

Università degli Studi di Pisa

Dipartimento di Informatica
Laurea Specialistica in Tecnologie Informatiche

Anno accademico 2005/2006

ConQueSt:
a Constraint-based Querying System
for Exploratory Pattern Discovery

Candidato:
Roberto Trasarti

Relatori:

Dott.sa Fosca Giannotti

Dott. Francesco Bonchi

Controrelatore:

Dott. Paolo Ferragina

Sommario

Il contributo di questa tesi è il disegno e lo sviluppo di un sistema di Knowledge Discovery denominato ConQueSt.

Basato sul paradigma del Pattern Discovery guidato dai vincoli, ConQueSt segue la visione dell'Inductive Database:

- il mining è visto come forma più complessa di querying,
- il sistema quindi è equipaggiato con un data mining query language, e strettamente collegato con un DBMS,
- i pattern estratti con query di mining diventano cittadini di prima classe e, seguendo il principio di chiusura, vengono materializzati accanto ai dati nel DBMS.

ConQueSt è già stato presentato con successo al workshop internazionale della comunità dello Inductive Database [49], e alla conferenza IEEE International Conference on Data Mining Engineering (ICDE 2006) [50]. A giugno sarà presentato alla conferenza italiana di basi di dati (SEBD 2006) [51]. E' attualmente in corso la sottomissione a rivista.

Esiste un sito web per la distribuzione della documentazione e del software stesso all'indirizzo <http://www-kdd.isti.cnr.it/~conquest/>.

Ringraziamenti

Desidero ringraziare tutte le persone che mi hanno permesso di raggiungere questo traguardo.

Per la loro fiducia, disponibilità e per l'opportunità di studiare e lavorare all'interno del CNR di Pisa, devo ringraziare la dott.sa Fosca Giannotti e il prof. Dino Pedreschi che, sin dal tirocinio della laurea triennale, hanno creduto in me e mi hanno sostenuto in ogni modo permettendomi di crescere come studente e come membro di un team di ricerca stimolante come il KDD-lab.

Un ringraziamento speciale va alle persone con cui ho portato avanti questo progetto: Francesco Bonchi e Claudio Lucchese, che mi hanno supportato/sopportato durante tutta la sua realizzazione, aiutandomi a vedere chiaro nei momenti difficili e per avermi insegnato che bisogna cercare di migliorarsi sempre sfruttando il 110% di se stessi per raggiungere gli obiettivi.

Ringrazio la mia Famiglia per avermi dato tutto quello che un figlio/fratello possa desiderare e per avermi sempre sostenuto in qualsiasi scelta io abbia preso, facendomi sempre sentire con le "spalle coperte" ma permettendomi un'autonomia completa che mi ha reso più forte di fronte ai fatti della vita.

Un aiuto fondamentale è stato quello dei miei amici di Pisa, in particolare: Matteo e Livio ~~senza i quali probabilmente sarei rimasto sano di mente~~ che sono diventati la mia seconda famiglia durante questi anni e Massimiliano per tutte le conversazioni-discussioni sulle scelte implementative di qualsiasi progetto.

Un grazie di cuore anche a tutti i miei amici di Viterbo che in questi anni mi hanno visto ad intermittenza e non si sono mai stufati di sentirsi dire "sto solo per il week-end poi devo tornare a Pisa che ho da fare...", in particolare a Simone, Livio e Agnese: miei storici "compagni di merenda" senza i quali non sarei quello che sono.

Anche se non "sempre presenti" voglio ringraziare delle persone che mi hanno permesso di vedere le cose sotto un punto di vista differente: Dumah, Hob, Omak, Mira, Papadopulos, Drupy, Scialpy e Corellus. Un ringraziamento va alla prof.ssa Vanna Serani e la prof.ssa Marilisa Carella per l'attività svolta presso la scuola media Fibonacci di Pisa.

La persona a cui devo tutto, che ogni giorno mi ha sostenuto e incoraggiato credendo ciecamente in me, è il mio Amore che non finirò mai di ringraziare.

INDICE

1. INTRODUZIONE	9
Contesto	9
Obiettivi	11
Contributo	11
Organizzazione tesi	12
2. INDUCTIVE DATABASE E CONSTRAINED PATTERN DISCOVERY	14
2.1 Inductive Database	14
2.2 Constraint Pattern Discovery	16
Origini e Definizioni	16
Evoluzione e Stato dell'arte degli algoritmi	18
Apriori	18
CAP	19
ExAMiner	20
FIC ^A e FIC ^M	22
PATTERNIST	23
2.3 Linguaggi d'interrogazione per il data mining	24
Linguaggi relazionali	24
Linguaggi logici	26
3. SIMPLE PATTERN QUERYING LANGUAGE	28
3.1 Il linguaggio SPQL	29
3.2 Il processing di una query SPQL	30
Estrazione dei dati dal database	31
Trasformazione dei dati da relazionali a transazionali	31
Esecuzione algoritmo di Mining	33
Algoritmi di Post-Processing	33
4. IL SISTEMA CONQUEST: GUIDA ALL'USO	34
4.1 Per iniziare	34
Requisiti di sistema	34
Installazione e esecuzione	34
4.2 Connessione al database	35
Workbench	36
Analisi dei dati	37
4.3 Strumenti	39
Estrazione conoscenza	42
Aggiungere vincoli	44
Navigazione dei risultati	46
Estrazione Regole	48
Materializzazione dei risultati	49
5. ARCHITETTURA E IMPLEMENTAZIONE	51
5.1 Architettura	51
Livello Gui	51
I Componenti Interface	52
I Componenti Workers	54
I Componenti Utility	56
IL Sistema di Caching	57
Livello Pre-Processing	57

Livello Mining	61
5.2 Implementazione	61
Tecnologie	61
Codice Sorgente	62
6. CONCLUSIONI E SVILUPPI FUTURI	64
7. BIBLIOGRAFIA	66

1. INTRODUZIONE

Contesto

L'esistenza di sempre più grandi database di informazione scientifica e commerciale ha fatto sì che si sviluppasse un crescente interesse nei metodi per la ricerca di strutture all'interno di questi dati. Tali metodi spesso vanno sotto il nome di *data mining*. Recentemente due tipi differenti di strutture sono state identificate nel data mining:

- *Modelli*: descrizioni globali dell'insieme dei dati
- *Pattern*: strutture locali che descrivono solo alcune parti dei dati.

Di conseguenza il *Pattern Discovery*, ha un ruolo distinto nelle tecnologie di data mining. In particolare, dato che la *frequenza* fornisce il supporto di una qualsiasi conoscenza estratta, è la più usata e probabilmente la più utile, misura di interesse per i pattern estratti. Per questa ragione durante l'ultima decade molti ricercatori hanno concentrato gli studi sul problema computazionale del *Frequent Pattern Discovery*, per esempio estrarre patterns che soddisfano una soglia minima di frequenza. [2,10,20,21].

La più semplice forma di pattern frequente è l'itemset frequente: dato un database di transazioni (una transazione è un insieme di item) si vuole trovare quel sottoinsieme di transazioni (itemsets) che appaiono insieme frequentemente (ovvero più volte della soglia specificata).

Questo problema computazionale è la base dell'estrazione di *Regole Associative*. L'idea dell'estrazione di regole associative [1] è originata dall'analisi dei dati tipo *market-basket* dove è interessante trovare regole che descrivano il comportamento di acquisto di prodotti da parte dei clienti. La loro diretta applicabilità ai problemi di mercato insieme alla loro facile comprensione, anche per i non esperti di data mining, hanno fatto sì che divenissero un metodo di mining molto popolare.

Tuttavia gli itemsets frequenti non sono importanti solo nel contesto dell'estrazione di regole associative, ma possono essere usati come elementi base in molti altri tipi di analisi, come la classificazione [56,57] o il clustering [81,96].

Recentemente la comunità di ricerca ha rivolto la sua attenzione a patterns frequenti più complessi estratti da strutture come sequenze, alberi o grafi. Tutti questi tipi diversi di pattern hanno differenti peculiarità e campi di applicazione (per esempio le sequenze sono particolarmente adatte per applicazioni, gli alberi ad esempio possono essere utilizzati per i documenti XML e i grafi per la biologia molecolare, in economia o in la creazione di nuovi farmaci o per il Web-mining), ma tutti condividono lo stesso aspetto computazionale: un input di

grosse dimensioni, uno spazio di ricerca esponenziale, e un insieme di soluzioni troppo grande. Questa situazione – troppi dati che producono troppi pattern – crea due distinti tipi di problemi:

1. *Degrado della performance*: il mining generalmente diventa inefficiente o, spesso, semplicemente infattibile.
2. *Difficile lettura dei risultati*: data la grossa quantità di risultati la conoscenza utile viene frammentata in mezzo a molti pattern inutili

Il paradigma del *constrained-based mining*, ovvero del mining guidato dai vincoli, è stato introdotto, per ovviare ad entrambi questi problemi. I vincoli permettono di focalizzare la conoscenza interessante e quindi ridurre il numero dei pattern estratti ai soli potenziali interessanti. Inoltre possono essere spinti in profondità nel processo di estrazione così da ottenere una migliore performance. Per questa ragione il problema di come spingere diversi tipi di vincoli nella computazione di itemsets frequenti è stato molto studiato [5,3,4,6,11,8,9,12,16,23,24,26].

La maggior parte del lavoro è stato fatto sugli algoritmi del pattern discovery per cercare di superare le difficoltà della ricerca a causa della esponenziale crescita dello spazio di ricerca. Un altro importante obiettivo è quello di definire una *query languages*, ovvero un linguaggio che l'utente possa utilizzare per specificare i parametri del processo di mining. Le informazioni necessarie che l'utente dovrebbe specificare sono:

- I dati di origine
- Il tipo di conoscenza da estrarre
- Il background o il dominio della conoscenza
- Misure interessanti per la valutazione del pattern
- La rappresentazione della conoscenza estratta

Sviluppando un linguaggio di query che possa incorporare tutte queste opzioni si otterrebbe, come nel caso dei database relazionali, un alto livello di espressività nell'espressione dei compiti del processo di mining, una chiara e ben definita separazione tra logica e implementazione fisica dei processi di mining, e una facile integrazione con varie fonti di informazioni. Chiaramente l'implementazione di questa visione presenta una grossa sfida.

E' proprio questa sfida che viene affrontata da ConQueSt che, oltre a definire e implementare un linguaggio con queste caratteristiche, mette a disposizione dell'utente un sistema per la visualizzazione, gestione e manipolazione della sorgente dei dati e dei risultati ottenuti con il processo di mining.

Obiettivi

L'obiettivo della tesi è il disegno e lo sviluppo di un sistema di Knowledge Discovery denominato ConQueSt, con lo scopo di colmare il “gap” tra database relazionali, SQL, database transazionali e strumenti di Mining, basandosi sulla visione di [35] attraverso le ricerche sviluppate all'interno del gruppo di ricerca dell'ISTI-CNR di Pisa [39], realizzando un sistema completo che possa permettere, in modo semplice e diretto, di manipolare il database e di utilizzare uno strumento specializzato nel *Constraint Pattern Discovery*. Questo processo permetterà all'utente di ottenere risultati che saranno considerati mattoncini di base per algoritmi di post-processing o di ricerche incrementali. Attualmente in ConQueSt è stata implementata l'estrazione di regole associative, ma è pronto per poter estendere l'utilizzo di questi mattoncini ad altri algoritmi come il clustering, alberi di decisione, ecc.. Vista la moltitudine di database esistenti ConQueSt è stato sviluppato con tecnologie JDBC/ODBC che permettono il collegamento e l'utilizzo di vari sistemi di DBMS compatibili.

Contributo

ConQueSt è stato “costruito” intorno al motore di mining PATTERNIST sviluppato all'interno del KDD-lab del CNR di Pisa in collaborazione con HPC-lab per creare un sistema interattivo con l'utente rimanendo robusto e efficiente dal punto di vista computazionale.

A questo scopo è stata studiata un'estensione dell'SQL: l'SPQL (simple pattern query language) che permette all'utente di specificare, oltre alle normali query SQL, query di ricerca attraverso il motore di mining direttamente sulle tabelle del database relazione.

Attraverso l'operazione di pre-processing, che trasforma i dati da relazionali a transazionali secondo le specifiche della query, il processo è completamente trasparente all'utente che percepisce gli strumenti di mining come estensione delle normali funzionalità del linguaggio di interrogazione. Su questa base è stato creato un sistema che permetta all'utente di:

- visualizzare informazioni contenute in un database
- mettere a disposizione dell'utente strumenti per la manipolazione dei dati
- specificare visualmente e testualmente i parametri per l'utilizzo di algoritmi di data mining attraverso il linguaggio SPQL
- esplorare i risultati dell'analisi
- raffinare i risultati ottenuti attraverso algoritmi di post-processing
- materializzare i risultati ottenuti per osservazioni future o operazioni incrementali

ConQueSt è già stato presentato con successo al workshop internazionale della comunità dello inductive database [49], e alla prestigiosa conferenza IEEE International Conference on Data Mining Engineering (ICDE 2006) [50]. A giugno sarà presentato alla conferenza italiana di basi di dati (SEBD 2006) [51]. E' attualmente in corso la sottomissione ad una prestigiosa rivista.

Organizzazione tesi

La tesi è organizzata come segue:

- Nel capitolo 2 viene descritto il contesto in cui si muove ConQueSt: l'idea dell'Inductive Database e del Constrained Pattern Discovery mostrando una carrellata degli algoritmi esistenti e di come si è arrivati allo sviluppo di un sistema completo come PATTERNIST.
- Nel capitolo 3 viene introdotto il linguaggio di query utilizzato in ConQueSt: lo SPQL (Simple Pattern Query Language), un nuovo linguaggio sviluppato, estendendo lo SQL standard, per dare la possibilità all'utente di specificare tutti i parametri necessari alla definizione delle "transazioni" e all'estrazione dei patterns da parte del motore di mining.
- Nel capitolo 4 viene presentata una guida all'uso dell'applicazione ed è realizzata in modo da poter essere considerata come una sezione a se stante come manuale d'uso. Verranno presentate inizialmente come collegarsi ai propri dati in un database e successivamente come utilizzare tutti gli strumenti di visualizzazione, estrazione, manipolazione e materializzazione dei dati.
- Nel capitolo 5 viene mostrata la struttura di ConQueSt, ovvero la sua architettura e dei dati statistici del codice sorgente. Nella sezione dell'architettura verranno presentate le classi di appartenenza di vari operatori dando una descrizione delle loro funzioni.
- Nel capitolo 6 vengono presentati i progetti e gli sviluppi futuri di ConQueSt con una loro breve descrizione.

Essendo ConQueSt un progetto di ricerca, le sue funzionalità sono in continua evoluzione, per cui verranno presentate delle indicazioni sulle tendenze che il progetto ha nel prossimo futuro.

Nonostante la presente tesi fornisca una panoramica esauriente del sistema, riteniamo che il modo migliore per apprezzare la robustezza e la facilità d'uso sia proprio quello di provarlo e di vederlo all'opera. A tal proposito allegato a questa tesi forniamo un CD contenente:

- il software ConQueSt,
- il codice sorgente,
- la documentazione in stile javadoc,

- un database di esempio per alcune rapide prove,
- la tesi corrente in formato elettronico.

2. INDUCTIVE DATABASE E CONSTRAINED PATTERN DISCOVERY

2.1 Inductive Database

L'inductive database nasce come idea in [35] dove viene espressa l'esigenza di sistemi integrati con i database per riuscire ad applicare gli algoritmi di mining. Nell'articolo vengono presentati alcuni punti fondamentali che questo nuovo tipo di sistemi dovrebbero avere:

1. *“By a KDD query we mean a predicate which returns a set of objects that can either be KDD objects or database objects such as records or tuples. The KDD objects typically will not exist a priori, thus querying the KDD objects requires generation at run time.”*
2. *“In relational database, the result of a query is a relation that can be queried further. In fact, a relational query “does not care” if its argument is the original database or if it is a result of other query. This typically referred to as a closure principle, and it is one of the most important design principles. We also would like a KDD query language to satisfy such a closure principle.”*
3. *“Another difficulty is that the border between querying and discovery is fuzzy; one might even say that there is no such a thing as real ‘discovery’. Discovery is just a matter of the expressive power of a query language.”*

Dalla 1 nasce il concetto di database “Induttivo” ovvero che genera al suo interno conoscenza intermedia per elaborare la richiesta dell'utente; nella 2 viene evidenziata la proprietà di “chiusura” che è alla base dell'evoluzione del database stesso esplorando la conoscenza su conoscenza già elaborata in precedenza; infine nella 3 viene evidenziata l'importanza dell'espressività del linguaggio di querying così da poter permettere all'utente di specificare “esattamente” ciò che sta cercando.

La prima nozione di *inductive database* è presentata in [36] e [37] che tentarono di formalizzare la nozione di “interactive mining process”.

In questo contesto avremo che:

Definizione 2.1 ([37,38]): Data una istanza \mathbf{r} di una relazione \mathbf{R} , una classe L di patterns, e un predicato di selezione q , lo scopo del pattern discovery è di trovare una *teoria induttiva* per cui vale:
 $Th(L, r, q) = \{s \in L \mid q(r, s) = true\}$

Da questa definizione traspare il concetto di conoscenza indotta che per l'utente finale è invisibile, ovvero la ricerca e la costruzione dei patterns partendo dal database su cui poi si applica il predicato.

In [36] si tenta anche di dare una prima definizione su cui poter basare un linguaggio di query per l'inductive database:

Definizione 2.2 ([36]): Un *inductive database schema* è definito come $R = (\mathbf{R}, (Q_R, e, V))$, dove

- \mathbf{R} è uno schema relazionale
- Q_R è una collezione di patterns
- V è un insieme di valori risultanti
- e è una funzione di valutazione che definisce i patterns semantici in termini di V

Una *istanza induttiva* è definita come una coppia (\mathbf{r}, s) , dove \mathbf{r} è una istanza di \mathbf{R} e $s \subseteq Q_R$.

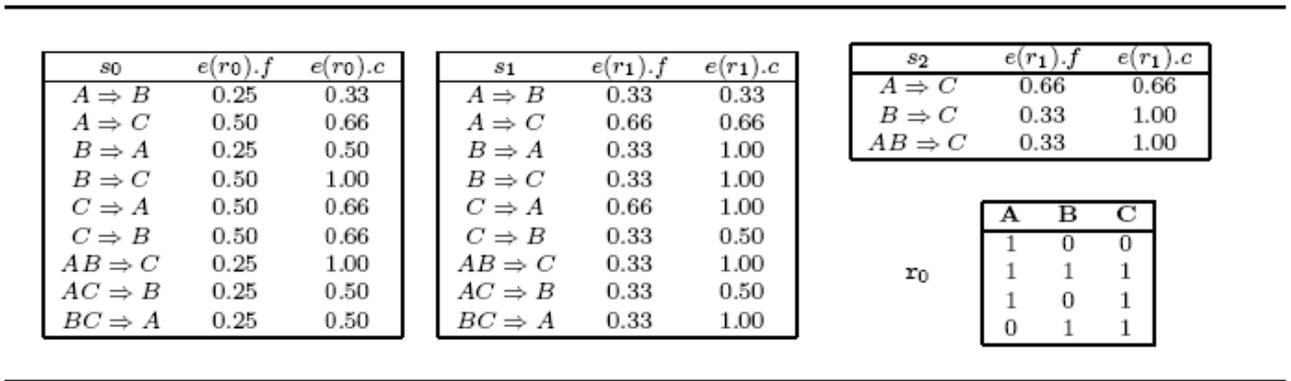


Figura 1: Componenti di una istanza di inductive database

Un algebra per questo schema definisce in modo più generale queries che una semplice algebra relazionale. Una tipica query di un inductive database può operare su entrambi i componenti di un inductive database: la parte deduttiva \mathbf{r} e la parte indotta s che rimane completamente trasparente all'utente che non vede le operazioni di mining se non come risultato di una computazione di \mathbf{r} .

Quindi possiamo dire che la definizione 2.1 e la definizione 2.2 sono equivalenti. Per ogni $Th(L, r, q)$ può essere formalizzata con una istanza di induttiva (r, s) , dove $Q_r=L$ e $e=q$, quindi ogni istanza induttiva è una teoria induttiva.

Nella 2.1 viene presentato il *predicato selettivo* che divide l'insieme dei patterns appartenenti alla Teoria (Th) e quali no, questo elemento corrisponde esattamente alla definizione di vincolo che porterà alla definizione del Constraint Pattern Discovery di cui parleremo nella prossima sezione.

2.2 Constraint Pattern Discovery

Gli studi e le ricerche sul Pattern Discovery hanno come presupposto che i dati siano presenti in un Database Transazionale definito come segue:

Definizione 2.3 (Transactional Database): Dato $I = \{x_1, \dots, x_n\}$ un set di *items*, dove un *item* x_i è un oggetto con qualche attributo (*Attributes*) predefinito. Un *itemset* X è un subset di I non vuoto. Se $|X| = k$ allora X è chiamato *k-itemset*. Un *database transazionale* D_T è una raccolta di itemset $t \in 2^I$, chiamati transazioni.

Un esempio della rappresentazione di questo “formato” può essere (basket analysis):

date	cust	itemset
11-2-97	cust1	{beer, chips, wine}
11-2-97	cust2	{wine, beer, pasta, chips}
13-2-97	cust1	{chips, beer}
13-2-97	cust2	{jackets, col_shirts}
13-2-97	cust3	{wine, beer}
15-2-97	cust1	{pasta, chips}
16-2-97	cust1	{jackets}
16-2-97	cust2	{wine, pasta}
16-2-97	cust3	{chips, col_shirts, brown_shirts}
18-2-97	cust1	{pasta, wine, chips, beer}
18-2-97	cust2	{beer, chips}
18-2-97	cust3	{pasta}

name	price	type
beer	10	beverage
chips	3	snack
wine	20	beverage
pasta	2	food
jackets	100	clothes
col_shirt	30	clothes
brown_shirt	25	clothes

Figura 2: Esempio di Dataset Transazionale

dove ogni riga contiene delle chiavi identificative della transazione (date e cust) e di un insieme di *item* che compongono la transazione stessa (itemset).

Origini e Definizioni

Il Constraint pattern discovery è la diretta evoluzione del frequent pattern discovery, ovvero l'estrazione della conoscenza basandosi sul concetto di frequenza [39]:

Definizione 2.4 (Frequent Itemset): Dato un database transazionale D_T , il supporto di una itemset X nel database D_T , rappresentato da $supp_D(X)$, è il numero di transazioni in D_T che sono superset di X . Dato un *supporto minimo* σ , un itemset X è chiamato *frequente* in D_T se $supp_D(X) \geq \sigma$.

Il “vincolo” della frequenza permette una riduzione della dimensione dei dati e costituisce la base per molti algoritmi di mining come la classificazione o il clustering.

Successivamente l'attenzione si spostò sull'analisi di insiemi di dati più vasti cercando di applicare strutture come sequenze, alberi o grafi; in questo caso il solo vincolo di frequenza non riusciva a ridurre a sufficienza la dimensione dei dati e soprattutto risultava rigido perché non si poteva intervenire nel processo di estrazione della conoscenza inserendo proprietà note del dominio dell'analisi.

Proprio per questo è nato il Constraint Pattern Discovery che è definito come:

Definizione 2.5 (Constrained Itemset Mining): Dato un database transazionale D_T , il supporto di una itemset X nel database D_T . Un vincolo su un itemset è una funzione $C:2^I \rightarrow \{\text{vero}, \text{falso}\}$. Diciamo che un itemset soddisfa un vincolo se e solo se $C(I) = \text{vero}$. Definiamo *teoria* di un vincolo come l'insieme di itemset che soddisfano il vincolo: $Tb(C) = \{X \in 2^I \mid C(X)\}$. Il Constraint Itemset Mining è definito come quel processo che calcola $Tb(C)$ su un database D_T .

A questo punto il constraint frequent pattern discovery non è altro che un caso particolare del Constraint Pattern Discovery, infatti anche la frequenza è un vincolo e può dunque essere definita come tale: $Tb(C_{freq})$, quindi:

Definizione 2.6 (Constrained Frequent Itemset Mining): Il Constrained Frequent Itemset Mining è quel processo che calcola $Tb(C) \cap Tb(C_{freq})$ su un database D_T .

Il concetto di funzione di “selezione” non è nuovo, infatti come abbiamo visto nell'inductive database, la 2.6 può essere messa a confronto con la 2.1 dove si parla di predicato di selezione q che ha lo stesso significato dell'attuale C .

Evoluzione e Stato dell'arte degli algoritmi

Come descritto, gli attuali algoritmi di Constraint pattern discovery, si basano sulla teoria dello “spingere” il vincolo all'interno della computazione per poter “tagliare” lo spazio di ricerca prima possibile così da avere una quantità di dati da analizzare inferiore. Per far ciò bisogna analizzare i tipi di vincoli e come poter sfruttare le loro proprietà.

Procediamo dunque ad una breve descrizione dell'evoluzione storica di questo metodo e dei risultati che sono stati raggiunti.

Apriori

Il primo passo fu quello di determinare come poter spingere il concetto di frequenza all'interno di un algoritmo di mining, questo fu fatto in [41] dove venne presentato l'algoritmo Apriori:

Algorithm 1 Apriori

Input: \mathcal{D}, σ

Output: $Th(\mathcal{C}_{freq}[\mathcal{D}, \sigma])$

```
1:  $C_1 \leftarrow \{\{i\} \mid i \in \mathcal{I}\}$ ;  $k \leftarrow 1$ 
2: while  $C_k \neq \emptyset$  do
3:    $L_k \leftarrow count(\mathcal{D}, C_k)$ 
4:    $C_{k+1} \leftarrow generate\_apriori(L_k)$ 
5:    $k++$ 
6:  $Th(\mathcal{C}_{freq}[\mathcal{D}, \sigma]) \leftarrow \bigcup_k L_k$ 
```

Figura 3 : Pseudocodice dell'algoritmo Apriori. C_k contiene i k-itemset frequenti.

In questo algoritmo presentato un database transazionale D e un valore di supporto σ si aveva come Output l'insieme di pattern frequenti all'interno di D .

Questo primo approccio sfruttava la proprietà di anti-monotonicità del vincolo della frequenza:

Definizione 2.7 (Anti-Monotonicity): Dato un Itemset X . Un vincolo C_{AM} è *anti-monotono* se:

$$\forall Y \subseteq X : C_{AM}(X) \Rightarrow C_{AM}(Y)$$

Se C_{AM} vale per X allora vale per ogni sottoinsieme di X .

All'interno dell'algoritmo (fig. 3) sono presenti due insiemi che vengono “mantenuti e aggiornati” ad ogni ciclo di computazione:

1. L_k : l'insieme di k-itemset frequenti
2. C_k : l'insieme di k-itemset candidati

La relazione tra questi due insiemi è la parte generativa dell'algoritmo (`generate_apriori`) che partendo dal set L_k estratti al passo k genera i candidati del passo $k+1$ per combinazione di elementi. In questo primo passo del mining non esisteva ancora il concetto vero e proprio di “vincoli generici”.

CAP

La prima suddivisione in classi dei vincoli che dimostrò di avere proprietà interessanti fu presentata in [40] dove si introduceva un algoritmo Apriori-like chiamato CAP, che sfruttava due proprietà dei vincoli: l'*anti-monotonicità* (non solo per la frequenza) e la *succintezza*.

Definizione 2.8 (Succintness): Definendo:

- Un Itemset $I_S \subseteq \text{Items}$ è un insieme *succinto*, se può essere rappresentato come $\sigma_P(\text{Items})$ secondo qualche predicato P , dove σ è un operatore di selezione.
- $SP \subseteq 2^{\text{Items}}$ è un sovrinsieme succinto, se c'è un numero fissato di insiemi succinti $I_1, \dots, I_k \subseteq \text{Items}$, tale che SP può essere espresso in termini di sovrainsiemi stretti di I_1, \dots, I_k usando unioni o sottrazioni.

Se un vincolo C_S è succinto allora vale:

$$SAT_{C_S} = \{S \mid S \subseteq \text{Items} \wedge C_S(S) = true\}$$

è un sovrinsieme succinto.

Venivano a trovarsi quindi 4 classi di vincoli:

1. Anti-monotoni ma non succinti
2. Anti-monotoni e succinti
3. Succinti ma non anti-monotoni
4. Non succinti e non anti-monotoni

Ad ognuna di queste classi veniva associata una strategia diversa. Come visto in precedenza l'algoritmo Apriori sfruttava già il concetto di anti-monotonicità, quindi l'algoritmo fu esteso sostituendo semplicemente il vincolo della frequenza con l'intersezione tra vincoli che godono di questa proprietà:

$$Th(C_{freq} \wedge_{AM}) = Th(C_{freq}) \cap Th(C_{AM})$$

che a sua volta è un vincolo anti-monotono più stretto dei due vincoli separati.

Nel caso in cui il vincolo da prendere in considerazione godeva della proprietà di succintezza si sfruttava la sua proprietà come segue:

Dato che

$$Th(C_S) = \{X \in 2^I \mid \exists i \in X : i \in A_1\} \text{ dove } A_1 \text{ è il set di items che soddisfano } C_S.$$

Allora si può fare un “taglio” a livello di pre-processing e su tutti i livelli della computazione su tutti gli item che non fanno parte della $Th(C_S)$, così senza perdita di informazione necessaria alla generazione dei candidati è stato ridotto lo spazio di ricerca.

Il caso in cui i vincoli presi in considerazione non sono né succinti, né anti-monotoni allora CAP introduce nel processo un “rilassamento” di questi ultimi che abbia una o tutte e due le proprietà.

ExAMiner

Fino a questo momento non c’era modo di sfruttare un’altra proprietà dei vincoli, la Monotonicità definita come:

Definizione 2.9 (Monotonicity): Dato un Itemset X . Un vincolo C_M è *monotono* se:

$$\forall Y \supseteq X : C_{AM}(X) \Rightarrow C_{AM}(Y)$$

Se C_{AM} vale per X allora vale per ogni sovrinsieme di X .

In questo caso C_{freq} e C_M non si comportano nello stesso modo e quindi non era stato trovato un sistema per sfruttare l’intersezione delle due teorie:

$$Th(C_{freq \wedge M}) = Th(C_{freq}) \cap Th(C_M)$$

In [42] fu presentato un sistema di pre-processing per far sì che i due vincoli sfruttassero le loro proprietà in modo “collaborativo”:

Definizione 2.10 (ExAnte property): Dato un database transazionale D_T e una congiunzione di vincoli monotoni C_M , definiamo una μ -riduzione di D_T come il dataset risultante dal taglio delle transazioni che non soddisfano C_M :

$$\mu_{C_M}(D_T) = \{t \in D_T \mid t \in Th(C_M)\}$$

Questa riduzione non compromette il supporto della soluzione degli itemsets:

$$\forall X \in Th(C_M) : \text{supp}_{D_T}(X) = \text{supp}_{\mu_{C_M}(D_T)}(X)$$

Questa proprietà permette una riduzione dello spazio iniziale, inoltre il vincolo C_M riducendo le transazioni (μ -riduzione) fa sì che si riduca anche il supporto degli item che quindi possono essere eliminati con C_{freq} (chiamiamola α -riduzione). A questo punto ci troviamo all'interno di un loop di riduzioni, infatti le transazioni ora ridotte di lunghezza (numero di item) possono essere analizzate per cercare di eliminare altre transazioni che non soddisfano più il vincolo monotono e così via fino ad un punto di stasi.

Anche se questo livello di pre-processing ottenesse una piccola riduzione dello spazio iniziale questo comporterebbe un effetto di forte riduzione dello spazio nel processo di generazione dell'algoritmo Apriori.

ExAMiner ([43] e [44]) fu realizzato generalizzando la proprietà di ExAnte e utilizzandola non solo a livello di pre-processing ma a tutti i livelli della computazione.

Algorithm 2 *count&reduce*

Input: $\mathcal{D}_k, \sigma, \mathcal{C}_M, \mathcal{C}_k, V_{k-1}$

Output: $\mathcal{D}_{k+1}, V_k, L_k$

```

1: forall  $i \in \mathcal{I}$  do  $V_k[i] \leftarrow 0$ 
2: forall tuples  $t$  in  $\mathcal{D}_k$  do
3:   forall  $i \in t$  do if  $V_{k-1}[i] < k - 1$ 
4:     then  $t \leftarrow t \setminus i$ 
5:     else  $i.count \leftarrow 0$ 
6:   if  $|t| \geq k$  and  $\mathcal{C}_M(t)$  then
7:     forall  $X \in \mathcal{C}_k, X \subseteq t$  do
8:        $X.count++$ ;  $t.count++$ 
9:       forall  $i \in X$  do  $i.count++$ 
10:      if  $X.count = \sigma$  then
11:         $L_k \leftarrow L_k \cup \{X\}$ 
12:        forall  $i \in X$  do  $V_k[i]++$ 
13:      if  $|t| \geq k + 1$  and  $t.count \geq k + 1$  then
14:        forall  $i \in t$  if  $i.count < k$ 
15:          then  $t \leftarrow t \setminus i$ 
16:        if  $|t| \geq k + 1$  and  $\mathcal{C}_M(t)$  then
17:          write  $t$  in  $\mathcal{D}_{k+1}$ 

```

Figura 4: Procedura di “Count&Reduce” che implementa il sistema di α e μ -reduction di ExAMiner

In fig. 4 è presente lo pseudocodice che rappresenta la procedura di ExAMiner che implementa il sistema di α e μ -reduction. I tre passi di riduzione sono:

- $G_k(i)$: un item che non è sottoinsieme di almeno k k -itemset frequenti può essere tagliato via da tutte le transazioni in \mathcal{D}_T (α -riduzione)
- $T_k(i)$: una transazione che non è sovrinsieme di almeno $k+1$ k -itemset frequenti può essere rimossa da \mathcal{D}_T (μ -riduzione)

- $L_k(i)$: dato un item i e una transazione t , se il numero di k -itemset frequenti che sono sovrinsiemi di i e sottoinsiemi di t sono meno di k , allora i può essere rimossa dalla transazione t .

FIC^A e FIC^M

Sia le proprietà monotone che antimonotone, vengono sfruttate per la riduzione dello spazio di ricerca, ma il problema è che molti vincoli non presentano nessuna delle due proprietà. Per questo si sono trovati metodi per “convertire” un vincolo in una delle due classi introducendo sistemi di ordinamento sui dati.

In [45] e [46] viene presentata una classe di vincoli convertibili e due algoritmi basati su FP-growth che introducevano questo nuovo concetto nella computazione: FIC^A e FIC^M.

Definizione 2.11 (Prefix itemset): Dato un ordinamento totale R su un set di items I , un imtest $S' = i_1, \dots, i_n$ è chiamato prefisso di un itemset $S = i_1, \dots, i_m$ rispetto a R se vale $n < m$ e gli item in tutti e due gli itemset sono ordinati secondo R .

Definizione 2.12 (Convertible constraint): Un vincolo C_{CAM} si dice convertibile anti-monotono se esiste un ordinamento R sugli items tale che per ogni itemset S che soddisfa C_{CAM} anche ogni suo prefisso lo fa. Un vincolo C_{CM} si dice convertibile monotono se esiste un ordinamento R sugli items tale che per ogni itemset S che viola C_{CM} anche ogni suo prefisso lo fa. Un vincolo si dice convertibile se è convertibile anti-monotono o convertibile monotono.

Definizione 2.13 (Strongly convertible constraint): Un vincolo è fortemente convertibile se dato un ordinamento R è convertibile anti-monotono e con R^{-1} è convertibile monotono.

La differenza dei due algoritmi stava nel tipo di vincolo che applicavano durante la ricerca :

- FIC^A : $Tb(C_{freq}) \cap Tb(C_{CAM})$
- FIC^M : $Tb(C_{freq}) \cap Tb(C_{CM})$

La maggiore limitazione di tutti gli algoritmi basati su FP-Growth è che inizialmente tutto il database, compreso in una struttura ad albero, deve rimanere in memoria e quindi non riesce a gestire grosse quantità di dati inoltre in questi algoritmi non viene sfruttata la proprietà di ExAnte (2.10) introducendo in FIC^M il vincolo monotono solamente come filtro di post-processing.

PATTERNIST

Pattern risulta essere l'evoluzione di ExAMiner (e quindi sempre Apriori-like) introducendo però una ulteriore classe di vincoli: gli *anti-monotni deboli*. La proprietà che questi vincoli è simile a quella dell'anti-monotonicità, ma rilassata, infatti per la 2.7 se abbiamo un k-itemset che soddisfa il vincolo C_{AM} allora tutti i suoi sottoinsiemi di cardinalità k-1 lo soddisfano, nel caso degli anti-monotoni deboli invece basta che ne esista almeno uno di cardinalità k-1:

Definizione 2.14 (Loose Anti-monotone constraint): Dato un itemset X con $|X| > 2$, un vincolo è *anti-monotono debole* (rappresentato con C_{LAM}) se: $C_{LAM}(X) \Rightarrow \exists i \in X : C_{LAM}(X \setminus \{i\})$

Questo tipo di vincoli viene sfruttato da Patternist, al livello $k > 2$, eliminando tutte le transazioni che non contengono itemset di cardinalità k che soddisfino i vincoli C_{freq} e C_{LAM} perché quelle transazioni non concorreranno mai a formare itemset di cardinalità maggiore di k.

In questo modo Patternist racchiude in se un insieme di vincoli molto ampio di cui sfrutta le diverse proprietà a seconda della tipologia:

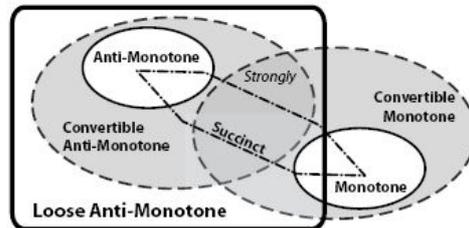


Figura 5: Classi di vincoli che Patternist supporta

Per chiarezza viene presentata una lista di vincoli “classici” e le classi di cui fanno parte:

Constraint	Properties
$S \subseteq V$	C_{AM}, C_S, C_{LAM}
$S \supseteq V$	C_M, C_S, C_{LAM}
$S.A \subseteq V$	C_{AM}, C_S, C_{LAM}
$S.A \supseteq V$	C_M, C_S, C_{LAM}
$\min(S.A) \geq v$	C_{AM}, C_S, C_{LAM}
$\min(S.A) \leq v$	C_M, C_S, C_{LAM}
$\max(S.A) \geq v$	C_M, C_S, C_{LAM}
$\max(S.A) \leq v$	C_{AM}, C_S, C_{LAM}
$\text{count}(S) \leq v$	C_{AM}, C_{LAM}
$\text{count}(S) \geq v$	C_M
$\text{count}(S.A) \leq v$	C_{AM}, C_{LAM}
$\text{count}(S.A) \geq v$	C_M
$\text{sum}(S.A) \leq v (\forall i \in S, i.A \geq 0)$	C_{AM}, C_{LAM}
$\text{sum}(S.A) \geq v (\forall i \in S, i.A \geq 0)$	C_M
$\text{range}(S.A) \leq v$	C_{AM}, C_{LAM}
$\text{range}(S.A) \geq v$	C_M, C_{LAM}
$\text{avg}(S.A) \theta v$	C_{LAM}
$\text{median}(S.A) \theta v$	C_{LAM}
$\text{var}(S.A) \theta v$	C_{LAM}
$\text{std}(S.A) \theta v$	C_{LAM}
$\text{md}(S.A) \theta v$	C_{LAM}

Figura 6: Vincoli e classi a cui appartengono

Proprio per questa sua caratteristica di completezza e efficienza PATTERNIST è il sistema integrato in ConQueSt per la ricerca di pattern frequenti e vincolati.

2.3 Linguaggi d'interrogazione per il data mining

Ci sono varie proposte per un linguaggio di querying per il data mining. A seconda del tipo di approccio possiamo considerare 2 categorie principali:

- *Relazionale*: il linguaggio di querying ha la stessa struttura e costrutti dei linguaggi relazionali (come lo SQL) allo scopo di utilizzare un supporto di questo tipo per l'applicazione di data mining.
- *Logico*: lo scopo principale di questi linguaggi è di costruire una interfaccia logica per gli algoritmi di data mining.

Linguaggi relazionali

Il linguaggio proposto in [27,28] estende lo SQL con un nuovo operatore MINE RULE, che permette l'estrazione di regole in formato relazionale. Considerando la relazione *transaction(Date,CustID,Item,Value)* che contiene le transazioni di vendite, la query in tav. 1 consente l'estrazione di regole con il supporto del 20% e la confidenza del 50%:

```
MINE RULE Associations AS
SELECT DISTINCT 1..n Item AS BODY, 1..1 Item AS HEAD, SUPPORT, CONFIDENCE
WHERE Body.Value > 100 AND HEAD.Value > 100
FROM transaction
GROUP BY CustID HAVING COUNT(Item) > 4
CLUSTER BY Date HAVING BODY.Date < HEAD.Date
EXTRACTING RULES WITH SUPPORT: 0.2, CONFIDENCE: 0.5
```

Tavola 1: Esempio di data mining query secondo [27,28]

Questo approccio riflette le seguenti caratteristiche :

- La sorgente dei dati è specificata come una entità relazionale, e il pre-processing sui dati è specificato con gli operatori normali dello SQL.

- L'estensione del linguaggio di query permette l'estrazione di regole associative unidimensionali. IL GROUP BY permette di specificare le transazioni mentre la SELECT gli Item.
- Limitata conoscenza di strutture complesse sia per la specificazione della query sia per i risultati che si otterranno.
- Permette l'utilizzo di vincoli come il supporto e la confidenza oltre a poter specificare vincoli sulla struttura delle regole da estrarre.
- La conoscenza estratta è rappresentata come tabelle relazionali contenenti i valori di: Body, head, Support e Confidence.

Imielinski e Virmani [22,23] mettono in evidenza l'importanza dell'integrazione dei risultati nel sistema relazionale. Lo MSQL che hanno proposto definisce una estensione dello SQL che materializza tutta la conoscenza estratta.

```

SELECT *
FROM transaction
WHERE VIOLATES ALL (
    GETRULES(transaction)
    WHERE BODY IS { (Item = *) }
    AND CONSEQUENT IS { (Item = *) }
    AND CONFIDENCE > 0.5 )

```

Tavola 2: Esempio di data mining query secondo Imielinski e Virmani

In tav.2 è mostrata una query che estrae tutte la parte di dati che viola la regola data.

On-Line Analytical Mining [7,15,64,33] è un approccio che integra il sistema on-line analytical processing con il data mining in un database multidimensionale. DMQL [17,19,30,18] è il corrispondente linguaggio di querying, molto simile ai linguaggi precedentemente mostrati ma con le seguenti differenze:

- I dati sono sottoforma di *cubi* (informazioni strutturate con dimensioni e misure)
- Permette la specifica di gerarchie
- I risultati sono in forma cubi “concettuali”.

Un esempio di query è presentata in tav.3:

```

MINE DATABASE Credit_Ranks
USE HIERARCHY location_hierarchy FRO C.Address
MINE CLASSIFICATION AS Credit_Risk
ANALYZE Risk

```

```

IN RELEVANCE TO C.Address, C.Age, P.type
FROM Client C, account I, account_type P
WHERE C.ID = I.client_ID AND P.ID = I.account_ID
WITH ERROR < 5%
DISPLAY AS TABLE

```

Tavola 3: Esempio di data mining query DMQL

Linguaggi logici

Il grosso vantaggio dei linguaggi di querying logici è la loro estrema espressività e flessibilità. L'approccio in [34] propone di specificare le richieste di mining sottoforma di clausole logiche. Per esempio, il problema di estrarre itemsets frequenti di dimensione 2 può essere specificato come in tav.4:

```

pair(I1,I2,count<T>) ← basket (T,I1), basket(T,I2), I1<I2
rules(I1,I2) ← pair(I1,I2,C), C ≥ 2

```

Tavola 4: Esempio di query logica di mining

Gli autori mettono in evidenza la “naturalzza” di queste espressioni e di come debbano essere poi ridefinite con delle strategie di esecuzione adeguate, come ad esempio usando le proprietà di *Apriori* [19],

if count(S,D) > T then for each S' ⊆ S, count(S',D) > T

(dove *cont(S,D)* è il numero di occorrenze dell'itemset S nel database D), così il programma può essere trasformato in uno equivalente ma molto più efficiente:

```

temp(I,count<T>) ← basket(T,I).
filter(I) ← temp(I,C), C ≥ 2.
pair(I1,I2,count<T>) ← filter(I1), filter(I2), basket(T,I1),basket(T,I2).
rules(I1,I2) ← pair(I1,I2,C), C ≥ 2.

```

Tavola 5: Ottimizzazione della query logica di mining precedente

Con questo sistema ogni query di mining deve essere ridefinita e analizzata per ottimizzare la ricerca rendendo difficile l'applicazione in un sistema interattivo con l'utente.

La prima proposta per utilizzare il sistema *deduttivo* in un linguaggio di esplorazione è in [31,32]. L'idea principale è quella di descrivere regole generali per poi cercare di istanziarle nei dati reali. Intuitivamente una meta-regola

$$P_1(X_1) \wedge P_2(X_2) \wedge \dots \wedge P_n(X_n) \Rightarrow Q_1(Y_1) \wedge Q_2(Y_2) \wedge \dots \wedge Q_m(Y_m)$$

dove P_i, Q_j sono predicati di variabili e X_i, Y_j sono (tuple di) termini di variabili, rappresentano un concetto ad altro livello su cui investigare.

I maggiori vantaggi di un approccio di questo genere sono:

- I dati di origine sono rappresentati come predicati del primo ordine sia *estensionali* o *intensionali*.
- L'approccio permette una espressività sia unidimensionale che multidimensionale anche se ha una forte rigidità nella struttura (per esempio è difficile esprimere regole con lunghezza arbitraria)
- E' facile estendere il linguaggio a strutture più complesse (per esempio gerarchie).
- E' facile introdurre vincoli specificando meta-query all'interno del processo
- Le regole estratte sono nella forma richiesta ma non c'è il concetto di chiusura nei risultati che devono essere elaborati successivamente con procedimenti ad-hoc per essere reintegrati nel sistema.

Di solito l'approccio logico è utilizzato come un motore computazionale per risolvere parti del processo di mining proprio per la difficoltà di reinserire i risultati all'interno del processo.

3. SIMPLE PATTERN QUERYING LANGUAGE

ConQueSt è un sistema di inductive database che cerca di ridurre al minimo il gap tra database e strumenti di mining (modello relazionale e modello transazionale) ispirandosi all'idea di [35] utilizzando un nuovo linguaggio chiamato SPQL (simple pattern query language) di tipo relazionale. La struttura generale di una query può essere così rappresentata:

```
[HEADER]
[SQL QUERY]
[MINING VIEW DEFINITION]
[CONSTRAINTS]
```

Tavola 6: Struttura generale di una query spql di data mining

Dove:

- la *SQL QUERY* è la sorgente dei dati;
- la *MINING VIEW DEFINITION* definisce i parametri per la creazione della rappresentazione transazionale a partire dai dati sorgenti, che a sua volta si suddividerà quindi nella definizione di transazione, item e attributo;
- l'*HEADER* e le *CONSTRAINTS* sono i parametri che il motore di mining utilizzerà per processare i dati in forma transazionale.

Un esempio di query SPQL di mining può essere:

```
MINE PATTERNS WITH SUPP>= 130 IN
SELECT articoli.cod_art, articoli.descrizione, articoli.prezzo
FROM articoli
TRANSACTION articoli.cod_art
ITEM articoli.descrizione
ATTRIBUTE articoli.prezzo
CONSTRAINED BY Sum(articoli.prezzo)>=100
```

Tavola 7: Esempio di una query spql di data mining

Lo SPQL è un'estensione dello SQL, infatti nel caso in cui non esistano header, mining view definition e constraints (ovvero non viene richiesto un processo di mining), i due linguaggi sono equivalenti.

3.1 Il linguaggio SPQL

La grammatica formale dello SPQL è descritta nella tav.8:

```

<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> SMOOTHINGH 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> ]

```

Il campo <SqlQuery> è definito come la grammatica SQL standard con il vincolo di avere nel campo SELECT i field contenuti in *Transaction*, *Item* e *Attribute* .

Tavola 8: Grammatica del linguaggio SPQL

Come descritto in [35] il linguaggio è fondamentale per un sistema di questo genere, ed è per questo che è stata creata una “estensione” dello SQL che garantisce la potenza espressiva di un linguaggio ben radicato e che abbia a disposizione un elevato numero di “funzioni base” ben ottimizzate. Inoltre la scelta di estendere il linguaggio più comune nel mondo dei database, permette un più semplice utilizzo del sistema da parte di utenti che operavano in precedenza sui dati immagazzinati.

3.2 Il processing di una query SPQL

Una query SPQL può essere di 3 tipi:

1. SQL standard
2. SPQL di discretizzazione
3. SPQL di mining

La prima tipologia è eseguita secondo le specifiche standard utilizzando il DBMS a cui ConQueSt è collegato mentre la seconda permette all'utente di manipolare i dati discretizzandoli specificando due parametri:

1. Metodi di discretizzazione
 - **EqualWidth:** partiziona lo spazio in sezioni di grandezza uguale
 - **EqualDepth:** partiziona lo spazio in sezioni di densità uguale, ovvero ogni sezione avrà circa lo stesso numero di "presenze"
 - **Free Partitioning:** permette all'utente di specificare liberamente un partizionamento dello spazio.
2. Metodo di smoothing (ovvero ciò che deve essere rappresentato per ogni intervallo discretizzato)
 - **Bin Boundaries:** come rappresentazione vengono utilizzati gli estremi dell'intervallo
 - **Average:** viene calcolata la media dei valori appartenenti all'intervallo
 - **Count:** viene utilizzato il numero di valori che sono finiti nell'intervallo (i valori che compaiono due volte vengono contati come 2 elementi separati)

Infine la terza tipologia permette all'utente di specificare i parametri e avviare l'algoritmo di data mining per la ricerca di itemset sui dati del DBMS, ovvero la funzione principale di ConQueSt.

Analizzeremo ora il processo che permette l'esecuzione di queste query, e nel resto del capitolo per query SPQL si intenderà proprio questa tipologia.

Utilizzando la suddivisione logica della tav.6 (*header, query SQL interna, mining view definition e Constraint*) vedremo come ognuna di queste parti sarà associata ad una diversa fase del processo di esecuzione (fig.7).



Figura 7: Le tre fasi del processo di ConQueSt

Estrazione dei dati dal database

L'estrazione dei dati avviene attraverso gli strumenti messi a disposizione del gestore del database che contiene i dati. In questo modo si possono sfruttare eventuali vantaggi e caratteristiche che i sistemi in circolazione mettono a disposizione sempre in accordo allo standard SQL.

Trasformazione dei dati da relazionali a transazionali

La fase "critica" di questo processo consiste nella trasformazione dei dati dal modello relazionale al modello transazionale, ovvero alla trasposizione che rende possibile l'utilizzo dei concetti del database relazionale ai presupposti necessari per l'applicazione di algoritmi di data mining.

Iniziamo definendo in modo più dettagliato cosa si intende per *Mining view definition* e successivamente analizzando il processo di trasformazione.

Definizione 3.1 (Mining view definition): Dato $R = \{r_1, \dots, r_n \mid r_p = \{c_{p1}, \dots, c_{pk}\} \ p=1..n, n>0, k>0\}$ il risultato di una query SQL su un *database relazionale* D_R dove c_{pj} sono i valori contenuti nella colonna j della riga p e dato l'insieme $Idc = \{idc_1, \dots, idc_k\}$ di identificatori delle colonne di R si dice *Mining view definition* :

$$\begin{aligned} Mvd = \{ & TransactionId, ItemId, AttributeId \mid \\ & TransactionId = \{t_1, \dots, t_\alpha\} \ 0 < \alpha < k, \\ & ItemId = \{i_1, \dots, i_\beta\} \ 0 < \beta < k, \\ & AttributeId = \{a_1, \dots, a_\gamma\} \ 0 \leq \gamma < k \} \end{aligned}$$

Tavola 9: Definizione di una Mining View Definition

dove i generici t, i, a sono identificatori delle colonne di R e vale $TransactionId \leq Idc, ItemId \leq Idc, AttributeId \leq Idc$ e $TransactionId \cap ItemId \cap AttributeId = \emptyset$.

Definizione 3.2 (Trasformazione From Relational to Transactional): data una *Mining view definition* Mvd e un database relazionale D_R indichiamo con ξ la funzione di *Trasformazione da relazionale a transazionale* tale che: $\xi(D_R) = D_T$.

La trasformazione ξ si occupa quindi di manipolare il risultato della query SQL relazionale e di trasformarlo nel formato definito in 2.3 attraverso le informazioni della *Mining view definition* come segue:

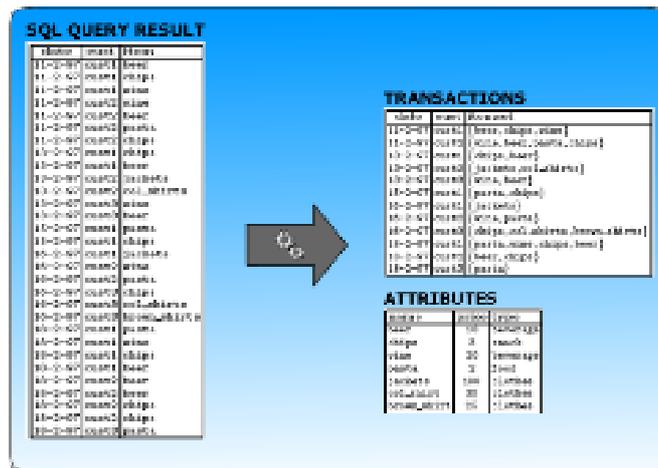


Figura 8: Esempio di trasformazione da dati relazionali a dati transazionali

La trasformazione viene completata eseguendo 4 fasi distinte:

- **Interpretazione SPQL query:** Inizialmente il motore di pre-processing interpreta e valida la query SPQL utilizzando la grammatica sopra descritta così da garantire la sua coerenza.
- **Generazione Items e Attributi:** esegue una interrogazione sul DBMS per ricavare tutti gli item e gli attributi utilizzando i campi definiti nella mining view nel campo ITEM e nel campo ATTRIBUTE, ottenendo la relazione ITEM-ATTRIBUTE. Come vedremo più avanti è possibile che alcuni ITEM abbiano più di un attributo, cosa che non può essere rispecchiata nel mondo transazionale, così questa fase prevede di aggregare i campi ATTRIBUTE secondo il campo ITEM in modo da avere un unico risultato calcolato su tutti i valori che esso può assumere (media, somma, count, ecc) oppure nel caso che l'utente voglia considerare solo il primo e l'ultimo valore dell'attributo nel processo successivo gli altri valori vengono ignorati. Tutte queste opzioni vengono previste a tempo di creazione della query SPQL e segnalate all'utente.

Oltre alla generazione

- **Mapping Items e Attributi:** dato che PATTERNIST, come la maggior parte degli algoritmi di mining, gestisce solamente valori numerici interi e non stringhe c'è una fase di *mapping* degli items e degli attributi. Questo processo fa sì che vengano generate altre due risorse: *item mapping* e *attribute mapping*. Queste risorse verranno poi utilizzate successivamente per interpretare i risultati che l'algoritmo di mining restituirà.
- **Generazione Transazioni:** Viene interrogato nuovamente il DBMS per ricavare tutti i valori dei campi definiti in TRANSACTION e ITEMS. Il risultato di questa query viene utilizzato per *costruire* le transazioni attraverso il concatenarsi dei valori dei campi ITEM per valori uguali della tupla formata dai campi TRANSACTION. Inoltre il valore

concatenato non è esattamente quello del campo ITEMS ma il suo valore intero riferito al mapping generato al passo precedente. In questo modo otterremo il dataset come un insieme di transazioni di valori interi generata secondo la definizione della mining view.

Tutte queste fasi verranno riprese in seguito per una analisi più dettagliata.

Esecuzione algoritmo di Mining

Il database transazionale è ora pronto per essere utilizzato dall'algoritmo di frequent pattern discovery che viene eseguito utilizzando i parametri dello Header e delle Constraints nella query iniziale. Il risultato di questa operazione può essere poi materializzata e reinserita nel database relazionale così da garantire il concetto di “chiusura” descritto in precedenza.

Algoritmi di Post-Processing

Anche se non specificato nella query SPQL, un ulteriore passo è quello di applicare algoritmi di post-processing sui dati così ottenuti. In ConQueSt è implementato il sistema di estrazione delle regole nella forma Body ->Head dove Body e Head sono sottoinsiemi degli itemsets risultanti dal mining. Anche in questo caso l'ulteriore raffinamento dei dati può essere materializzato nel database.

4. IL SISTEMA CONQUEST: GUIDA ALL'USO

4.1 Per iniziare

Requisiti di sistema

ConQueSt necessita dei seguenti requisiti:

Processore 300 MHz o superiore

256 MB RAM (consigliati 512 MB)

Windows XP

Java Runtime Environment Version 5.0 Update 5 o superiore

Inoltre, essendo un sistema di analisi e manipolazione di DBMS, necessita di una piattaforma qualsiasi tra quelle compatibili che contenga i dati.

Installazione e esecuzione

Estrarre il contenuto dell'archivio in una qualsiasi directory (es. C:\Programmi\Conquest\), assicurandosi che il path di java sia incluso tra quelli di sistema. In caso contrario procedere con le seguenti operazioni:

- Cliccare su "Start->Impostazioni->Pannello di Controllo"
- Doppio click su l'icona "Sistema"
- Selezionare in alto il tab "Avanzate"
- Cliccare in basso "Variabili d'ambiente"
- Nella lista inferiore selezionare "Path" e cliccare su "Modifica"
- Nel campo "Valore variabile" aggiungere il path degli eseguibili di Java (es. C:\Programmi\java\jdk1.5.0_05\bin) separandolo dai valori già esistenti con un ";"
- Cliccare su "OK" per 3 volte sulle varie finestre aperte e riavviare il computer.

Per eseguire ConQueSt doppio click su ConQueSt.bat

4.2 Connessione al database

Appena avviato Conquest appare come un piano di lavoro vuoto pronto all'uso. La prima cosa da fare è connettersi ad un database che contenga i dati da analizzare: per fare ciò selezionare "Preferences..." dal menu "Action" e inserire i dati per la connessione.

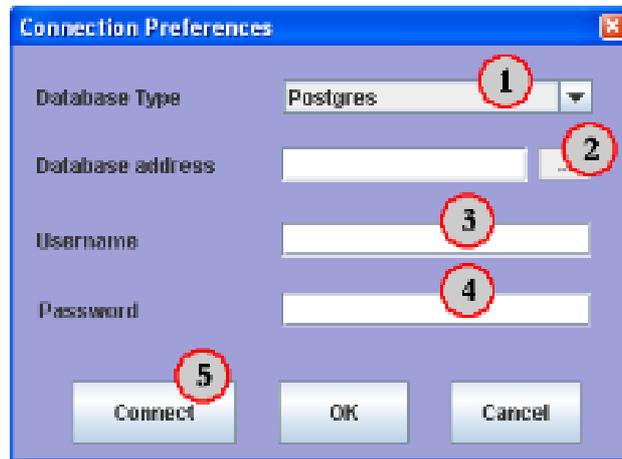


Figura 9: Finestra per le preferenze di connessione

Descrivendo la finestra di preferenze di connessione (Fig. 9) che viene visualizzata avremo:

1. **Database Type:** Selezioniamo il tipo di DBMS che vogliamo utilizzare, ConQueSt supporta molti tipi di database come: Postgres, MS Access, My SQL, Oracle, SQL Server, ecc.
2. **Database Address:** Inseriamo l'indirizzo del database che può essere un indirizzo per una connessione ad un database non locale oppure un file come nel caso di MS Access.
3. **Username:** il nome utente da utilizzare per l'accesso al DBMS
4. **Password:** la password da utilizzare per l'accesso al DBMS
5. **Connect, Ok, Cancel:** Cliccando su "Connect" si procede direttamente alla connessione al database, altrimenti si può cliccare su "OK" per poi connettersi con i dati inseriti cliccando "Connect" nel menu "Action". Con "Cancel" si annullano le modifiche ai dati di connessione.

Appena connesso ConQueSt comincia a estrarre informazioni dal database selezionato visualizzando nella Table List le tabelle esistenti, icona grigia, e quelle correntemente "processate", icona verde, ovvero le tabelle lette e di cui ora sono disponibili statistiche come il massimo, il minimo, la media, ecc... che potranno essere visualizzate in maniera rapida perché precalcolate durante questa fase.

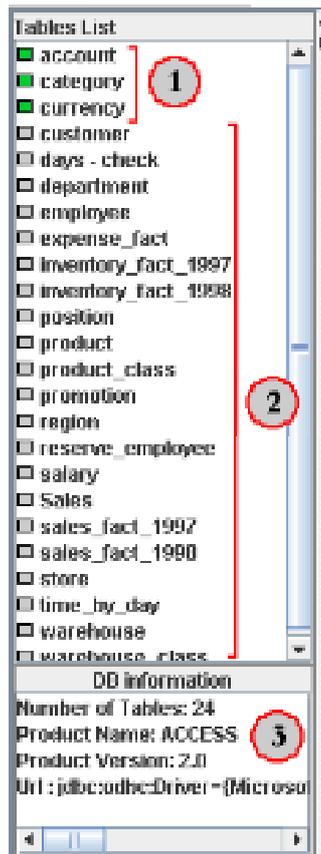


Figura 10: Pannello Table List durante la connessione ad un Database. 1- Tabelle pronte all'uso, 2- Tabelle non ancora processate, 3- informazioni del database correntemente in uso.

In fig.10 viene mostrato un esempio di caricamento di un database (FoodMart.mdb) dove vengono visualizzate oltre alla lista delle tabelle anche le informazioni del database a cui si è connessi.

In qualsiasi momento dal menu "Action" ci si può disconnettere dal Database o Ricaricarlo nel caso in cui si sono apportate modifiche al database durante l'esecuzione di ConQueSt con un software esterno.

Workbench

Al termine di questo processo tutte le tabelle saranno processate e pronte all'uso e quindi marcate con l'icona verde, nello stesso tempo le tabelle vengono posizionate in maniera casuale sul workbench in forma compatta (solo il nome della tabella) e vengono visualizzate le connessioni tra chiavi primarie e chiavi esterne evidenziandole con colorazioni differenti a seconda che siano uscenti (rosse) o entranti (blu) nella tabella selezionata. Nel caso di database che non abbiano sistemi standard compatibili con i driver JDBC di chiavi primarie e esterne, ConQueSt esegue un

algoritmo di creazione automatica dei collegamenti (similare al sistema usato da Microsoft Access basato sui nomi dei campi: a nome uguale corrisponde riferimento logico uguale).

Selezionando una tabella verranno evidenziati i collegamenti di quest'ultima con colori differenti a seconda della tipologia: blu se uscenti (la tabella corrente ha una chiave primaria che punta ad un'altra tabella) o rossi se entranti (la tabella corrente ha una chiave esterna puntata da un'altra tabella). Per visualizzare una tabella in forma estesa doppio click su di essa o sul nome nella "Table list".



Figura 11: Icone rappresentanti le proprietà dei campi delle tabelle

Nella forma estesa avremo l'elenco dei campi che questa tabella contiene, inoltre attraverso un sistema di icone, verranno evidenziate le proprietà dei campi come in fig.11.

Le tabelle possono essere spostate trascinandole in un qualsiasi punto del workbench con il pulsante sinistro del mouse, oppure nascoste selezionandole e utilizzando il tasto "delete" (non verranno cancellate dal database, ma solamente nascoste) per farle ricomparire basta fare doppio click sulla tabel list.

Analisi dei dati

Attraverso il processamento delle tabelle iniziale ConQueSt ha la possibilità di visualizzare un insieme di informazioni riguardanti tabelle e campi a loro associate:

In fig. 12 è raffigurato un esempio di esecuzione di ConQueSt collegato ad un database di tipo MS Access (FoodMart.mdb) dove vengono visualizzate solo 3 tabelle e selezionato il campo net_weight della tabella product. Descrivendo le 4 sezioni di visualizzazione dati di analisi:

- 1. Table Information:** tutte le informazioni riguardanti la tabella selezionata: il numero di righe della tabella e i vari campi con il tipo riconosciuto (Integer, Double o String)
- 2. Field Information:** le informazioni calcolate durante la connessione al database del campo selezionato all'interno della tabella: il nome, il massimo, il minimo, la media e i valori distinti, nel caso il campo selezionato è di tipo String le informazioni disponibili saranno solamente il nome e i valori distinti.
- 3. Distribution Graph:** Un grafico che descrive la distribuzione dei valori dell'attributo, può essere in forma di Istogramma (Bar) o grafico a Torta (Pie), automaticamente viene

selezionata la rappresentazione più “adeguata” (Auto), ma può essere anche scelta a discrezione dell’utente dall’apposito pannello.

4. **Table Visualization:** Viene visualizzato il contenuto della tabella selezionata, automaticamente la visualizzazione viene limitata a 5000 righe, ma questo limite può essere variato dall’utente a seconda delle proprie necessità. Su questa visualizzazione è possibile ridimensionare, spostare o ordinare colonne in maniera visuale (che quindi non avrà effetti reali sulle tabelle del database).

The screenshot displays the ConQueSt application interface. On the left, a 'Tables List' pane shows a tree view of database tables including 'account', 'category', 'currency', 'customer', 'days - check', 'department', 'employee', 'expense_fact', 'inventory_fact_1997', 'inventory_fact_1998', 'position', 'product', 'product_class', 'product_category', 'product_department', 'product_family', 'promotion', 'region', 'reserve_employee', 'salary', 'Sales', 'sales_fact_1997', 'sales_fact_1998', 'store', 'time_by_day', 'warehouse', and 'warehouse_class'. Below this is a 'DB information' section showing 24 tables, product name 'ACCESS', version '2.0', and a JDBC driver for Microsoft.

The central workspace shows three table diagrams: 'product_class' (with columns: product_class_id, product_subcategory, product_category, product_department, product_family), 'Sales' (with columns: customer_id, Product_id, product_class_id, time_id, store_id, unit_sales), and 'product' (with columns: product_class_id, product_id, brand_name, product_name, SKU, SRP, gross_weight, net_weight, recyclable_package, low_fat, units_per_case, cases_per_pallet, shelf_width, shelf_height, shelf_depth). Red lines indicate foreign key relationships between 'product_class_id' in 'product' and 'product_class_id' in both 'product_class' and 'Sales'.

The right sidebar contains several panels: 'Constraints' with an 'Add New' button; 'Mining view definition' with tabs for 'Transaction', 'Item', and 'Attribute'; 'Table Information' (1) showing 1560 rows and data types for 'product_class_id' (Integer), 'product_id' (Integer), 'brand_name' (String), 'product_name' (String), and 'SKU' (Double); 'Field Information' (2) for 'net_weight' showing a maximum of 20.8, minimum of 0.0, average of 11.79, and 332 distinct values; and 'Distribution Graph' (3) showing a histogram of 'net_weight' values with a legend for 'Auto', 'Bar', and 'Pie'.

At the bottom, the 'Table Visualization' tab (4) displays a data table with columns: 'product_id', 'product_name', 'SKU', 'SRP', 'gross_weight', 'net_weight', 'recyclable_package', 'low_fat', 'units_per_case', 'cases_per_pallet', 'shelf_width', 'shelf_height', 'shelf_depth'. The first few rows are: (1, 'Washi...', '9.074...', 2.85, 8.39, 6.39, 0, 0, 30, 14, 16.9, 12.6, 7.4), (2, 'Washi...', '9.651...', 0.74, 7.42, 4.42, 0, 1, 18, 8, 13.4, 3.71, 22.6), (3, 'Washi...', '5.842...', 0.83, 13.1, 11.1, 1, 1, 17, 13, 14.4, 11.0, 7.77), (4, 'Washi...', '6.441...', 3.64, 10.6, 9.6, 1, 0, 26, 10, 22.9, 18.9, 7.93), (5, 'Washi...', '8.556...', 2.19, 6.66, 4.65, 1, 0, 7, 10, 20.7, 21.9, 19.2), (6, 'Washi...', '2.980...', 1.15, 15.8, 13.8, 0, 0, 14, 10, 6.42, 18.1, 21.3).

A warning message at the bottom states: 'WARNING: Foreign keys not present in the database. Links among tables were created automatically by ConQueSt'.

Figura 12: Esempio di esecuzione di Conquest in fase di analisi dei dati

4.3 Strumenti

ConQueSt mette a disposizione una serie di strumenti per la manipolazione dei dati e l'estrazione di conoscenza da essi.

Lo strumento di base messo a disposizione è lo SPQL, un linguaggio che estende lo SQL standard, formalmente è definito come segue:

<pre> <SpqQuery> ::= (<SqlQuery> <MineQuery> <Discretize>) <MineQuery> ::= <Header>
 <SqlQuery>
 <MiningDefinition>
 <Constraints> <Header> ::= MINE PATTERNS WITH SUPP >= <Number> <MiningDefinition> ::= TRANSACTION <Transaction>
 ITEM <Item>
 [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> ::= ,
 ::= \n <Set> ::= <String> [<Separator> <Set>] <Discretize> ::= DISCRETIZE <Field> AS <Field>
 FROM <String>
 IN (<Method> <Intervals>) BINS
 SMOOTHING 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>] </pre>

Il campo <SqlQuery> è definito come la grammatica SQL standard con il vincolo di avere nel campo SELECT i field contenuti in *Transaction*, *Item* e *Attribute*.

Tavola 10: Grammatica del linguaggio SPQL

Con esso quindi possiamo specificare una qualsiasi query SQL standard e quindi manipolare i dati o interrogarli direttamente all'interno di ConQueSt (Attenzione, non è direttamente ConQueSt ad eseguire la query ma la piattaforma di DBMS a cui ci si è collegati, dunque eventuali "eccezioni" o

“estensioni” di quest’ultima avranno effetto sul linguaggio utilizzato da ConQueSt). Per eseguire una query SPQL bisogna selezionare il tab “SPQL Query”, digitare la query da eseguire e cliccare sul pulsante “Exec SPQL Query”, a questo punto la query verrà eseguita e verrà visualizzato il risultato o aggiornato lo stato del database (se la query è di modifica e non di interrogazione, es. “DROP TABLE”). In fig. 13 è riportato un esempio di esecuzione di un query SQL.

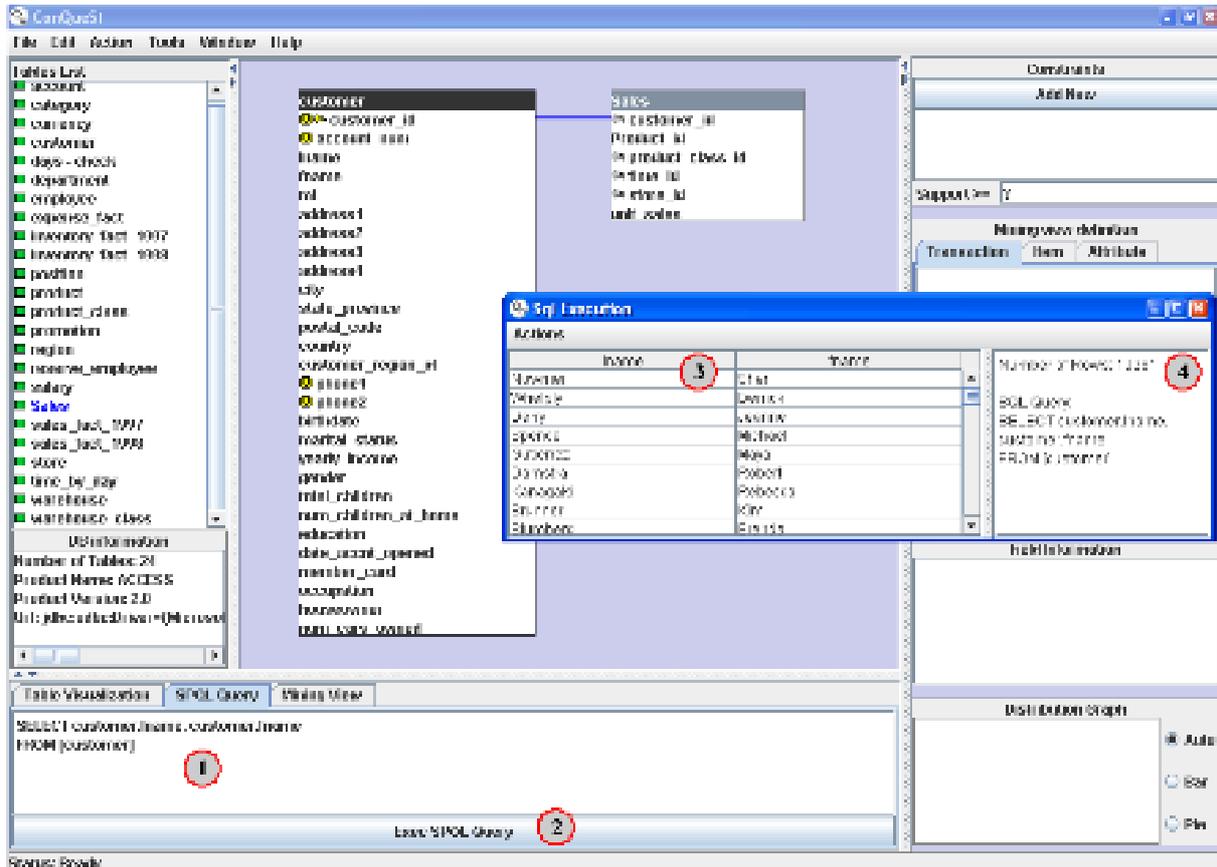


Figura 13: Esempio di esecuzione di una query SQL. 1- Tab per l’inserimento della query, 2- Pulsante di esecuzione, 3- Finestra di visualizzazione dei risultati, 4- metadati dei risultati.

Come detto in precedenza lo SPQL è una estensione dello SQL infatti, oltre alle interrogazioni standard, ci permette di estrarre conoscenza attraverso un algoritmo di Data Mining che vedremo in seguito.

Un altro strumento messo a disposizione è il discretizzatore, che ci permette di creare un campo aggiuntivo in una tabella discretizzandone un altro. Come mostrato in fig.14, per accedere a questa funzionalità basta selezionare la tabella e il campo da discretizzare e cliccare sul grafico di distribuzione (1), nel menu Action della finestra ingrandita del grafo, se il campo è di tipo numerico, ci sarà l’opzione “Discretize”(2).

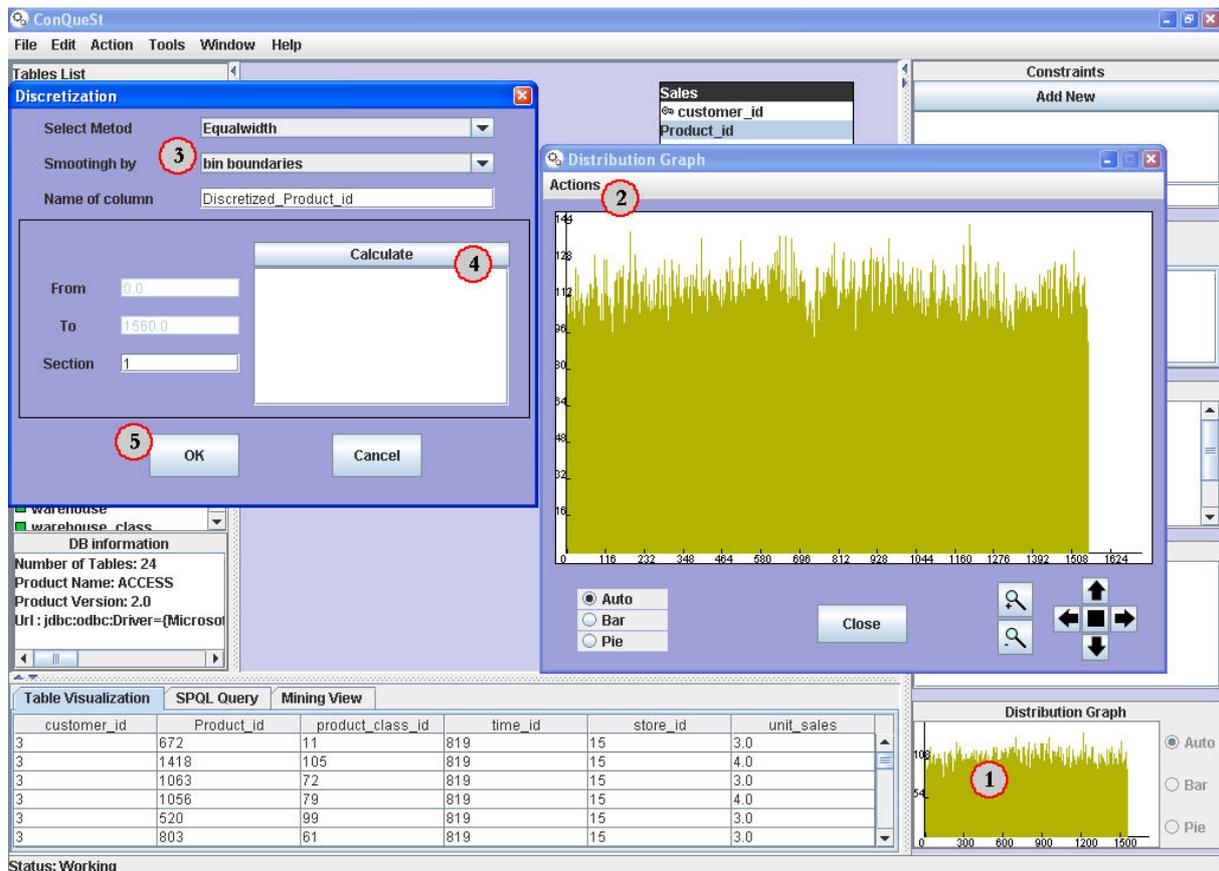


Figura 14: Finestre di discretizzazione.

Nella finestra di discretizzazione dobbiamo specificare il metodo di discretizzazione, il metodo di *smoothing*, e il nome della colonna da creare per inserire i dati discretizzati (3):

- *Metodi di discretizzazione*
 - **EqualWidth:** partiziona lo spazio in sezioni di grandezza uguale
 - **EqualDepth:** partiziona lo spazio in sezioni di densità uguale, ovvero ogni sezione avrà circa lo stesso numero di “presenze”.
 - **Free Partitioning:** permette all’utente di specificare liberamente un partizionamento dello spazio.
- *Metodo di smoothing* (ovvero ciò che deve essere rappresentato per ogni intervallo discretizzato)
 - **Bin Boundaries:** come rappresentazione vengono utilizzati gli estremi dell’intervallo
 - **Average:** viene calcolata la media dei valori appartenenti all’intervallo
 - **Count:** viene utilizzato il numero di valori che sono finiti nell’intervallo (i valori che compaiono due volte vengono contati come 2 elementi separati)

- *Nome della colonna*: il nome della colonna da aggiungere alla tabella selezionata dove i dati discretizzati verranno inseriti, di base viene indicato il nome originale del campo discretizzato preceduto dal tag “Discretized_”.

Una volta specificati i parametri, con il pulsante “Calculate/Add” (4) viene visualizzato sia graficamente che testualmente i limiti delle partizioni create. Per confermare la discretizzazione e dare inizio al processo basta cliccare su “OK” (5).

Il processo di discretizzazione non è solo visuale ma può essere specificato anche sotto forma di query seguendo la grammatica dell’SPQL:

```
DISCRETIZE amount AS discretized_amount
FROM expence_fact
IN 10 EQUALWIDTH BINS
SMOOTHING BY AVERAGE
```

Tavola 11: Esempio di query SPQL di discretizzazione

dove le parti evidenziate sono la struttura standard della query di discretizzazione e il resto i parametri del processo.

Estrazione conoscenza

Per “estrarre conoscenza” si intende estrarre informazioni basate sul concetto di itemset frequenti. Questi itemset sono definiti dall’utente attraverso l’SPQL in modo da essere estratti dal database e adattarsi alle esigenze dell’analisi.

Un esempio di query SPQL è il seguente:

```
MINE PATTERNS WITH SUPP>= 10 IN
SELECT vendite.cod_cliente, articoli.descrizione, articoli.prezzo
FROM vendite, articoli
WHERE vendite.cod_art = articoli.cod_art
TRANSACTION vendite.cod_cliente
ITEM articoli.descrizione
ATTRIBUTE articoli.prezzo
```

Tavola 12: Esempio di una query spql di data mining

Considerando di avere due tabelle dove sono presenti tutte le vendite dell’anno (vendite) e tutti gli articoli in vendita con i rispettivi dati (articoli) la query costruirà le transazioni in modo da

avere un insieme di transazioni rappresentanti ciò che ogni cliente ha comperato con il loro rispettivo prezzo; questo stato di rappresentazione è chiamato dataset.

Successivamente verranno processate dall'algorithmo di mining che estrarrà tutti gli itemset frequenti (secondo il parametro specificato nella query, in questo caso che siano presenti almeno 10 volte in tutto il dataset). L'insieme di campi specificati nelle categorie TRANSACTION, ITEM e ATTRIBUTE formano la Mining Definition.

La creazione della query può essere fatta manualmente come nell'esempio dello SQL precedentemente visto oppure generato in maniera visuale, che permette una rapida selezione dei campi desiderati. Per far ciò basta selezionare una tabella e un campo nel workbench (fig 15) e cliccare con il pulsante destro su di esso, apparirà un menù a tendina per specificare in quale categoria quel campo deve essere inserito (1) acquisendo una colorazione differente a seconda della categoria assegnata (2). Un riassunto dei campi selezionati e le loro rispettiva categorie è visualizzato nella parte destra all'interno della Mining view definition (3).

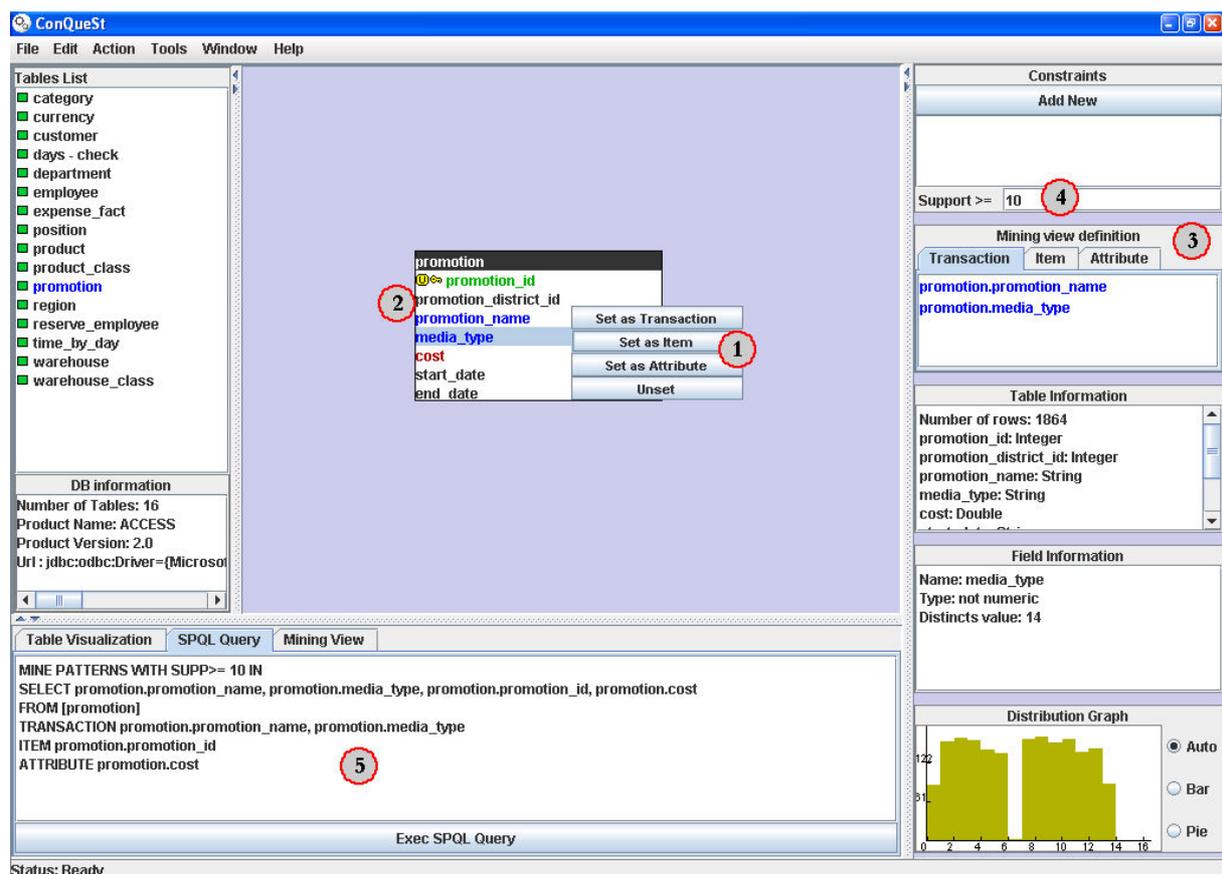


Figura 15: Costruzione visuale di una query SPQL

Come visto in precedenza bisogna specificare anche il parametro della frequenza, da inserire nel campo "Support >=" nella sezione "Constraints" che può essere specificato in forma assoluta,

specificando una soglia numerica, oppure in forma relativa, specificando una percentuale che si baserà sul numero totale di transazioni del dataset generato (4). Durante questo processo la query viene generata automaticamente in forma testuale nel tab “SQPL Query” pronta ad essere eseguita (5).

A fianco dei campi assegnati alla categoria `ATTRIBUTE` potrebbe apparire una icona come in fig.16, questo sta a significare che c'è la possibilità di un conflitto nella generazione degli attributi:



Figura 16: Icona possibile conflitto nella generazione degli attributi

durante il processo di generazione degli attributi per un valore di un `ITEM` potrebbe esistere più di un valore di attributo, questa eventualità deve essere risolta in modo da avere una corrispondenza univoca tra item e attributo, come default viene considerato solo il primo valore trovato, ma l'utente può selezionare uno dei seguenti metodi di risoluzione cliccando con il pulsante destro sul campo segnalato e selezionando la voce “Resolve”:

- Take first (default): conserva solo il primo valore trovato come attributo
- Take last: conserva solo l'ultimo valore trovato come attributo
- Take count: utilizza come attributo il numero di valori differenti trovati come attributo
- Take maximum: conserva il valore massimo trovato come attributo
- Take minimum: conserva il valore minimo trovato come attributo
- Take average: utilizza la media dei valori trovati come attributo
- Take sum: utilizza la somma dei valori trovati come attributo

E' importante sottolineare che in caso di attributi non numerici alcuni di questi metodi sono disabilitati.

Aggiungere vincoli

Il linguaggio SPQL permette di specificare anche vincoli sull'estrazione di conoscenza, utilizzando il costrutto `CONSTRAINED BY` seguito dal vincolo (o una congiunzione di vincoli) specificato sui campi assegnati alla categoria `ATTRIBUTE`, utilizzando l'esempio precedente avremo:

MINE PATTERNS WITH SUPP >= 10 IN

```
SELECT vendite.cod_cliente, articoli.descrizione, articoli.prezzo
FROM vendite, articoli
WHERE vendite.cod_art = articoli.cod_art
TRANSACTION vendite.cod_cliente
ITEM articoli.descrizione
ATTRIBUTE articoli.prezzo
CONSTRAINED BY Sum(articoli.prezzo) >= 100
```

Tavola 13: Esempio di una query sql di data mining con l'aggiunta di un vincolo

Nell'eseguire questa query si costruirà come in precedenza il dataset di transazioni ma durante l'esecuzione dell'algoritmo di data mining il vincolo verrà utilizzato per eliminare gli itemset che non lo soddisfano. Questa tipologia di estrazione di conoscenza viene chiamata “constrained frequent pattern discovery”.

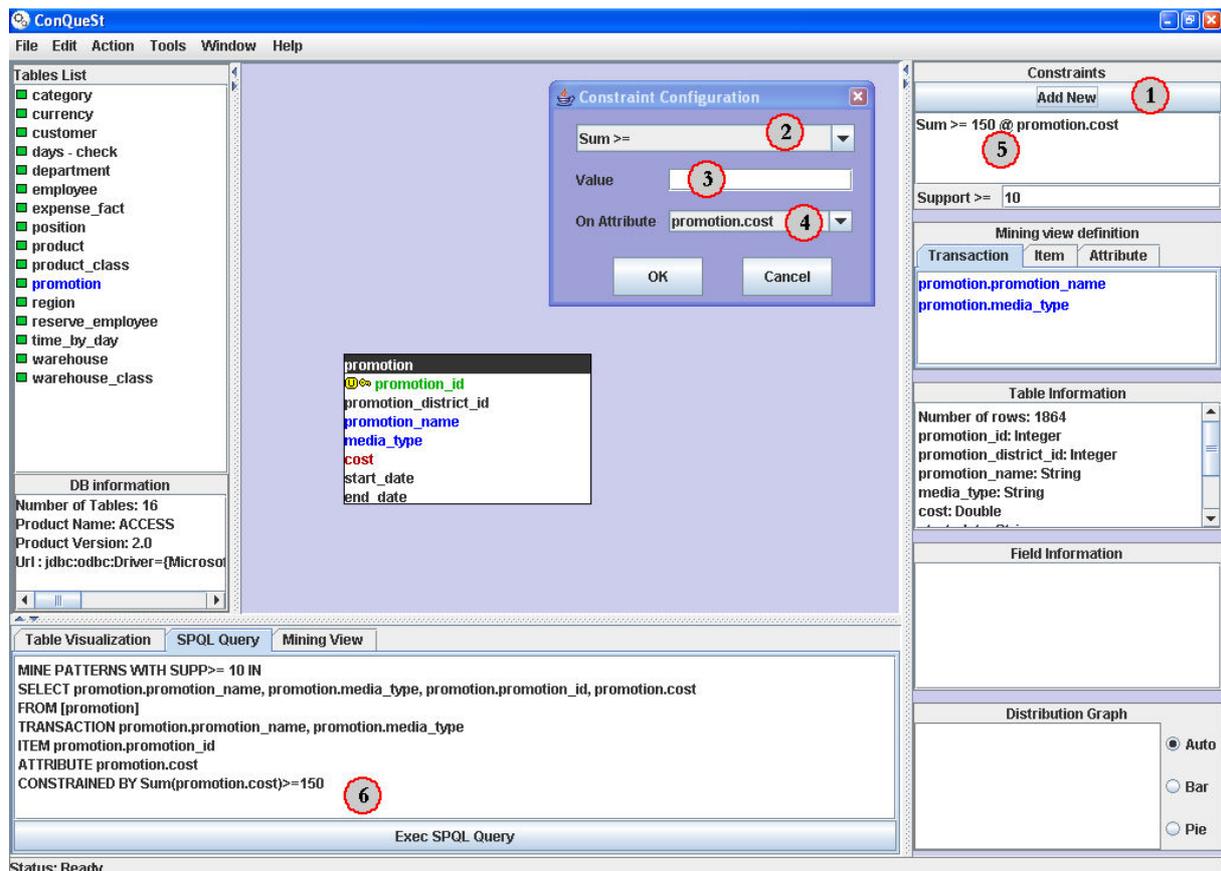


Figura 17: Aggiungere un vincolo ad una query SQL

Come il resto della query si può aggiungere il vincolo in maniera più semplice visualmente (fig.17) cliccando sul pulsante “Add new” nel pannello “Constraints” (1), si aprirà un finestra nella quale si devono specificare 3 parametri:

- **Tipo di vincolo** (2): Conquest include in se molte tipologie di vincoli:
 - *Minimum* \geq e *Minimum* \leq
 - *Maximun* \geq e *Maximun* \leq
 - *Range* \geq e *Range* \leq
 - *Variance* \geq e *Variance* \leq
 - *Std_Deviation* \geq e *Std_Deviation* \leq
 - *Median* \geq e *Median* \leq
 - *Average* \geq e *Average* \leq
 - *Sum* \leq e *Sum* \geq

Tutti queste tipologie hanno in comune il fatto di essere numeriche e che vengono applicate sui campi ATTRIBUTE, ne esistono di diverso genere:

- *Attribute_Subset_of* e *Attribute_Superset_of*: il vincolo non è di tipo numerico ma insiemistico.
 - *Subset_of* e *Superset_of*: il vincolo è sui valori dei campi ITEM e non su quelli ATTRIBUTE ed è di tipo insiemistico.
 - *Lenght* \geq e *Lenght* \leq : il vincolo è sulla lunghezza dell’itemset trovato.
- **Value** (3): il parametro numerico (o insiemistico) da abbinare al vincolo.
 - **On Attribute/On Item** (4): il campo sul quale il vincolo deve essere applicato.

Una volta confermato il vincolo viene aggiunto alla lista dei vincoli attualmente inseriti nella sezione “Constraints” (5) e nella query SPQL (6)

A differenza dei metodi di risoluzione conflitti sopra descritti le funzioni numeriche applicate sugli attributi non numerici possono essere applicate perché verrà utilizzato invece che il valore reale un valore sostitutivo basato sull’ordinamento dei valori.

Navigazione dei risultati

I risultati ottenuti attraverso l’esecuzione di query SPQL sono visualizzati all’interno di un “Pattern Browser” in due differenti modalità (i termini pattern e itemset sono equivalenti):

- Itemset View
- Item View

Di base viene visualizzato nella prima modalità così da permettere all’utente una visione rapida degli itemset estratti, infatti ogni riga della tabella a sinistra è formata da:

1. *Id*: l'identificatore del pattern, questi valori sono consecutivi da 1 al numero dei pattern trovati.
2. *Frequent Itemset*: i pattern veri e propri formati dagli item separati da una virgola.
3. *Support*: il supporto del pattern (numero di volte in cui il pattern è stato trovato all'interno del dataset)
4. *Constraint Functions Value*: il valore delle funzioni su cui è stato fatto il controllo per i vincoli specificati nella query SPQL.

La seconda modalità invece prevede la visualizzazione dei pattern trovati in modo da avere un item per riga ponendo che item appartenenti allo stesso pattern abbiano un id uguale, quindi avremo:

5. *Id*: l'identificatore del pattern, in questo caso possiamo definire un pattern come l'insieme di item con lo stesso id.
6. *Item*: un singolo item di un pattern trovato
7. *Attribute Value*: gli attributi dell'item specificati dalla query SPQL

In entrambe le modalità è presente un riepilogo dei risultati (8) contenente informazioni come: il numero di patterns trovati, la loro massima lunghezza, la minima lunghezza, la lunghezza media e la query che li ha generati.

Mining result: Pattern browser - Itemset view

id	Frequent Itemset	Support	Sum product
0	CDR Apple Preserves	43	20.0
1	Washington Berry Juice	54	6.38999...
2	Amigo Lox	70	18.2999...
3	Faux Products Silky Smooth Hair Conditioner	76	17.7000...
4	Tri-State Canned Peanuts	76	18.2000...
5	Tri-State Green Pepper	77	10.0
6	Pleasant Canned String Beans	78	6.5
7	Queen City Map	79	11.6000...
8	Plato Grape Jelly	79	7.30000...
9	National Small Eggs	79	19.2000...
10	Club Large Curd Cottage Cheese	80	12.3000...
11	High Top Plums	81	12.6000...
12	Colony Bagels	81	6.71999...
13	Cutting Edge Sliced Turkey	82	9.89000...
14	Sunset Scissors	82	15.1999...
15	Landslide Apple Preserves	83	7.34999...
16	PigTail Lime Popsicles	83	15.0
17	CDR Extra Chunky Peanut Butter	83	9.5

Number of Patterns: 348725
Maximum length: 5
Minimum length: 1
Average length: 2

SPQL Query:
MINE PATTERNS WITH SUPP>= 5 IN
SELECT product.product_name, product.gross_weight,
product.net_weight, sales_fact_1998.customer_id,
sales_fact_1998.store_id
FROM [product], [sales_fact_1998]
WHERE
sales_fact_1998.product_id=product.product_id
TRANSACTION sales_fact_1998.customer_id,
sales_fact_1998.store_id
ITEM product.product_name
ATTRIBUTE product.gross_weight, product.net_weight
CONSTRAINED BY Sum(product.net_weight)>=5

Mining result: Pattern browser - Item view

id	item	product.gross_weight	product.net_weight
0	CDR Apple Preserves	21.0	20.0
1	Washington Berry Juice	8.39	6.39
2	Amigo Lox	21.4	18.3
3	Faux Products Silky Smooth Hair Conditioner	19.7	17.7
4	Tri-State Canned Peanuts	21.2	18.2
5	Tri-State Green Pepper	12.0	10.0
6	Pleasant Canned String Beans	7.5	6.5
7	Queen City Map	13.6	11.6
8	Plato Grape Jelly	9.3	7.3
9	National Small Eggs	21.3	19.2
10	Club Large Curd Cottage Cheese	14.3	12.3
11	High Top Plums	14.7	12.6
12	Colony Bagels	8.72	6.72
13	Cutting Edge Sliced Turkey	12.9	9.89
14	Sunset Scissors	17.3	15.2
15	Landslide Apple Preserves	9.36	7.35
16	PigTail Lime Popsicles	17.0	15.0
17	CDR Extra Chunky Peanut Butter	11.5	9.5

Number of Patterns: 348725
Maximum length: 5
Minimum length: 1
Average length: 2

SPQL Query:
MINE PATTERNS WITH SUPP>= 5 IN
SELECT product.product_name, product.gross_weight,
product.net_weight, sales_fact_1998.customer_id,
sales_fact_1998.store_id
FROM [product], [sales_fact_1998]
WHERE
sales_fact_1998.product_id=product.product_id
TRANSACTION sales_fact_1998.customer_id,
sales_fact_1998.store_id
ITEM product.product_name
ATTRIBUTE product.gross_weight, product.net_weight
CONSTRAINED BY Sum(product.net_weight)>=5

Figura 18: Il pattern browser nelle 2 modalità: Itemset View e Item View

Per navigare i risultati in maniera più completa, il pattern browser mette a disposizione dell'utente vari tipi di ordinamenti sulle colonne:

- *Ascending*: ordina in maniera crescente secondo la colonna selezionata
- *Descending*: ordina in maniera decrescente secondo la colonna selezionata
- *Cardinality*: ordina in maniera decrescente secondo la colonna selezionata in base alla cardinalità del pattern (questo ordinamento è disponibile solo sulla colonna dei Frequent Itemset)
- *Cardinality * Support*: ordina in maniera decrescente secondo la colonna selezionata in base alla cardinalità moltiplicata per il supporto del pattern (questo ordinamento è disponibile solo sulla colonna dei Frequent Itemset)

Estrazione Regole

Oltre all'estrazione di conoscenza ConQueSt include un algoritmo di post-processing in grado di estrarre partendo dai risultati trovati dalla query SPQL una informazione ancora più raffinata: le regole associative nella forma “*Body -> Head*”.

Per eseguire questo ulteriore passo, basta selezionare “Extract Rules” dal menù “Actions” del pattern browser. Inizialmente sarà richiesto di inserire la minima confidenza delle regole che si desiderano visualizzare: la confidenza è calcolata come segue:

$$\frac{\alpha}{(\alpha + \beta)}$$

dove α è il numero di volte in cui la regola è “vera” all'interno dei patterns e β è il numero di volte in cui la regola è “falsa”. Il significato di vero e falso nei pattern corrisponde alla presenza del *Head* in un pattern in cui è presente l'*Body*.

Insieme alle regole estratte verranno visualizzati anche dati come il supporto delle regole e la confidenza (ovviamente superiore alla soglia specificata).

Id	Body	Head	Support	Confidence
0	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional	customer.education:Bachelors Degree	897	0.951336
1	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional, customer.marital_status:M	customer.education:Bachelors Degree	482	0.960159
2	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional, customer.marital_status:M, cust...	customer.education:Bachelors Degree	226	0.957627
3	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional, customer.marital_status:S	customer.education:Bachelors Degree	515	0.943223
4	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional, customer.marital_status:S, custo...	customer.education:Bachelors Degree	283	0.956081
5	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional, customer.gender:F	customer.education:Bachelors Degree	488	0.945736
6	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional, customer.gender:F, customer.m...	customer.education:Bachelors Degree	256	0.962406
7	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional, customer.gender:F, customer.m...	customer.education:Bachelors Degree	232	0.928
8	customer.yearly_income:\$50K-\$70K, customer.occupation:Professional, customer.gender:M	customer.education:Bachelors Degree	509	0.956767
9	customer.yearly_income:\$10K-\$30K	customer.education:Partial High School	2062	0.927993
10	customer.yearly_income:\$10K-\$30K, customer.marital_status:M	customer.education:Partial High School	1032	0.928993
11	customer.yearly_income:\$10K-\$30K, customer.marital_status:M, customer.gender:M	customer.education:Partial High School	513	0.929348
12	customer.yearly_income:\$10K-\$30K, customer.marital_status:S	customer.education:Partial High School	1030	0.927093
13	customer.yearly_income:\$10K-\$30K, customer.marital_status:S, customer.gender:M	customer.education:Partial High School	524	0.934046
14	customer.yearly_income:\$10K-\$30K, customer.gender:F	customer.education:Partial High School	1025	0.924256
15	customer.yearly_income:\$10K-\$30K, customer.gender:F, customer.marital_status:M	customer.education:Partial High School	519	0.928444
16	customer.yearly_income:\$10K-\$30K, customer.gender:F, customer.marital_status:S	customer.education:Partial High School	506	0.92

Figura 19: Visualizzazione delle regole estratte dal pattern browser

Materializzazione dei risultati

ConQueSt mette a disposizione un ulteriore strumento che permette all'utente di riutilizzare i risultati ottenuti per future osservazioni o per utilizzarli per analisi incrementali, questo grazie alla funzione di materializzazione dei dati sul database di origine. Attraverso questa funzione, accessibile cliccando su "Materialize" nel menu "Actions" del pattern browser o del visualizzatore di regole, ConQueSt trasforma i risultati in forma relazionale e li immagazzina nel database di origine dei dati sotto forma di tabelle, un esempio di questo processo in fig.20

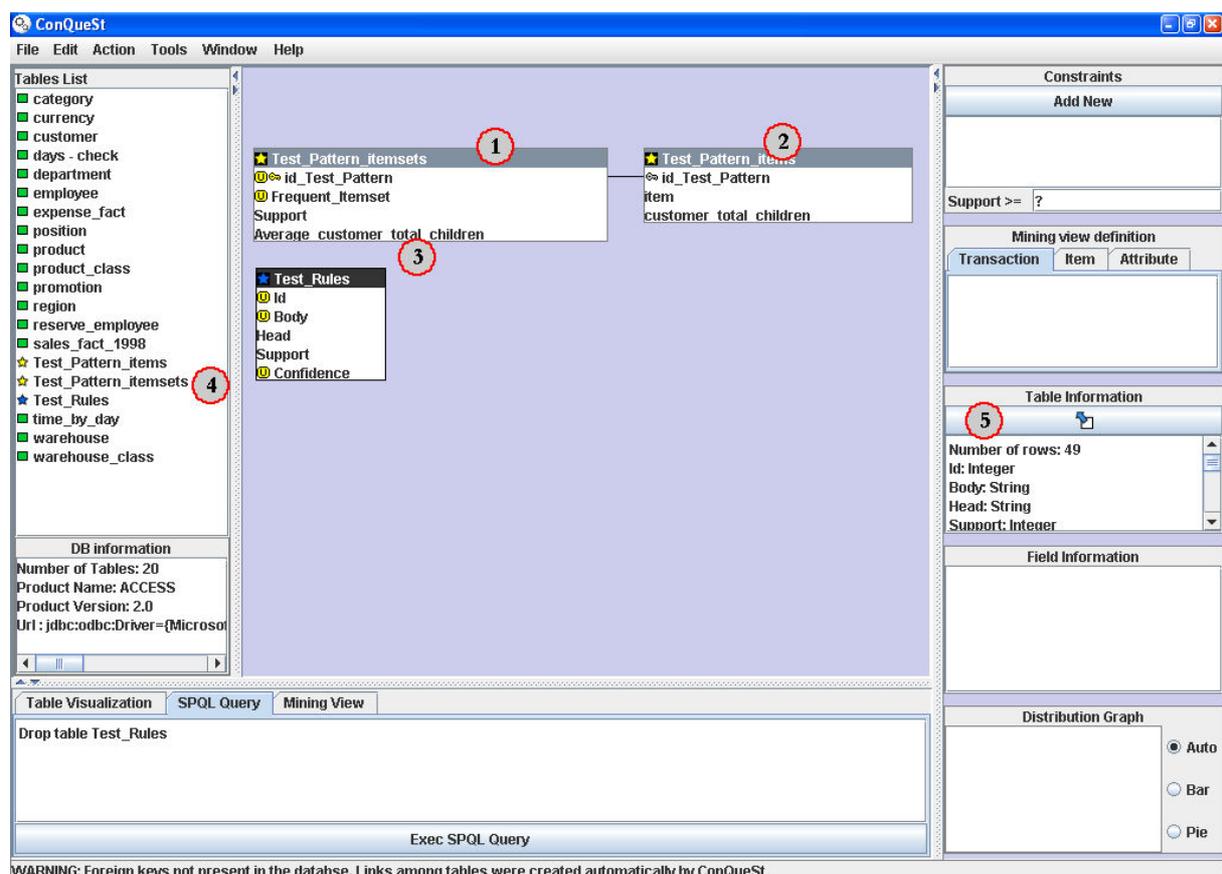


Figura 20: Esempio materializzazione Patterns e Regole

Come si può osservare i patterns vengono materializzati in due tabelle: la prima che rappresenta per esteso degli itemset (1), con i valori delle funzioni dei vincoli e il supporto, collegata ad una seconda tabella contenente gli item singoli (2) e il valore dei loro attributi. Le regole vengono materializzate in un'unica tabella (3). Queste tabelle sono riconosciute da ConQueSt come generate e quindi avranno informazioni e strumenti aggiuntivi alle solite tabelle:

- vengono segnalate in modo differente all'interno della table list attraverso l'icona a forma di stella, gialla per i pattern e blu per le regole.

- nella sezione “Table information” (5) verranno visualizzate anche informazioni riguardanti la loro generazione.
- È abilitato il pulsante di ritorno alla visualizzazione, (5) che permette di ritornare al pattern browser o il visualizzatore delle regole.

Queste tabelle sono a tutti gli effetti presenti nel database e come tali possono essere utilizzate per qualsiasi tipo di ricerca o manipolazione all'interno o all'esterno di ConQueSt.

5. ARCHITETTURA E IMPLEMENTAZIONE

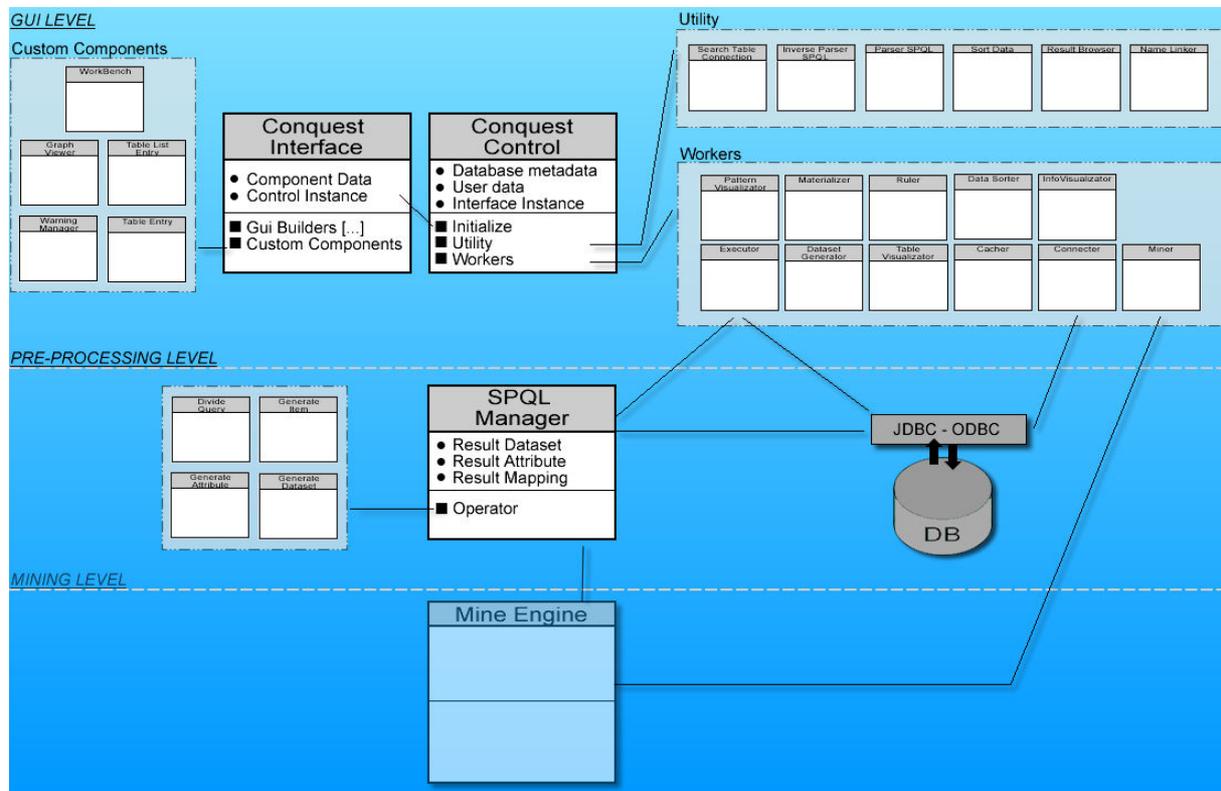


Figura 21: L'Architettura logico-implementativa di ConQueSt

5.1 Architettura

L'architettura di ConQueSt si sviluppa su 3 livelli:

- Livello Gui: si occupa di gestire l'interfaccia grafica, mandare in esecuzione i workers e utility per processare le richieste dell'utente.
- Livello Pre-processing: riceve le query SPQL, esegue un parsing di quest'ultima e le risolve
- Livello Mining: utilizza tutti i dati provenienti dal livello di pre-processing per eseguire il mining

I primi due livelli sono stati implementati in Java, mentre il livello di Mining in C++.

Livello Gui

Il Livello Gui è composto da 2 macrostrutture:

- **Interface:** l'interfaccia vera e propria, si occupa solamente di visualizzare i componenti in essa contenute e di gestire gli eventi legati a essi.
- **Control:** inizializza l'interfaccia e gestisce tutte le operazioni legate ad essa attraverso due tipi di operatori:
 - Componenti Workers
 - Componenti Utility

I Componenti Interface

Si occupano della istanziazione, della composizione e della gestione degli eventi legati a questi. La loro gerarchia può essere così riassunta (escludendo dalla rappresentazione tutti componenti minori):

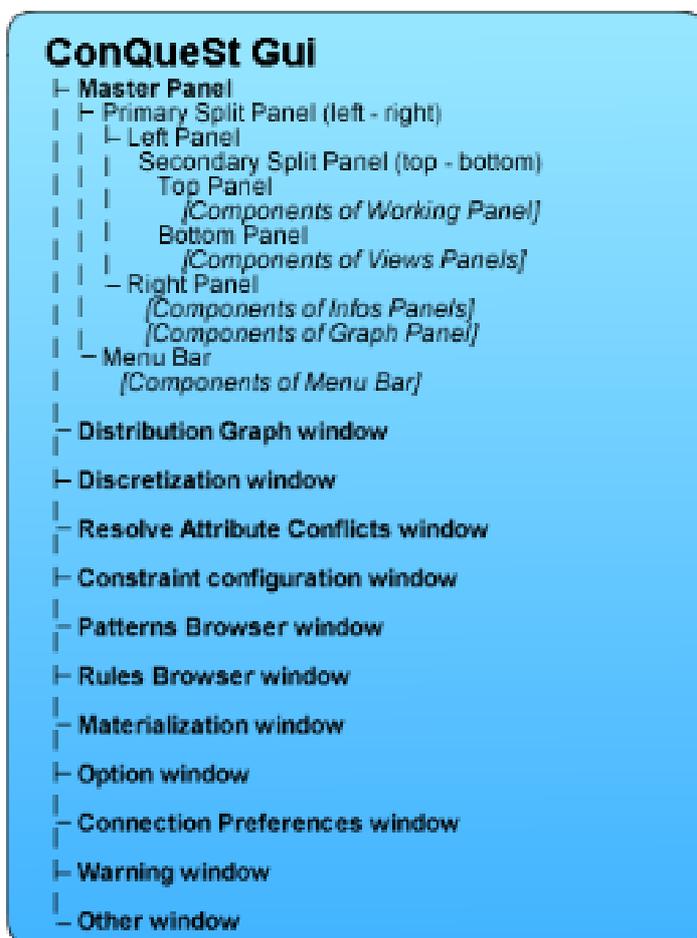


Figura 22: La gerarchia dei pannelli grafici di ConQueSt

- *Master Panel:* è il componente più grande all'interno di ConQueSt e si occupa di istanziare tutti i componenti della schermata iniziale, inoltre gestisce tutti gli indirizzamenti dei segnali ai vari componenti workers, utility o agli altri pannelli secondari.

- *Distribution Graph window*: gestisce la rappresentazione grafica dei dati ricavati dal cacher di ConQueSt e li visualizza in due modalità (attraverso la scelta automatica del sistema che considera la quantità e qualità dei dati oppure a scelta dell'utente): istogramma o torta. Inoltre gestisce lo zoom grafico.
- *Discretization window*: permette all'utente l'immissione di tutti i parametri per discretizzare il campo selezionato.
- *Resolve Attribute Conflict window*: è il componente che permette di impostare i parametri per risolvere gli eventuali conflitti degli attributi
- *Constraint Configuration window*: gestisce la selezione delle varie constraint e i parametri con cui applicarle sugli attributi.
- *Patterns Browser window*: questo componente gestisce la visualizzazione dei pattern trovati nel processo di mining.
- *Rules Browser window*: visualizza l'output del processo di estrazione di regole.
- *Materialization window*: una semplice maschera per l'immissione dei dati necessari alla materializzazione dei patterns o delle regole.
- *Options window*: è il componente che permette la selezione o la specifica delle opzioni di ConQueSt
- *Connection Preferences window*: questo componente si occupa di settare i parametri di connessione e la tipologia di database a cui connettersi.
- *Warning window*: è il componente che permette la visualizzazione di messaggi di errore.

La maggior parte di questa tipologia di componenti è quella fornita dalla libreria standard *javax.swing* ma nella *Interface* sono implementati anche cinque componenti nuovi utilizzati all'interno della struttura:

1. *WorkBench*: estende il componente standard *JDesktopPane*, ma aggiunge funzionalità per la visualizzazione di collegamenti tra tabelle.
2. *Graph Viewer*: estende il componente *JPanel* e permette la creazione e visualizzazione di Grafici partendo dai dati messi a disposizione dal *Control*, inoltre gestisce un sistema di caching grafico in modo da visualizzare velocemente un grafico già disegnato e non modificato.
3. *Warning Manager*: estende il componente *JLabel* e si occupa di gestire i messaggi di errore e segnalati dal *Control*.
4. *Table Entry*: estende il componente *JLabel* e permette la visualizzazione con icone multiple e colorazione differente delle caratteristiche dei campi delle tabelle.

5. *Table List Entry*: ha le stesse origini e funzionalità del componente *Table Entry* ma specifiche per la *Table List window* (componente all'interno del Master Panel).

I Componenti Workers

Come descritto sopra i componenti interface invocano i componenti workers e i componenti utility che eseguono richieste fatte dall'utente. La distinzione tra queste due classi di componenti è più implementativa che logica, infatti, mentre i componenti workers eseguono operazioni molto pesanti, le utility devono gestire piccole funzionalità a basso costo computazionale. I primi vengono invocati su thread separati da quello della costruzione e aggiornamento e quindi in maniera asincrona dall'interfaccia grafica, mentre le utility vengono eseguite in maniera sequenziale.

Proprio per la loro caratteristica i componenti workers estendono la classe standard Thread di Java, e possono essere divisi in 3 sottocategorie:

1. Visualizators:

- a. *Table Visualizator*: imposta e trasferisce i dati di una tabella, presente in cache, all'interno del pannello di visualizzazione della tabella (che si trova nel master panel sezione bottom del secondary split panel) e le informazioni generali sulla tabella correntemente selezionata nel pannello Table Information (che si trova nel master panel sezione right del primary split panel). Questo componente generalmente non avrebbe bisogno di essere un worker perché le informazioni sono tutte in cache e quindi pre-elaborate, ma in caso non lo fossero automaticamente quest'ultime vengono elaborate e messe in cache e quindi il tempo computazionale crescerebbe bloccando l'interfaccia grafica.
- b. *Info Visualizator*: imposta le informazioni del campo correntemente selezionato nel pannello Field Information (che si trova nel master panel sezione right del primary split panel). Come il precedente questo componente è stato implementato come worker per gestire l'eventuale mancanza di dati in cache in maniera automatica.
- c. *PatternVisualizator*: esegue operazioni di post-processing sui dati per la loro corretta visualizzazione nel pattern browser eseguendo il remapping degli item e eliminando informazioni superflue (che l'utente in ogni caso può riabilitare dal rispettivo menu del pattern browser)
- d. *DataSorter*: si occupa di eseguire gli ordinamenti richiesti dall'utente sui dati che possono essere di notevoli dimensioni.

2. Connecters:

- a. *Connector*: connette ConQueSt al database di cui sono stati inseriti i dati, ovvero locazione, tipologia, username e password e ne ricava la struttura in tabelle (istanziando vari Table, componente interface visto in precedenza) e i metadati generali relativi ad esse (chiavi primarie, chiave esterne)
- b. *Cacher*: questo componente carica in memoria il contenuto delle tabelle per un accesso rapido da parte di ConQueSt, inoltre calcola informazioni sui vari campi delle tabelle (come massimo, minimo, media, valori distinti, ecc..) determinando anche quali sono i campi univoci della tabella. Questo componente è stato progettato per poter generare una cache totale del database oppure solo parziale, una scelta di progetto ha poi fatto sì che il cacher esegua una elaborazione totale del database al momento della connessione ad esso così da avere uno stato di caricamento iniziale più lungo e un utilizzo più fluido durante l'esecuzione delle funzionalità di ConQueSt.

3. Executors:

- a. *Dataset Generator*: questo worker esegue materialmente la query corrente per estrarre una vista dello strato pre-processing. Può essere avviato in due modalità: preview o complete. Nella prima esegue il pre-processing con una finestra sul dataset limitata (settabile tra le opzioni di ConQueSt) e visualizza il contenuto nel mining view panel (che si trova nel master panel sezione bottom del secondary split panel), nella seconda invece esegue il pre-processing vero e proprio e genera i file del dataset.
- b. *Executor*: esegue le chiamate di puro SQL al database processando il risultato e visualizzandolo nello SQL browser.
- c. *Miner*: esegue le chiamate di SPQL, controllando se in cache esiste già materiale che può essere riutilizzato per evitare computazioni superflue (vedere sistema di caching), in caso contrario lancia un worker a sua volta (il Dataset Generator) per la generazione dei file del dataset, dopo di che costruisce una chiamata di sistema per il miner e lo fa eseguire facendo utilizzare i file appena generati.
- d. *Materializer*: ha il compito di materializzare sul DBMS le tabelle riguardanti il pattern browser e il rule browser e generare i collegamenti tra di esse.
- e. *Ruler*: si occupa della estrazione delle regole a partire dal risultato di un mining utilizzando il motore di regole.
- f. *Discretizator*: esegue la discretizzazione di un campo di una tabella secondo i parametri specificati attraverso l'interfaccia o la query SPQL.

I workers anche se eseguiti su thread separati operano su informazioni condivise a tutto il sistema, quali i dati di connessione al database, metadati generati sul database (transaction id, item id, attribute id, le informazioni dei vari campi, ecc.), i vari settaggi dell'utente e le tabelle in cache.

I Componenti Utility

L'ultima tipologia di componenti è quella delle utility, ovvero le funzionalità a basso costo computazionale integrate in ConQueSt:

- *Parser SPQL*: è l'utility che permette a ConQueSt di "interpretare" la query SPQL specificata dall'utente (suddividendola nei componenti visti in precedenza: intestazione, query SQL, estensioni SPQL, constraint) mantenendo la coerenza con l'interfaccia grafica. Questa utility viene eseguita ogni volta che la query subisce modifiche.
- *Inverse Parser SPQL*: questo componente è necessario per mantenere una corrispondenza coerente tra utilizzo dell'interfaccia e query SPQL, ovvero tutte le modifiche che l'utente fa a livello grafico si riflettono sulla query SPQL che si aggiorna.
- *Search Table Connection*: durante la creazione della query SPQL controlla se esiste una connessione, tramite chiavi primarie e chiavi esterne, tra le tabelle selezionate per la generazione del dataset, in caso positivo restituisce i vari vincoli da aggiungere nel campo "where" della query.
- *Result Browser*: questo componente non fa altro che leggere il risultato di una query SPQL dai file di output del mine engine e trasferirli sul SPQL browser Panel.
- *Sort Data*: utility che permette il riordinamento secondo vari metodi del risultato di una query SPQL all'interno del SPQL browser panel. I metodi sono: ascending, descending, cardinality e cardinality*support.
- *Name Linker*: questa utility permette di generare chiavi esterne basandosi euristicamente sui nomi dei campi delle tabelle, con l'ipotesi che nomi uguali corrispondono a riferimenti a dati uguali. Questa utility permette di rendere più organica una tabella con pochi metadati originali come vedremo in seguito.
- *Name Generator*: per la gestione della cache bisogna generare nomi differenti per memorizzare le ricerche già eseguite, questa utility ha il compito di generare partendo dai dati dell'operazione eseguita un nome univoco per il file che contenga i risultati. Il sistema è implementato utilizzando l'hashcode di un determinato insieme di fattori a seconda del tipo di ricerca.

Ci sono anche altre sotto-utility di poco conto che non sono state riportate. Anche i componenti utility come i workers utilizzano le strutture dati condivise del sistema per svolgere le loro mansioni.

IL Sistema di Caching

Il sistema di Caching permette all'utente una più rapida esplorazione dei dati e un riutilizzo di lavoro già svolto per altre ricerche simili. La prima parte è svolta durante la connessione di ConQueSt al DBMS, infatti caricando le tabelle e le loro proprietà vengono pre-calcolate tutte le informazioni che le riguardano come: minimo, massimo e media dei campi, grafico di distribuzione, ecc. La seconda parte del caching invece viene effettuata durante il processing di una query SPQL, infatti nel caso in cui sia stata già effettuata una ricerca uguale il sistema recupera direttamente i risultati risparmiando l'intero processo. Nel caso in cui venga processata una query SPQL che abbia la mining view definition uguale negli ITEM e TRANSACTION id ad una processata in precedenza ma con differenti ATTRIBUTE id allora viene riutilizzato gran parte del processo e si ricalcolano solamente gli attributi degli item.

Livello Pre-Processing

Come visto in precedenza, il dataset deve essere ricavato attraverso i parametri specificati in una query SPQL. Il processo avviene attraverso la concatenazione di "operatori": *Parser*, *Validator*, *Processor*, *Generator*.

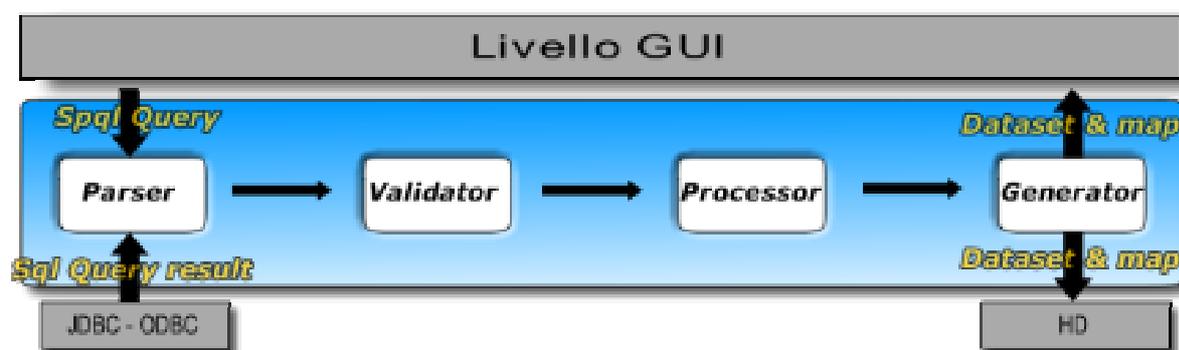


Figura 23: Le fasi del pre-processing di ConQueSt

Ricevuta la query dal livello Gui, la query viene passata al *Parser* che esegue un controllo di sintassi della query e la suddivide in sezione SQL e metadati per la generazione del dataset.

Successivamente il *Validator* data la query SPQL così definita:

```

MINE PATTERNS WITH SUPP >= σ IN
SELECT s1, s2, ..., sn
(1)
TRANSACTION t1, t2, ..., tm
ITEM i1, i2, ..., ik
ATTRIBUTE a1, a2, ..., ar
CONSTRAINT BY [2]

```

Tavola 14: Struttura di una query SPQL.

Con $(n \geq m+k+r)$ e dove (1) rappresenta una qualsiasi continuazione della query SQL e [2] un insieme qualsiasi di constraint, garantisce la coerenza delle informazioni verificando che valga:

$$\{t_1, t_2, \dots, t_n\} \cup \{i_1, i_2, \dots, i_k\} \cup \{a_1, a_2, \dots, a_r\} \leq \{s_1, s_2, \dots, s_n\}$$

ovvero che l'unione dei campi selezionati come transaction Id, item Id e attribute Id siano presenti nella SELECT principale della query SQL estratta dalla query SPQL. Il *Validator* ha anche il compito di mantenere in memoria la corrispondenza tra stringhe e posizioni all'interno della select dei campi transaction, item e attribute:

es.

```

MINE PATTERNS WITH SUPP >= σ IN
SELECT campoA campoB campoC campoD
(1)
TRANSACTION campoA campoD
ITEM campoB
ATTRIBUTE campoC

```

Tavola 15: Esempio di costruzione di una query SPQL.

Dove (1) è una qualsiasi continuazione della query SQL, in memoria viene mantenuto:

```

TRANSACTION = {0,3}
ITEM = {1}
ATTRIBUTE = {2}

```

Tavola 16: Posizioni dei vari elementi della mining view in relazione con la select

In questo modo conoscendo le colonne della query associate ai vari Id sarà successivamente più veloce la costruzione del dataset.

A questo punto il *Processor* per prima cosa genera gli item e gli eventuali attributi eseguendo sul DBMS query SQL generate dal *parser* (continuando ad utilizzare l'esempio precedente):

```
SELECT campoB campoC
(1)
```

Tavola 17: Esempio di una query SQL

Utilizzando il risultato per creare l'elenco degli item esistenti e gli attributi a loro associati. Vista la necessità successiva di avere solamente numeri interi e consecutivi per gli item, viene anche generato il mapping. Questo processo deve garantire l'univocità tra valore dell'item e numero intero che lo rappresenta: indicando con l'espressione $\downarrow I_k$ l'insieme degli item ricavati dall'itemid k processando la query avremo che dato:

$$\downarrow I_{k-1} = \{item1 \dots itemn\} \text{ e } \downarrow I_k = \{item1 \dots itemm\}$$

$$k = 1 \dots r \quad r = \text{numero itemid}$$

i valori di mapping saranno:

$$Map(\downarrow I_k) = \{n+1, \dots n+m\}$$

Inoltre il valore originale dell'item arricchito di un *marker* che permette la loro distinzione nel caso in cui 2 colonne del risultato della query abbiano lo stesso valore.

Esempio: Poniamo che esistano 2 item: Sex: M e Style: M Hanno lo stesso valore ma non sono lo stesso item, per cui verranno memorizzati come M(_1_) e M(_2_) dove 1 e 2 sono il numero delle colonne di origine.

Nel caso esistano due Item uguali con più valori come attributi di default si prende solamente il primo valore riscontrato, altrimenti nel caso di metodi differenti di risoluzione dei conflitti scelti dall'utente nell'interfaccia grafica la query utilizzata per ricavare item e attributi sarà nella forma:

```
SELECT campoB Method(campoC)
(1)
GROUP BY campoC
```

Tavola 18: Funzione di aggregazione aggiunta per la costruzione degli attributi

Dove Method è il metodo selezionato: avg, sum, min, max, count, ecc..

Nel caso in cui gli attributi non fossero numerici si applica un mapping anche su questa categoria.

Successivamente il *Processor* che, come spiegato nella teoria di ConQueSt, aggiunge un ordinamento alla query SQL contenuta nella query: considerando l'esempio precedente, avremmo che la query SQL processata sarebbe:

```
SELECT campoA campoB campoC CampoD
(1)
Order By campoA CampoD
```

Tavola 19: Ordinamento per la costruzione delle transazioni

In questo modo avremo le righe appartenenti alla stessa transazione raggruppate e ben distinte.

Il risultato viene passato al *Generator*.

Questa fase è il vero e proprio cuore del pre-processing level e ha lo scopo di ricavare tutte le informazioni necessarie per la creazione del dataset. Il processo si divide in 2 sottofasi che vengono ripetute per tutte le righe del risultato della query:

1. *Estrazione delle informazioni necessarie:* dalla riga correntemente presa in considerazione vengono estratte le informazioni necessarie per la creazione del dataset, ovvero:
 - o la “transactionKey”: l’unione dei campi marcati come transactionid che identificano la transazione corrente (ricordiamo che i risultati sono ordinati quindi ci sarà sempre e solo una transazione in costruzione e quando questa chiave cambia vuol dire che la precedente è conclusa)
 - o l’insieme degli item, marcando la loro “origine” con il sistema dei marker di cui abbiamo parlato in precedenza
 - o mapping delle transazioni in modo che i valori originali degli item vengano sostituiti con il loro valore di mapping
2. *Creazione delle transazioni:* se la transazione precedente è completa viene salvata su disco e liberata la memoria usata in precedenza.

A conclusione di questa visita dei dati abbiamo su disco un file contenente il dataset in formato transazionale, oltre ai file riguardanti gli item, gli attributi e i loro rispettivi mapping:

1. *[datasetName].att* : gli attributi del dataset
2. *[datasetName].amap* : il mapping degli attributi
3. *[datasetName].imap* : il mapping degli item
4. *[datasetName].desc* : la descrizione del dataset contenente dati come massima, minima, media della lunghezza delle transazioni, numero massimo transazioni, ecc. Tutti questi dati sono salvati in formato XML.
5. *[datasetName].fimi* : il dataset vero e proprio in formato fimi

Questi dati vengono anche mantenuti in memoria per una lettura rapida da parte della Gui per visualizzare informazioni e risultati del pre-processing (es. mining view).

Il nome del dataset è passato come parametro dal livello Gui, questo per permettere la gestione del sistema di caching.

Livello Mining

Per terminare l'esecuzione di una query SPQL che richiede l'utilizzo di un algoritmo di data mining, tutte le informazioni determinate dal livello di pre-processing vengono passate a PATTERNIST insieme ai parametri necessari (specificati nello *Header* e nelle *Constraint* della query).

Al termine dell'esecuzione di PATTERNIST, i patterns trovati verranno restituiti sotto forma di un file di testo:

```
... ..  
1 2 (121)  
1 2 3 (115)  
1 3 (189)  
6 8 (721)  
... ..
```

dove ogni riga contiene gli item appartenenti al pattern e tra parentesi il supporto, queste informazioni vengono manipolate successivamente da ConQueSt per eseguire un *reverse mapping* sui vari item per avere transazioni più leggibili da parte dell'utente.

5.2 Implementazione

Per prima cosa introduciamo le tecnologie usate nell'implementazione di ConQueSt, per poi scendere in dettaglio su alcune delle parti più importanti del codice sorgente.

Tecnologie

Il linguaggio utilizzato per il livello *Gui* e *Pre-Processing* è Java 1.5.0 Update 6, mentre per il livello *Mining* è C++.

Le librerie utilizzate sono state essenzialmente 2:

- javax.swing per la *Interface*
- JDBC per il *Control* e lo *SPQL Manager*

Le Javax.Swing sono un insieme di definizioni di componenti standard grafici che permettono il rapido utilizzo di funzionalità a loro connesse, inoltre attraverso l'overriding di alcuni metodi è stato possibile realizzare "componenti grafici ad hoc" per il progetto come il *WorkBench*, il *Graph Viewer* e i *Table Entry*. La gerarchia dei componenti è riportata in forma estesa al sito [LINK 3] .

Le JDBC permettono di realizzare strumenti di gestione e analisi indipendentemente dal tipo di database a cui ci si vuole connettere, questo in accordo con la filosofia di java "Write Once, Run Anywhere". La struttura generale delle JDBC e della loro interazione con i database è la seguente [47]:

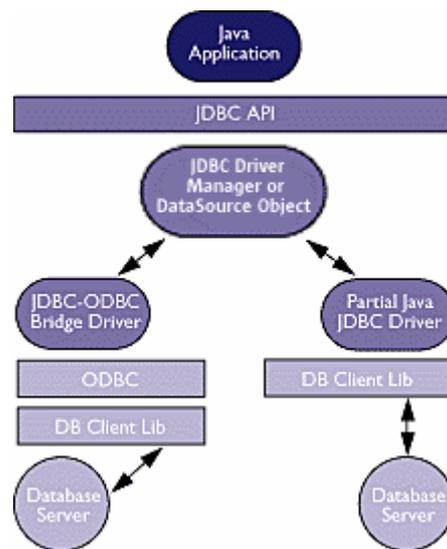


Figura 24: Schema generale delle interazioni tra applicazione, JDBC e database

Come si può osservare JDBC permette la connessione a tutti i database che hanno un client "JDBC compatibile" oppure utilizzando un "bridge" tra librerie con ODBC riuscire a connettersi anche a quei database che hanno solamente un client "ODBC compatibile".

In questo modo ConQueSt riesce ad abbracciare un insieme di database molto vasto come ad esempio: Postgres, Microsoft Access, MySQL, Oracle, Sql Server o qualsiasi altro database che abbia un client compatibile con la tecnologia sopra descritta.

Il motore di mining invece è stato implementato in C++ per rendere più veloce ed efficiente il livello di mining.

Codice Sorgente

Il codice sorgente la relativa documentazione (javadoc) è allegata in formato elettronico per ragioni di dimensioni, mentre le scelte implementative sono state già discusse all'interno della descrizione delle varie parti che costituiscono ConQueSt (worker, utility, interface, ecc..)

Alcune statistiche implementative riguardanti ConQueSt (escludendo il Miner che non appartiene a questa tesi) sono:

Linee di codice	9751
Classi globali (escluse classi interne)	5
Classi totali	34
Metodi	332~
Tempo impiegato (progettazione e sviluppo)	2000 o.u. ~

Tavola 20: Statistiche di implemetazione del sistema ConQueSt

6. CONCLUSIONI E SVILUPPI FUTURI

ConQueSt è stato sviluppato all'interno del KDD-Lad del CNR di Pisa collaborando a stretto contatto con Francesco Bonchi (per la teoria, la progettazione iniziale e in corso d'opera del sistema) e Claudio Lucchese (che ha implementato il motore di mining: PATTERNIST) con la supervisione di Fosca Giannotti, Raffaele Perego e Salvatore Orlando.

Per quanto riguarda il futuro implementativo di ConQueSt, si prevede l'estensione delle sue funzionalità come segue:

- **Miglioramento della materializzazione:** la materializzazione su DB dei risultati ottenuti può essere più efficiente se materializzata su 3 tabelle: ITEMS, ITEMSETS, LINK. Nella prima esistono solo gli item con i propri attributi con un item id, nella seconda gli itemset con le funzioni di aggregazione con un itemset id e infine la terza che realizza la connessione molti a molti tra le due tabelle.
- **Collegamento Regole-Itemsets:** ovvero la materializzazione delle regole dovrà legarsi alla tabella itemset sopra descritta (ancora da specificare esattamente come)
- **Ottimizzazione dell'inverse mining:** dopo aver materializzato un qualsiasi risultato si può tornare indietro alla visualizzazione dei dati che lo hanno generato, attualmente è realizzato con la riesecuzione della query (che se contenuta nella cache viene semplicemente ricaricata dalla memoria), sarebbe più efficiente recuperare i dati eseguendo delle query adeguate sulla tabella materializzata.
- **Cache persistente del DB:** una volta chiuso il programma tutte le metainformazioni calcolate all'avvio sono perse e ogni esecuzione devono essere ricalcolate, creare una cache persistente di questi dati darebbe più efficienza al sistema, fermo restando un controllo di coerenza all'avvio del sistema per eventuali modifiche al DB da fonti esterne.
- **Constraint con duplice sistema di applicazione:** patternist contiene al suo interno, per alcuni tipi di constraint, vari modi di applicarla durante il processo, questa possibilità non è ancora integrata completamente nel sistema
- **Specifiche di parametri aggiuntivi al processo di mining:** estendere il linguaggio SQL includendo specifiche di parametri aggiuntivi per il processo di mining (che PATTERNIST implementa) come la specifica del numero di passi di $\alpha - \mu$ reduction iniziali da compiere.
- **Funzionalità grafiche di ordinaria amministrazione:** implementare estensioni grafiche per poter eseguire velocemente query di amministrazione del DB come eliminare o creare una tabella o un campo, ecc..

- ***Aggiunta di algoritmi di post-processing:*** oltre all'estrazione di regole verranno implementati e/o integrati altri sistemi per la minopolazione dei risultati ottenuti dal mining: Clustering, Alberi di decisione, ecc..
- ***Ottimizzazioni varie:*** rendere sempre più efficiente il sistema operando su tutti i livelli di manipolazione e gestione dei dati.

7. BIBLIOGRAFIA

- [1] Agrawal, R., Imielinski, T., and Swami, A. N. Mining association rules between sets of items in large databases. In Proceedings ACM SIGMOD (1993).
- [2] Agrawal, R., and Srikant, R. Fast Algorithms for Mining Association Rules in Large Databases. In Proceedings of the 20th VLDB (1994).
- [3] Bonchi, F., Giannotti, F., Mazzanti, A., and Pedreschi, D. Adaptive Constraint Pushing in frequent pattern mining. In Proceedings of 7th PKDD (2003).
- [4] Bonchi, F., Giannotti, F., Mazzanti, A., and Pedreschi, D. ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In Proc. of ICDM (2003).
- [5] Bonchi, F., Giannotti, F., Mazzanti, A., and Pedreschi, D. Exante: Anticipated data reduction in constrained pattern mining. In Proceedings of the 7th PKDD (2003).
- [6] Bonchi, F., and Lucchese, C. Pushing tougher constraints in frequent pattern mining. In (ICDT05) (2005). submitted.
- [7] Choenni, S., and Siebes, A. P. J. M. Query Optimization to Support Data Mining. In Proc. of the Int'l. Workshop on Database and Expert Systems Application (September 1997).
- [8] Grahne, G., Lakshmanan, L., and Wang, X. Efficient mining of constrained correlated sets. In Proceedings of 16th ICDE (2000).
- [9] Han, J., Lakshmanan, L. V. S., and Ng, R. T. Constraint-based, multidimensional data mining. Computer 32, 8 (1999), 46-50.
- [10] Han, J., Pei, J., and Yin, Y. Mining frequent patterns without candidate generation. In Proceedings of ACM SIGMOD (2000).
- [11] Jeudy, B., and Boulicaut, J.-F. Optimization of association rule mining queries. Intelligent Data Analysis Journal 6(4) (2002), 341-357.
- [12] Lakshmanan, L. V. S., Ng, R. T., Han, J., and Pang, A. Optimization of constrained frequent set queries with 2-variable constraints. ACM SIGMOD Record (1999).
- [13] Li, W., Han, J., and Pei, J. CMAR: Accurate and efficient classification based on multiple class association rules. In Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM'01) (2001).
- [14] Liu, B., Hsu, W., and Ma, Y. Integrating classification and association rule mining. In 4th Int. Conf. Knowledge Discovery and Data Mining (KDD'98) (New York, 1998), pp. 80-86.
- [15] Han, J., Chee, S., and Chiand, J. Issues for On-Line Analytical Mining of Data Warehouses. In Proc. of 1998 SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98) (1998).

- [16] Ng, R. T., Lakshmanan, L. V. S., Han, J., and Pang, A. Exploratory mining and pruning optimizations of constrained associations rules. In Proc. of SIGMOD (1998).
- [17] Han, J., Fu, Y., Koperski, K., Wang, W., and Zaiane, O. DMQL: A Data Mining Query Language for Relational Databases. In SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96) (1996).
- [18] Han, J., and Kamber, M. Data Mining: Concepts and Techniques. Morgan Kaufman, 2000.
- [19] Han, J., Lakshmanan, L., and Ng, R. Constraint-Based, Multidimensional Data Mining. COMPUTER 32, 8 (1999), 46{50.
- [20] Orlando, S., Palmerini, P., and Perego, R. Enhancing the Apriori Algorithm for Frequent Set Counting. In Proc. of the 3rd Int. Conf. on Data Warehousing and Knowledge Discovery, LNCS 2114 (Germany,2001), pp. 71-82.
- [21] Orlando, S., Palmerini, P., Perego, R., and Silvestri, F. Adaptive and Resource-Aware Mining of Frequent Sets. In Proc. of the 2002 IEEE Int. Conference on Data Mining (ICDM'02) (Maebashi City,Japan, Dec. 2002), pp. 338-345.
- [22] Imielinski, T., and Virmani, A. MSQL: A Query Language for Database Mining. Data Mining and Knowledge Discovery 3, 4 (1999), 373{408.
- [23] Imielinski, T., Virmani, A., and Abdulghani, A. DMajor – Application Programming Interface for Database Mining. Data Mining and Knowledge Discovery 3, 4 (1999), 347{372.
- [24] Pei, J., Han, J., and Lakshmanan, L. V. S. Mining frequent item sets with convertible constraints. In (ICDE'01) (2001), pp. 433-442.
- [25] Pei, J., Zhang, X., Cho, M., Wang, H., and Yu., P. Maple: A fast algorithm for maximal pattern-based clustering. In Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03) (2003).
- [26] Srikant, R., Vu, Q., and Agrawal, R. Mining association rules with item constraints. In Proceedings ACM SIGKDD (1997).
- [27] Meo, R., Psaila, G., and Ceri, S. A new SQL-like operator for mining association rules. In VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases (Mumbai (Bombay), India, 3{6 Sept. 1996), T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds., Morgan Kaufmann, pp. 122{133.
- [28] Meo, R., Psaila, G., and Ceri, S. A Tightly-Coupled Architecture for Data Mining. In International Conference on Data Engineering (ICDE98) (1998), pp. 316{323.
- [29] Yiu, M. L., and Mamoulis, N. Frequent-pattern based iterative projected clustering. In Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03) (2003).

- [30] Ng, R., Lakshmanan, L., Han, J., and Pang, A. Exploratory Mining and Pruning Optimizations of Constrained Associations Rules. In Proc. ACM Conf. on Management of Data (Sigmod98) (June 1998).
- [31] Shen, W., and Leng, B. A Metapattern-Based Discovery Loop for Integrated Data Mining - Unsupervised Learning of Relational Patterns. IEEE Trans. on Knowledge and Data Engineering 8, 6 (1996), 898{910.
- [32] Shen, W., Ong, K., Mitbender, B., and Zaniolo, C. Metaqueries for Data Mining. In Advances in Knowledge Discovery and Data Mining. AAAI Press/The MIT Press, 1996, pp. 375{398.
- [33] Siebes, A. P. J. M., and Kersten, M. L. Keso: Minimizing Database Interaction. In Proc. of the Int.'l Conf. on Knowledge Discovery and Data Mining (August 1997).
- [34] Tsur, D., et al. Query Flocks: A Generalization of Association-Rule Mining. In Proc. ACM Conf. on Management of Data (Sigmod98) (1998), pp. 1 {12.
- [35] T. Imielinski and H. Mannila, A Database Perspective on Knowledge Discovery. Communications of the ACM Vol. 39, No. 11 (Nov. 1996), Pages 58-64
- [36] Boulicaut, J., Klemettinen, M., and Mannila, H. Querying Inductive Databases: A Case Study on the MINE RULE Operator. In Proc. 2nd European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD98) (1998), vol. 1510 of Lecture Notes in Computer Science, pp. 194-202.
- [37] Mannila, H. Inductive databases and condensed representations for data mining. In International Logic Programming Symposium (1997), pp. 21 {30.
- [38] Mannila, H., and Toivonen, H. Levelwise Search and Border of Theories in Knowledge Discovery. Data Mining and Knowledge Discovery 3 (1997), 241-258.
- [39] F. Bonchi, F. Giannotti and D. Pedreschi. A relational Query Primitive For Constraint-based Pattern Mining
- [40] R. T. Ng, L. V. S. Lakshmanan, J Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In Proceedings of the ACM sigmod'98.
- [41] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In Proceedings of VLDB'94.
- [42] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAnte: Anticipated data reduction in constrained pattern mining. In Proceedings of PKDD'03.
- [43] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Efficient Breadth-first Mining of Frequent Pattern with Monotone Constraints. To appear in Knowledge and Information Systems - An International Journal (KAIS). Springer, Berlin.
- [44] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAMiner: Optimized

level-wise frequent pattern mining with monotone constraints. In Proceedings of ICDM'03.

[45] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In Proceedings of ACM SIGKDD'00.

[46] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In (Proceedings of ICDE'01).

[47] Java Sun, <http://java.sun.com/products/jdbc/overview.html>

[48] Gerarchie javax.swing: <http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/package-tree.html> .

[49] Black Forest Workshops, Talk of Francesco Bonchi, Introducing ConQueSt: a Constraint-based Querying System for Exploratory Pattern Discovery, and Showing its Application on a Biomedical Problem.

[50] Francesco Bonchi, Fosca Giannotti, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, Roberto Trasarti: ConQueSt: a Constraint-based Querying System for Exploratory Pattern Discovery. ICDE 2006: 159

[51] Francesco Bonchi, F. Giannotti, Lucchese Claudio, S. Orlando, R. Perego, R. Trasarti: On Interactive Pattern Mining from Relational Databases.