Guided Interaction: Rethinking the Query-Result Paradigm

Arnab Nandi
Dept. of EECS
University of Michigan, Ann Arbor
arnab@umich.edu

H. V. Jagadish
Dept. of EECS
University of Michigan, Ann Arbor
jag@umich.edu

ABSTRACT

Many decades of research, coupled with continuous increases in computing power, have enabled highly efficient execution of queries on large databases. In consequence, for many databases, far more time is spent by users formulating queries than by the system evaluating them. It stands to reason that, looking at the overall query experience we provide users, we should pay attention to how we can assist users in the holistic process of obtaining the information they desire from the database, and not just the constituent activity of efficiently generating a result given a complete precise query.

In this paper, we examine the conventional query-result paradigm employed by databases and demonstrate challenges encountered when following this paradigm for an information seeking task. We recognize that the process of query specification itself is a major stumbling block. With current computational abilities, we are at a point where we can make use of the data in the database to aid in this process.

To this end, we propose a new paradigm, *guided interaction*, to solve the noted challenges, by using interaction to guide the user through the query formulation, query execution and result examination processes. The user can be given advance information during query specification that can not only assist in query formulation, but may also lead to abandonment of an unproductive query direction or the satisfaction of information need even before the query specification is complete. There are significant engineering challenges to constructing the system we envision, and the technological building blocks to address these challenges exist today.

1. INTRODUCTION

The explosion of information in today's world has made direct human interaction with data a common occurrence. Unlike the past where a domain-specific *application layer* was responsible for the communication between the user and the database, today's users commonly consume data in its raw form, e.g. through spreadsheets, keyword search engines, music libraries and social networking websites, each of which surface a plethora of rich, heterogenous structured data. Given the wide variety of ways in which users may want to access these pieces of data (e.g. a user may want to search their social network for friends with common tastes in music,) it is intractable to conceive an application layer for each intent. Further, as the rate of interaction with data increases, the heterogeneity of the information need and the expectations of time efficiency also increase. These patterns are not just restricted to the end-user: ad hoc querying and querying with diverse or approximate intents is also becoming an increasingly popular use case in data warehouses, where unfamiliarity with data is a significant challenge.

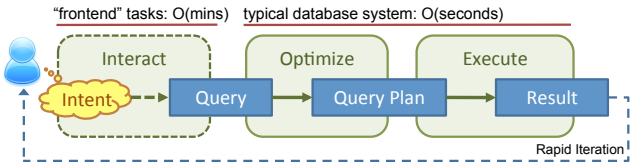

Figure 1: Construction of a database query is often challenging for the user, commonly takes longer than the execution of the query itself, and does not use any insights from the database. We propose (shown in dashed lines) that this process be considered as a fundamental part of the database query paradigm, enabled by *guided interaction*.

Since the inception of databases, the model has constantly been that of a *query* and a *response*. The query has typically been assumed to be precise and unambiguous, and running it on a database produces an expected answer. We observe that this is only one part of the data interaction process. Often, more time is spent constructing the query and familiarizing oneself with the data, schema and query language than actually executing the query (we will see an example of this in Sec 2.) Further, despite being declarative, queries are typically highly iterative and incremental in construction. In Fig. 1, we articulate the traditional process of going from *Query* to *Query Plan* by using an optimizer, followed by execution of the plan to generate *Results*. In the same figure, we point out that while the latter steps are well articulated and researched, the process of going from a user's *Intent* to an unambiguous structured *Query* has typically been considered a "front-end" or query modeling task, depicted by the dashed box. Even if one were to put user convenience aside, even purely from a system performance perspective, we note that bad queries (e.g. searching against the wrong table) often cost much more (too many results, scan of an unindexed table, etc.) than the correct query.

Proceedings of the VLDB Endowment, Vol. 4, No. 12

Databases are now part of a rapidly changing landscape. Query execution is typically fast enough that a significant part of data interaction is expected to be done in real time, at least for databases of moderate size. The increased availability of computational resources now allow us to process and digest data in a much more comprehensive manner than before. Given these changes, it is clear that we are at a point where we can make use of data and computational power not only to execute queries efficiently, but also help the user *form* these queries in the first place.

In traditional database systems deployment, the solution to the query specification problem is either to have a human expert database administrator (DBA) or to have a carefully constructed application interface. While a DBA is usually knowledgable about the schema, data and indexes in the database, it can be expensive and time-consuming to consult such an expert for each user query intent. Application interfaces, on the other hand, suffer from providing a very limited access to underlying databases. There is usually no support for users with queries that were not envisaged by the application designer. Neither solution supports direct interaction of the user with the data.

Given this scenario, there is a pressing need for systems that enable direct user interaction with databases. However, based on the challenges presented in the next section, we need to approach this problem in a principled manner such that all challenges are dealt with. To this end, we describe a paradigm called ***guided interaction***, a tractable solution that can be incorporated into existing database systems.

Contributions: In this paper, we propose a framework for interaction with the database that assists users to form precise queries from vague query intent. We observe that the data itself can be used to support this interaction paradigm. We describe *guided interaction*, comprising three principles for efficient data interaction, requiring that database interfaces be *guided, responsive* and *intuitive* for the end user.

2. AN EXAMPLE

We use an often-encountered situation to better explain the challenges associated with interacting with databases:

Alex and Bob are new employees. Alex is new to computers, while Bob is an expert at databases. They met a senior person at the company orientation who would make a great mentor, but they have unfortunately forgotten what her name was; all they remember is that she was manager of a small group of researchers. While they both have full access to a corporate social network "LinkedBook", and the company's employee database, finding her name and contact information still seems to be a daunting task.

We will use this scenario to explore two possible use cases: one where Alex is new to computers, and one where Bob is an expert at SQL. In both these use cases, we will demonstrate that there are challenges common to both scenarios.

Naive user Alex: Alex logs in to the social network LinkedBook, and is presented with a wide variety of pages, sections and "advanced search" forms, each with a large number of fields. He skims each of them, trying to piece together all the information he has and determining which form is most applicable. After a few minutes of browsing around, he finds a faceted search interface [8] and naively issues a query for everyone in his company: this yields an unhelpful set of thousands of results, which he reformulates by looking at the facets. He then realizes that he cannot find a facet to drill down by "seniority", but there is an "age" facet. Not knowing what the proper age for a manager is, Alex has to go back to a different "Birthday Search" form which allows him to search for people's birth dates. He uses this to browse through some employee profiles, gauging that the average manager is around 50 years old. He uses this information to drill down to a small set of managers, and selects "Databases" and "Theory" as two departments to further look into, resulting in a successful search.

As we can see, performing a simple employee search took a large number of refinement steps, including one where the user had to abandon a query session to look up external information (relation between seniority and age.) Clearly, regardless of how efficient the end-user interface was, the lack of complete information made it impossible to find this information in a single step.

Database Expert Bob: Bob preferred to express his information need in SQL. Given that the `employee` database is large and comprises about 50 tables, he must first list all the tables and inspect the schema:

```
SHOW tables;
```

And then for each table, he inspects the columns (there are 10-100 columns in each):

```
DESC tables; // assume this lists schema for all tables
```

After a little reading, he gains enough familiarity with the database to construct some SQL queries. He executes them carefully, worrying about bogging down the server with an investigative query that involves more data than he expects. After many minutes of trial-and-error, he solves the problem by performing two SQL queries:

```
SELECT emp.project, COUNT(*) as c, AVG(emp.age) AS a
FROM emp JOIN dept ON (emp.deptID = dept.ID)
GROUP by emp.project ORDER BY c ASC,a DESC LIMIT 3
```

```
SELECT emp.name,emp.cubicleID
FROM emp JOIN dept ON (emp.deptID = dept.ID)
WHERE dept.name='Research' AND emp.project='DatabasePrj'
AND emp.designation='Manager'
```

He uses the first query to identify *which* project the manager was on by browsing the list of projects. Then, he picks the most plausible project from the first query and adds it as a predicate to the second query, which performs the join again. Initially he tries "TheoryPrj", but notices that the employees are all on a different campus, and decides to try the next project, "DatabasePrj", resulting in a successful search. As we can see, performing a single final SQL query took more than a few intermediate SQL queries. It was not possible to fold all of them into a single declarative step. In addition to the user challenges, let us consider the execution times. The employee database is comprised of a few hundred million tuples in its entirety, consuming less than 15GB with all possible indexes built. Such a database was easy to store completely in memory even with a medium-grade server (i.e. sub-2000$) machine, and even expensive `JOIN` queries took less than 30 seconds. Much of Alex and Bob's time was spent **constructing the right query** in an incremental fashion, and running several queries that re-

turned irrelevant answers, some of which involved wasteful scans of the entire database.

3. CHALLENGES

Both our users faced similar hurdles in their querying process, regardless of their skill level. The challenges they faced resulted in a time-consuming query session and wasteful querying of the database. To understand the challenges better, we categorize them into four classes:

Lack of Knowledge: The primary challenge encountered when dealing with data is the knowledge required to query it. First, the user needs to be familiar with the **query language** at hand. Second, the user needs to be aware of the **schema,** to be able to construct the right query. Third, awareness of the values of the database, i.e. the **data** is also important; so as to provide the right set of predicates in the query. In our example, the user was required to construct a `JOIN` on two tables and filter on *"Research"*, which requires knowledge of all three. In the absence of this knowledge, the user has to resort to trial-and-error, issuing wasteful and redundant queries just to understand the database.

Dependency of Information: An important observation is that *the construction of the query* itself often requires information from the database. While there are cases where this can be declaratively specified as a subquery (represented either as an outer `JOIN`, a nested query or even a conditional `IF/THEN` expression in a scripting language wrapped around SQL,) the intermediate information may be in a form that cannot be represented as a subquery. For example, Alex's decision that *age* is a good representative of employee seniority is possible only by looking at the data and corroborating with current knowledge about coworkers, an expensive intermediate step.

Iterative and Incremental Querying: An important consideration is that the cognitive capacity of the user is in fact limited [5]. While it may be possible to construct a single declarative query for many complex tasks, the human in the loop has a small finite memory. This limitation is overcome by rapid iteration and incremental construction of the query at hand. First, a simple query is constructed and executed on the database. Then, more complex predicates are added and the query re-executed, until the desired query is constructed and a satisfactory result is achieved. This paradigm of querying is both incremental and iterative: the query is constantly evolving during the query session, and the user is going through the *intent→query→execution→result* process many times. Due to the independence of queries, any notion of state is completely ignored, resulting in expensive requerying.

Imprecise Query Intent: The first three challenges make it harder for the user to express his intent. Another challenge is that the user may not have precisely expressed intent in the first place. In our example, when mapping concepts to elements in the schema, Bob has no way to map the fact that a group of people is represented by people who share the same `emp.project`, even after reading the schema documentation. The only way to confirm this is to actually execute the query and assess the results, and ensure that they are satisfactory (in our case, we execute the query and ensure that projects usually comprise 10-100 people.) Again, the database has to execute various trial-and-error queries that test various hypotheses for the user.

4. PRINCIPLES OF GUIDED INTERACTION

Based on these challenges, it is clear that we need to rethink the query-result paradigm for today's data-related tasks. As we have seen with interactions on the Web, end-users benefit heavily from responsive, interactive applications, such as instant web search and maps. We recognize these advantages and believe that they will prove just as useful in the database setting. By aiding the user in navigating the database, and by providing fluid and instantaneous feedback on their actions, it is possible to solve the challenges at hand. As a result, in this section we propose *guided interaction*, which is comprised of three complementary principles to follow when designing a database system. These principles hold true irrespective of the interaction mode, whether SQL, form-based, or imprecise keyword search.

Enumeration: *The database is responsible for effectively enumerating possible valid interactions with the data.*
By enumerating all possible queries to the user, we replace the burden of database (schema, data and query language) knowledge with a set of options presented in friendly ways to the user. In our example, prefix-matched *column:value* pairs from the database could be presented on each keystroke, enabling a successful search for "DatabasePrj" using automatic query completion (autocompletion.)

Insights: *The database must attempt to surface as many insights from the data as possible.*
In order to remove the informational dependencies observed in the previous section, the system should prompt the user with unsolicited insights into the data, such as value distributions or information mined from the data (association rules, template patterns, data clusters.) Like query fragment suggestions, the goal of these insights is to aid the user in expressing their original information need. Thus, the presentation of the insights needs to be done so as not to overwhelm the user, but at the same time present information most likely to be useful. We refer the reader of a wide body of prior work in the area of mixed-initiative models [11] that solves this problem. In our example, surfacing the correlation between age and seniority to the users would have helped them pick the right column to search for.

Responsiveness: *All interactions must be* instantaneous *even if inaccurate.*
Traditional query processing generates correct and complete answers, however long that takes. During query formulation, the user needs a sense of the data ("insights",) but these are useful only if they come quickly, i.e. while the user is actually entering the query. Therefore, we require that the database system be constantly responsive, even at the price of accuracy. The introduction of a finite time limit (100ms has been shown [3] to be an acceptable threshold for perceptibly instantaneous interactions) forces us to rethink many of the computations performed. In our example, the users may trigger an expensive query (e.g. complex JOIN on all employees) that could run for hours with no indication or guarantee of success. We propose to augment existing systems with summary data structures that can provide approximate answers quickly to the user, deferring to the actual database for a more accurate answer when it is ready.

5. RELATED WORK

Database Usability: While there is a body of work on making database systems usable [12], ranging from in schema summarization to query interfaces, it still conforms to the existing query-result paradigm. Guided interaction blurs the line between query and result, in order to satisfy the user's information need more effectively.

Query Construction: A large variety of interfaces have been built to aid construction of database queries; e.g. Query-By-Example [18], which allows the user to express the query using examples of what a result might look like. These systems require the user to be familiar with the data model they are querying against, which may not be true for the naive user. An alternative approach is to eschew all structure in the input query and assume that the query is a bag of words. Systems implementing *keyword search in databases* [17] accept a set of keywords as a query, and heuristically generate a collection of results. This paradigm fails to surface insights such as the data and the schema to the user, and forces the user to perform trial-and-error queries. To aid the user in constructing queries, web search engines provide query completion based on past query logs. In [16], autocompletion is used to suggest predicates to the user in order to create conjunctive `SELECT-PROJECT-JOIN` queries. In a similar light, work has also been done to use mine SQL query logs to derive and suggest reusable query templates [13].

Iterative Querying: In contrast to autocompletion, the concept of *find-as-you-type* allows the user to quickly iterate through the query process by providing results on each keystroke. In *Completesearch* [4], Bast et al. modify the inverted index data structure to provide incrementally changing search results for document search. In the information retrieval area, Anick et al. [2] achieve interactive query refinement by extracting key concepts from the results and presenting them to the user. Faceted search [8] extends this to present the user with multiple facets of the results, allowing for mixing of search and browse steps.

Online, Adaptive and Approximate Querying: The work done in the AQUA [1], CONTROL [9] and Telegraph projects [10] focus on surfacing approximate answers to the user in an online fashion using sampling-based techniques. This aligns well with the *responsiveness* requirement described in Sec. 4, and can be considered be an excellent execution of this requirement. However, in addition to being unfamiliar with the results, guided interaction posits that the user is also unfamiliar with the data, schema and the query language itself, motivating the need for *enumeration* of all possible interactions with the database. This is in contrast to the traditional approach of using a distance metric to infer a best-effort exact relational query from approximate queries [15].

Presentation of Results and Insights: The principles of *insights* and *enumeration* pose an important challenge: how does one combine and present feedback from each iteration to the user? This issue has been discussed at length in the context of mixed-initiative user interfaces [11], by employing multiple agents, where each agent can contribute to the task that it does best. This approach can be used to expose insights from the data based on the expectation of each query. In addition to prioritization, the need for visualization of results is an important component, especially for analytical queries [6] . Interactive data mining has been approached by implementing query languages and combining them with an easier-to-use visualization layer [7]. Tableau [14] performs this by translating VizQL queries to SQL, leveraging the query optimizer to remove inefficiencies. These platforms are excellent candidates for guided interaction, since we can incorporate the enumeration and insights as visual cues.

6. CONCLUSION

In this paper, we propose *guided interaction* as a paradigm for data interaction. We point out shortcomings with the existing query-result model, extracting challenges encountered that can be addressed by adhering to three core principles. We mandate that databases be *responsive* to the user, that all possible actions be *enumerated* so as to allow discovery and exploration, and that the database preemptively deliver *insights* to aid in query construction. These principles are of value independent of user capability and the specific interaction interface, whether that be SQL-writing, form-filling, keyword-typing or any other interface. We suggested how information in the database can be leveraged to guide a user during query construction by following these core principles.

7. REFERENCES

[1] S. Acharya, P. Gibbons, and V. Poosala. AQUA: A Fast Decision Support Systems using Approximate Query Answers. *VLDB*, pages 754–757, 1999.

[2] P. Anick et al. The Paraphrase Search Assistant: Terminological Feedback for Iterative Seeking. *SIGIR*, pages 153–159, 1999.

[3] B. Bailey et al. The Effects of Interruptions on Task Performance in the User Interface. *INTERACT*, pages 593–601, 2001.

[4] H. Bast and I. Weber. Completesearch: Interactive, Efficient, & Towards IR/DB Integration. *CIDR*, 2007.

[5] B. Britton et al. Effects of Prior Knowledge on Use of Cognitive Capacity. *Verbal Learning & Behavior*, 21(4):421–436, 1982.

[6] U. Fayyad, G. Grinstein, and A. Wierse. Info. Vis. in Data Mining & Knowledge Discovery. *M. Kaufmann*, 2002.

[7] M. Ferreira de Oliveira and H. Levkowitz. Visual Data Exploration to Visual Data Mining. *Visualization & Computer Graphics*, pages 1–8, 2003.

[8] M. Hearst. Design Recommendations for Hierarchical Faceted Search Interfaces. *ACM Faceted Search*, 2006.

[9] J. Hellerstein, R. Avnur, et al. Interactive Data Analysis: the Control Project. *Computer*, 32(8):51–59, 2002.

[10] J. Hellerstein et al. Adaptive Query Processing: Technology in Evolution. *TCDE*, pages 7–18, 2000.

[11] E. Horvitz. Principles of Mixed-Initiative User Interfaces. *Human Factors*, pages 159–166, 1999.

[12] H. Jagadish et al. Making Database Systems Usable. *SIGMOD*, pages 13–24, 2007.

[13] N. Khoussainova et al. SnipSuggest: A Context-Aware SQL-Autocomplete System. *PVLDB*, 4(1):22–33, 2011.

[14] J. Mackinlay, P. Hanrahan, and C. Stolte. Automatic Presentation for Visual Analysis. *IEEE TVCG*, pages 1137–1144, 2007.

[15] A. Motro. VAGUE: A user interface to relational databases that permits vague queries. *TOIS*, 6(3):187–214, 1988.

[16] A. Nandi and H. Jagadish. Assisted Querying Using Instant-Response Interfaces. *SIGMOD*, page 1156, 2007.

[17] M. T. Ozsu, L. Chang, and J. Yu. *Keyword Search in Databases*. Morgan & Claypool Publishers, 2010.

[18] M. Zloof. Query-By-Example: a Data Base Language. *IBM Systems Journal*, 16(4):324–343, 1977.