Chapter 5

SQuO validation

The porupose of this chapter is to present some test cases for evaluating the efficiency of the model proposed as solution (Chapter 3) to carry out query optimization in ubiquitous environments. Test cases allow us to appreciate the query optimization challenges that this project addresses. The evaluation consists in analyze the solutions that SQuO prototype suggest in order to solve the problems that test cases involve. These solutions exemplify the different functions that SQuO is capable to perform, all of them supported on the basis exposed in the model for query optimization using case-base reasoning.

This chapter is organized as follows: Section 5.1 presents an ubiquitous enviornment description and some examples in the daily life to clarify this notion. Section 5.2 describes the context validation used for these experiments. Section 5.3 describes the model of the data sources used for SQuO validation. Section 5.4 explains the method used for the validation of the proposed solution exposing the SQuO functions. Section 5.5 Shows the validation results. Section 5.6 concludes the chapter.

5.1 Ubiquitous environments

Ubiquitous environments require sophisticated computing and communication capabilities in order to cooperate with a wide variety of activities that, nowdays, could be essential for the human daily life. The handy interaction between users and several computational and electronic devices is an indispensable scopes for ubiquitous computing, the presence of multiple users will further complicate this challenge. Combination of global wireless and wired connectivity along with increasingly small and powerful devices enables a wide array of new applications that will change the nature of computing [13].

Mobility is an integral part of every life, so, if devices become truly ubiquitous, then they will be used constantly in a wide range of continually changing situations. Existing applications must be able to operate in varied and dynamic communication and computation environments. Additionally, user must be exempt from context variations, which implies complete disappearance of pervasive computing technology from a users consciousness, otherwise, users should be acutely aware from technology due to all the impacts produced by their activities [27].

An ubiquitous space could be an enclosed area such as a meeting room or corridor by embedding computing infrastructure in building infrastructure. The rate of penetration of pervasive computing technology into the infrastructure will vary considerably depending on many technical (i.e. context aware, mobile, collaborative and autonomous devices.) and nontechnical factors (i.e. organizational structure, economics, business models, among others). These scenarios tend to involve users with several portable devices, moving between different environments (e.g., home, car, office, conference).

Beyond new devices and communications mechanisms, there exist a computational requirement that is a key to make ubiquitous computing a reality, data retrieving and

management. Users require technology to solve their needs for communication, storage, maintenance, delivery, and presentation of shared data. Data lies at the heart of all ubiquitous computing applications, but these applications and environments impose new and challenging requirements for data management technology. Variable nature of data availability, real-time data production and demand, as well as processing and management of this dynamic data flows represent daunting challenges.

As an example, we can appreciate an ubiquitous environment in a intra and inter enterprise level, where a comunity of users retrieve and share distributed information that is represented in meny different ways. Resources for data production (i.e. data bases, servers, sensor networks, video cameras, etc.) are static, dynamic and present phisical limitations. A busines manager wants to perform the following query:

Q2: Retrieve the list of customers at the meeting room

To solve this query a sensor network is required, it must detect the profile of people that is in some specific room. In this case, data update must be continous since people enters and leaves the room. All the variations must be taken in to account.

Ubiquitous computing environments must provide appropriate techniques for data retrieving, query optimization is essential for their improvement. As we said before, the core of this investigation project is to propose a query optimization technique capable to deal with data retrieving challenges in these type of environments, and this chapter aims to present the results from the proposal solution.

The previous scenario, is a simple example that allow us to appreciate important requierements for query optimization, such as the need to optimize a query in terms of a wide range of parameters or dimensions. Additionally optimization techniques must be useful to calculate the cost of query execution plans in despite of metadata absence. Finally, thye must to deal with resources physically constrained, and a constant execution

environment variation.

Query optimization is fundamental for data retrieving, it is the core of our investigation project. Classical query optimization techniques are not apropriate to meet the requirements for data retrieving on ubiquitous environments. Statistics and metadata absence is one of the main reasons, since they are used to determine the cost of an execution plan by means of an adaptive cost function that optimize a query in terms of different parameters according to the user requirements. In addition, classical query optimization techniques aim to determine the optimal execution plan to perform a query according to one optimization objective, tipically, processing time.

Thus, some in industry have begun to talk in terms of delivering a certain type of user experience rather than a particular application or suite of applications. The prototype SQuO can be used in this kind of environments because it is not recuired the metadata as in classical query optimization The project will provide the (things) required so that workers and their clients can perform daily business processes collaboratively and safely according to determined security policies. Something about mobility and business applications and context sensitivity and secure business applications. Mobile workers will be enabled to perform business tasks in arbitrary environments - devices, networks, and domains - as well as with any user.

5.2 Validation context

The validation context is not a real ubiquitous environment, however, it is enough to accomplish the validation process. The objective is twofold: (i) observe the behavior of the SQuO prototype (case-based reasoning optimizer) and (ii) study the variations on evaluation time and memory consumption when the case-base is preloaded with different amounts of knowledge. Thus, it is possible to demostrate if the proposed

solution is capable to deal with the challenges previously explained (metadata absence, several query optimization objectives, some execution environment variations) or not. Successful resoults will allow to apply the solution in different contexts in a future.

5.3 Sources

For the validation we used business data sources in a simple document format, any existing data base maneger (e.g. MySQL, Postres, etc.) was used. We implement an execution plan generator with the basic operations to propose new solutions without the classical statistics and metadata use. In addition, we use a simple model for the query plan implementation in order to facilitate access to the data sources and data retrieving. Figure 5.1 illustrates the schema of data sources.

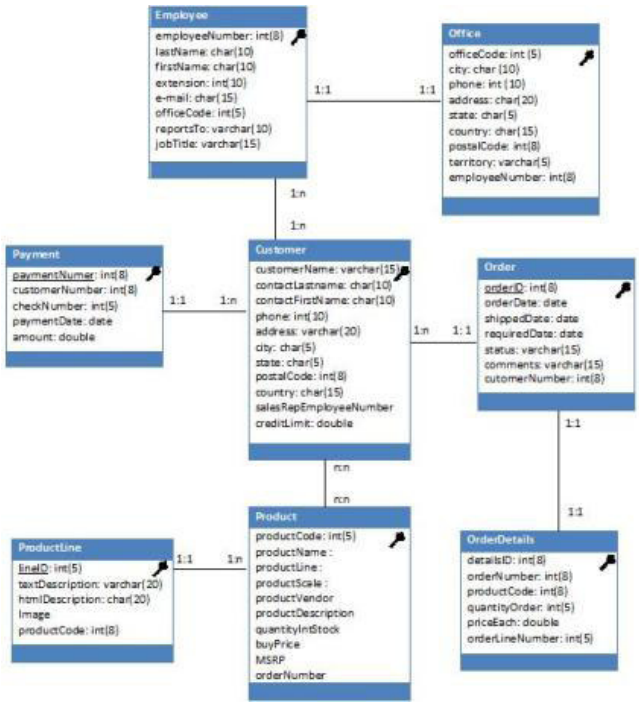

Figure 5.1: Database schema

5.4 Experimenting SQO

SQuO is exposed as a case-based resoner optimizer, it involves a very wide ranges of functionalities since it is incharge to evaluate the execution environment conditions, analyze the knowledge obtained from experiences and to propose solutions (query plans) for the optimization of new queries according to different optimization objectives.

We propose a pseudo-random query plans generation, different with respect to classical procedures in order to deal with the absence of information that is usually required. For the experimentation, we propose a query plan generation technique that promotes a simple execution of basic query plans. They include Selection, Projection, Sort and different Join operations. A detalied description is presented in Chapter 5.

Since the prototype is based on a query optimization technique, as we said in chapter 4, supported for the case-based reasoning approach is important to demostrate the SQuO functionality for storage, manage and knowledge retrieving for solving a problem. Finally, is necessary to validate that the prototype is capable to take into account the execution environment characteristics and different query optimization objectives.

SQuO validation is apreciated by query execution over data sources related to enterprise production. Query execution was carried out in a personal computer with the following characteristics: AMD Turion processor, 2 GB RAM Memory, operative system Microsoft Windows XP and JDK version 1.6.0.

The experiment that we proposed was the following: Frist, the knowledge base was fed with a set of queries produced randomly and pertaining to five different query families (see Chapter 4) and registrating the measures related to the computational resources that were consumed during their execution. Next, one houndred rounds of query evaluations were performed, once again, the queries were randomly selected and taken from five different query clasess. For each of them, the complete case-based

reasoning optimization process was applied.

The SQuO general process to be evaluated is as follows: Search a relevant case, it contains a total or partial query solution (exection query plan). Parcial solutions must be adapted to the new situation.

Otherwise, a new solution (pseudo-random query) plan must be suggested. Each time, this solution can be improved and the knowledge related to its execution must be actualized. Some times, this knowledge (computational resources) is stabilized, which inidcates that probably this is the optimal point to solve a query for a specific state of the execution environment. Following section presents a simple example to appreciate the main SQuO functionalities.

5.4.1 Relevant case retrieval

A bussines manager needs to know the name of the customers present at the meeting room. SQuO must to search a relevant case to solve the query. Relevant case retrieving includes a detalied analysis to determine if avaliable computational resources are enough to carry out the solution according to the resurces that it consumes. Additionally, the optimization objective must be appropriate, this means, minimize as possible all the computational resources or the most urgent (i.e. energy consumption).

The knowledge base is group queries according to their similarity, as is describen in Chaper 4, in order to avoid a exaustive seaarch. It is possible to find a relevan case that contains query plan that solves the query, and the resource consumption is convenient acording to the optimization objectives. The plan is executed and new knowledge is registred (i.e. a particular characteristic in the execution environment).

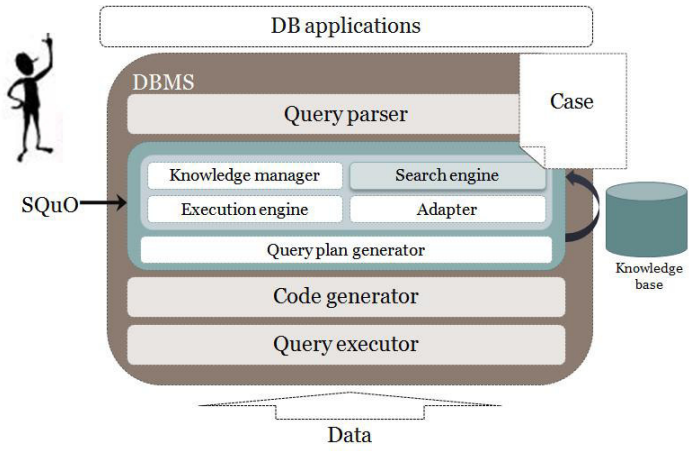

Figure 5.2: Relevant case retrieval

5.4.2 Query plan generation

If a relevant case was not retrieved, new knowledge must be produced (See figure 5.3). The query plan generator produce a pseudo-random query plan, it is ejecuted and the resource consumtion measures are taken.

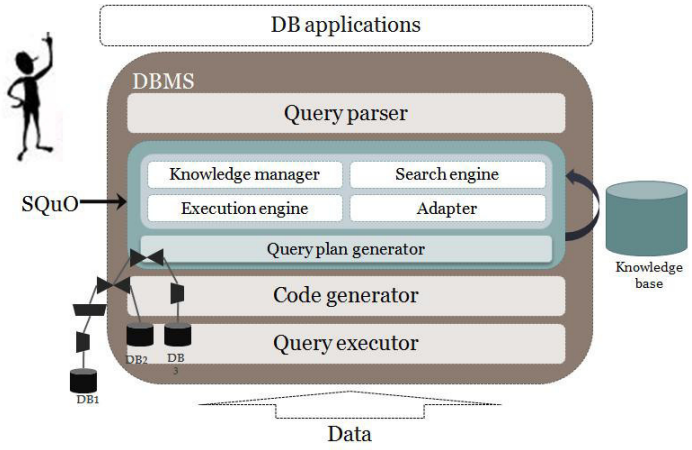

Figure 5.3: Query plan generation

If a run time error detected (i.e. insufficient resources), a new solution is proposed in order to improve the previous one. When the proposed solution is acceptable to

retrieve the required data, useful knowledge is retained and managed in the knowledge base. Whenever possible, knowledge evolve and the solutions are improved to achieve the optimization according to different objectives and execution environment conditions for the same query.

5.4.3 Query plan adaptation

A relevant case contain a total or parcial solution. It is possible to retrieve an identical query to the query problem, but in other cases, a similar query is founded, simple adjustments can be made. SQuO can adapt different attribute values, comparison operators and data combination functions (i.e. Join and Merge-Join).

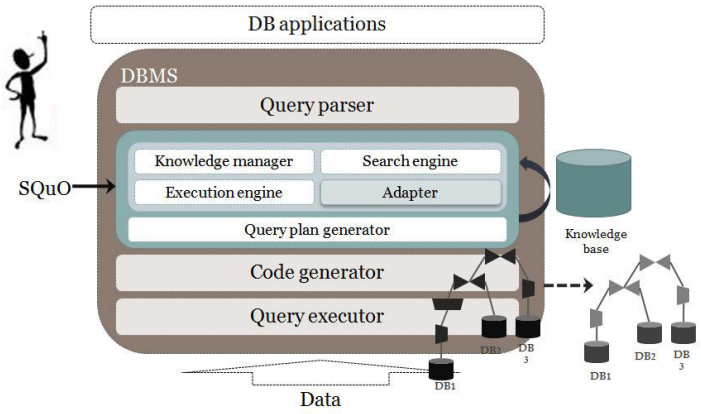

Figure 5.4: Query plan adaptation

5.5 Validation results

This section presents the first experimental results of our prototype. The experiment suggest tje knowledge as a threshold under which execution plans are systematically generated instead of extracted from the case base.

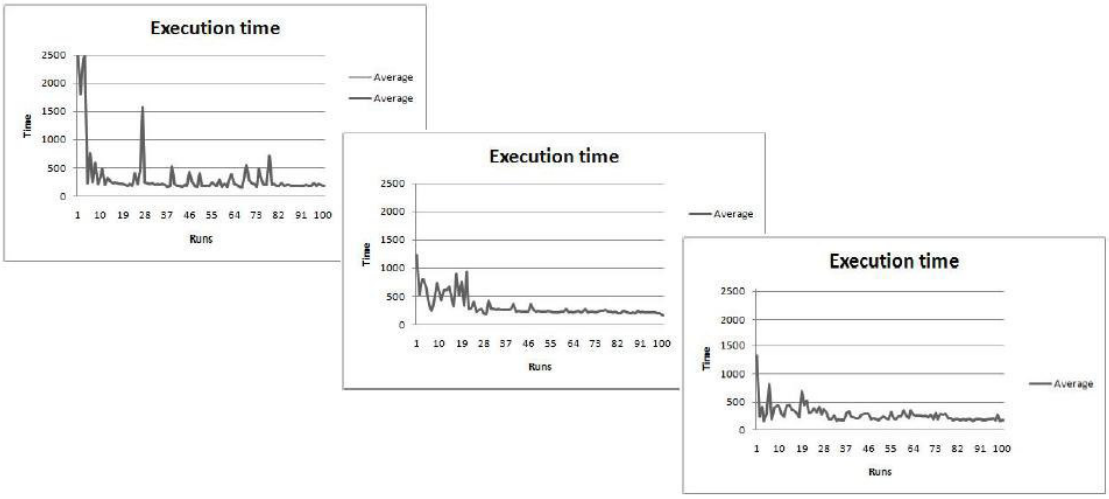

Figure 5.5: Evaluation time results (a) threshold = 10, (b) threshold = 20, (c) threshold = 30

The experiment that we propose consists in executing a set of queries related to five different query classes and registrating the measures related to the computational resources that were consumed during their execution. Each figure is composed of 100 rounds of query evaluations where queries are taken from five different query classes. The executed query is randomly selected. The X axis represents each round. The Y axis represent the consumed ocmputational resources.

Figure 5.5 shows the query evaluation time w.r.t. different thresholds. Here, the optimization objective is minimizing the total evaluation time. Something we observe is that this evolution is decomposed in two phases: (i) a learning phase where evaluation time est very cahotic; (ii) a stabilized phase where the query optimizer reuse known query plans and updates the execution measures.

We can also observe that low values of threshold lead to poor performance stability and longer learning phase (Figure time(a)) while highest values reduce learning phase but do not improve stability (comparing threshold of 20 and 30 on figures time(b) and time(c)). The gain in evaluation time is a factor from two to five.

Figures 5.6 shows memory consumption measures under the same experimental conditions, except optimization objective that is minimizing memory consumption. In a similar way to the first series of experiments, we observe that the memory consumption is divided in two phases: chaotic initial learning phase, and stabilized exploitation phase. One major point is that with a vert low threshold value, the gained optimization is only a two factor (Figure memory (a)) while we obtain a factor four with greater values (figures memory (b) and memory (c)). Once again a threshold of 30 is not clearly better than a threshold of 20. Appendix A and B descibe the tools that we use to take measures. Apendix C explains the basic principles to measure Java objects [8] [10] [11].

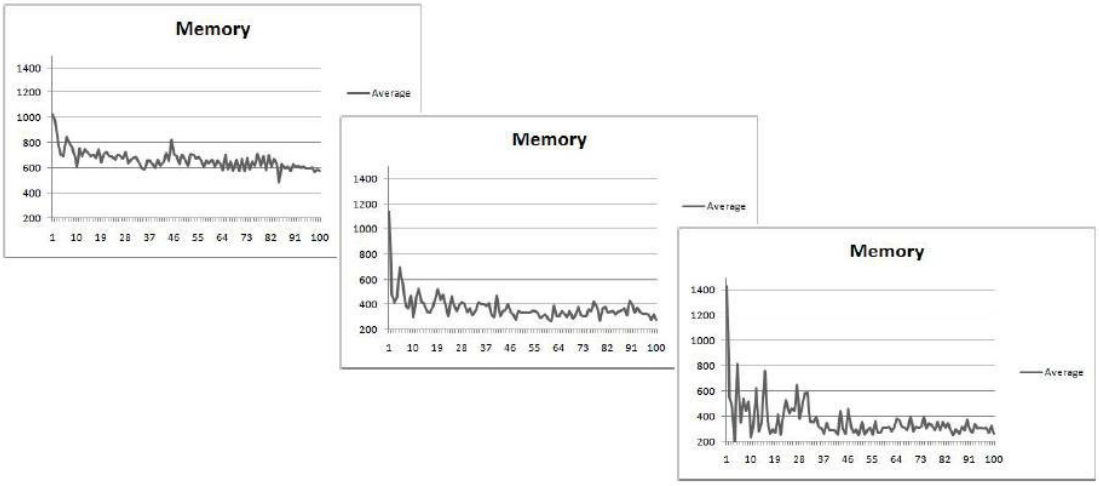

Figure 5.6: Memory consumption results (a) threshold = 10, (b) threshold = 20, (c) threshold = 30

This graphic shows the average of the computational resources consumed during the experiments. It reflex the reduction of the consumed computational resources as more knowledge within in the case base. This experimental evaluation will be improved by dynamically varying other parameters, such as number of query families, database volume and evaluation rounds.

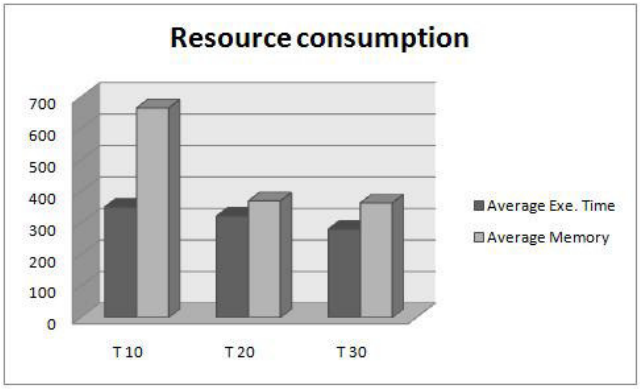

Figure 5.7: Resource consumption results

5.6 Conclusions

The valiation context is not a real ubiquitous computing environment, however, the experimentation proved the capacity of knowledge generation ase well as learning. This means, the proposed solution and the implementation are capable to generate solutions whitout the information that classical query optimization techniques use. Also it allows the optimization objective customization according to users (people, computational tools, devises, etc.) requirements. Finally, available computational resources in the computational environment can be measured and taked into account to select or suggest appropriate solutions. Could be very interesting to improve the prototype implementation and adjust its functionality to more complex computing environments.