Chapter 2

Classical query optimization

This chapter presents a preliminary study of the general aspects that query evaluation and more specifically, query optimization involves, as well as the existent query optimization approaches and application contexts. Section 2.1 addresses the general query evaluation process in order to identify and remark the importance of the optimization process. Section 2.2 presents a general view about different query optimization approaches. Section 2.3 describes some alternative query optimizer architectures. Section 2.4 presents the application of the query optimization process in different computational environments. Finally, Section 2.5 lay down a discussion about the presented approaches as well as the identification of the challenges that different application contexts represent for query optimization.

2.1 General query evaluation process

The modules that participate in the classical query evaluation process are the query parser, query optimizer, code generator and the query executor. The query parser is in charge to verify if the query is syntactically (well formed) and semantically correct. The output of this module is a tentative algebraic query tree. It is a sequence of algebraic

operations (e.g. selection, projection and joins) that indicate the operations that must be performed on the data for solving the query [17]. Then, a valid query must to be optimized, this is carried out in by the Query optimizer module. That estimates the best order to perform the operations included by the algebraic query tree and assigns to each algebraic operator an algorithm to execute it. The result is the execution plan. When the query is optimized, a codification of the execution tree must be performed by the code generator, to be executed by the last module, the query executor. Finally, the data that solve the query is obtained [14].

2.2 Query optimization approaches

Given a query, there are multiple execution plans that a database system can complete to process it, the optimal plan, according to a set of parameters (time, memory, energy, etc.), must be selected. Query optimization is divided in two phases: rewiring phase and planning phase. Rewiring phase consist in apply a sequence of transformation to query and produce a set of equivalent queries more efficient. Planning phase consist in generate all possible execution plans to each query produced in the previous phase and choose the less expensive plan to generate a data output that respond the original query. The most important aspects to address in query optimization are (i) execution plan generation and execution plan evaluation thought (ii) a search strategy and (iii) a cost function. The objective is to determine the optimal execution plan according to a set of parameters of interest (dimensions) [17] [28].

2.2.1 Search strategies: Top-down and Bottom-up

Query optimization research in the literature can be divided in two classes, which can be described as bottom-up and top-down. Researchers found the overall query optimiza-

tion problem to be very complex. Theoretical work began with a bottom-up approach, studying special cases, such as the optimal implementation of important operations and evaluation strategies for certain simple subclasses of queries. Subsequently researchers attempted to compose larger building blocks from these early results [19].

A need for working systems triggered the development of full-scale query evaluation procedures, which stressed the generality of solutions and handled query optimization in a uniform and heuristic manner (Astrahan and Chamberlin 1975; Makinouchi et al. 1981; Niebuhr et al. 1976; Palermo 1972; Schenk and Pinkert 1977; Wong and Youssefi 1976). As this often did not achieve competitive system efficiency, the current trend seems to be a top-down approach that incorporates more knowledge opportunities into the general procedures. At the same time, the general algorithms themselves have been augmented by combinatorial cost-minimization procedures for choosing among strategies [19].

The steps for accomplish this technique are the following:

- Step 1. Find an internal query representation into which user queries can easily be mapped that leaves the system all necessary degrees of freedom to optimize the evaluation.

- Step 2. Apply logical transformations to the query representation that (1) standardize the query, (2) simplify the query to avoid duplication of effort, and (3) ameliorate the query to streamline the evaluation and to allow special case procedures to be applied.

- Step 3. Map the transformed query into alternative sequences of elementary operations for which a good implementation and its associated cost are known. The result of this step is a set of candidate access plans.

- Step 4. Compute the overall cost for each access plan, choose the cheapest one, and execute it. The first two steps of this procedure are to a large degree data independent and thus often can be handled at compile time.

Quality of Steps 3 and 4, that is, the richness of the access plans generated and the optimality of the choice algorithm, heavily depends upon knowledge about the values in the database. The consequences of data dependence are twofold. First, if the database is volatile, Steps 3 and 4 can be done only at run time. This means that the possible gain in efficiency must be traded off against the cost of the optimization itself. Second, a metadatabase (e.g., an augmented data dictionary) must maintain general information about the database structure as well as statistical information about the database contents. As in many similar operational research problems (e.g., inventory control), the costs of obtaining and maintaining this additional information must be compared to its value [14].

2.2.2 Cost-based query optimization

Cost-based query optimization techniques typically consider a large number of candidate execution plans, and select one for execution. The choice of an execution plan is the result of various, interacting factors, such as database and system state, current table statistics, calibration of costing formulas, algorithms to generate alternatives of interest, and heuristics to cope with the combinatorial explosion of the search space. A fundamental technique used in cost estimation is cardinality estimation - optimizers take as input the cardinalities of tables at the leaves of a query tree, and then use selectivity of operators in the tree to estimate the cardinality of the input to operators further up in the tree. To convert cardinalities to costs, optimization techniques use functions that estimate the cost of each execution plan. Cost model is an important

component in cost-based optimization [22].

A cost model specifies the cost functions expressed by arithmetic formulas that are used to estimate the cost of execution plans. For every step in a query execution plan, there is a formula that gives its cost. Given the complexity of many of these steps, most of these formulas are simple approximations of what the system actually does and are based on measures related to computational resources like buffer management, CPU performance, memory capacity, etc. Also use statistics about the size of data and distribution of values.

Unidimensional cost model

Cots based query optimization techniques use cost models that evaluate a query execution plan in terms of some parameters. Unidimensional cost models evaluate an execution plan in terms of a single dimension. Typically, the optimization goal of this cost models are CPU path length, amount of disk buffer space, disk storage service time, memory consumption [29], query performance, I/O data rates [22], among others. This kind of cost models are often used in query optimization techniques applied in computational systems with low degree of complexity i.e. in centralized and static database systems.

Multidimensional cost model

More complex computational applications demand query optimization techniques that allow to optimize query execution in terms of a set of dimensions. Since the cost model reflects optimization aims, it must specify a multidimensional function or a set of functions that can be applied in order to evaluate a set of query execution plans. The cost functions must be capable to ponder the optimization of query execution in terms of multiple dimensions. It is expressed by means of a formula that includes a

set of parameters that represents those dimensions. A multidimensional cost model also can offer a set of functions that optimize a query in terms of different dimensions; a function from this set must be selected and applied according to user requirements (person, application and devise).

2.2.3 Rule-based query optimization

Rule-base query optimization techniques are applied in the rewriting phase involved in query optimization general process. The original query can be poorly expressed, that means that it includes nested queries, data combinations, data projections and another type of operations that are unnecessary to obtain the desired data. A query plan related to this query is a suboptimal plan. To eliminate this problem, given a query, a set of equivalent but optimal queries must be rewritten. Two queries are equivalent if they produce de same result. Equivalent queries are derived from the original query by means of the application of correspondence rules specified in the rewriting module included within optimizer architecture. Additionally, optimizer includes a rule engine that is capable to process rules and produce equivalent queries. Rewriting phase is very important in modern query optimization techniques [28].

2.3 Query optimizer architectures

A query optimizer is the component of a database system responsible to determine the optimum plan for a query execution. Figure 5 Illustrate the general query optimizer architecture. Modules that compose this architecture are grouped according to query optimization phases. Rewriter is the module that executes the rewriting phase. Generator, Selector, Cost estimator, Statistics estimator and Planner are the modules that execute the planning phase [9]. Figure 2.1 illustrates the classical general optimizer

architecture [17] [19].

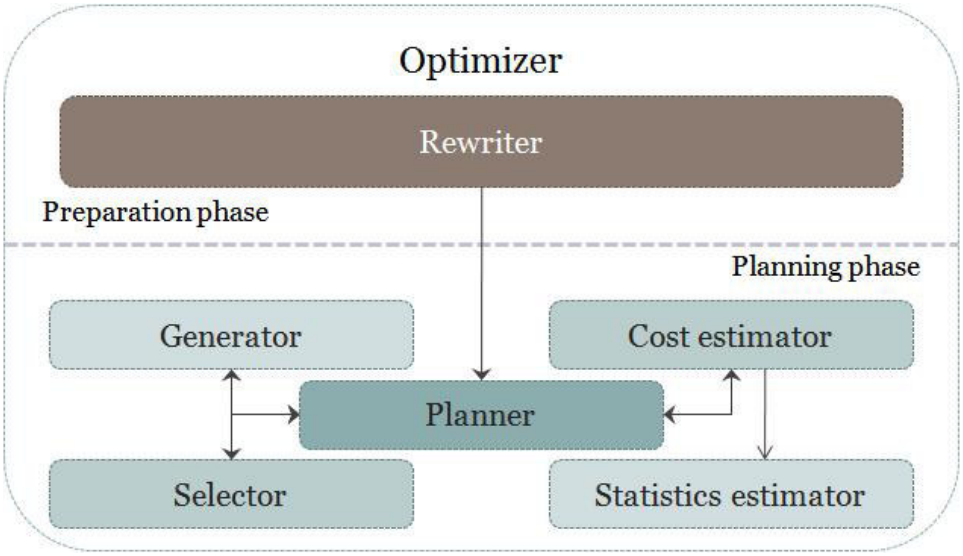

Figure 2.1: General optimizer architecture

Rewriter. Rewrites from the original query, a set of equivalent queries by applying a set of transformations. Queries Q1 and Q2 are equivalent. O1 and O2 are the data output sets related to Q1 and Q2 respectively. This two queries are semantically equivalent if O1 O2. New queries processing must be more efficient according some specific parameters. Transformations depend on declarative characteristics of the original query.

1. **Generator.** Generates a set of query execution plans to process a query. This execution plans are represented by algebraic expressions or three structures equivalent to those formulas.

2. **Selector.** Select a set of algorithms to execute each operator that is included in algebraic expressions related to a query execution plan proposed by the Generator module.

3. **Cost estimator.** It specify calculus functions (in accordance a cost model) to evaluate execution plans to estimate their cost.

4. **Statistical estimator.** It specifies a set of functions to estimate relations size, indexes and results, as well as, the distribution feculence of the values associated to an attribute include by a relation.

5. **Planner.** It employs a search strategy to examine the execution plans space (with an associate cost) and select the less expensive in order to process a query and generate the result.

2.4 Ubiquitous environments

Over the last years, we have seen the power of many computational and electronic tools. An equally rapid increase applies to some other technological parameters such as storage capacity and communications bandwidth. This continuing trend means that computers will become considerably smaller, cheaper, and more abundant they are becoming ubiquitous, and are even finding their way into everyday objects. This is resulting in the creation of smart things that can access the Internet and its varied resources in order to optimize their intended purpose, and maybe even cooperate with each other. A ubiquitous environment must be accessed at anytime and anywhere trough any kind of tool (application or devise). Figure 2.2 illustrates the ubiquitous environment functionalities [25].

The functionalities that such kind of environments imply must be categorized. This categorization can then be examined to determine the requirements that are imposed on data management. The functionalities can be classified into the following [13]:

- *Support for mobility.* the compactness of the devices combined with wireless com-

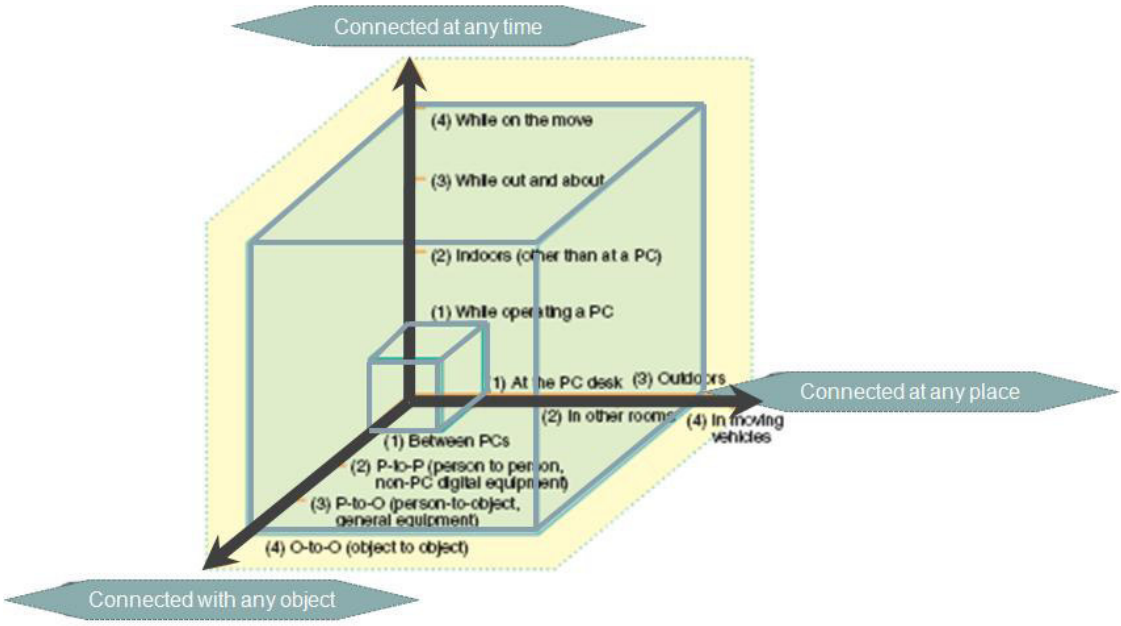

Figure 2.2: Ubiquitous environment functionalities

munication means that the devices can be used in mobile situations. Thus, existing applications must be able to operate in varied and dynamic communication and computation environments, possibly 3 Challenges in Ubiquitous Data Management moving from one network or service provider to another. Furthermore, new applications that are location-centric will also be developed.

- *Context awareness.* if devices become truly ubiquitous, then they will be used constantly in a wide range of continually changing situations. For the devices to be truly helpful, they must be aware of the environment as well as the tasks that the user is performing or will be performing in the near future. Context aware applications range from intelligent notification systems that inform the user of (hopefully) important events or data, to smart spaces, that is, rooms or environments that adapt based on who is present and what they are doing.

- *Support for collaboration.* another key theme of ubiquitous computing applications is the support of groups of people. This support consists of communications and conferencing as well as the storage, maintenance, delivery, and presentation of shared data. Collaborations may be performed in real-time, if all of the participants are available, or may be done asynchronously otherwise. In addition to supporting on-going collaboration, access to and analysis of traces of past activities is also required.

These functionalities provide a host of challenges for data management techniques, but one requirement is present across all of them, Ubiquitous computing environments are characterized because the devices that compose them are [12] [27] [5]:

- *Heterogeneous*, since they integrate a wide variety of devices as portable computers, PDAs, cell phones, censor networks, embedded systems, among others. These devises present important physical and logical different characteristics. Storage and processing capability, memory availability, and network transfer rate are some differences since physical point of view. Data representation models, storage structures and calculus algorithms are some differences since logical point of view.

- *Dynamic*, because of their continuous change, typically, due to their mobility, e.g. when a resource is moved, its location and availability change. Also, the communication network properties and the set of resources with which it interacts are different.

- *Distributed* in diverse locations within a physical environment to which the user interact to carry out a set of activities. The information used to complete these activities is distributed among these resources. It must be available in order to access it at any time since any location.

- *Autonomous*, this means that it can change its availability (It can be available to interact with other resources within the environment or not) status at any time. Every type of resource (mobile or stationary) can be disconnected during its interaction with other resources and there is no guarantee that it will be available again.

- *Physically constrained*, because resources present physical limitations that constrain their appropriate operation, e.g. processing and storage capability, energy consumption, location, among others. In accordance to the measures of these characteristics is determined if a resource is suitable for completing a task.

- *Lack of metadata*, this is a characteristic that results from the previous. Evaluation of queries, specifically, classical query optimization, requires metadata to be accomplished, e.g. distributed systems related to data management and retrieval are based in the notion of global schema (a united and integrated representation of the local schemas exported by each source to represent its own data). In ubiquitous environments there is not a global schema, in general, the acquisition and maintenance of metadata is very expensive as result of the high heterogeneity, dynamicity and autonomy of its devices.

2.5 Conclusions

This chapter presents an overview about the general query evaluation process, particularly, query optimization. The chapter is focused on the description of classical query optimization techniques, such as, Top-down and Bottom-up strategies, Cost-based query optimization and Rule-based query optimization. Also, general query optimizer architectures are presented. Finally, an ubiquitous environment description

is presented in order to explain its main characteristics and the requirements that these kind of environments demand.