Fast Cartography for Data Explorers

Thibault Sellam
CWI
thibault.sellam@cwi.nl
Supervised by: Martin Kersten

ABSTRACT

Exploration is the act of investigating unknown regions. An analyst exploring a database cannot, by definition, compose the right query or use the appropriate data mining algorithm. However, current data management tools cannot operate without well defined instructions. Therefore, browsing an unknown database can be a very tedious process. Our project, Atlas, is an attempt to circumvent this problem. Atlas is an active DBMS front-end, designed for database exploration. It generates and ranks several *data maps* from a user query. A data map is a small set of database queries (less than a dozen), in which each query describes an interesting *region* of the database. The user can pick one and submit it for further exploration. In order to support interaction, the system should operate in quasi-real time, possibly at the cost of precision, and require as little input parameters as possible. We draft a framework to generate such data maps, and introduce several short- to long-terms research problems.

1. ANSWERING QUERIES WITH QUERIES

In business as well as in science, the volume and complexity of the data to be handled has been rapidly expanding. This has led to a massive research effort, in at least two directions. On one hand, database scientists have been focussing on the storage and retrieval of this data. This led to scalable, robust, and now well established data management systems. On the other hand, data miners have been trying to make sense of this data by exploiting statistical regularities. In both directions, many of the research challenges set several decades ago have been successfully tackled. It is now possible to detect dubious credit card transactions among Terabytes of transactions, or detect interesting target populations for marketing with acceptable latency and precision. Indeed, coupling massive data repositories and smart data mining algorithms allows solving many of the data analyst's problems. That is, when the problem is clearly defined. However, what if a user wants to *explore* the data? A database can only give a precise answer to a precise query. Similarly, the type of "knowledge" to be extracted by current data mining software must be clearly defined (classifiers, item sets, clusters...) and parameterized. In both cases, the user must already have a good understanding of his data and his goal.

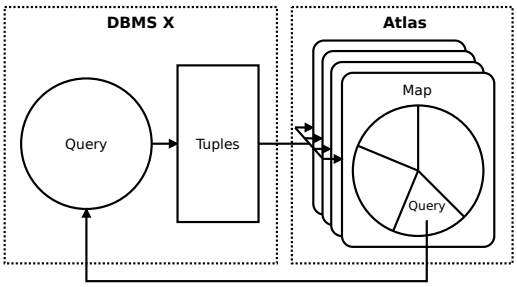

Figure 1: Answering queries with queries

Suppose a user has access to a large dataset stored in a SQL-based DBMS. All he knows about the data is the semantics of the columns. He may just want a preliminary "feel" for the data, or he is looking for facts with only a vague idea of how to reach them. In other terms, he is browsing. In the file system world, file managers offer a simple way to conduct such explorations. In the database world, we believe that the query-tuple paradigm lacks lots of flexibility. More precisely, most database front-ends fail to fulfill three requirements. First, a database interface should not require any complex query or statistical model specifications. Instead, it should present simple exploration options among which the user can choose. Second, the query latency should be close to zero even with large sets. Finally, it should favour intuition over accuracy: exploration requires rough impressions of the data, not lists of tuples or full statistical reports.

In this direction, I will dedicate my PhD to a novel data exploration scheme: answering queries with queries. Our system, Atlas sits on top of a traditional DBMS. The DBMS takes a query as input and returns a list of tuples. The idea behind Atlas is to analyze these tuples, and summarize them with a few queries. We call such a summary a *data map*. Each query describes a *region*. The user can do two things. He can refine his exploration by submitting one of the queries for further analysis. The region will then be itself broken down. Otherwise, if he is not satisfied, he can request a new map. This is illustrated in Figure 1.

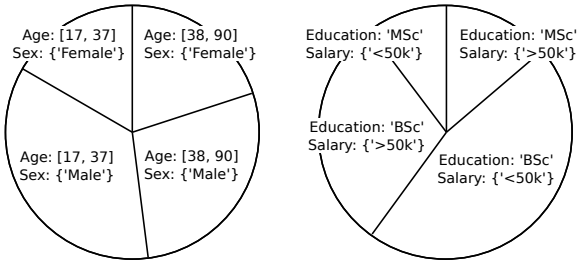

Figure 2: Two maps of the same data

Figure 2 gives an example. Consider a survey dataset. We issue a simple conjunctive query:

```
Sex:       any
Salary:    any
Age:       [17, 90]
Eye color: {'Blue','Green','Brown'}
Education: {'Bsc', 'Msc'}
```

Instead of returning a long list of tuples, Atlas clusters the results and describes the subsequent groups with queries. This can be done with the attributes "Age" and "Sex", as depicted on the left side. Alternatively, the attributes "Education" or "Salary" may also be used, as shown on the right side. Actually, Atlas generates several possible ways to describe the data (more than two), and ranks them by interest. The aim of my PhD is to develop methods to generate such views over any relational database, in real time.

2. INFORMATIVE DATA MAPS

We want to create data maps to support database exploration. These maps should be informative: they should be *representative* (they "make sense") and *convenient*. This is what we describe in this section.

As stated previously, a map is a set of queries over a dataset. These queries describe a subset of items with a subset of attributes. We consider that a query is representative if two requirements are fulfilled. First, its attributes should be semantically related. Consider the introductory example. It seems more natural to group "Education" with "Salary" rather than with "Eye color". A simple way to detect this relation is to group variables that are statistically dependent. Second, the items covered by each query should be similar. This means that they should be close in the space defined by the chosen attributes.

In order to enable an intuitive exploration, the maps should be convenient. Instead of returning one exhaustive solution as most clustering algorithms would (for instance, a dendogram), Atlas should return several easily understandable maps. Inside these maps, the number of regions should be kept as low as possible. As a rule of thumb, we consider that a map with more than 8 regions is hard to read. Also, the queries should be simple, with very few predicates (we target less than 3). Finally, the order in which we display the maps matters. The system should rank them, displaying the most interesting options first.

It is now possible to formulate our research problem. From a user query, we generate a ranked list of partitionings. Each of these partionings should group similar items on a set of mutually dependent attributes. The number of partitions and attributes should be kept low. The algorithm should work on any data type present in a DBMS, with a small latency and few input parameters.

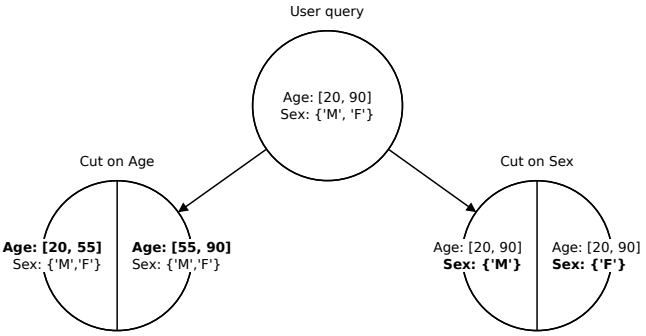

Figure 3: The CUT operation

In data mining terms, Atlas may be seen as a "lazy" projective clustering system, designed for interactive exploration. A more detailed comparison between clustering and our approach is given in Section 6.

3. MAP GENERATION FRAMEWORK

In this section, we present the "skeleton" of an algorithm which, we believe, is a good first step in this direction. The algorithm is based on four steps. First, we generate several simple maps from the data, each based on a single attribute. This is the candidate set. Second, we cluster this set into groups of dependent maps. The intuition is that if two maps are statistically dependent (and therefore in the same cluster), then they describe the same aspect of the data. The third step is to merge the maps in each cluster; thus we form the result set. In the last step, we rank this set and present it to the user.
The algorithm is not yet finalized: for each of these steps, we discuss several alternatives. We introduce the following notations:

- $P_k : att_k \in S_k$ is a predicate, with k in $[1, N]$
- $Q = P_1 \wedge \ldots \wedge P_N$ is a conjunctive query which describes a region.
- $C(Q)$ is the *cover* of a query: the number of items described by a query divided by the total number of tuples in the database
- $\mathcal{M} = \{Q_0, \ldots, Q_M\}$ is a map

3.1 Candidate maps

The aim of this step is to create several simple maps, which will be combined later to form more complex ones. These maps are obtained by successively breaking down each predicate of the user query. This is the CUT_k primitive. It takes a query as input, and splits the range covered by the k^{th} variable.

Definition 1. Consider the query $Q = P_1 \wedge \ldots \wedge (att_k \in S_k) \wedge \ldots \wedge P_N$. The primitive CUT_k creates a map of M regions as follows:

$$CUT_k(Q) = \left\{ P_1 \wedge \ldots \wedge (att_k \in S_k^j) \wedge \ldots \wedge P_N \right\}_{j \in [1,M]}$$

with $\bigcup_{j=1}^{M} S_k^j = S_k$, and $S_k^i \cap S_k^j = \emptyset$ for any i,j in $[1, M]$

The *CUT* operation may be seen as one-dimension clustering. Its simplest form splits each predicate in two. Consider the example pictured in Figure 3. The user provides a query based on *Age* and *Sex*. We decompose this query on each of these attributes. In this case, we cut the attribute *Age* around the value 55, and separate males from females. Two questions immediately follow: how should we partition the predicate ranges S_k, and how many partitions should we create?

Cutting method

There are many different ways to split the value range of an attribute into "interesting" portions. Consider an ordinal attribute (e.g. date, integer). We consider two options. First we could use equi-width binning. This gives fast and intuitive results. However, this does not tell much about the shape of the underlying distribution. An alternative is to split the data such that the intra-cluster distance is maximized within each partition (as in K-means [5]). This tells much more about the data but requires more calculations.
For categorical attributes, creating homogeneous partitions is harder because there is no natural order. We could however consider the values in the order in which the user gives them. If the user does not provide such information, we could use the frequency of occurrence of each value. Alternatively, if the data has a high cardinality (name, codes), we can use a simple alphabetic order.

Number of splits

Clearly, the more partitions per attribute we create, the more the subsequent calculations will be accurate: the algorithm will have a smaller chance of error when it will identify the map dependencies in the next step. However, this comes at the cost of more expensive computations. As we value performance to accuracy, we choose to restrict the number of partitions per attribute to two.

3.2 Clustering

In this step, we cluster all the maps that are *related*, that is, all the maps which describe the same aspect of the data. To do so, we need a distance function and a clustering algorithm.

Distance

We need a distance function d that quantifies how related two maps are. We propose to use a metric based on statistical dependency. First, we need to associate a random variable to each map.

Definition 2. Consider a dataset, described by a map $\mathcal{M}$. We take a random tuple in this set. X is the variable which describes its partition in $\mathcal{M}$. We call it the underlying variable of $\mathcal{M}$.
The possible outcomes of X are the queries of $\mathcal{M}$, and its distribution is determined by the cover of the associated areas.

If $\mathcal{M}^1$ and $\mathcal{M}^2$ are two maps, we need a distance function such that the more X^1 and X^2 are mutually dependent, the lower $d(\mathcal{M}^1, \mathcal{M}^2)$ gets. There are many such functions in the statistics literature. One approach would be to use a metric based on the mutual information [3]. Intuitively, the mutual information measures how much knowing the value of one variable reduces the uncertainty about the other. Although it can be quantified, it is not a metric in the mathematical sense (the triangle inequality does not hold). However, the literature offers several variations with better properties. For instance, the Variation of Information [10] seems to be a good candidate, as it is well established and relatively simple.

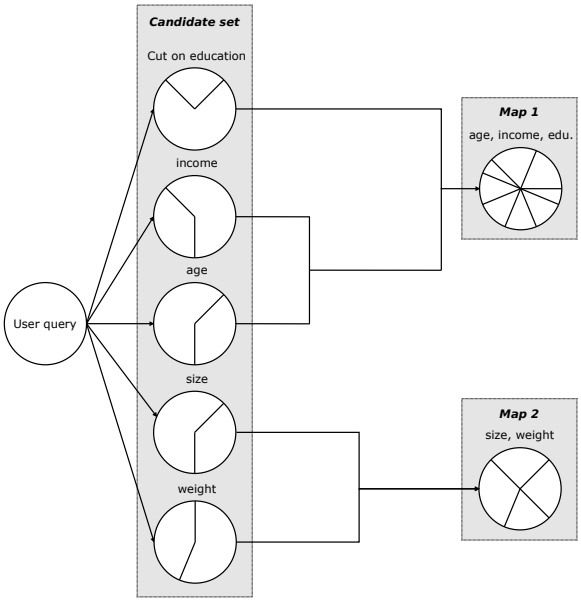

Figure 4: Agglomerative map clustering

Algorithm

Now that we defined the distance between two maps, we have to choose a clustering algorithm. The literature provides many suitable candidates. We favour agglomerative hierarchical methods such as SLINK [14]. These methods start with one cluster per map, then progressively merge pairs of similar clusters. We believe this is the best choice for two reasons. First, we do not know a priori the numbers of clusters to form. This discards all centroid-based methods such as K-Means. Second, recall that we set a limit on the complexity of the maps that we wish to generate. A hierarchical algorithm allows us to control the size of the clusters, and thus the number of areas in the result. Figure 4 gives an example of such an algorithm. In total, three merge operations are performed.
As the clusters grow, the maps they contain are less and less dependent. There should be a point after which two maps are too far away to be aggregated. However, it is not yet clear how to set this parameter.

3.3 Merging

During this step, we combine all the candidates of the same cluster into a representative map. We propose two methods, illustrated in Figure 5.

3.3.1 Product

This is the simplest way to merge maps. We create the product of two maps by intersecting each region of the first with each region of the second.

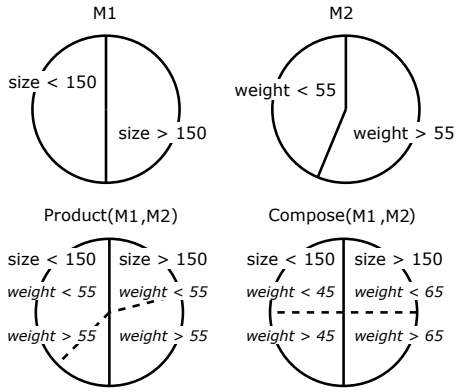

Figure 5: Product and composition of two maps

Definition 3. Consider $\mathcal{M}^1 = \{Q_0^1, \ldots, Q_K^1\}$ and $\mathcal{M}^2 = \{Q_0^2, \ldots, Q_L^2\}$. We define the *product* operator $\times$:

$$\mathcal{M}^1 \times \mathcal{M}^2 = \{Q_i^1 \wedge Q_j^2 \mid (i,j) \in [0,K] \times [0,L]\}$$

As this operation is associative and commutative, it can be trivially extended to any number of maps. The main advantage of this method is that it gives fairly "natural" partitionings of the space, especially if we apply equi-width splitting in the first part of the algorithm. However, if there are any clusters in the data, it is unlikely that they will appear on the map.

3.3.2 Composition

The idea behind composition is to cut the queries of one map on the attributes of the other.

Definition 4. Consider $\mathcal{M}^1 = \{Q_0^1, \ldots, Q_K^1\}$ and $\mathcal{M}^2 = \{Q_0^2, \ldots, Q_L^2\}$. Suppose that the queries $\mathcal{M}_2$ are based on the set of attributes att_1, att_2,..., att_N. We define the *composition* operator $\circ$:

$$\mathcal{M}_1 \circ \mathcal{M}_2 = \bigcup_{k=1}^{K} CUT_1\left(CUT_2\left(\ldots CUT_N\left(Q_k^1\right)\ldots\right)\right)$$

Consider by example Figure 5. The queries of $\mathcal{M}_1$ are based on *size*, those of $\mathcal{M}_2$ on *weight*. We compose these two maps by splitting the queries of $\mathcal{M}_1$ on *weight*.
If we use the intra-cluster distance as a cutting criteria for the *CUT* operation, then this method has a higher chance of revealing the clusters in the data. This chance depends on the distribution of the data: this is to be evaluated empirically or theoretically in future works.

3.4 Ranking

In the previous step, we generated a result set of several maps. We propose to rank them by decreasing order of entropy. The maps with many queries will have a high score. If two views have the same number of queries, then the entropy favors the most balanced one. Thus, the maps with many large areas will appear first, while the last ones will tend to reveal small subsets of outliers.

4. ARCHITECTURE

Ultimately, this PhD project should lead to a real-life system. Therefore, we are not only interested in the map derivation algorithm, but also how we implement it.

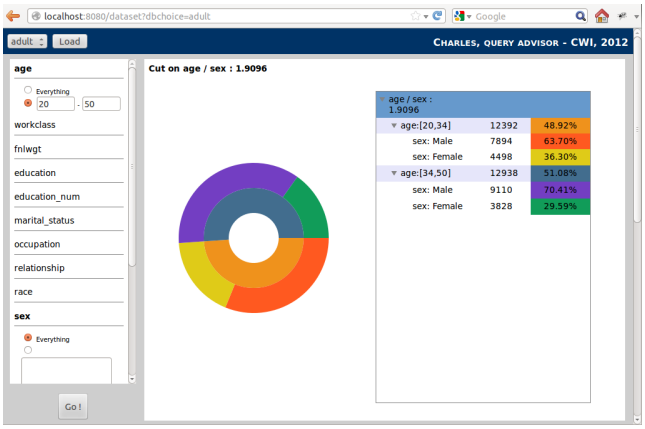

Figure 6: Screenshot of Atlas' GUI

We are currently developing a prototype for Atlas. It consists of three layers. At the bottom, a DBMS serves the dataset. Currently, Atlas is implemented on top of MonetDB [9], using its native C driver (MAPI). Ideally, the system, should be completely generic, and therefore support standard APIs such as ODBC or JDBC. However this limits the scope of the operations that can be pushed to the database, as only SQL may be used. The second layer is the actual mapping engine. This is where the main algorithms reside. The user can directly interact with this layer through a proprietary query language [13]. This language is a restriction of SQL which can only express conjunction of predicates. We consider that a well designed GUI is vital for such a an application: this is the third layer. As demonstrated in Figure 6, it is currently designed as a Web application. In a near future, we plan on using handheld devices such as iPads.

5. DEVELOPMENTS

This section presents several mid-term research questions that have to be tackled for the completion of this project.

5.1 Volume and time

Our approach is still too complex for large data sets. One of the main challenges of this project is to bring query and computational costs down. We propose three research directions.

Algorithm optimization

All the steps of the algorithm can be accelerated with small scale optimizations.
For instance, the primitive *CUT* is called many times. It is therefore vital to make it run as fast as possible. We remind that it takes a set of tuples as input, and splits one of its column according to some criteria (currently, we use the median). This could be approximated with one-pass algorithms such as sketches [1].

Sampling and refinement

One way to reduce the cost of all the computations would be to work on a sample of the full database. Nevertheless, determining a priori a trade-off between accuracy is running time is far from trivial. Therefore, the ideal algorithm would be an *anytime* variation of our framework [16]: the

quality of the results would improve as computation time increases. It would continually take small samples of the data and update a set of approximate results. This way, the user would have instant results and the system could interrupt the exploration after a timeout.

Anticipative computations

The idea of this approach is to perform calculations offline, by anticipating what the user will ask. There are two periods during which this is possible: before the first query, and during the idle time between each query. Deciding what to compute is an open question.

5.2 Real life challenges

Real life databases

The type of datasets we are interested in is quite different from the usual data mining sets: we would like to work directly on relational databases (e.g., the SDSS or TPC data). This brings several issues.
First, the logical layout of the data is more complex than one large table: we have to consider multiple tables with foreign key relationships. The naive way to deal with this would be to materialize the join into one large temporary table. However, this operation may be very costly. It is necessary to work on subsets only, or push some computations down to individual tables. How to do this is not yet clear. Second issue, some columns may have a very large cardinality and/or no semantics (codes, names, comments or keys). A failure to detect this could lead to very long and useless computations. Third issue, the raw data may be imprecise or contain mistakes. We believe that more issues will appear as the system is developed and tested.

Real life users

Our system targets human user. Therefore, the clarity of the interface and the visualization is a crucial component of the project. Representing the maps and the description of the regions without cluttering the screen space is difficult, especially with high cardinality attributes such as names. In the future, it could be interesting to describe the regions with random or, if possible, representative examples. Even better, one research direction would be to explain *why* a region is interesting, by charting the attributes of the subset versus those of the whole database. Another direction would be to propose *personalized* sessions, during which what is proposed depends on the past behavior of the user or his peers (as in collaborative filtering).
Finally, note our scheme is very well suited to mouse and keyboard free devices, such as tablets. On a more user interface perspective, this direction seems promising.

6. RELATED WORK

Clustering

Our work may be seen as a form of cluster analysis. More precisely, it is a variant of subspace clustering [8]: we detect groups of similar items in the subspaces of the dataset. Our system differs from existing projects on two points. First, the goal is different. We do not aim at finding all the clusters in the data, we want to support data exploration. Therefore, our requirements concerning statistical accuracy are lower but we target high speed and ease of use. Second, our method explores the data space *lazily*: after each iteration, we propose several several "embryo" of search directions among which the user chooses. It seems that all other approaches return one exhaustive list of clusters/subspaces. Another research directions in the clustering literature resembles this project: some GUIs for clustering propose interactive hierarchical interfaces. For instance, gCluto [11] gathers many clustering tools under one graphical application, and has this feature. However, we believe that these interfaces require too many "knobs" and tuning to allow database browsing.

Visual mining

Visual data mining is a vast and growing field of research. Several concise representations of multivariate data have been proposed [6]. Also, several systems have gained commercial success, such as Tableau [15]. We believe that our approach is a nice complement to these techniques, but in no way an overlap.

Visual querying

Visual querying has been an open research topic for many years. For instance, one of the oldest and most famous systems is Query By Example [17]. In a sense, faceted search [4] serves a similar purpose. There is to our knowledge no other system that uses the clustering of the data to suggest queries.

Query suggestion

Several authors have proposed query recommendation algorithms. We divide their work in two groups. The first group uses the query log of the database [2] [7], drawing inspiration from collaborative filtering systems proposed on the Web. As our algorithm does not use the query log, we can drop the assumption that several experts have been independently using the same database for the exact same purpose. The second group is closest to us: the algorithms use statistics over the data. For instance, the system of Sarawagi et al. [12] detects abnormal behaviors in data cubes. There is to our knowledge no other work which uses the clusters in the data to suggest queries.

7. CONCLUSION

Current data management algorithms excel at solving specific problems. Nevertheless, the act of exploration is by definition empirical and imprecise. We believe that the ideas presented in this proposal will help building more flexible, intuitive and eventually inspiring systems.

8. ACKNOWLEDGMENTS

This work was supported by the Dutch national program COMMIT.

9. REFERENCES

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16, 2002.

[2] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *Scientific and Statistical Database Management*, pages 3–18, 2009.

[3] T. Cover, J. Thomas, J. Wiley, et al. *Elements of information theory*. Wiley Online Library, 1991.

[4] J. English, M. Hearst, R. Sinha, K. Swearingen, and K. Lee. Flexible search and navigation using faceted metadata. Technical report, University of Berkeley, 2002.

[5] G. Gan, C. Ma, and J. Wu. *Data clustering*. SIAM, 2007.

[6] D. Keim and H.-P. Kriegel. Visualization techniques for mining large databases: a comparison. *IEEE Transactions on Knowledge and Data Engineering*, pages 923–938, 1996.

[7] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: context-aware autocompletion for sql. *Proceedings of the VLDB Endowment*, pages 22–33, 2010.

[8] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, pages 1–58, 2009.

[9] S. Manegold, M. Kersten, and P. Boncz. Database architecture evolution: mammals flourished long before dinosaurs became extinct. *Proceedings of the VLDB Endowment*, pages 1648–1653, 2009.

[10] M. Meilă. Comparing clusterings – an information based distance. *Journal of Multivariate Analysis*, pages 873–895, 2007.

[11] M. Rasmussen and G. Karypis. gcluto - an interactive clustering, visualization, and analysis system. Technical report, University of Minnesota, 2004.

[12] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *Advances in Database Technology – EDBT'98*, pages 168–182, 1998.

[13] T. Sellam and M. Kersten. Meet charles, big data query advisor. *CIDR*, 2013.

[14] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, pages 30–34, 1973.

[15] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, pages 52–65, 2002.

[16] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, page 73, 1996.

[17] M. M. Zloof. QBE/OBE: a language for office and business automation. *Computer*, pages 13–22, 1981.