ConQueSt User Manual

Francesco Bonchi, Fosca Giannotti, Claudio Lucchese,
Salvatore Orlando, Raffaele Perego, Roberto Trasarti

December 2007

Contents

1 Introduction

1.1 What is ConQueSt ?

This is the user manual of ConQueSt a constraint based querying system devised with the aim of supporting the intrinsically exploratory (i.e., human-guided, interactive, iterative) nature of pattern discovery. Following the Inductive Database vision, this framework provides users with an expressive constraint based query language which allows the discovery process to be effectively driven toward potentially interesting patterns. This comprehensive mining system can access real world relational databases from which extract data. After a preprocessing step, users mining queries are answered by an efficient pattern mining engine which entails several data and search space reduction techniques. Finally, results are presented to the user, and then stored in the database.

1.2 Who should read this document?

This document is for the user that approach ConQueSt having a basic knowledge in database and want to improve the analysis using constraint pattern discovery to find regularities in the data. Next we describe some theoretical requirement useful for understand how use all the potentially of ConQueSt .

1.3 Acknowledgements

ConQueSt is the result of a joint work of the Pisa KDD (*Knowledge Discovery and Delivery*) and HPC (*High Performance Computing*) Laboratories: it is built around a scalable and high-performance constraint-based mining engine exploiting state-of-the-art constraint pushing techniques, as those ones developed in the last three years by our two labs.

1.4 Theoretical requirement

The constraint-based pattern mining paradigm has been recognized as one of the fundamental techniques for Inductive Databases: user-defined constraints drive the mining process towards potentially interesting patterns only. Moreover, they can be pushed deep inside the mining algorithm in order to deal

with the exponential search space curse, achieving better performance. Constraint based frequent pattern mining has been studied a lot as a query optimization problem, i.e., developing efficient, sound and complete evaluation strategies for constraint-based mining queries. To this aim, properties of constraints have been studied comprehensively, e.g., anti-monotonicity, succinctness, monotonicity , convertibility, loose anti-monotonicity, and on the basis of such properties efficient computational strategies have been defined.

Definition 1 (Constrained Frequent Itemset Mining) *Let $\mathcal{I} = \{x_1, ..., x_n\}$ be a set of distinct items, where an item is an object with some predefined attributes (e.g., price, type, etc.). An itemset X is a non-empty subset of $\mathcal{I}$. A transaction database $\mathcal{D}$ is a bag of itemsets $t \in 2^{\mathcal{I}}$, usually called transactions. A constraint on itemsets is a function $\mathcal{C} : 2^{\mathcal{I}} \rightarrow \{true, false\}$. We say that an itemset I satisfies a constraint if and only if $\mathcal{C}(I) = true$. We define the theory of a constraint as the set of itemsets which satisfy the constraint: $Th(\mathcal{C}) = \{X \in 2^{\mathcal{I}} \mid \mathcal{C}(X)\}$. The support of an itemset X in database $\mathcal{D}$, denoted $supp_{\mathcal{D}}(X)$, is the number of transactions which are superset of X. Given a user-defined minimum support, denoted σ, an itemset X is called frequent in $\mathcal{D}$ if $supp_{\mathcal{D}}(X) \geq \sigma$. This defines the minimum frequency constraint: $\mathcal{C}_{freq[\mathcal{D},\sigma]}(X) \Leftrightarrow supp_{\mathcal{D}}(X) \geq \sigma$.*

In general, given a conjunction of constraints $\mathcal{C}$ the constrained frequent itemsets mining problem requires to compute $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$.

1.5 Some intended purposes

Developing ConQueSt we have tried to reduce as much as possible the gap existing between real-world data stored in relational DBMS, and the constraint-based pattern discovery paradigm, as defined above. In fact, the data is usually stored in relational databases, and thus in the form of relations and not of transactions. In Section 4 we explain how transactions are defined and constructed both at the query language and at the system level. Moreover, the constraint-based mining paradigm assumes that the constraints are defined on attributes of the items that are in functional dependency with the items. This rarely the case in real-world data: just think about the price of one item in a market over a period of one month. ConQueSt provides support to solve this cases, allowing the user to reconstruct the ideal situation of functional dependency, for instance, by choosing to take the average of

prices of the item in the period as price attribute. ConQueSt as Inductive Database system provide the following features:

Compatible with commercial DBMS. Easy connect to commercial DBMS to access to the data storage.

Coupling with a DBMS. The analyst can retrieve the portion of interesting data (for instance, by means of SQL queries). Moreover, extracted patterns should also be stored in the DBMS in order to be further queried or mined (closure principle).

Expressiveness of the query language. The analyst interact with the pattern discovery system by specify declaratively how the desired patterns should look like and which conditions they should satisfy. The analyst is supposed to have a high-level vision of the pattern discovery system, without worrying about the details of the computational engine, in the same way as a database user has not to worry about query optimization. The task of composing all constraints and producing the most efficient mining strategy (execution plan) for a given query should be thus completely demanded to the underlying system.

Efficiency of the mining engine. Keeping query response time as small as possible is, on the one hand necessary, since our goal is to give frequent feedbacks to the user allowing realistic human-guided exploration. On the other hand, it is a very challenging task, due to the exponential complexity of pattern discovery computations. To this end, data and search space reduction properties of constraints should be effectively exploited by pushing them within the mining algorithms. Pattern discovery is usually a highly iterative task: a mining session is usually made up of a series of queries (exploration), where each new query adjusts, refines or combines the results of some previous queries. It is important that the mining engine adopts techniques for incremental mining; i.e. reusing results of previous queries, in order to give a faster response to the last query presented to the system, instead of performing again the mining from scratch.

Graphical user interface. The exploratory nature of pattern discovery imposes to the system not only to return frequent feedbacks to the user, but also to provide pattern visualization and navigation tools.

These tools should help the user in visualizing the continuous feedbacks form the systems, allowing an easier and human-based identification of the fragments of interesting knowledge. Such tools should also play the role of graphical querying interface: the action of browsing pattern visualization should be tightly integrated (both by a conceptual and engineering point of view) with the action of iteratively querying.

To do this ConQueSt implements tools which help the user from the beginning to the end of his analysis giving the possibilities to guide the process and obtain the knowledge he search for.

1.6 Features

ConQueSt give a set of tools that can be divided in 4 big familes. Now we give only a short description of this family, which will be deepened in the next sections.

Visualization Tools Help user to organize the tables in the workbench and to analyze their dependencies.

Analysis Tools Those tools give information about tables or field and permit to manipulate the data to prepare at the mining process.

Mining tools Give the possibilities to define the mining task and view the transformation from relational to transactional view of the data and then navigate the result of query in different way.

Post-processing tools Reached the result of a query, there are tools that use this knowledge to extract more refined information or put it into the database in relational form that can be used as source of further analysis or mining process.

1.6.1 System requirement

ConQueSt need at least this requirement:

- Processor 800Mhz
- Ram 256 MB
- Windows 9*/XP

- Java Runtime Environment Version 1.5.0 Update 5
- 20 MB free hard disk space

1.6.2 Compatibility with DBMS

ConQueSt can be connected to all the DBMS that use the interface JDBC/ODBC. The basic set of driver included is:

- Microsoft Access
- PostgreSQL
- Oracle
- MySQL
- SQL Server
- ODBC Sources

Is easy to add a new database having his driver, simply add in the "DriverDB" directory the driver and then add in "DBlist.ini" an entry in this format: *[DBName] [DBDriverName] [FormatOfConnection].*

1.6.3 Obtaining the source, binary, and document distributions

The source code, the binary and this document is published on the site of ConQueSt : [www.conquest ???] where you can find other information about this system and you can download a video tutorial of the system. [subscribe ???]

2 User Interface

2.1 Introduction

ConQueSt is equipped with a simple but powerful interface, that permit the user to explore the data, guide the mining process and navigate the result. When it start the main window appear as show in fig. 1. Next it's described how to connect to a database use all the possibilities that ConQueSt give to users.

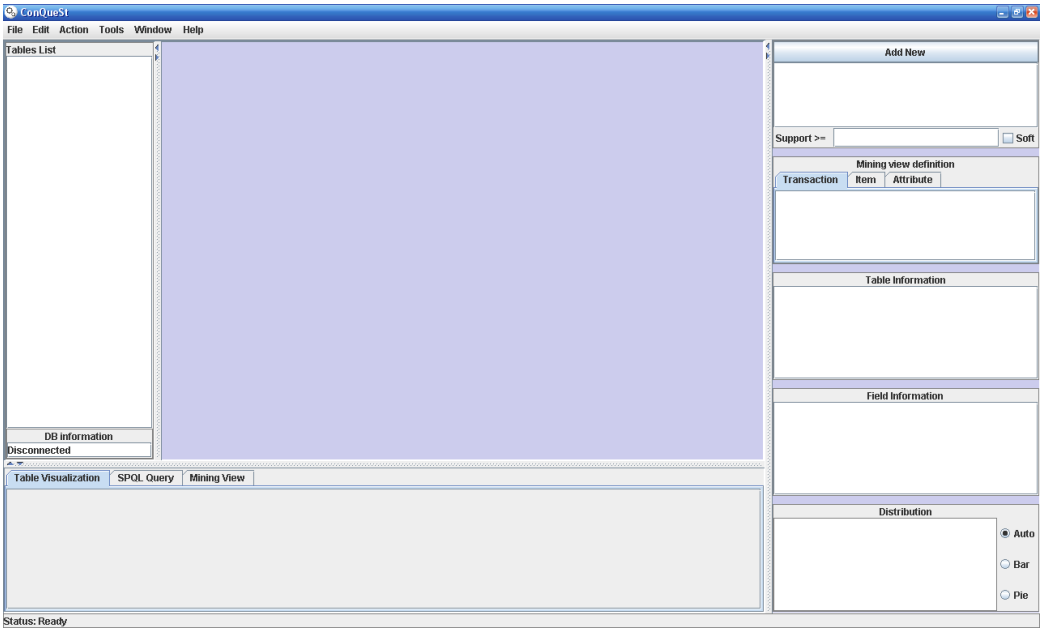


Figure 1: ConQueSt main window

2.2 Connect to a database

To connect ConQueSt to a database select from the toolbar *File* and then *Connect*, the system show the window showed in fig.2 where the user can insert all the information needed to establish a connection with the dbms:

Database Type: the type of database which ConQueSt must be connected

Database Address: is the string that identify the location of database with format: *[Url]:[Port]:[DbName]*

Username: the username of a user in the dbms

Password: the password of the user in the dbms

Schema Name: the name of the schema of the dbms which contains the data

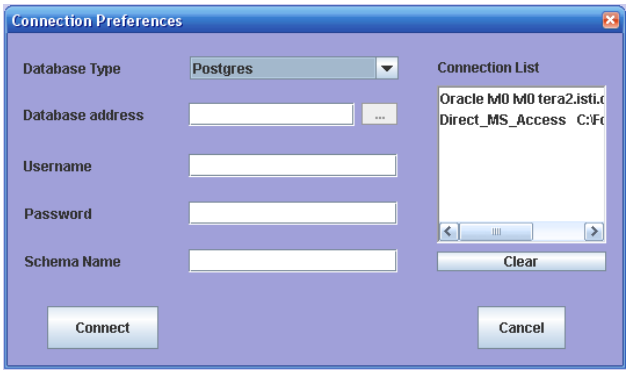


Figure 2: ConQueSt main window

On the right side of the window there is a list of connection used by the user for a rapid reconnection. The *Clear* button delete the history of the connection. When all the information is ready simply click on the *Connect* button. In the first phase of connection ConQueSt read the list of the tables in the schema of the dbms specified and the start to gather information of it reading them. This process is showed on the left side of the main window on the *table list* pane.

2.3 The Table List pane

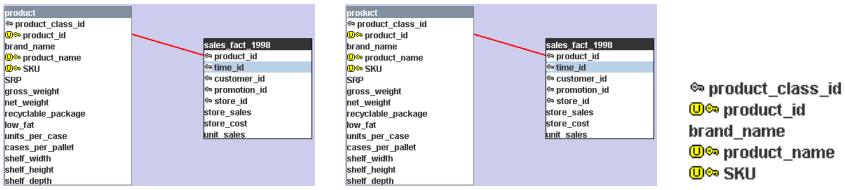
The table list, after connecting to a dbms, contains the list of the tables of the schema which the system is connected. After a first phase of reading all the table is scanned to calculate statistics on their fields. At the end, near all tables names, there is an icon that show the status of scanning and if ConQueSt have metainformation about the creation of the table as show in fig.3

If some table in the list is marked as *not scanned* after the scanning phase it means that their size is grater then a threshold specified in the option to denied a too long computation of statistics. In the section 2.5 is described how to change this parameter. Under the table list there are some information about the dbms which ConQueSt is connected that show:

- Number of tables
- Product name and version

Icon	Description
	not scanned
	there are error in the table
	informations avaibles for this table
	table materialized from a patterns discovery process
	table materialized from a rules extraction process

Figure 3: Icon of table list



(a) (b) (c)

Figure 4: link and icon in workbench

- The Url of the dbms

A table can be show in workbench with a double click on his name in the table list. In the table list is always showed the table currently selected.

2.4 The workbench

The workbench is the place where the tables are visualized and can be organized by the user. Not only the tables are showed into the workbench but the relation between they and the typology of fields are represented by lines and icon: In fig.4(a) and fig.4(b) is showed the relation between the two table and the different color of the line in the case the selected table is the one that have the primary key or the foreign key for the link. In fig.4(c) is the detailed view ot a part of the table where there are information about some field: intuitively the yellow key represent the primary key and the grey one represent the foreign key, in addition the yellow box with U is for the unique property that the field have (it is call property because isn't necessary the unique constraint of database). All the tables can be moved with drag and drop or can be minimized by double clicking on their header (or restored in

the same way). In case of big schema it's necessary have a lot of space, so ConQueSt give to users a tool to organize and navigate it: clicking on free space of workbench with right mouse button a navigation control window appear, with this the user can perform zoom and can move all the tables in workbench to create big diagram. With the *Global View* button see all in one window.

2.5 The toolbar menu

At the top of main window there is the toolbar divided into six different menu, in the next sections we describe them.

- **File menu:**

  Save diagram. Save the diagram created into into the workbench.

  Exit. Exit from ConQueSt .

- **Edit menu:**

  Copy text. Copy the text from SPQL query pane into clipboard.

  Paste text. Paste the text from the clipboard into the SPQL query pane.

  Clear text. Clear the text into the SPQL query pane.

- **Action menu:**

  Connect. Open the window for connection preferences.

  Reset all id. Clear the definition of mining view.

  Option. Open the window for ConQueSt preferences (e.g. maximum number of rows displayed in Table view pane)

  Reload database. Establish a new connection to a database and reload the data.

  Exec SPQL query. Exec the SQPL query written into the SPQL query pane.

- **Tools menu:**

Create table link. Create links between tables based on the name and existing primary key (in some type of database this information isn't stored so this feature is applied during connection)

Maximize all. All the tables are maximized (show the name and the field of the tables).

Minimize all. All the tables are minimized (show only the name of the tables).

Show all. All the table are showed into the workbench.

Hide all. All the table are hidden.

Clear dataset cache. Clear the cache of ConQueSt .

- **Window menu:**

  Add constraint. Open the window to add a constraint to the SPQL query.

  Resolve attribute conflict. Open the window to resolve an attribute conflict.

- **Help menu:**

  Help. Open the online user manual of ConQueSt .

  About. Show some information about ConQueSt .

2.6 The Statistics pane

In the right side of ConQueSt there are three pane that give to user some statistical information about the selected table as show in fig.5. The top one is the Table information pane that show information about the number of rows and the type of all the fields contained in the table. If a field of a the table is selected, the Field information pane show the name of this one and some aggregate information as maximum, minimum, average and number of distinct value. Under these two panel there is a graph that show the distribution of the values of the field selected. The three ratio button on the right of the graph permits to select the type of graph or if the system selects automatically the better one. Clicking on it with left mouse button a window with the graph appear. In this new window the user can navigate the graph as show in fig.6 or trough Action menu discretize the values and

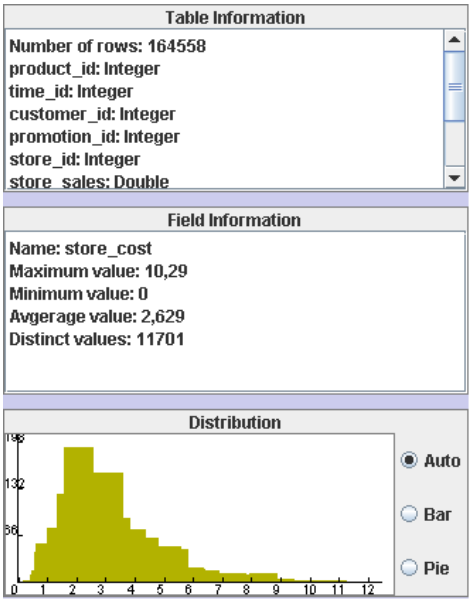


Figure 5: ConQueSt statistic pane

create a new field in the table, this feature is specified in the section 3. If a table is not scanned, during the connection phase, when is selected in these panel can't be showed any information, clicking on it the scanning process will be initialized and executed, and then the results is showed.

2.7 The Constraints pane

In the top right side of main window there is the Constraints pane, that show the list of constraints of the current SPQL query. Clicking on Add constraint button the user can add a new constraint, more information about it is in section 4.3.

2.8 The Table Visualization pane

This pane is in the group of three tab under the workbench as show in fig.7. In this pane is showed a preview of the selected table (the number of the rows showed is a system parameter, specified in Option menu) and the user can perform an ordination clicking on the header of the columns.

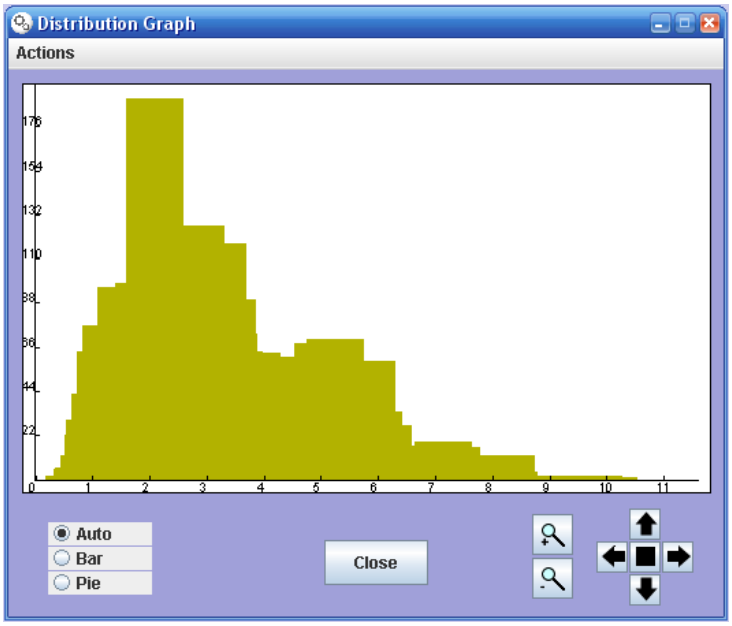
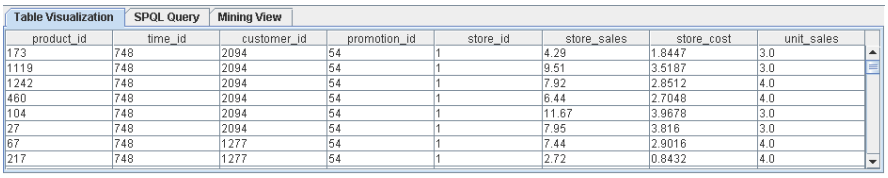


Figure 6: Distribution graph window



product_id	time_id	customer_id	promotion_id	store_id	store_sales	store_cost	unit_sales
173	748	2094	54	1	4.29	1.8447	3.0
1119	748	2094	54	1	9.51	3.5187	3.0
1242	748	2094	54	1	7.92	2.8512	4.0
460	748	2094	54	1	6.44	2.7048	4.0
104	748	2094	54	1	11.67	3.9678	3.0
27	748	2094	54	1	7.95	3.816	3.0
67	748	1277	54	1	7.44	2.9016	4.0
217	748	1277	54	1	2.72	0.8432	4.0

Figure 7: Table visualization pane

2.9 The SPQL query pane

This is the space where the SPQL (or SQL) query is written, as showed in fig.8 there is a button for execute the query. The user can write the query directly in this pane or see what the system create meanwhile the user use the interface to specified the parameters of the query.

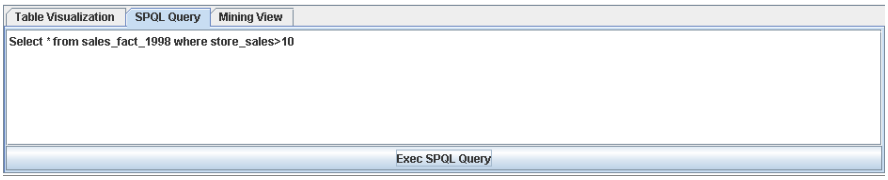


Figure 8: SPQL query pane

2.10 The Mining View pane

This pane permits to user to create a preview of the dataset generated by the mining view definition (section **??**) given in the SPQL query. To start the process click on the button on the right as show in fig.9 The pane is divided

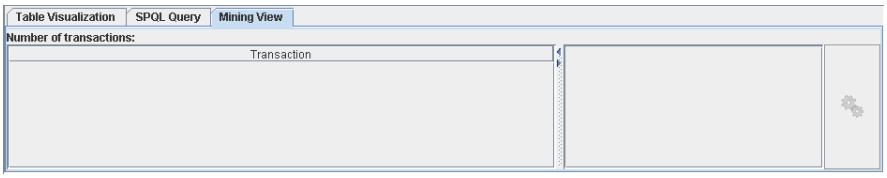


Figure 9: Mining view pane

in two, the left side show the transactions of the dataset and the right side the attribute of the items. On the top there are some statistic of the dataset as the number of transactions.

2.11 The Status and progress bar

The status bar is the bottom part of the main window, and visualize the status of ConQueSt or warning and error message of the system. When ConQueSt perform some process a progress bar and a little window on the right side of workbench appear displaying the name of the work in progress, with the abort button the work is terminated.

3 Discretization

As showed in previous section, ConQueSt give to user a powerful tool which can discretize a field of a table. To use this tool click on the distribution

graph pane and then select Discretize from the Action menu. A new window appear as in fig.10, where the user can select the discretization method the smoothing type, the name of the new field creataed in the table by the process and the number of partition in which the value space must be divided. The

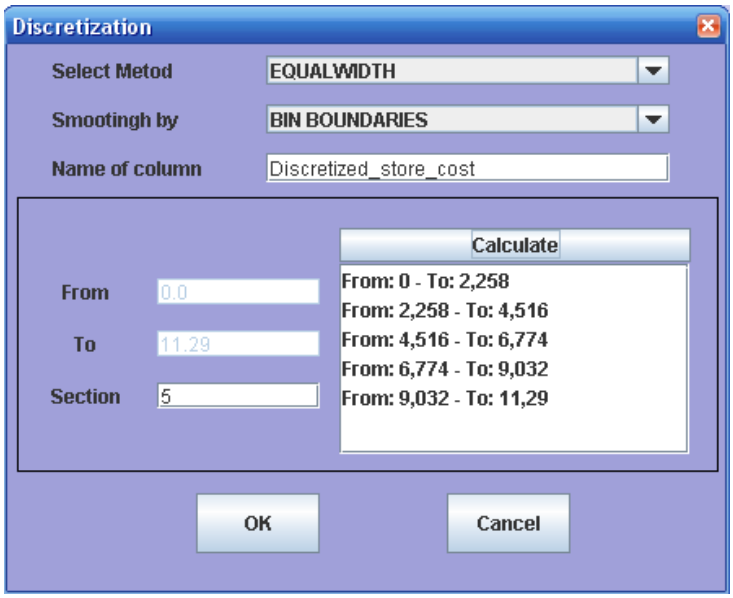


Figure 10: Discretize window

discretization method can be:

Equal width. The space are divided into equal part with the same length (fig.11(a)).

Equal depth. The space are divided into part that have the same numbero of objects (fig.11(b)).

Free Partitioning. The partitions is defined by the user.

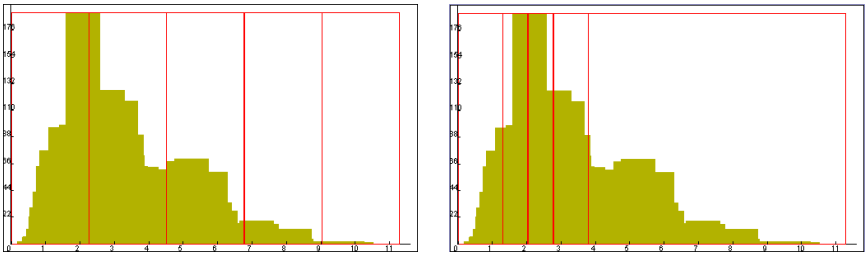
In the first two cases after specified the number of partitions the user must click on calculate to apply the parameter, in the third the user specified the boundaries of each partition and then click on add for each one. The smoothing is the type of information that will be stored into the new field:

Bin boundaries. The boundaries of the partition where the field is in.

Average. The average of the boundaries of the partition where the field is in.

Count. The count of the items that there are in the partition where the field is in.

When all are set the user can click on OK button and the discretization will be applied on the table. Discretize a field can be performed not only using



(a) (b)

Figure 11: sample of discretion method

the interface but also executing a SQPL query, the syntax is:
DISCRETIZE *old field* AS *new field*
FROM *table*
IN *number of partition disctretization method* BINS
SMOOTINGH BY *smoothing type*

In case of free partition the syntax change:

DISCRETIZE *old field* AS *new field*
FROM *table*
IN $(i_1^\alpha, i_1^\beta), \ldots, (i_\kappa^\alpha, i_\kappa^\beta)$ BINS
SMOOTINGH BY *smoothing type*

where $(i_\kappa^\alpha, i_\kappa^\beta)$ are the boundaries of the interval κ.

4 SPQL language

In this section is presented the SPQL language, this include three type of query: discretization query, standard SQL query and Mine query, but in the

next paragraph we use the term SPQL query to indicate only Mine query. The formal grammar of the language is presented in appendix 8.1.

To understand the process of a SPQL query and then the language an introduction to the theory is needed. According to the constraint-based mining paradigm, the data analyst must have a high-level vision of the pattern discovery system, without worrying about the details of the computational engine, in the very same way a database designer has not to worry about query optimizations. The user must be provided with a set of primitives to be used to communicate with the pattern discovery system, using a query language. The user just needs to declaratively specify in the pattern discovery query how the desired patterns should look like and which conditions they should obey (a set of constraints). Such rigorous interaction between the user and the pattern discovery system, can be implemented following, where Mannila introduces an elegant formalization for the notion of interactive mining process: the term *inductive database* refers to a relational database plus the set of all sentences from a specified class of sentences that are true w.r.t. the data. In other words, the inductive database is a database framework which integrates the raw data with the knowledge extracted from the data and materialized in the form of patterns. In this way, the knowledge discovery process consists essentially in an iterative querying process, enabled by a query language that can deal either with raw data or patterns.

Definition 2 *Given an instance* $\mathbf{r}$ *of a relation* $\mathbf{R}$*, a class* $\mathcal{L}$ *of sentences (patterns), and a selection predicate* q*, a pattern discovery task is to find a theory*

$$\mathcal{T}h(\mathcal{L}, \mathbf{r}, q) = \{s \in \mathcal{L} | q(\mathbf{r}, s) \textit{ is true}\}$$

The selection predicate q indicates whether a pattern s is considered interesting. In the constraint-based paradigm, such selection predicate q is defined by a conjunction of constraints. In this Section, going through a rigorous identification of all its basic components, we provide a definition of constraint-based frequent pattern mining query over a relational database DB.

The first needed component is the data source: which table must be mined for frequent patterns, and which attributed do identify transactions and items.

Definition 3 (Data Source) *Given a database* DB *any relational expression* $\mathcal{V}$ *on preds*(DB) *can be selected as* data source*.*

Definition 4 (Identifier) *Given a data source* V *and let* $sch(\mathcal{V})$ *his fields. Any subset of fields* $\mathcal{H} \subset sch(\mathcal{V})$ *can be selected as identifier, and named* id.

Definition 5 (Transaction id) *A* Transaction id *is an identifier on* $\mathcal{V}$ *that uniquely identify a transaction.*

Definition 6 (Item id) *An* Item id *is an identifier* $\mathcal{I}$ *on* $\mathcal{V}$ *that define the values of items in the transactions and satisfy* $\mathcal{I} \cap \mathcal{T} = \varnothing$ *where* $\mathcal{T}$ *is the transaction id associated.*

Definition 7 (Transactional definition) *A transactional definition is a triple* $\xi = \langle \mathcal{V}, \mathcal{T}, \mathcal{I} \rangle$ *where* $\mathcal{V}$ *is the data source,* $\mathcal{T}$ *is the transaction id and* $\mathcal{I}$ *is the item id.*

Definition 8 (Attribute id) *Given a transactional definition* $\xi = \langle \mathcal{V}, \mathcal{T}, \mathcal{I} \rangle$*, an* Attribute id *is an identifier* $\mathcal{A}$ *on* $\mathcal{V}$ *that define the values of item attributes and satisfy* $\mathcal{A} \cap \mathcal{T} = \varnothing$*,* $\mathcal{A} \cap \mathcal{I} = \varnothing$ *and have a functional dependency* $\mathcal{I} \rightarrow \mathcal{A}$ *holds in* V*.*

Definition 9 (Mining View definition) *A Mining view definition is a quadruple* $\xi' = \langle \mathcal{V}, \mathcal{T}, \mathcal{I}, \mathcal{A} \rangle$ *where* $\mathcal{V}$ *is a data source,* $\mathcal{T}$ *is a transaction id,* $\mathcal{I}$ *is a item id and* $\mathcal{A}$ *is an attribute id. We can define a mining view definition as* $\xi' = \langle \xi, \mathcal{A} \rangle$ *too.*

Definition 10 (Derived Attributes list) *Given a mining view definition* $\xi' = \langle \mathcal{V}, \mathcal{T}, \mathcal{I}, \mathcal{A} \rangle$*, a derived attribute list is*

$$\mathcal{L}_{\xi'} = \{\langle i_k, a_k \rangle \in V | i_k \in \delta_V(\mathcal{I}), a_k \in \delta_V(\mathcal{A})\}$$

Definition 11 (Mining View) *Given a mining view definition* $\xi' = \langle \xi, \mathcal{A} \rangle$ *the mining view is* $\langle \mathcal{D}_\xi, \mathcal{L}_{\xi'} \rangle$

Since the data source is in relational form, a pre-processing step is needed to create a set of transactions, which are the input of any frequent pattern mining system. Transaction are created by grouping `ITEM` by the attributes specified in the `TRANSACTION` clause and associating to each item the value of the `ATTRIBUTE` selected.

4.1 Mining view definition

Definition 12 (Derived Transactional dataset) *Give a transactional definition $\xi = \langle \mathcal{V}, \mathcal{T}, \mathcal{I} \rangle$, a derived transactional dataset is*

$$\mathcal{D}_\xi = \{(t_{tid} = \{i_1 \dots i_n\}, i_k \in \delta_V(\mathcal{I}), i_a \neq i_b, a \neq b) \Leftrightarrow \exists \langle tid, i_k \rangle \in \mathcal{V}\}$$

Definition 13 (Mining View definition) *A Mining view definition is a quadruple $\xi' = \langle \mathcal{V}, \mathcal{T}, \mathcal{I}, \mathcal{A} \rangle$ where $\mathcal{V}$ is a data source, $\mathcal{T}$ is a transaction id, $\mathcal{I}$ is a item id and $\mathcal{A}$ is an attribute id. We can define a mining view definition as $\xi' = \langle \xi, \mathcal{A} \rangle$ too.*

Definition 14 (Derived Attributes list) *Given a mining view definition $\xi' = \langle \mathcal{V}, \mathcal{T}, \mathcal{I}, \mathcal{A} \rangle$, a derived attribute list is*

$$\mathcal{L}_{\xi'} = \{\langle i_k, a_k \rangle \in V | i_k \in \delta_V(\mathcal{I}), a_k \in \delta_V(\mathcal{A})\}$$

Definition 15 (Mining View) *Given a mining view definition $\xi' = \langle \xi, \mathcal{A} \rangle$ the mining view is $\langle \mathcal{D}_\xi, \mathcal{L}_{\xi'} \rangle$*

In ConQueSt the definition of mining view can be performed in two way: by writing a query in the SPQL query pane, or by clicking and select the field we want to define as transaction id, item id and attribute id; to do this open the all the table you need and maximize them on the workbench, then click with right mouse button on each field and select the type of definition as show in fig.12 The fields change their color when a definition is assigned:

Color	Definition associated
Blue	Transaction id
Green	Item id
Red	Attribute id (descriptive)

In the example showed in the figure the fields aren't all in the same table, so ConQueSt joins the source tables using the links. In the same time a user define the Mining view, an inverse parser create the SPQL query associated showed in the bottom of the figure, there is a Mining view definition pane too on the right of the main window with three tab one for each definition: fig.13(a) transaction view, fig.13(b) item view, fig.13(c) attribute view

The definition of the mining view is the first step to generate the SPQL query, but, before continue the process, the user can have a preview of the

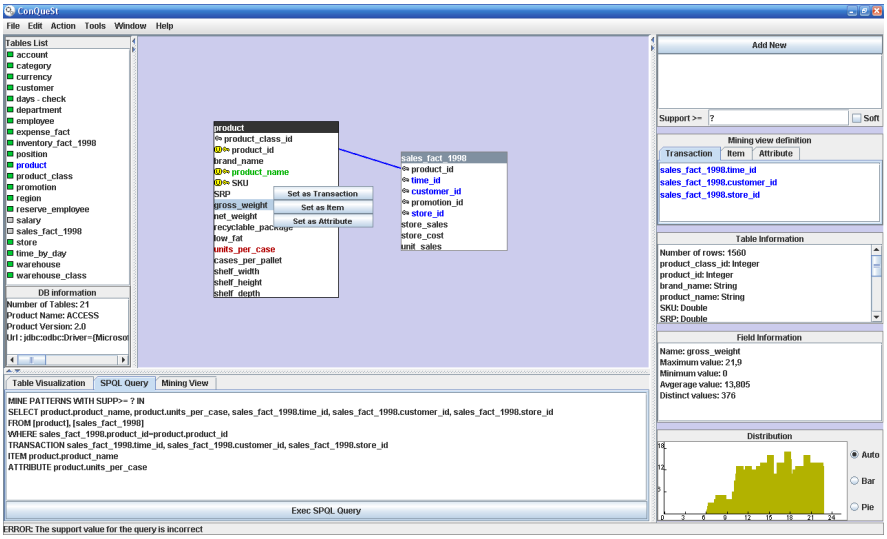
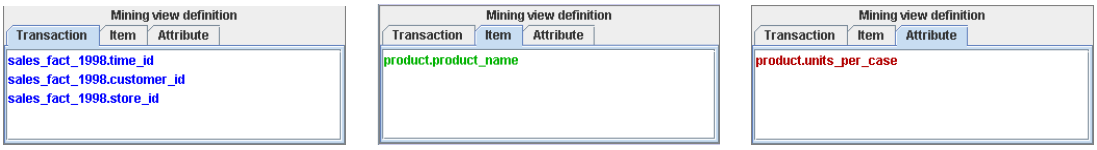


Figure 12: Mining view definition process



(a) (b) (c)

Figure 13: Mining View definition pane

dataset that the definition generate using the Mining view pane. Clicking on the button on the right side of this panel the pre-processing described above start (an example of result is in fig.14). In this way the user can see how the data is transformed in and if there are error in his definition before executing the SPQL query. Adding a support value in the space under the constraints pane (fig.15)the user can execute the SPQL query. Without constraints the query task will be find frequent pattern w.r.t. the support value in the transaction model of the data generated (the attribute definition in this case is useless). The second way to define a mining view is to write the query in the SPQL query pane. For simplicity consider the query:

```
1. MINE PATTERNS WITH SUPP>= 5 IN
```

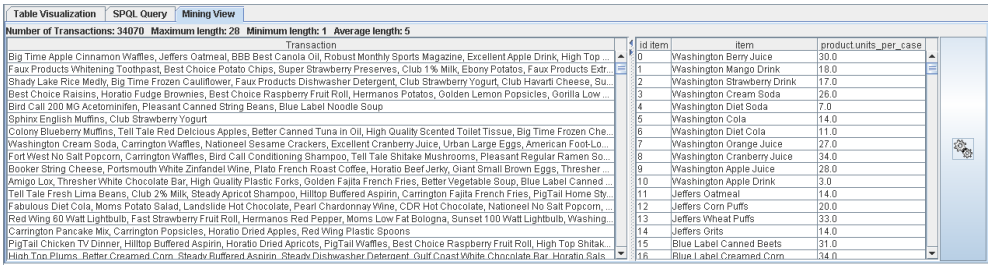


Figure 14: Mining view definition process

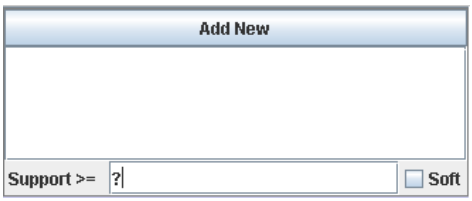


Figure 15: Constraint pane and support value space

```
2. SELECT product.product_name, product.gross_weight, sales_fact_1998.time_id,
   sales_fact_1998.customer_id, sales_fact_1998.store_id
3. FROM [product], [sales_fact_1998]
4. WHERE sales_fact_1998.product_id=product.product_id
5. TRANSACTION sales_fact_1998.time_id, sales_fact_1998.customer_id,
   sales_fact_1998.store_id
6. ITEM product.product_name
7. ATTRIBUTE product.gross_weight
```

In line 1 there is the header that classify the query as a *mine* query and specify the support value, next in line 2-3-4 there is a SQL query that extract for the dbms all the information needed: all the field and table used in mining definition must be in this part. From line 5 to 7 we specify what fields identify a transaction (line 5), what the is the item of transaction (line 6) and what is the attribute of this items (line 8). The Syntax is SQL like and very simple, additional information about it are in appendix 8.1.

4.2 Resolve Attributes conflicts

In def.8 an Attribute is described as a value that have functional dependency $\mathcal{I} \rightarrow \mathcal{A}$, but in the real world is not guarantee this property (e.g. price of a item can change over time). ConQueSt analyze the mining definition given and if there are possibilities of a conflict, near the attribute field appears a warning icon as in fig.16(a), clicking with right mouse button on the field and selecting Resolve the window in fig.16(b), where the user can select one method to resolve the conflict:

Take first. Is the default one and take the first value associated with item, the other is ignored.

Take last. Each time a new value is fended associated with a item, the relation is updated.

Take count. The value of attribute is ignored, each item is associated to the number of distinct value of the attribute field.

Take maximum. Take the maximum of the attribute field values.

Take minimum. Take the minimum of the attribute field values.

Take average. Take the average of the attribute field values.

Take sum. Take the sum of the attribute field values.

(a) (b) (c)

Figure 16: How to resolve an attribute conflict

When a method is specified the warning became gray as in fig.16(c).

4.3 Add Constraints

The previous section provided all the needed components for defining a constraint-based frequent pattern query as follows.

Definition 16 (Constraint-based frequent pattern query) *Using all the definition above (def. 3 - 15): given a Mining view $\phi = \langle \mathcal{D}_\xi, \mathcal{L}_{\xi'} \rangle$, the minimum support threshold σ and a conjunction of constraints on itemsets $\mathcal{C}$, the task is to extract all the itemset:*

$$freq(\phi, \sigma, \mathcal{C}) = \{(i, S) \mid i \in \delta_V(\mathcal{I}) \wedge \mathcal{C}(i) \ \wedge \ supp_\xi(i) = S \ \wedge \ S \geq \sigma\}$$

where $\delta_V(\mathcal{I})$ indicate the set of values of the item in the transactional dataset $\mathcal{D}_\xi$.

this definition is the same of def.1 but in this case the source data is the processed data from the DBMS selected by the user. Using the same example of SQPL query showed above we complete it with a constraint:

```
1. MINE PATTERNS WITH SUPP>= 5 IN
(...)
8. CONSTRAINED BY Average(product.gross_weight)<=15
```

The two new elements are the first line where there is the header that classify the query as a *mine* query and specify the support value and the last line with the constraint on the attribute: $\sigma, \mathcal{C}$. The normal way to specify a constraints in $\mathcal{C}$ is the crisp one (as in the example), but ConQueSt give the possibility to use a different type of constraints named Soft where the result of $Th(\mathcal{C})$ is continuous.

4.3.1 Crisp Constraints

A *Crisp Constraint* is a function $\mathcal{P}(x) \rightarrow \{0, 1\}$ that is used to prune the space of solutions in the process. The constraints that can be used in ConQueSt are showed in table 1. This set of constraints can be divided into 2 type:

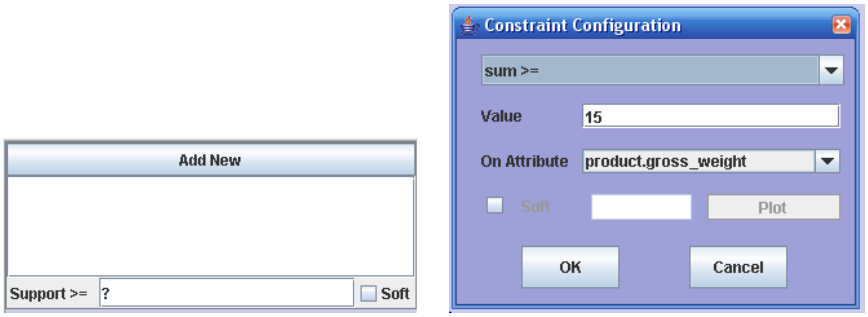
On Attributes Constraints means that the constraint is based on the values of on item's attribute in the pattern (e.g. max(price)).

subset	subset	**supset**	superset
asubset	attributes are subset	**len**	length
asupset	attributes are superset	**acount**	attributes count
min	minimum	**max**	maximum
range	range	**sum**	sum
avg	average	**var**	variance
std	standard deviation	**spv**	sample variance
md	mean deviation	**med**	median

Table 1: The set of available constraints.

On Properties Constraints means that the constraint is based on a properties of the pattern (e.g. lenght() in this case is implicit that the argument of the constraint is the pattern).

With the combination of constraints on all the attributes selected by the user, he can represent the specific type of knowledge he want to extract. In ConQueSt the crisp constraint is named only constraints because, in the literature, is considered as the *normal* type of constraints. To add a constraints in ConQueSt the user can write them into the query or can use the top right constraints panel (fig.17(a)) clicking on *Add New* button. The system show a window (fig.17(b)) where the user can select the constraints, the threshold and the attribute based on (the system show only the attributes in the current mining view definition).



(a) (b)

Figure 17: Constraints panels

If the user select a set constraint as subset, superset, attribute subset or attribute superset, the system show a window where all the values available (fig.18), the user can select all the values needed by double clicking on it (or writing it on the parameter space separated by a comma).

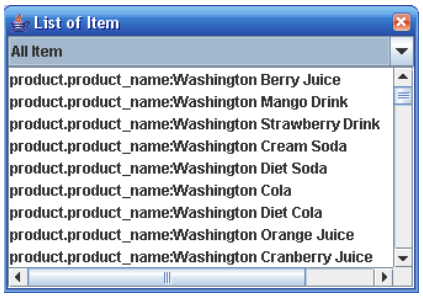


Figure 18: List of values panel

4.3.2 Soft Constraints

Soft constraints are an advanced feature of ConQueSt , the *Soft constraints* aren't boolean predicates that describe if an attribute/property satisfy it but continuous function $\mathcal{P}(x) \rightarrow [0, 1]$ that give the interesting value of the pattern as showed in fig.19.

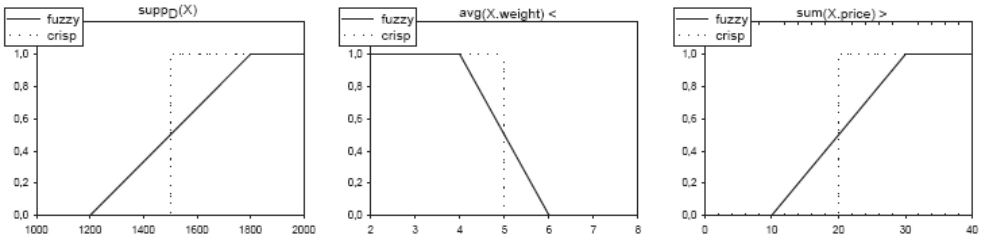


Figure 19: Difference between Crisp and Soft Constraints output

To Know how *soft* is a constraint the user must specify the *softness* value: a percentile of the threshold value of the constraint that specify the inclination of the line graph. The idea of this type of definition is that the crisp constraints can be too rigid and can eliminate patterns with value of

constraint near the threshold but which don't satisfy it completely, or in other hand, there are more constraints but not all of them have the same importance, in fact With this new concept of constraints we have two different way to calculate the interest value for a conjunction of constraints:

Fuzzy We take the minimum of the interesting values of all the constraints.

Probabilistic We take the multiplication of the values of all the constrains.

In this way we can perform two new task:

extract the pattern with interesting value greater then a threshold the user specify an interesting threshold and the system extract all the pattern that have the interesting value greater than this value.

extract the top-k interesting pattern the user specify how many top pattern want to extract and the system perform a search in the space to find them. If there are more than one pattern that have the same interesting value in the last position the result will be greater than the values specified by the user.

In ConQueSt is simple to create a soft constraint: in the constraints panel in fig.17(b) check the option soft and specify the softness in the box on the left (the user can visualize graph of the soft constraint clicking on plot button). All the constraints created (soft and crisp) are showed int the constraints panel showed as in fig.20 The frequency constraint can be soft too, in this

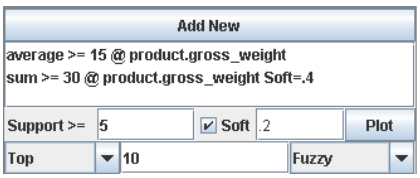


Figure 20: How the constraints are showed in the panel

case check the soft option on the left right and then specify the softness value for it (fig.20). Using the Soft constraints the format of a SPQL query change a little:

```
1.MINE TOP 10.0 FUZZY PATTERNS WITH SUPP>= 10.0 SOFT 0.3 IN
(...)
8.CONSTRAINED BY sum(product.gross_weight)>=35 SOFT .2
  AND minimum(product.gross_weight)>=5
```

The changes in the relative SPQL query is on line 1 and 8 where there are the information about the soft method and the parameters. For more details see the appendix 8.1.

5 Execute SPQL query

Now the user have all the elements to create a SQPL query, to execute it he can select the SPQL query pane and than clicking on the Execute button, or from the Action menu the Execute item. When the query is executed the a progress bar appear and show the process of the following steps:

Create Dataset Create the dataset from the mining view definition.

Create Attributes Create the attribute of item from the mining view definition.

Mining Search for patterns in the dataset.

All the step can be chaced from queries executed before by the users and then skipped. On the Workbench there is another windows with abort button to stop the process.

6 Navigate Results

This section show how the user can visualize, navigate and manipulate the result of a query. In fig.21 is showed a schema of the possibilities of navigation.

6.1 Pattern Browser

After the execution of a query the pattern browser appears as in fig.22. The patterns are divided in two parts: in the left side show the extracted patterns, in the right side there are the source query and general information. The

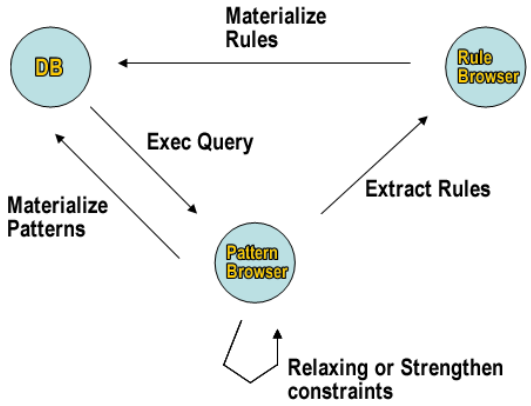


Figure 21: Navigation Schema

patterns are in tabular format and have two type of visualization: itemset view or item view, the default mode is the first one. In the Itemset view we have the following columns:

Id. A number that identify the pattern.

Frequent itemset. The pattern in transactional representation.

Support. The support value of the pattern (see def.1).

Constraints value. The values of all constraints in the source query

Otherwise in the item view we have:

Id. A number that identify the pattern, in this case if a pattern is formed by more item there are multiple rows with same id.

Item. A item of a pattern, the Id of the item is the same of the pattern in the itemset view.

Attribute value. The values of all attributes specified in the query of the item

To change type of visualization and other option use the visualization menu in the pattern browser. In case of the query have soft constraints in the visualization menu the user can use the command show interesting value to

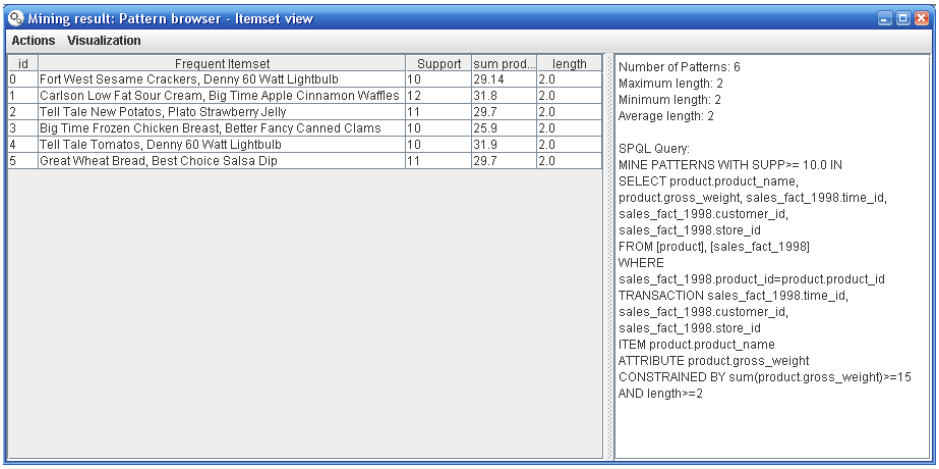


Figure 22: Pattern Browser

change visualization of the value of constraints from the real values and the interesting values of them. Clicking with right button on one of the columns (not all the ordering are available for all the columns) of pattern browser the user can perform different type of sorting:

Ascending. Sort by ascending value.

Descending. Sort by descending value.

Cardinality Sort by the number of item in a pattern.

Cardinality*Support. Sort by the product of Cardinality and the support value of the pattern.

6.1.1 Relaxing or Strengthen constraints

The pattern browser isn't only a visualization of pattern, but the user can tuning the constraints specified in the source query by clicking with right button on one of the constraint column and select *Modify Constraint*, the window in fig.23 is showed. The user can modify the threshold of the constraint selected and then clicking on Apply the change are applied to the pattern browser. During the change in the modify constraint window there are the description of the current operation the user are going to perform:

Figure 23: Pattern Browser

Relaxing. The $Th(\mathcal{C}) \subseteq Th(\mathcal{C}')$.

Strengthen. The $Th(\mathcal{C}') \subseteq Th(\mathcal{C})$.

using the same symbolism of def.1. Checking the box *Apply on change* the change is applied immediately.

6.1.2 Materialize patterns

6.2 Extract Rules

7 Example of use

8 Appendix

8.1 Formal SPQL grammar

```
<SpqlQuery> ::= (<SqlQuery>| <MineQuery> | <Discretize>)
<MineQuery> ::=<Header><br><SqlQuery><br><MiningDefinition><br><Constraints>
<Header> ::= MINE PATTERNS WITH SUPP >= <Number>
<MiningDefinition> ::= TRANSACTION <Transaction><br> ITEM
<Item><br>[ ATTRIBUTE <Attribute>]
<Transaction> ::= <Field>[<Separator><Transaction>]
<Item> ::= <Field>[<Separator><Item>]
<Attribute> ::= <Field>[<Separator><Attribute>]
<Field> ::= <String>.<String>
<Constraints> ::= CONSTRAINED BY <Function>
<Function> ::= (<FunctionM>(<Field>)<Op><Number> | <FunctionS>(<Field>)<Op><Set> |
                <FunctionN>()<Op><Number> ) [<Separator>(Function)]
<FunctionM> ::= (Minimum | Maximum | Range | Variance | Std_Deviation | Median | Average | Sum)
<FunctionS>::= (Subset_of | Superset_of | Attribute_Subset_of | Attribute_Superset_of )
<FunctionN>::= Length
<Op> ::= (>|<|>=|<=)
<Separator> ::= ,
<br> ::= \n
<Set> ::= <String>[<Separator><Set>]
<Discretize>::= DISCRETIZE <Field> AS <Field> <br> FROM <String> <br> IN
(<Method>| <Intervals>) BINS <br> SMOOTINGH BY <Smethod>
<Method> ::= (EQUALWIDTH | EQUALDEPTH)
<Smethod> ::= (AVERAGE | COUNT | BIN BOUNDARIES)
<Intervals>::= (<Number> <Separator> <Number>)[<Separator> <Intervals>]
<Number> ::= (0-9) [<Number>]
<String> ::= (a-z|A-Z|0-9) [<String>]
```

Table 2: A portion of SPQL formal grammar definition.