PatternFinder in Microsoft Amalga: Temporal Query Formulation and Result Visualization in Action

Stanley Lam
University of Maryland, Department of Computer Science

Honors Paper for Spring 2008 under the supervision of Dr. Ben Shneiderman

Abstract

As electronic health records (EHR) become more widespread, they enable clinicians and researchers to pose complex queries that can benefit immediate patient care and deepen understanding of medical treatment and outcomes. However, current query tools make complex temporal queries difficult to pose, and physicians have to rely on computer professionals to specify the queries for them. This paper describes our efforts to develop a novel query tool implemented in a large operational system at the Washington Hospital Center (Microsoft Amalga, formerly known as Azyxxi). We describe our design of the interface to specify temporal patterns and the visual presentation of results, then summarize the feedback gathered during early testing with physicians. The use case described in the paper focuses on adverse reactions following radiology studies using contrast.

Introduction

As the use of electronic health records (EHRs) spreads, there are growing opportunities for use in clinical research and patient care. Queries often have a temporal component. For example "Find all patients who were discharged from the emergency room then admitted again within a week". Another example is "Find patients who had a normal serum creatinine lab test less than 2 days before a radiology test with intravenous contrast, followed by an increase in serum creatinine by more than 50% and of more than 1.0 mg/dl within 5 days after the contrast administration". Currently available user interfaces make possible simple queries such as "Find patients who had a radiology test with contrast and a high value of creatinine, leaving users with the burden of shuffling through large numbers of results in search for matching patients.

Specifying temporal queries in SQL is difficult even for computer professionals specializing in such queries. Scientific research has made progress in representing temporal abstractions and executing complex temporal queries (e.g. Shahar98; Shahar00, Augusto05, Stacey07) but there is very little research that focuses on making it easy for clinicians and medical researchers to specify the queries and to examine the results visually.

We believe that interactive query interfaces allowing researchers and clinicians to explore data that have specific temporal patterns in both numerical and categorical data can dramatically increase the benefits of EHR databases. Comprehensive presentation of the results can then help users see patterns and exceptions in the data they retrieved, and refine their query accordingly. This paper describes our efforts to implement such an interface in a large operational working system at the Washington Hospital Center (Microsoft Amalga, formerly known as Azyxxi).

Temporal searches are used in many situations, from clinical trial recruitment, clinical research, general patient care (e.g. tracking the application of guidelines) or alarm specification. For example, setting an alarm for patients on Heparin with a precipitous drop in platelet counts (heparin-induced thrombocytopenia) requires specificity around the definition of "precipitous". By querying existing EHR databases, physicians designing the alarm can iteratively test the logic of the alarm (e.g. Should an absolute drop or a relative drop be used as a marker? What is the appropriate time range to use?) and validate it with a large amount. Clinical medicine is always concerned about changes from some baseline state. A blood pressure of 90/60 may be normal for a 25 year old female but may represent severe

hypotension in a 65 year old male hypertensive patient whose blood pressure during previous visits was 160/100. Clinicians are always seeking changes in the value of some clinical measure to determine whether or not an intervention should be taken.

We believe that interactive query interfaces allowing researchers and clinicians to explore data that have specific temporal patterns in both numerical and categorical data can dramatically increase the benefits of EHR databases. Comprehensive presentation of the results can then help users see patterns and exceptions in the data they retrieved, and refine their query accordingly. This paper describes our efforts to implement such an interface - called PatternFinder, in a large operational system (Microsoft Amalga, formerly known as Azyxxi) at the Washington Hospital Center (Figure 1).

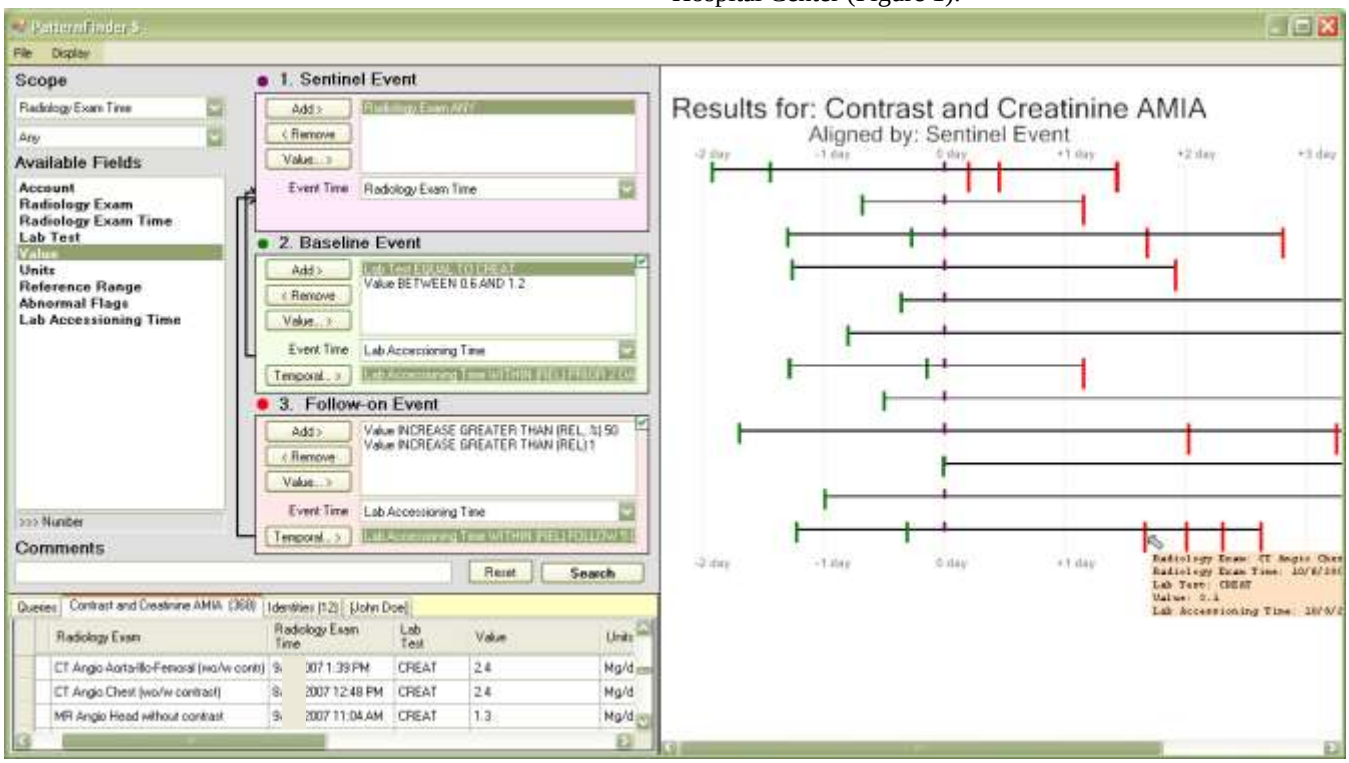


Figure 1 PatternFinder's main screen. The user has specified the following query: "Find patients who had a normal serum creatinine lab test (the baseline event) less than 2 days before a radiology test with intravenous contrast (the index event), followed by an increase in serum creatinine by more than 50% and of more than 1.0 mg/dl relative to the baseline measurement, within 5 days after the contrast administration". The query returned 12 patients. For each patient a timeline show the timing of events that match the query, aligned by the index event. Each event is drawn as a color coded tick marks. In this query all baseline events appear on the left side of the index event, and follow on are on the right side. Options allow users to display the numerical values as well. Zooming reveals more details. ***NOTE: Medical records and dates have been modified and hidden to produce this figure.***

Related Work

Handling time-related concepts is essential in medicine. A survey (Augusto05) lists the many applications that use the history of temporal events, and review the latest contributions to time-aware decision support systems. Research into causality, natural language, argumentation is very hot topic in AI and Medicine research community. However, Augusto observed that there are still opportunities for providing much needed search tools. Some research systems provide temporal access languages to support limited visual queries from end-users (Catarci97; Cheng97; Cheng 99; Jensen 99), but many of these suffer the same accessibility difficulties of SQL or they require an understanding of the underlying database structure. The large body of related computer science work can be grouped into three general areas: time theory, databases, and visualizations. None of the systems discussed in those papers enable query or visualizations of patterns across multiple entities (e.g. temporal cross-patient queries). This is a major contribution of our work which will combine power of temporal queries

with graphical visualizations, integrated in an operational EHR system.

Time Theory: Much of the seminal work in computer science relating to time stems from artificial intelligence, time reasoning, and early natural language processing (Bruce72; Allen83; Kahn77). Time can be characterized by point or interval events. Snodgrass defines instance events as being absolute and intervals as being relative distances between two instances (Snodgrass99). Allen introduces time intervals as the primitive for automated reasoning over temporally structured data (Allen83). Our early prototype PatternFinder supports intervals, including: start/stop event pairs, and as sets of events that share a user-defined characteristic, which together span an extent of time. Although overlapping queries are possible, our proposed approach supports temporal sequence patterns more directly.

Databases: Due to the complexity of formulating SQL queries, several approaches have made database query more accessible to a broader spectrum of users, e.g. Query By Example (QBE) presents the structure of the database as skeleton tables, and is the visual query mechanism used in Microsoft's Access (Microsoft00). Although visual query languages facilitate database query by avoiding SQL syntax, users must still understand the relational tables and formulate queries, using variables and other difficult concepts. Our work will presents users with visual constructs for querying the temporal relationships among events.

Numerous extensions to the relational model have been proposed to incorporate time, such as TSQL2 (Jensen99), an extension to the SQL-92 (ANSI86) language standard. A hybrid between QBE and Extended Entity-Relationship diagrams (EER) represents queries visually as EER objects for which attributes and variables can be instantiated (Kouramajian95). Temporal queries are supported by temporal operator objects in the diagram including the qualifiers: before, during, after, start and end. This work was expanded by (Silva97). Neither work visually encodes temporal aspects and relationships in a succinct or orderly way taking advantage of the ordered nature of time. MQuery (Dionisio96) targets various types of streaming data. Temporal features can be captured by before/after date specifications and left to right positioning of query objects, but it does not provide a higher level representation of the temporal aspects of the data or the query itself.

Visualizations: Chittaro and Combi proposed three alternative visual metaphors for querying temporal intervals (Chittaro03). Hibino and Rudensteiner introduced a direct manipulation Temporal Visual Query Language (TVQL) (Hibino97) to support Allen's 13 relational primitives. Interestingly, none of the work described above address the visualization of the returned results. However, applications such as TimeSearcher (Hochheiser04), DataJewel (Ankerst03), KNAVE (Cheng97) and LifeLines (Plaisant98) offer visualizations that cluster results and highlight temporal patterns. LifeLines provides a compact hierarchical timeline visualization for personal histories organized by facets, such as doctor visits, lab tests, and medications (Plaisant98). It focus on a single record and does not offer a query mechanism for discovery across multiple records. Many systems have built on LifeLines (e.g. Bade04). Shahar proposes a Knowledge Based Temporal Abstraction model RÉSUMÉ (Shahar98)

Again, we emphasize that none of the systems discussed above enable query or visualizations of patterns across multiple entities (e.g. cross-patient query). This is one of the major contributions of the proposed work which will combine power of temporal queries with graphical visualizations.

The implementation of PatternFinder in Amalga builds on early prototypes developed at the Human-Computer interaction Lab (Fails06). A limited set of temporal filters could be chained together with changes always relative to the previous event. Those prototypes only read small datasets from flat files. We worked with physicians to refine the prototypes and develop a taxonomy of query types. Later on class student project usability tested the early user interfaces and generated recommendations (www.cs.umd.edu/hcil/patternfinder). Lifelines 2 focuses in the exploration of the results once a query has been specified, and demonstrated the benefit of alignement (e.g. on index events) to see patterns in the data (Wang 2008).

Description of PatternFinder

A successful application to enable physicians, clinical researcher, or administrators to search electronic health records must include components for query specification, rapid execution, and display of results.

Interfacing with Amalga: While Amalga has a wide range of databases, the basic concept is that the databases are presented to the PatternFinder application using an event-value table data format. An event-value table joins patient information with event information, requiring a field for categorical or numerical event data in addition to a date-time associated with the event. Each row in the event-

value table contains information describing at least one event. Although Amalga uses a post-relational data model, the event-value tables used in Amalga are constructed as virtual tables by creating views that join multiple databases and tables together, which can have up to hundreds of fields. Figure 1 shows the interface linked to a data table containing radiology events with contrast and creatinine labs.

Query Specification Interface: To minimize the learning time for Amalga users, the PatternFinder interface has the look and feel of Amalga's existing data filtering tool, to which we added support for temporal functionality, i.e. the ability to specify multiple events, define relationships between events using a time basis, and compare values between events. While we aim to create a very general interface to handle a large number of medically interesting temporal patterns, we chose to focus our initial attention on the relationships of index events, baseline and follow-on events.

Users first define a specific temporal scope for the query (top left) e.g. choosing a date range for the radiology exam. Next they specify the index event, baseline event and follow-on event of interest in the three separate filter widgets. In the example of Figure 1, the index event is the radiology exam during which intravenous contrast was administered, the baseline event is a lab test result of a normal serum creatinine level, and the follow-on event is another serum creatinine lab test which shows an increase in serum creatinine level of at least 50% relative to the baseline value and of at least 1 mg/dl. The events are linked together using the temporal definition fields and dialog boxes (Figure 2). A date-time field asks users to select the appropriate database field to use if multiple date-time fields exist in the data source – a likely scenario in Amalga's database tables and views. The addition of these fields in the event definition interface serves as a visual reminder for the steps necessary for fully defining a temporal pattern (Our first iterations of PatternFinder did not include this separate date-time widget, so some users forgot to temporally link the events together). Most operators are relative to other event times or values and we allow users to select the event to be used as reference. Arrows on the left of the filter widgets indicate the chosen temporal connections between widgets (Figure 1).

Figure 2 Defining the temporal relationship

The temporal relationships use various operators and allow users to specify the reference time (e.g. here the index event or the baseline event).

The 'operators' available in any given event allow users to specify changes in field values between events. For numeric fields, examples of new operators include:

- "relative increase greater than X"
- "relative increase greater than X%"
- "relative decrease greater than X"
- "relative decrease greater than X%"
- "less than value in event X"
- "equal to value in event X
- "not equal to value in event X"

For date time fields, new operators include:

- "within X prior to (relative)"
- "within X following (relative)"
- "after X (relative)"
- "before X (relative)"
- "is equal to (relative)"

For text fields, new operators include:

- "equal to value in event X"
- "not equal to value in event X"

Display of Results: PatternFinder for Amalga uses two methods for showing results, a table view that mirrors the existing Amalga "Grid" view and a graphical view which combines aspects from LifeLines (Plaisant 98) and an earlier PatternFinder version using a ball and chain metaphor (Fails 06). On the lower left part of the screen the table view breaks result pairings up and places each event in a single row. Users can review all events, the list of patient ID that satisfy the query, or view all events for a single selected patient. To increase readability,

duplicate rows were eliminated from the table as events could trigger multiple result pairs/triplets. At the right end of each row, additional columns indicate which filter the event was matched to.

Lab Accessioning Time	Filter	Filter 1	Filter 2	Filter 3
4/5/2007 8:41 AM	1	X		
3/20/2007 10:07 AM	2		X	
4/1/2007 12:38 PM	3			X

Figure 3 Additional filter columns shown on the grid result display for a three-event temporal query.

On the right side of the screen the graphical display shows all records in the result set, labeled by the patient number (hidden in figure 1). For each patient a simplified LifeLines-based timeline summarizes all the events that match the query for that patient. A tick mark is drawn to indicate the time of each event, colored to match the event type, allowing users to see which events relate to each filter of the temporal query. Numerical values can be displayed below the tick marks (in our example the creatinine serum level value would be shown there)

Based on the result of our previous research (Wang08) an "align by" control was added to interactively align the records (either by the index event or the 1st baseline or 1st follow-on event). This action makes the time scale relative to the aligned event. In figure 1, the LifeLines have been aligned by the index event i.e. the time of the radiology exam.

Figure 4 A snapshot of the lifelines visualization of result-set using "align by" capability. Green hashes indicate the first event, purple hashes indicate the second event, and the red hashes indicate the third event. All patterns have been aligned by the purple secondary event.

Users can click on a "LifeLine" to expand it and see more details about the matches. Two options are available: 1) a single triplet display to show only one set of events matching the filters (i.e. events that are the closest to the index event) or 2) a display of all the possible triple combinations of events that match the query for that patient (this display is called a ball and chain display (Fails06).) Users can also zoom in to enlarge the display, or zoom out to see more patients and events at once. By hovering over any of the tick-marks from the LifeLines display (or balls in the ball-and-chain display) users can see detailed textual information for that specific event (date and time, test name, units etc.)

Rapid Execution

The PatternFinder converts the specified temporal query into a T-SQL (Transact-SQL, a Microsoft/Sybase extension of SQL) query using a series of self-joins to sequence multiple events together. A series of self-joins can be prohibitively expensive on large databases such Microsoft Amalga's and can form a bottleneck preventing rapid execution due to the massive amount of data the database is required to handle. These self-joins combine rows from the table with itself based on the conditions presented in a query, and have a worst case asymptotic complexity of roughly $O(n^2)$ for each self-join used. We found that preprocessing the 25.6 million rows of data from the virtual tables that our application queried from greatly helped in our goal of achieving rapid execution.

Sample SQL Query:

```
SELECT TOP 200
  *
FROM
  Labs Labs_1,
  Labs Labs_2
WHERE
  Labs_1.Observation = 'HGB'
AND
  Labs_1.Value BETWEEN '8' AND '9'
AND
  Labs_1.DtTm > '2005-12-30
  01:36:00.000'
AND
  Labs_2.Observation = 'HGB'
AND
  Labs_2.Value > (Labs_1.Value *
  (1 + .2))
AND
  Labs_2.DtTm > Labs_1.DtTm AND
  Labs_2.DtTm < dateadd(Hour, 5,
  Labs_1.DtTm)
AND
   Labs_1.MRN = Labs_2.MRN
```

Figure 5 A generated SQL query, searching for a 20% rise in hemoglobin levels within 5 hours

To accommodate for data sources that use multiple tables, a SQL view can be created in advance joining the tables of interest together into a single virtual table. This technique is used to narrow the scope of the queries possible and reduce the amount of data

exposed to PatternFinder, increasing the relevancy of the search data and reducing the overall query execution time. Preprocessing the source data and converting the virtual table into a real one, before using PatternFinder, also eliminates some of the work the SQL server has to do when performing temporal searches and was found to be an essential step towards providing results in real-time.

Preprocessing a virtual table with over 25.6 million rows of lab and patient information took 1 hour 13 minutes, and created a new table with only 600,000 rows. The table was later analyzed and optimized with additional indices. Without indices loaded into memory, a query against the 25.6 million row table took 20 minutes to finish whereas the same query against the 600,000 row table took 15 seconds to complete. With indices loaded into memory, the same query respectively took 6 minutes and 1 second to finish.

Readers should note the widely varying y-axes for the execution times, indicating the many orders of magnitude variations in performance with large data sets.

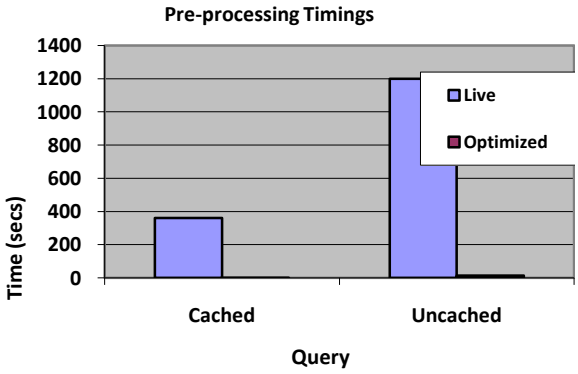


Figure 6 Speed advantages of preprocessing data. Preprocessing the data in advance showed a 80 to 360-fold decrease in query execution time.

Figure 6 above illustrates the speed differential between queries between preprocessed data and raw data, and cache indices and uncached indices. The first query in a multiple-query set against a new data source typically runs the slowest since Microsoft SQL Server has not loaded the indexes into memory yet. Once the indices have been loaded, significant speedups in query performance times can be gained. Therefore for consistency purposes, all performance test timing results, unless otherwise noted, are returned from a database using cached indices.

For a given dataset, the execution time for queries with multiple events can grow at a geometric rate, depending on the number of results returned by each event during the actual query. Therefore by placing tight constraints on the initial event in the temporal query, the total execution time can be significantly reduced.

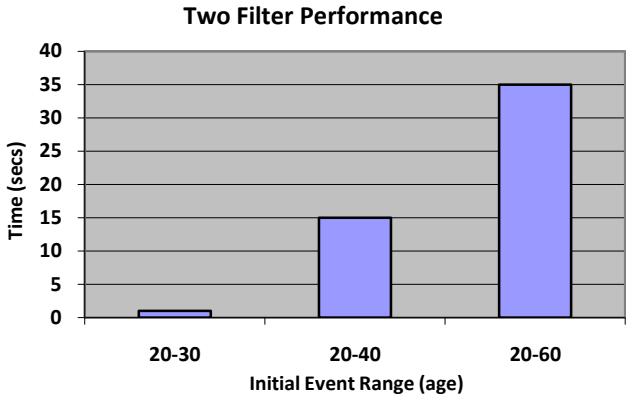


Figure 7 The performance of a two-filter query shows series degradations as the event range expands. Going from a 10-year to 20-year to 40-year produced a total 35-fold increase in the query execution time.

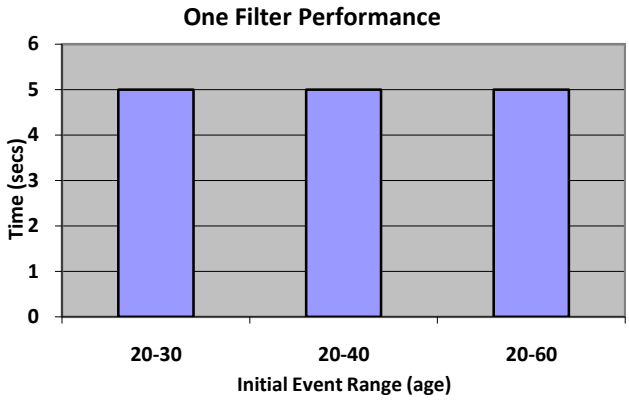


Figure 8 The performance of a one filter query with increasing initial event sets. Despite the increasing range of age conditions, the query execution time stayed the same.

In a two event filter performance test, we found that modifying the initial event and increasing the range of patient ranges greatly affected performance times as shown in figure 7. However, in a single filter query scenario, figure 8 demonstrates that modifying the event range seemed to provide no improvements.

Figure 9 The performance of a two filter query with increasing distance between events. Increasing the time range between events from 1-year to 5- years produced a 60-fold increase in query execution time.

Increasing the temporal aspect of the query and changing acceptable time between the events from 1 year to 5 years caused a 60x increase in query execution times, as shown in figure 9.

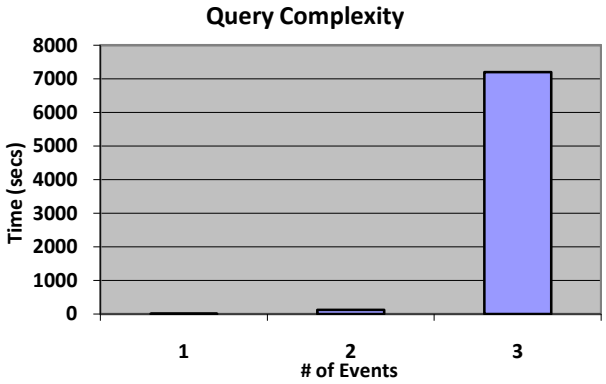


Figure 10 The performance of queries with varying # of events. Each increase in the number of events caused a 60-fold increase in query execution time.

A similar relationship was also demonstrated in our dataset by increasing the number of events in the temporal query, as shown in figure 10. For each event filter added to the temporal query, an additional self-join is used in the SQL query – which can quickly increase the overall query execution time.

During our initial testing, performance, however, did not seem to be a major concern for physicians using the temporal querying tool. Users were willing to trade performance and execution times for the ability to query as much data as possible. This propelled the application towards an integrated client-server model where the query would be handled by Microsoft Amalga's separate reporting server and a separate replicated database as well as a batch querying processing system.

The option to save and load the raw results returned for the temporal queries was added to PatternFinder, allowing users to save and export the data for later analysis. In the intended Microsoft Amalga environment, users are expected to protect the saved and exported information if the data contains confidential patient information.

Saved queries could also be executed against different data sources as long as the data sources contain the same table and fields names used by the query, allowing users to reuse previously defined queries in multiple environments.

Implementation Difficulties

Query performance became a recurring issue during the implementation phase, especially when all of the initial tests were performed on Washington Hospital Center's development and research servers. The development server acted as a staging ground for building the advanced querying tool as an integrated Microsoft Amalga component. For testing purposes, PatternFinderS was instructed to use the Washington Hospital Center's research database server. The research server allowed the complex temporal queries the tool produced to run on replicated snapshots of the hospital's 7 terabyte database, without putting excessive load on real Amalga servers used on the hospital floor. Although this configuration was immensely helpful in testing the tool in a realistic environment, getting new iterations of the tool onto the servers was a multiple step process alone and generating and running queries could take upwards of one hour depending on the table used and the complexity of the temporal query. By creating a stand-alone version of the application that could run independently outside of Microsoft Amalga and using a smaller test dataset, time between iterations was greatly improved, allowing for more rapid development to occur.

Another issue experienced with the SQL based PatternFinder is that the query can timeout depending on the complexity of the query and the size of the dataset. Without the proper database optimizations, query performance deteriorated to the point where it became difficult to verify whether or not queries were correctly constructed as no results were returned within a reasonable timeframe. To combat the extensive query times found by querying against the 7 TB datasets, smaller datasets of interest were extracted from the live Amalga database for

development purposes. Using a smaller limited dataset reduced the scope of queries available but allowed the more complex queries to complete within a few minutes. As the confidence level in the tool's query specification and SQL generation capabilities rose, the timeout time was increased.

As the waiting time for each query to complete grew, it became more and more obvious that some kind of query management system for running queries was needed. The job management system implemented stored query results and queries in a binary file on the Amalga client's machine, which later proved to be problematic in terms of providing backward-compatibility for previously created queries even though query parameters were stored using XML. Due to the changing definitions used to generate the SQL queries in the tool, old queries could be invalidated when functionality in the tool was added, replaced or removed. An alternate method to store the query history, such as .NET class serialization, was initially used during development but was later dropped due to difficulties experienced in the actual Amalga environment.

Another factor which made the implementation more difficult was limited domain-knowledge with medical terminology and the databases within Microsoft Amalga made it difficult to create meaningful test queries that accurately represented the same type of queries our intended users, physicians, would be creating. Even physicians who were already familiar with the existing Microsoft Amalga system reported that it was difficult to create certain queries since it required users to know in advance where the data of interest was located, and what each field in the database meant. During an initial demonstration, physicians at the Washington Hospital Center who were more familiar with the Microsoft Amalga databases made corrections to our demonstration temporal query that searched for patients with heparin induced thrombocytopenia.

Design Evolution

The first iteration of the design split each event in the temporal queries into multiple tabs, with each tab representing a single event. Although this allowed users to specify a lengthy series of events without cluttering the query building interface, this tab-based approach was later replaced with a more compact, inline approach. Placing the controls for multiple events into a single window instead of spreading them out across multiple tabs reduced the number of interactions required to both define and view queries.

The graphical visualizations only currently display data that match the specified temporal query, and do not show all event / patient information like the original LifeLines and PatternFinder applications do. Only a subset of the information is available since providing full patient information requires the application to have a full understanding of the data-source, which requires an appropriately created ontology that is outside of the scope of the PatternFinder for Microsoft Amalga project

Various changes were made to the query specification interface to allow events to be defined in a non-sequential order. The contrast and creatinine query formulated during our tests sequenced the events in a 2 – 1 – 3 order, positioning the first event specified (a radiology exam) in the middle of the temporal sequence of events. To reduce confusion, arrows were added into the interface to indicate how each event is related to the others.

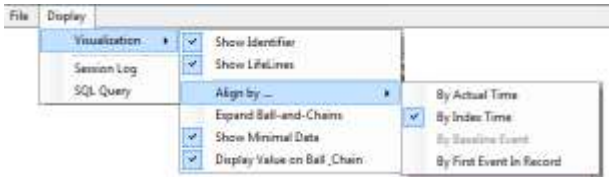


Figure 11 - Menu bar addition to UI, allowing the user to specify various visualization options

In the later iterations, many simplifications were made to improve usability. For example, many of the buttons and checkboxes for various features and options in the tool window were moved out of plain sight into the tool's menu bar. This addition greatly reduced the amount of clutter on the interface while preserving previous functionality. When specifying the temporal operator between events, all of the non-relational operators were removed to reduce confusion.

Based on user feedback, various assumptions were added in an attempt to more closely align the tool to the user's expectations. For example, the temporal time field, when not specified, defaults to be the same as the time field used in the user specified data-scope. The concept of implicit parameter inheritance, where old parameters would "fall-through" to other events unless overridden with new parameters, was also tested but was found to be confusing, particularly in the case of non-sequential temporal queries.

Initial Evaluation

While laboratory-based controlled experiments are effective to measure the performance of interfaces, our expertise evaluating user interfaces suggests that the complex cognitive skills and domain expert knowledge involved here mean that usability testing and case study methods are more effective in revealing problems and identifying strengths in interfaces such as PatternFinder. Adoption and the value of the findings obtained using the interface eventually become the measures of success.

During the past 3 months we worked with 4 senior emergency medicine physicians and researchers from the Washington Hospital Center. During each individual meeting we reviewed the interface, let physicians use it on their own, and discussed needed improvements. The interface was revised between each meeting. We first used a small de-identified dataset provided by the Washington Hospital Center's emergency department, then recently connected to a replicated database mirroring the live database. Being familiar with the Amalga interface and the data available, participants had no problems getting starting using the interface and were able to define some temporal queries with some limited help. The first iteration of the interface was very generic causing some confusion understanding what was possible or not, they suggested simplifying the interface to tackle simple problems more elegantly. For example, originally the interface had filter labels such as Filter 1, 2 and 3 and had no notion of index events. Providing a more guided interface seem to naturally lead users toward useful queries which can be correctly specified. Identifying the needed data table in Amalga remains a challenging step for many users. It may have a significant impact on the overall usability of our interface (even if unrelated to our design) which we hope will be addressed with the development of an interactive data dictionary for Amalga. Participants also identified additional temporal operators that are missing from the initial application: "event count" and "absence of". All physicians were very eager to see the prototype upgraded to the live Amalga system to access the live data, which is now possible. A simple interface allows users to submit multiple queries in batch mode, monitor the progress of long queries, review past queries and results and attach comments to them.

The case study of adverse reaction to contrast materials illustrates the potential benefit of PatternFinder. A significant decline in renal function after administration of intravenous contrast material is a not uncommon and potentially serious consequence of modern medical care. Its incidence is estimated at 2% - 7% of all administrations of contrast media. Most hospitals are unable to specify in real time (or even retrospectively) which patients and the percentage of patients who have had contrast nephropathy. The ability to identify and track these patients in real time is helpful at several levels. At the individual patient level, the dose of many medications needs to be modified in the face of reduced renal function. Second, at a population level, hospitals are increasingly being asked to provide objective measurement of quality of care. The overall hospital incidence of contrast-induce nephropathy reactions (which typically is not tracked in most hospitals) could become a standardized measure across hospitals if there were a simple way to gather the information. The temporal query methodology we describe here would provide such a way.

While the query does a good job at identifying populations of interest, PatternFinder needs to link each result directly to the detail views of the individual patient records (e.g. seeing all the lab results of a patient, instead of just those that match the query) to achieve maximum effectiveness. Similarly it will be useful to define an alarm based on a query, or to schedule queries to run at regular intervals to track performance metrics.

Project Progress

Major features / changes to the PatternFinder for Microsoft Amalga tool made during this development period included:

1) minimal configuration, domain-agnostic SQL support using Open Database Connectivity (ODBC)
2) multiple event temporal query specification interface, modeled off of Microsoft Amalga's filter interface
3) a client-side job management system which saves results and their respective queries while running queries in the background
4) the initial data-scope filter to the interface, as well as the option to do out-of-order temporal queries
5) a visualization core, which include "align by" and other options
6) session logging, tracking user actions and queries performed
7) Microsoft Amalga component integration as well as an independent stand-alone mode.

Future Work

Directly integrating PatternFinder's temporal query building capabilities into Microsoft Amalga is ultimately the Washington Hospital Center and Microsoft's goal for this project. By extending the current filter options in Amalga with the temporal functionality provided by this tool, Amalga will be able to explore a new dimension within the data that it manages as a Unified Intelligence System.

Adding integration hooks or data export capabilities will allow other systems, such as Taowei David Wang's Align-Rank-Filter visualization tool, to leverage the functionality provided by the tool.

Removing the restriction of a single view or table and adding support for multi-table joins or dynamic view (virtual table) generation would allow the tool to be used against normalized databases. Currently, the tool can produce multi-table SQL queries with the appropriate metadata describing how tables are joined together, but no interface into Amalga's system has been developed yet.

Server-side logging and job management on Amalga servers would allow users to operate the tool from any machine without losing previous query history and to close the tool without stopping the currently running query. This functionality is necessary for continued development into the Amalga environment since users routinely use their Amalga accounts on different computers. Without server-side support, users lose session data as they move from machine to machine.

The addition of alarm-based querying and repeating queries for reporting purposes, will also allow users to generate a set of performance metrics in which they can use to improve the quality of service provided in their hospitals. The use of a robust and real-time metrics system will help hospitals meet and exceed standards on the 17 key performance indicators used to assess hospitals. The current Amalga system has report-building functionality, but does not support the temporal queries that PatternFinder is capable of creating.

Conclusion

By bridging the worlds of data bases, user interface design and information visualization, we believe that we can create the next generation of potent visual analytic tools and information technology-enabled work environments. Each participant in our study found ways to use PatternFinder to search for patterns that their current tools could not locate today. Continued development into this area will allow users to potentially find answers to questions that they were never able to ask, and formulate a whole new set of questions that allow for a retrospective view of previous hospital performance.

Acknowledgements

We would like to thank Mark Smith, David Roseman and Greg Marchand from the Washington Hospital Center for supporting this project through funding the work, providing a research database, provisioning development systems, giving valuable and insightful feedback, participating in our usability tests and more. We would also like to thank Michael Gillam, Craig Feied, Jonathan Handler and Hank Rappaport from the Microsoft Corporation for providing their expertise on electronic health systems, project coordination and technical guidance.

Lastly, we would like to thank Ben Shneiderman and Catherine Plaisant from the University of Maryland's Human-Computer Interaction Laboratory and Department of Computer Science for their continued support and guidance through the project, making this project possible.

References

1. Allen, J. F., Maintaining knowledge about temporal intervals, *Communications of the ACM*, 26, 1983, 832-843
2. Ankerst, M., D. H. Jones, A. Kao, and C. Wang, DataJewel: Tightly integrating visualization with temporal data mining, *ICDM Workshop on Visual Data Mining*, 2003.
3. Augusto, J. C. 2005. Temporal reasoning for decision support in medicine. *Artif. Intell. Med*. 33, 1, 2005, 1-24
4. Bade, R., Schlechtweg, S., and S. Miksch, Connecting time-oriented data and information to a coherent interactive visualization, *Proc. Conf. on Human Computer Interaction CHI 2004*, ACM, 2004, 105-112.
5. Bruce, B. C., A model for temporal references and its application in a question answer program, *Artificial Intelligence*, 3, 1-25, 1972.
6. Catarci, T., Costabile, M. F. Levialdi, S. and C. Batini, Visual query systems for databases: a survey, *Journal of Visual Languages and Computing*, 8, 215-260, 1997.
7. Cheng, C., Shahar, Y. , Puerta, A. and D. Stites, Navigation and visualization of abstractions of time-oriented clinical data *Stanford University Section on*

medical informatics technical report SMI-97-0688, 1997.
8. Cheng C. and Shahar Y., Intelligent visualization and exploration of time-oriented clinical data, *Topics in Health Information Management*, 20, 15-31, 1999.
9. Chittaro L. and Combi, C. Visualizing queries on databases of temporal histories: new metaphors and their evaluation, *Data and Knowledge Engineering*, 44, 239-264, 2003.
10. Dionisio J. D. N. and Cardenas, A. F. MQuery: A visual query language for multimedia timeline and simulation data, *Journal of Visual Languages and Computing*, 7, 377-401, 1996.
11. Fails, J., Karlson, A., Shahamat, L., Shneiderman, B. A Visual Interface for Multivariate Temporal Data: Finding Patterns of Events over Time in *Proc. of the IEEE Symposium on Visual Analytics Science and Technology*, IEEE, 2006, 167-174
12. Hochheiser, H. and Shneiderman, B., Dynamic query tools for time series data sets, Timebox widgets for interactive exploration, *Information Visualization* 3, 1, 2004, 1-18.
13. Hibino S. and Rundensteiner, E. A. User interface evaluation of a direct manipulation temporal visual query language, *ACM Multimedia*: ACM, 1997, 99-107.
14. Jensen C. S. and R. T. Snodgrass, Temporal data management, *IEEE Transactions on Knowledge and Data Engineering*, 11, 1999, 36-44,
15. Kahn K. M. and Gorry, A. G. Mechanizing temporal knowledge, *Artificial Intelligence*, 9, 1977, 87-108,
16. Kouramajian V. and Gertz, M. A graphical query language for temporal databases, in *Proc. of Object-Oriented and Entity Relationship Modeling*: Springer-Verlag, 1995, 388-399.
17. Plaisant, C., Mushlin, R., Snyder, A., Li, J., Heller, D., Shneiderman, B., LifeLines: Using visualization to enhance navigation and analysis of patient records, *American Medical Informatics Association 1998 Annual Fall Symposium.* AMIA, Bethesda MD, 1998, 76-80
18. Shahar, Y., Dynamic Temporal Interpretation Contexts for Temporal Abstraction, *Annals of Mathematics and Artificial Intelligence*, 22, 1998, 159-192.
19. Shahar, Y, Dimension of Time in Illness: An Objective View, *Annals of Internal Medicine*, 132 , 1, 2000, 45-53
20. Silva, S. F., Schiel, U. and T. Catarci, Visual query operators for temporal databases, in International *Workshop on Temporal Representation and Reasoning*, 1997, 46-53.
21. Snodgrass, R. T. *Developing time-oriented database applications in SQL*. San Francisco: Morgan Kauffman Publishers, Inc, 1999.
22. Stacey, M., McGregor, C., Temporal abstraction in intelligent clinical data analysis: A survey, *Artificial Intelligence in Medicine*, 2007, 39, 1-24
23. Wang, T., Plaisant, C., Quinn, A., Stanchak, R., Shneiderman, B., and Murphy, S., Aligning temporal data by sentinel events: Discovering patterns in electronic health records, *ACM CHI2008 Conference* (April 2008, to appear).
24. Weber, M. Alexa, M., and W. Muller, "Visualizing time series on spirals," *IEEE Symposium on Information Visualization:* IEEE Press, 2001, 7-14.

Appendix

Classes

Container Objects

- **Field** – DB field
- **Table** – DB table

- **Join** - connects two tables together for multi-table queries (not in use)

- **Filter** – contains an operator and the appropriate values
- **Operator** – a SQL operator

- **DBDescription** – holds database information

Utility Classes

- **SQLGen** – generates a SQL query based on filter information
- **Expander** – splits rows that contains multiple event data into multiple rows
- **Logger** – writes to a log file and to standard out
- **SaveLoad** – performs loading and saving functions

UI Objects

- **BallChainLifeLinesVisual** – interactive PiccoloX visualization component

- **DBIdentityListGridForm** – grid that displays a list of identities
- **DBSingleListGridForm** – grid that displays expanded result data for a single identity
- **DBResultGridForm** – grid that displays all expanded result data
- **JobBrowser** – tree that stores queued and finished queries

- **CommentsDialog** – dialog asking if the user wants to continue without writing a comment
- **DatesSelector** – dialog with up to two DateSelectors, used for the data scope
- **IdentitySelector** – dialog with fields, allows user to select an "identity" field
- **LogMonitor** – dialog with the current session log
- **DBInfo** - dialog prompt for db information when initial db connection attempt fails

- **Guided/FilterControl** – user control (available fields, scope, filters)
- **FilterPanel** (the user control for each "filter")
- **TemporalValueForm** – editing form for temporal Filter component
- **ValueForm** – editing form for Filter component

- **SnowForm** – main application window (menu bar, status bar, resize, etc)
- **snowman** – user control (contains query specification interface, results, and visualization)

- **SnowControl** – UserControl wrapper around SnowForm (launch inside an Amalga environment)
- **snowmanapp** - application wrapper around SnowForm (launch outside an Amalga environment)

Code Structure

PatternFinder-S

- **SnowForm**
  - **snowman** – user control (contains query specification interface, results, and visualization)
    - ***Query specification interface***
      - **Guided/FilterControl** – user control (available fields, scope, filters)
        - **FilterPanel** (the user control for each "filter")
          - **TemporalValueForm** – editing form for temporal Filter component
          - **ValueForm** – editing form for Filter component
    - ***Results interface***
      - **JobBrowser** – user control, tree (running jobs, finished jobs)
      - **DBResultGridForm** – processes SQL results and stores raw & processed result data. gridview shows expanded results with duplicates removed
      - **DBIdentityListGridForm** – gridview shows identities
      - **DBSingleListGridForm** – gridview shows data from a single identity
    - ***Visualization interface***
      - **BallChainLifeLinesVisual** – PiccoloX user component

PatternFinder-S User Manual

Last updated: 4/17/2008

Remote Desktop Connection

If *PatternFinder-S* is not available as a standalone application or a live Amalga component, *PatternFinder-S* for Amalga is available for demonstration and testing purposes on a test Washington Hospital Center server under the **AZ-Hosp – Emergency** app, and is currently connected to a replicated database of live Amalga data. To establish a remote desktop connection, start the **Remote Desktop Connection** application under:

Start Menu > All Programs > Accessories > (Communications >) Remote Desktop Connections

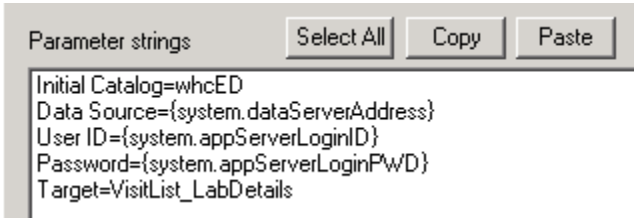
and connect to

172.26.190.39

Launching from Amalga

PatternFinder-S has been added under the Dev tab and should launch when you click the Dev button.

If the data source for *PatternFinder-S* needs to be adjusted, it can be configured through the **Amalga Manager** app by editing the component method information. The default is configured to use view **VisitList_LabDetails** in **whcED** database on the local server. (A stand-alone version is also available.)



Launching Standalone

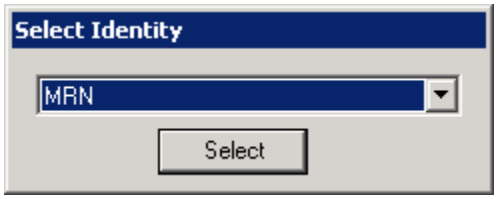
To launch *PatternFinder-S* in standalone mode, navigate to the folder that contains the PatternFinderS executable file and double-click it. If the data source for PatternFinder-S needs to be adjusted, it can be configured by editing the settings.cfg file in the same folder. Here is a sample settings string:

```
Initial Catalog=master; Data Source=localhost; User ID=12user34;
Password=abpasscd; Target=contrast; Identity=[Account][MRN]
```

Initial Catalog:	database name
Data Source:	address of server
User ID:	db login name
Password:	db password
Target:	table to use
Identity:	possible fields to use for the unique identity. Each identity is enclosed in brackets

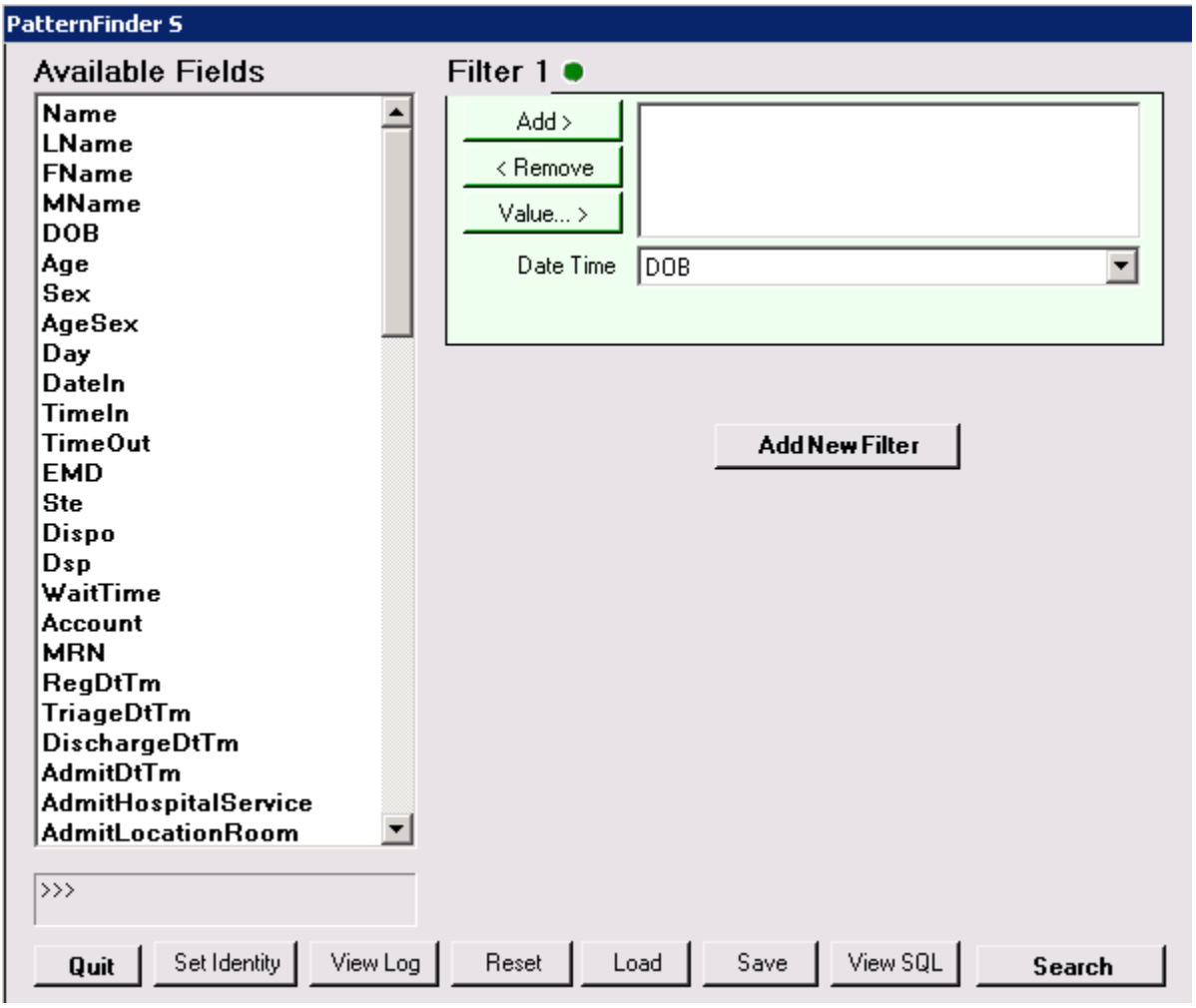
Selecting an Identity

Once *PatternFinder-S* is launched and connected, a prompt will pop up asking you to select the field from the database you wish to use as the unique identifier if more than one identity is available. For the **VisitLab_LabDetails** target, the **MRN** field is appropriate for identifying per-patient patterns and the **Account** field is appropriate for identifying per-visit patterns. Clicking **Select** will launch the *PatternFinder-S* application.

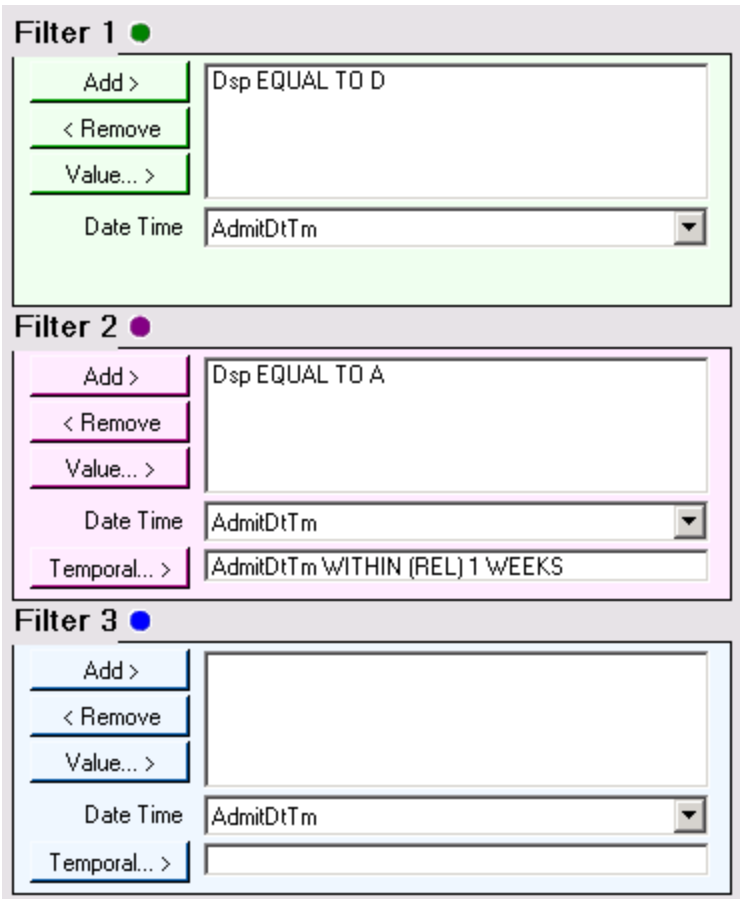


Specifying Your Query

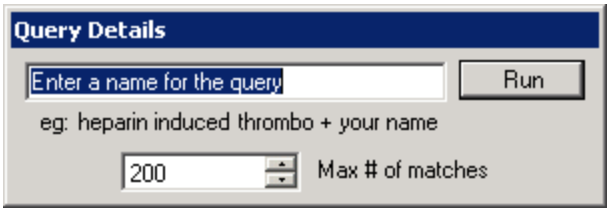
PatternFinder S will launch with multiple parts in a single window. The query specification portion will have the fields from the database target listed, as well as the option to use up to three filters.



Each filter has a Date Time field that allows users to specify which field from the database should be used to chronologically compare each event with the last one. The second and third filters have a temporal field that lets users define how one filter is related to the previous one. Shown below is an example of a query that searches for patients who have been discharged by the ER only to be re-admitted within one week.

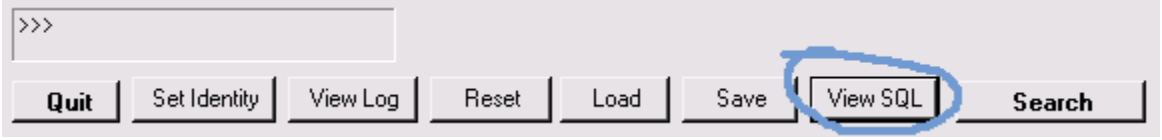


Once the query has been specified, clicking the Search button will bring a prompt up asking for additional query details. Enter a descriptive name for the query (for user slam, eg: heparin induced thrombo + slam) and the desired maximum number of matches and click **Run**.

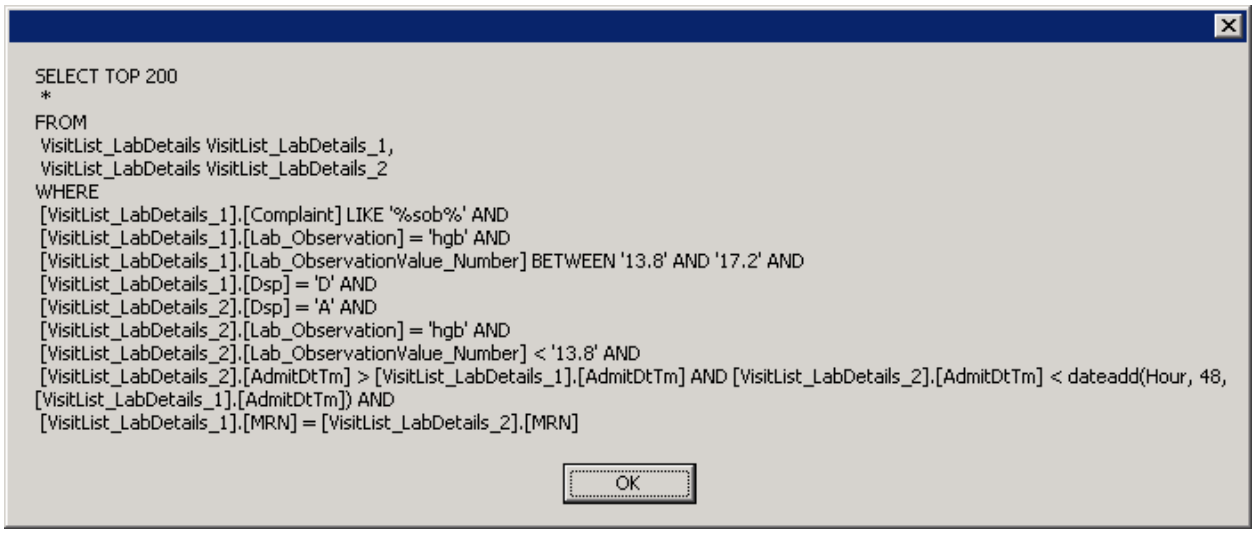


The query will be added to the job browser in the query results area.

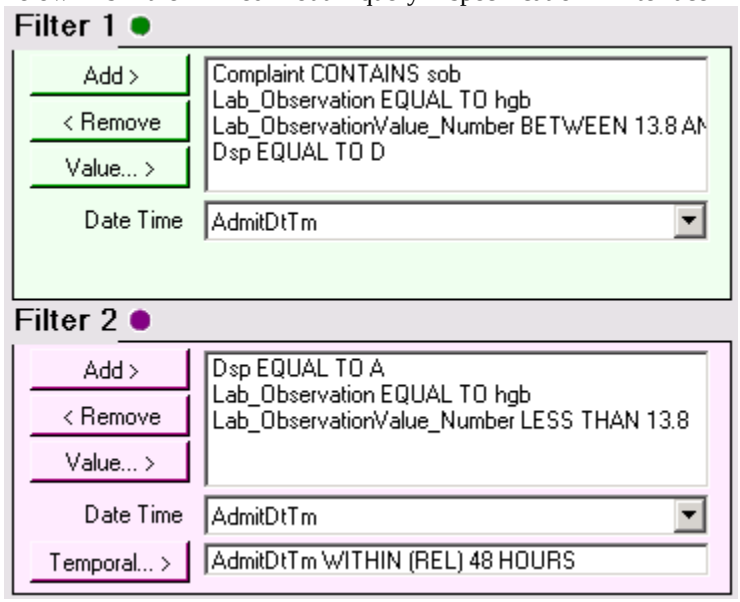
Clicking the **View SQL** button for any query will show a dialogue with its SQL statement equivalent.



In the screenshot shown below, the SQL statement describes a search for patients who come in and report shortness of breath ("sob" in the Complaint field) and are discharged with normal hemoglobin levels (between 13.8 and 17.2), but come back within 48 hours and are admitted to the ER with less than normal hemoglobin levels.

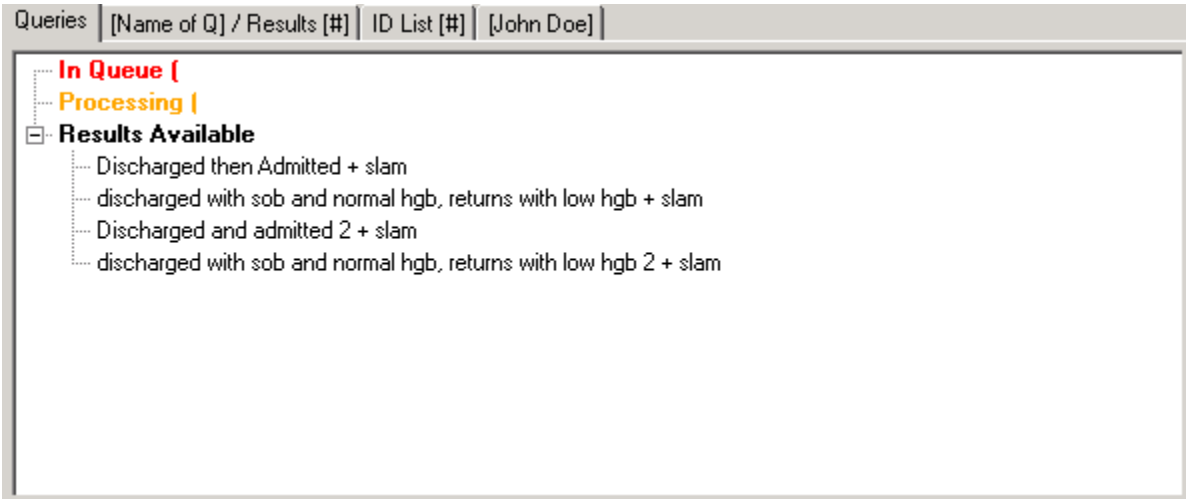


Below is the filled out query specification interface that produces the SQL shown above.

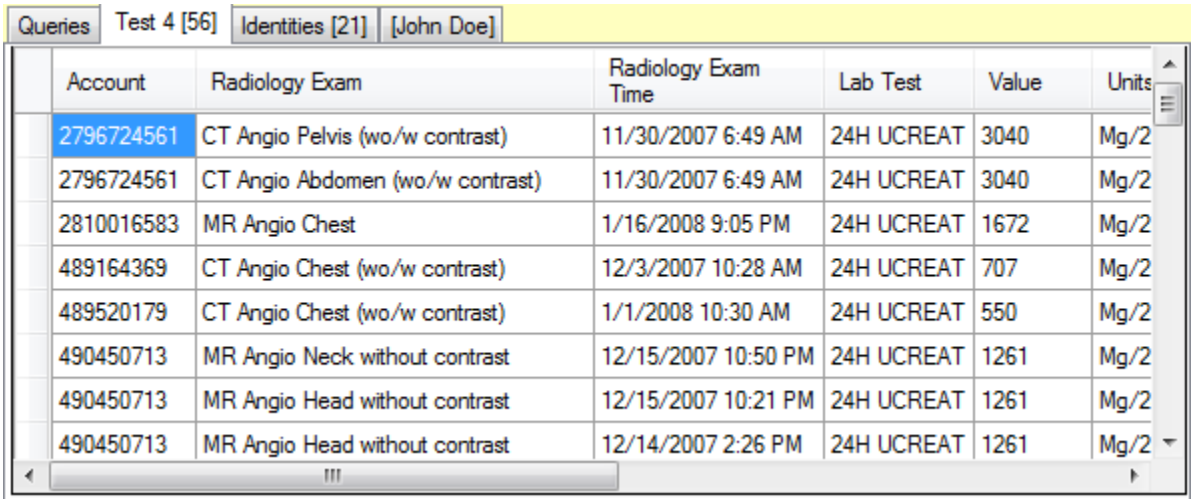


Accessing Query Results

Below the query specification interface is the query results area. The query will initially show up under the **In Queue** area, be moved down into **Processing** when the database is processing the query and be moved to **Results Available** when finished.



Double clicking the query will populate the graphical visualization as well as the 2nd and 3rd tabs in the results area with the matches returned from the database. Double clicking on any of the data rows in the 2nd and 3rd tabs will populate the 4th tab with results for the same identity. The visualization will also scroll to the appropriate location.



Account	Radiology Exam	Radiology Exam Time	Lab Test	Value	Units
2796724561	CT Angio Pelvis (wo/w contrast)	11/30/2007 6:49 AM	24H UCREAT	3040	Mg/2
2796724561	CT Angio Abdomen (wo/w contrast)	11/30/2007 6:49 AM	24H UCREAT	3040	Mg/2
2810016583	MR Angio Chest	1/16/2008 9:05 PM	24H UCREAT	1672	Mg/2
489164369	CT Angio Chest (wo/w contrast)	12/3/2007 10:28 AM	24H UCREAT	707	Mg/2
489520179	CT Angio Chest (wo/w contrast)	1/1/2008 10:30 AM	24H UCREAT	550	Mg/2
490450713	MR Angio Neck without contrast	12/15/2007 10:50 PM	24H UCREAT	1261	Mg/2
490450713	MR Angio Head without contrast	12/15/2007 10:21 PM	24H UCREAT	1261	Mg/2
490450713	MR Angio Head without contrast	12/14/2007 2:26 PM	24H UCREAT	1261	Mg/2

Controlling the Graphical Visualization

The visualization model has several options that can be toggled on and off.

Show Identifiers – causes labels to be drawn.

Show LifeLines view – causes a summary-view to be drawn for each identity.

Align By First – aligns results for each identity by the first event

Expand Ball-and-Chain View – causes each set of matches to be displayed individually.