems.com/ideas/6456/set-attribute-name-when-group-by-id-to-entitynameid

Continuation of Table A.1		
ID	**Issue**	**Description**
Figured Out	Analysis and Study, Community Ideas, and User Testing	
Interface Components Related	Aggregation Functions	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-mapping	
Task Subcategory	Navigation	
Nielsen Heuristics	Visibility of system status	
Community Posts		
Post		Likes
Aggregate Group By Tab[46]		19
29	**When the user clicks in "add attribute"(calculated attribute), set the cursor automatically to value**	**This is useful to turns the process of adding faster, reducing the user's effort, increasing the pleasant of use.**
Figured Out	Analysis and Study, and Community Ideas	
Interface Components Related	Calculated Attributes	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-facilitation	
Task Subcategory	Keeping the user task on track	
Nielsen Heuristics	Consistency and standards, and Recognition rather than recall	
Community Posts		
Post		Likes
Set cursor to 'value' after adding new attribute to an aggregate[47]		1
Continued on next page		

---

[46]https://www.outsystems.com/ideas/7322/aggregate-group-by-tab
[47]https://www.outsystems.com/ideas/7528/set-cursor-to-value-after-adding-new-attribute-to-an-aggregate

Continuation of Table A.1		
ID	**Issue**	**Description**
30	**Inconsistent data formats between filters and test values**	**"The test values for Aggregate filters requires YYYY-MM-DD, but data in the results is displayed as MM/DD/YYYY. If you enter MM/DD/YYYY into the test values, you get an error."**
Figured Out	Community Ideas	
Interface Components Related	Filters	
Main Action Hampered	Query Formulation and Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Language	
Artifact Subcategory	Nameling/labeling	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Consistency and standards	
Community Posts		
Post		Likes
Allow date entry into test values in same format as results are displayed[48]		4
31	**Inconsistent of sorting between aggregate properties and aggregate popup on action.**	
Figured Out	Community Ideas	
Interface Components Related	Actions and Nodes	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Consistency and standards, and Flexibility and efficiency of use	
Continued on next page		

---

[48]https://www.outsystems.com/ideas/4985/allow-date-entry-into-test-values-in-same-format-as-results-are-

Continuation of Table A.1		
ID	**Issue**	**Description**
Community Posts		
Post		Likes
Consistency between Aggregate properties and popup info balloon when hovering[49]		10
32	**Aggregates' Search engine.**	**General query search box.**
Figured Out	Analysis and Study, and Community Ideas	
Interface Components Related	General Layout, Sources, Joins, Filters, Sorting, Aggregation Functions, Calculated Attributes, and Query Result Table	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Keyboard press	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Flexibility and efficiency of use, and Recognition rather than recall	
Community Posts		
Post		Likes
Aggregate - Highlight/Find Usage by Source[50]		4
33	**Freeze formulation areas while scroll horizontal in query result table.**	
Figured Out	Analysis and Study, User Interviews, and Community Ideas	
Interface Components Related	General Layout	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Screen layout	
Task Category	Task-mapping	
Task Subcategory	Navigation	
Nielsen Heuristics	Visibility of system status	
Continued on next page		

[49]https://www.outsystems.com/ideas/5740/consistency-between-aggregate-properties-and-popup-info-balloon-v

[50]https://www.outsystems.com/ideas/3412/aggregate-highlight-find-usage-by-source

	Continuation of Table A.1	
ID	**Issue**	**Description**
Community Posts		
Post		Likes
Freeze Sources, Filters, Sorting and Test Values in AG-GREGATE[51]		10
34	**Duplicate tables.**	
Figured Out	Community Ideas	
Interface Components Related	Sources	
Main Action Hampered	Query Formualtion	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Keyboard press	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use	
Community Posts		
Post		Likes
Aggregates: Duplicate Tables / Joins[52]		1
35	**Duplicate joins.**	
Figured Out	Community Ideas	
Interface Components Related	Joins	
Main Action Hampered	Query Formualtion	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Keyboard press	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use	
Community Posts		
Post		Likes
Aggregates: Duplicate Tables / Joins[53]		1
36	**Duplicate filters.**	
Figured Out	Analysis and Study	
	Continued on next page	

---

[51] https://www.outsystems.com/ideas/5935/freeze-sources-filters-sorting-and-test-values-in-aggregate
[52] https://www.outsystems.com/ideas/7582/aggregates-duplicate-tables-joins
[53] https://www.outsystems.com/ideas/7582/aggregates-duplicate-tables-joins

Continuation of Table A.1		
ID	**Issue**	**Description**
Interface Components Related	Filters	
Main Action Hampered	Query Formualtion	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Keyboard press	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use	
Aggregates: Duplicate Tables / Joins[54]	1	
37	**Distinct function**	**Without extending aggregates' expressiveness is the "Group by of all columns of the result".**
Figured Out	Analysis and Study, and Community Ideas	
Interface Components Related	Query Result Table	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Language	
Artifact Subcategory	Nameling/labeling	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Consistency and standards, and Match between system and the real world	
Community Posts		
Post	Likes	
Aggregate with Distinct[55]	107	
38	**Count Distinct function**	**Expressiveness problem (distinct in all could be "replaced"by group bys, but COUNT DISTINCT cannot be reached using group bys).**
Continued on next page		

[54]https://www.outsystems.com/ideas/7582/aggregates-duplicate-tables-joins

[55]https://www.outsystems.com/ideas/Idea_View.aspx?IdeaID=2179&IdeaName=aggregate-with-distinct&utm_source=community&utm_medium=email&utm_campaign=idea-comment

ID	Issue	Description
	Continuation of Table A.1	
Figured Out	Community Ideas	
Interface Components Related	Aggregation Function, and Query Result Table	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Flexibility and efficiency of use	
Community Posts		
Post		Likes
Aggregate: Count Distinct[56]		13
39	**Drag and drop input parameters to automatically create filters into Aggregates.**	
Figured Out	Community Ideas	
Interface Components Related	Actions and Nodes, and Filters	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Object movement	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use	
40	**Button to clear all the test values inserted**	**It could be a good resource if users have many values inserted and they want to clear all. It could be very annoying if they need to clear one at a time.**
Figured Out	Analysis and Study	
Interface Components Related	Test Values	
	Continued on next page	

---

[56]https://www.outsystems.com/ideas/1889/aggregate-count-distinct

Continuation of Table A.1		
ID	**Issue**	**Description**
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Mouse click	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Flexibility and efficiency of use	
41	**Add variable into aggregate puts again the entity already inserted**	**When dragging a variable of type identifier (e.g. CompanyID) the entity Company is added even if it was added before. The only thing that it should add is the filter (if there isn't one already).**
Figured Out	Analysis and Study	
Interface Components Related	Actions and Nodes, and Sources	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Object movement	
Task Category	Task-facilitation	
Task Subcategory	Keeping the user task on track	
Nielsen Heuristics	User control and freedom	
42	**Query compreension at a glance**	**To comprehend what data the query will retrive (query structure) the user need to switch between tabs.**
Figured Out	Analysis and Study, and User Interviews	
Interface Components Related	General Layout	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Continued on next page		

Continuation of Table A.1		
ID	**Issue**	**Description**
Artifact Category	Representation	
Artifact Subcategory	Screen layout	
Task Category	Task-mapping	
Task Subcategory	Navigation	
Nielsen Heuristics	Visibility of system status	
43	**"Hide others"bug in group by**	**When someone clicks on "hide others" above an aggregation column, there is no visible effect (gray columns stay visible).**
Figured Out	Analysis and Study	
Interface Components Related	Aggregation Functions, and Query Result Table	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Flexibility and efficiency of use	
44	**Highlight new additions (feedback)**	**Highlight elements added in last interaction. For example sources, joins, group bys sorts, etc...**
Figured Out	Analysis and Study	
Interface Components Related	Sources, Joins, Filters, Sorting, Aggregation Functions, Calculated Attributes, Static Entities, and Query Result Table	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	No-message feedback	
Task Category	Task-facilitation	
Task Subcategory	Keeping the user task on track	
Nielsen Heuristics	Visibility of system status	
Continued on next page		

ID	Issue	Description
45	**Turn visible query formulation options**	**Some query formulation options like aggregation functions or calculated attributes are only visible if the user uses accelerators like right-click on columns. As well as possible, try to show these options without pain simplicity of interface. Improve "Recognition, not recall"without harm "Aesthetic and minimalist design".**
Figured Out	Analysis and Study, and User Testing	
Interface Components Related	General Layout	
Main Action Hampered	Query Formulation and Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Object appearance	
Task Category	Task-facilitation	
Task Subcategory	Alternatives	
Nielsen Heuristics	Recognition rather than recall	
46	**It's not possible to remove more than one aggregated attribute at the same time.**	
Figured Out	Analysis and Study, and User Testing	
Interface Components Related	Aggregation Functions, Calculated Attributes, and Query Result Table	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Flexibility and efficiency of use, and Consistency and standards	
Continued on next page		

Continuation of Table A.1		
ID	**Issue**	**Description**
47	**It's not possible to change aggregation functions**	**To change an aggregation function (e.g. count, max, etc.) is necessary to remove the attribute and to add again.**
Figured Out	Analysis and Study, and User Testing	
Interface Components Related	Aggregation Functions, and Query Result Table	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Flexibility and efficiency of use	
48	**Add another join of two tables that are already merged.**	**When for example already exists a join between A and B with foreign key Key1 and we need to add another join between A and B but with the foreign key Key2, the interface should add automatically the entities twice (e.g., A2 join B using Key2).**
Figured Out	User Testing	
Interface Components Related	Sources and Joins	
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-facilitation	
Task Subcategory	Task/function automation	
Nielsen Heuristics	Error prevention, and Match between system and the real world	
Continued on next page		

Continuation of Table A.1		
ID	**Issue**	**Description**
49	**The users don't know that exist hidden attributes**	**If the users don't have experience using the platform they don't know that some attributes are hidden.**
Figured Out	User Testing	
Interface Components Related	Query Result Table	
Main Action Hampered	Query Formulation and Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Object appearance	
Task Category	Task-facilitation	
Task Subcategory	User error tolerance	
Nielsen Heuristics	Visibility of system status, and Match between system and the real world	
50	**Function symbol of aggregated attributes isn't clickable.**	
Figured Out	User Testing	
Interface Components Related	Aggregation Functions, and Query Result Table	
Main Action Hampered	Query Comprehension	
Usability-ODC Framework Attributes		
Artifact Category	Manipulation	
Artifact Subcategory	Mouse click	
Task Category	Task-facilitation	
Task Subcategory	Keeping the user task on track	
Nielsen Heuristics	Consistency and standards, and Visibility of system status	
51	**No visual identifier to show that is a filter edition option**	**Reported by a user in an User Test.**
Figured Out	User Testing	
Interface Components Related	Filters	
Continued on next page		

ID	Issue	Description
Continuation of Table A.1		
ID	**Issue**	**Description**
Main Action Hampered	Query Formulation	
Usability-ODC Framework Attributes		
Artifact Category	Representation	
Artifact Subcategory	Presentation of information/results	
Task Category	Task-mapping	
Task Subcategory	Functionality	
Nielsen Heuristics	Visibility of system status	
End of Table A.1		

USER TESTING SCENARIOS

In order to test the usability of the existing interface and its followed prototypes, the following test scenarios were created in order to evaluate the usability of the system[1]:

Query Comprehension 1: Employees of "Portugal" or "Japan" of departments "Services Support West" or "Services Support East" who have a job position different from "Services Representative" and never have created any notification. The employees must be presented ordered by their last name (descending order).

Query Modification 1: Change the existing query to consider only the employees of offices "Australia"or "Japan", their department must be "Marketing"or "Services Support East", and they must have any job position. Moreover, create an attribute to present employees' full name.

Query Comprehension 2: List for each AccountNumber the amount sum of transactions of type "Eating Out". Moreover, the transaction is only shown if its source account is managed by employees of department "Credit Control" and office "United Kingdom". Account Numbers are sorted by their amount sum in descending order.

Query Comprehension 3: List the number of employees by department and office. The number of employees is presented in descending order.

Query Formulation 1: Notifications created by employees who are owners of a set of accounts which, at least, must combinedly average a balance of 20.000 (average of balances of accounts owned by each employee).

Query Comprehension 4: List all requests assigned to employees of department "Services Support West" ordered by priority in the first place ("High" first), and secondly by creation date (oldest dates first).

---

[1]**Note:** The 5.1 illustrated the relational database diagram of the database used to test the following scenarios.

Query Modification 4: Considering the previous requests, change the query to show only the ones that were requested by employees from other departments (not the same).