INDICE

INTRODUZIONE 3

CAPITOLO 1 5

1.1 OTTIMIZZAZIONE SEMANTICA DI QUERY 9
1.1.1 APPRENDIMENTO DELLE REGOLE DI TRASFORMAZIONE PER L'OTTIMIZZAZIONE SEMANTICA DELLE QUERY 14
1.1.2 VINCOLI DI INTEGRITÀ E REGOLE DI TRASFORMAZIONE DELLE QUERY 15
1.1.3 REGOLE DI TRASFORMAZIONE DEFINITE DALL'UTENTE O SCOPERTE 17
1.2 DEFINIZIONE DELLE CARATTERISTICHE DELLE REGOLE 18
1.3 ALCUNI APPROCCI AL PROBLEMA DEL SQO 20
1.3.1 ACQUISIZIONE DI CONOSCENZA NEL DATABASE PER L'OTTIMIZZAZIONE SEMANTICA .. 20
1.3.2 TECNICHE DI APPRENDIMENTO E DI SCOPERTA NELL'AI 21
1.3.3 ACQUISIZIONE DELLE REGOLE DI TRASFORMAZIONE BASATO SU UN ALGORITMO DI GENERAZIONE DI QUERY EQUIVALENTI 23
1.3.4 APPRENDIMENTO DI REGOLE PER SQO BASATO SULLA DISTRIBUZIONE DEI DATI 25
1.4 DEFINIZIONI PER IL TRATTAMENTO DELLE REGOLE 29

CAPITOLO 2 33

2.1 ACQUISIZIONE AUTOMATICA DELLE REGOLE SEMANTICHE PER L'OTTIMIZZAZIONE SEMANTICA 33
2.2 DEFINIZIONE DELLE EURISTICHE 41
2.2.1 SGUARDO D'INSIEME 41
2.2.2 IL LINGUAGGIO DELLE QUERY 43
2.2.3 IL LINGUAGGIO DELLE REGOLE 46
2.2.4 EURISTICA H1 48
2.2.5 EURISTICA H2 51
2.2.6 EURISTICA H3 53
2.2.7 EURISTICA H4 56
2.2.8 EURISTICA H5 60
2.3 PROCESSO PER LA CREAZIONE DELLE REGOLE PROPOSTE 63
2.3.1 PROCESSO PER LA CREAZIONE DELLE REGOLE PROPOSTE 65
2.3.2 ALGORITMO PER IL CALCOLO DELLA CHIUSURA DI UN INSIEME DI RESTRIZIONI 66
2.3.3 GENERAZIONE DELLE REGOLE PROPOSTE 78
2.4 VALUTAZIONE E SCELTA DELLE REGOLE PROPOSTE PER IL PROCESSO DI DERIVAZIONE 83
2.5 PROBLEMA DELL'UTILITÀ E UTILITÀ DI UNA REGOLA 83

2.6 RULE CLASS 86
2.7 PROCESSO DI VALUTAZIONE DELLE REGOLE PROPOSTE 89
2.8 DERIVAZIONE DELLE REGOLE PROPOSTE 99

CAPITOLO 3 106

3.1 GENERAZIONE REGOLE CON UN APPROCCIO DATA-DRIVEN 106
3.2 GENERALITÀ SUL SISTEMA DI DISTRIBUZIONE DEI DATI GRID-BASED 108
3.3 GENERAZIONE GRIGLIA 115
3.3.1 GENERAZIONE DI UNA GRIGLIA A 2 DIMENSIONI 117
3.3.2 GENERAZIONE DI UNA GRIGLIA A 3 DIMENSIONI 122
3.3.3 COSTO DI COSTRUZIONE DELLA GRIGLIA 126

CAPITOLO 4 129

4.1 MANUTENZIONE DEL RULE SET 129
4.2 CONTROLLO E RIPRISTINO DELLA VALIDITÀ DELLE REGOLE DEL RULE SET 139
4.2.1 CONTROLLO VALIDITÀ DELLE REGOLE ATTRAVERSO UN ANALISI SINTATTICA DELLE REGOLE 141
4.2.2 CONSIDERAZIONI SUL CONTROLLO E RIPRISTINO DELLA VALIDITÀ DI REGOLE DERIVATE 142

CAPITOLO 5 158

5.1 CONCLUSIONI E LAVORI FUTURI 158
5.1.1 RICAPITOLAZIONE DEGLI ARGOMENTI TRATTATI 159
5.1.2 CONCLUSIONI E LAVORI FUTURI 161

BIBLIOGRAFIA 163

INTODUZIONE

L'ottimizzazione semantica di interrogazioni può ridurre significativamente il tempo di risposta del database attraverso la riformulazione delle interrogazioni. Il successo nel ridurre il costo di esecuzione di un interrogazione è fortemente condizionato dai metodi di apprendimento e di manutenzione della conoscenza semantica per fornire al sistema un insieme utile di regole di trasformazione dell'interrogazione.

In questa tesi verranno discusse le problematiche di acquisizione e manutenzione di conoscenza nell'ambito dell'ottimizzazione semantica delle interrogazioni. Questi due aspetti sono fortemente correlate tra loro per quanto riguarda il processo di ottimizzazione.

In relazione all'acquisizione di conoscenza, verrà analizzato una possibile integrazione tra un approccio di tipo query-driven con un approccio di tipo data-driven in modo da compensare le lacune di una metodologia con i pregi dell'altra.

Un primo obbiettivo che si ci propone è quello di integrare i due approcci. Il procedimento che si utilizzerà, sarà quello di raccogliere informazioni attraverso un approccio query-driven, al fine di dedurre su quali relazioni è possibile effettuare una ricerca più efficace con un approccio data-driven.

Un secondo obbiettivo è quello di proporre un sistema per la generazione di regole con un approccio data-driven. In particolare verrà proposto un algoritmo per la scoperta di regole basato sull'analisi della distribuzione dei dati attraverso la costruzione di una griglia multidimensionale .

Per manutenzione delle regole scoperte si intende il processo di controllo e di ripristino della validità delle regole. Nella tesi, dopo l'analisi di alcuni approcci presentati in letteratura, verranno descritti alcuni metodi per la manutenzione delle regole come la riscrittura delle regole in caso di una loro violazione. Inoltre verrà considerato il problema del controllo della validità delle regole in seguito ad aggiornamenti del database. Infine si mostrerà un tipo di manutenzione attraverso la generazione di procedure.

Il continuo di questa dissertazione è organizzata nel seguente modo. Nel capitolo 2 viene descritto il processo di acquisizione delle regole attraverso un approccio query-driven. Tale approccio prevede l'utilizzo di euristiche [(1)King(1981)], da applicare alle interrogazioni generate dagli utenti, per estrarre la conoscenza semantica necessaria per l'ottimizzatore. Partendo dal sistema proposto da [(56) Siegel (1992)] si considererà una sua estensione per quanto riguarda l'applicazione delle euristiche e per la raccolta delle informazioni necessarie, per generare una ricerca delle regole attraverso un approccio data-driven. Il capitolo 3 presenta un metodo data-driven per la generazione delle regole. Esso si basa sul concetto di griglia vista come una regolare decomposizione dello spazio di tutte le possibili tuple di una data relazione in una collezione di celle disgiunte.

Nel capitolo 4 si parlerà della manutenzione delle regole scoperte, inteso come il processo per il controllo e il ripristino della validità delle regole. Nel capitolo verranno descritti alcuni metodi per la manutenzione delle regole come la riscrittura delle regole in caso di una loro violazione. Inoltre si considera il problema del controllo della validità delle regole in seguito ad aggiornamenti del database. Nel capitolo 5 verranno riassunti i principali temi trattati . Il capitolo si conclude con una descrizione di alcuni lavori futuri.

CAPITOLO 1

Introduzione

L'ottimizzazione di query è una componente molto importante di un sistema di gestione dei dati. Un buon ottimizzatore di query può automaticamente ricercare un efficiente procedura per il ritrovamento di dati. Esso ci permette di interrogare un sistema di informazioni senza aver bisogno di capire i suoi meccanismi interni. Comunque, è difficile per le tecniche di ottimizzazione di query convenzionali risolvere tutti i problemi per la prossima generazione dei sistemi di informazione. *L'ottimizzazione semantica di query* (SQO) [(2) Hammer and Zdonik (1980), (1) King (1981), (10) Siegel (1988), (6)Shekhar (1988), (45) Sun and Yu (1994)] è una promettente tecnica di ottimizzazione di interrogazioni che può complementare tecniche convenzionali per ridurre considerevolmente il costo di esecuzione. L'idea base dell'ottimizzazione semantica di query è di usare regole semantiche sui dati per riformulare query in una forma equivalente (stessa risposta per tutti gli stati della base di data) che può essere risolta in maniera più efficiente, soprattutto in termini di tempo di esecuzione. Ad esempio per un database di una biblioteca si può considerare la regola "*tutti i libri di poesia hanno un anno di pubblicazione antecedente al 1980"*. Data questa regola supponiamo di avere la seguente query :

Query : Trova tutti i libri di poesia con un anno di pubblicazione antecedente al 1950 e con una collocazione 'S-10'.

Il sistema può riformulare la query in una nuova query :

Query (ottimizzata) : trova tutti i libri di poesia con una collocazione 'S-10'.

Questa query ottimizzata è equivalente all'originale poiché per la regola data non c'è bisogno di controllare che l'anno di pubblicazione sia antecedente al 1950, cosicché la query ottimizzata avrà lo stesso risultato di quella iniziale. L'esecuzione della query ottimizzata è meno dispendiosa dell'esecuzione della query originale perché il sistema risparmia il tempo per confronti ridondanti. Questo è solo uno delle possibili trasformazioni che possiamo ottenere sfruttando un certo insieme di regole semantiche.

Sebbene l'ottimizzazione semantica di query sia una tecnica efficace e promettente, essa richiede una sufficiente conoscenza semantica per produrre un alta riduzione dei costi. Precedenti lavori nel SQO, come [(1) King (1981)], assumono che l'ottimizzatore possa usare vincoli di integrità semantici dati dagli utenti. I vincoli di integrità semantica esprimono regole che devono essere verificate dai dati in qualunque stato del database. Vincoli di integrità semantica sono utili per garantire il corretto uso di un database, ma non necessariamente rispecchiano una distribuzione dei dati che incidono sui costi di esecuzione di query. Uno dei motivi può essere che gli utenti già sono in possesso della conoscenza semantica riguardo il dominio di applicazione del database è quindi improbabile che essi generino una query dove i vincoli di integrità possono essere applicati nell'ottimizzazione della query.

In questa dissertazione si vuole proporre un approccio basato sull'utilizzo di euristiche all'apprendimento di regole di trasformazione che possono essere usate per stabilire dei procedimenti per l'individuazione di regole utili. Regole utili permettono all'ottimizzatore SQO di raggiungere alti riduzione dei costi nell'ottimizzazione di query. Regole inconsistenti, cioè che non rispettano più lo stato corrente del database, non sono utili per l'SQO poiché usando regole inconsistenti l'ottimizzatore può riformulare una query in una nuova query non equivalente all'originale producendo risultati incorretti nel sistema.

Un punto importante di questo approccio è la nozione di regola proposta e di trasformazione euristica. Le regole proposte ci dicono quali dovrebbero essere le regole di trasformazione utili all'SQO per ottimizzare una data query. Verrà introdotto

anche il concetto di *trasformazione euristica*. Le trasformazioni euristiche sono applicazioni che generano le regole proposte. I principi base su cui si basano le trasformazioni euristiche derivano dall'osservazione del funzionamento di un ottimizzatore classico e di quali sarebbero le migliori condizioni che permetterebbero all'ottimizzatore di generare il migliore piano di accesso ai dati. Così una trasformazione euristica che permetta di introdurre un vincolo su un attributo indicizzato permette all'ottimizzatore di poter utilizzare tale indice per accedere ai dati.

Il sistema che viene proposto non vuole essere alternativo ad un ottimizzatore classico ma complementare ad esso. Lo scopo dell'ottimizzatore è quello di creare delle regole semantiche che racchiudano parte della conoscenza intrinseca del database (*conoscenza intensionale*) e tali da poter essere utili nel generare delle query alternative a quelle originali come descritto precedentemente. Le query alternative vengono successivamente passate all'ottimizzatore classico che ha la possibilità di generare migliori piani di accesso rispetto alla query iniziale. Risulta inoltre essere importante anche la gestione delle regole di trasformazione che vengono generate nel processo di ottimizzazione semantica. Le regole che vengono generate dall'analisi di una query non vengono cancellate ma memorizzate per essere utilizzate per l'ottimizzazione di successive query. La gestione dell'insieme delle regole è una fase importante nel nostro sistema. La scelta di quale trasformazione usare è cruciale ; non considerando una trasformazione può mancare un importante ottimizzazione, mentre includendo una trasformazione irrilevante possiamo aggiungere (rimuovere) una inutile (utile) configurazione e incrementare il tempo di esecuzione. Decidere quale sia l'insieme ottimale di trasformazioni prende del tempo, così limitare la dimensione dell'insieme di regole può aiutare a ridurre il costo dell'ottimizzazione delle query.

Il capitolo è strutturato nel seguente modo : nella sezione 1.1 viene introdotto il processo dell'ottimizzazione semantica di una query in modo più approfondito tramite un esempio. Nella sezione 1.2 sono introdotte alcune definizioni che riguardano la rappresentazione della conoscenza in un database attraverso regole e i vari tipi di

regole che possiamo considerare, inoltre vengono espresse le caratteristiche delle regole utilizzate nell'SQO. Nella sezione 1.3 vengono presentati alcuni tipi di approcci all'ottimizzazione semantica di query. In ultimo , nella sezione 1.4 sono riportate alcune definizioni e tipi di operazioni riguardanti le regole utilizzate nell'SQO.

```
Q1:   select SHIP_CLASS.Class, SHIP_CLASS.Draft
      from SHIP_CLASS, SHIP
      where SHIP_CLASS.Class=SHIP.Class
      and SHIP_CLASS.Draft<50
      and SHIP.Status='Active' ;

Q1.1: select SHIP_CLASS.Class, SHIP_CLASS.Draft
      from SHIP_CLASS, SHIP
      where SHIP_CLASS.Class=SHIP.Class
      and SHIP_CLASS.Draft<50 ;

Q1.2: select SHIP_CLASS.Class, SHIP_CLASS.Draft
      from SHIP_CLASS, SHIP
      where SHIP_CLASS.Class=SHIP.Class
      and SHIP_CLASS.Draft<50
      and SHIP.Year-built>1945

Q1.3: select SHIP_CLASS.Class, SHIP_CLASS.Draft
      from SHIP_CLASS, SHIP
      where SHIP_CLASS.Class=SHIP.Class
      and SHIP_CLASS.Draft<50
      and SHIP.Year-built>1945
      and SHIP_CLASS.Containers='Y';
```

Query equivalenti a Q1 dedotte dalla conoscenza semantica

Tabella 1

1.1 Ottimizzazione semantica di query

In questo paragrafo vedremo come viene eseguita l'ottimizzazione semantica di una query usando un esempio concreto. Prima diamo le seguenti tre definizioni.

Definizione di implicazione : Si consideri il database D che si trovi nello stato x, si consideri inoltre le due espressioni logiche A e B.

Diremo che A implica B, scritto $A \rightarrow B$, se le query Q1(A) e Q2(B) danno come

risposta due insiemi di valori per cui

$$R_x(Q1) \subseteq R_x(Q2).$$

Dove con $R_x(Q_i)$ si indica la risposta della query Q_i rispetto allo stato x del database.

Definizione di applicabilità di una regola derivata : Consideriamo la regola RD: $C \rightarrow D$ e la query Q(A) dove A risulta essere un espressione logica che vincola la query data. Diremo che sarà possibile applicare la regola RD alla query Q se abbiamo che

$$A \rightarrow C.$$

L'obbiettivo dell'ottimizzazione semantica di query è di trovare, per una query in input, una query semanticamente equivalente che presenti un piano di esecuzione più efficiente.

Definizione di equivalenza semantica : Due query sono dette *semanticamente equivalenti* se ritornano un risultato identico per qualunque stato del database soddisfacente un dato insieme di vincoli di integrità.

Consideriamo la query Q1 nella tabella 1 che vuole ritrovare la classe e il pescaggio delle navi che soddisfano le seguenti condizioni : il pescaggio deve essere inferiore a 50 piedi (draft<50), e la nave deve essere ancora in attività (Status= 'Active').

Rules :

R1.1 : Se il draft massimo della nave è minore di 50 allora il suo stato è active.
SHIP_CLASS.Class=SHIP.Class ∧ SHIP_Class.Draft<50 → SHIP.Status='Active'

R1.2 : Se una nave è active allora è stata costruita dopo il 1945.
SHIP.Status='Active' → SHIP.Year-built>1945

R1.3 : Se una nave è active allora è stata costruita dopo il 1945.
SHIP_CLASS.Class=SHIP.Class ∧ SHIP.Year-built >1945
→ SHIP_CLASS.Containers='Y'

Regole semantiche applicabili a Q1

Tabella 2

Supponiamo che l'ottimizzatore possieda l'insieme di regole semantiche poste nella tabella 2. Basandosi sulla conoscenza semantica, l'ottimizzatore suggerirà una serie di riformulazioni per ottimizzare la query in input. Possibili riformulazioni sono: inserimento di una nuova espressione nella query, cancellazione di un espressione e rifiuto dell'intera query (è il caso per cui il risultato della query è un insieme vuoto). Le riformulazioni coinvolgono l'applicazione delle regole semantiche che sono mostrate nella tabella 2. Una regola è considerata applicabile a una query se l'antecedente della regola è una conseguenza logica dell'espressioni della query.

Una volta trovate le regole applicabili alla query l'ottimizzatore può procedere alla formulazione di query alternative a quella originale. L'ottimizzatore può cancellare un espressione dalla query se il conseguente di una regola applicabile implica un espressione della query, oppure inserire il conseguente come una nuova espressione. Alcune delle query riformulate equivalenti a Q1 sono mostrate nella tabella 1. La query Q1.1 deriva dalla riformulazione di Q1 applicando la regola R1.1. Da Q1 e R1.1 posiamo dedurre che la restrizione sullo status 'Active' è implicato dalla restrizione

Draft<50 e dunque è ridondante. Perciò può essere cancellata e il risultato è la query Q1.1 che è ancora equivalente a Q1. Oltre alla cancellazione possiamo aggiungere nuovi vincoli ad una query. Per esempio possiamo aggiungere year-built>1945 a Q1.1 da R1.2, producendo un'altra query Q1.2 equivalente a Q1. La regola R1.3 è applicabile a Q1.2. Introduciamo la restrizione su Containers a Q1.2 ottenendo la query Q1.3. Si osservi che la regola R1.3 non è applicabile alla query Q1 mentre risulta applicabile per la query Q1.2 grazie alla restrizione su year-built introdotta dalla regola R1.2. Quindi l'applicazione di una regola può essere utile anche al fine di poter applicare ulteriori regole. La query iniziale Q1 è stata trasformata nelle query Q1.1, Q1.2 e Q1.3 sintatticamente differenti a Q1, ma semanticamente equivalenti.

L'ottimizzatore può rifiutare una query quando scopre che un'espressione della query contraddice una regola. Con questo si esprime che non vi sono dati che soddisfano la query. E' possibile che l'ottimizzatore possa dedurre direttamente dalle regole la risposta alla query, in questo caso non si ha nemmeno bisogno di accedere al database per rispondere all'interrogazione. Il caso precedente riguarda una di quelle condizioni che si possono verificare per le quali è possibile portare a termine l'esecuzione della query senza accedere ai dati del database.

Consideriamo le seguenti due possibilità. La query ha una *contraddizione*. Se in una query esiste un conflitto tra i vincoli nella clausola where, allora non ha soluzione, nel qual caso verrà restituita una risposta nulla. La query implica una *tautologia*, è possibile ottenere il risultato della query derivandolo direttamente dai vincoli della query stessa, nel qual caso la risposta è già presente all'interno della query.

Ad esempio consideriamo la seguente query :

```
select SHIP.Shipname
from SHIP                                                    (Q2.1)
where SHIP.Deadweight > 700
and SHIP.Registry = 'France' ;
```

Se esiste una regola del tipo

R2 : SHIP.Registry = ‘France’ → SHIP.Deadweight < 500

allora c’è una contraddizione nei predicati della query. Senza dovere accedere al database possiamo affermare che la risposta della query è nulla.
Vediamo un esempio di una tautologia considerando la query Q3.1 e la regola R3.

```
select SHIP.Shiptype
from SHIP                                                    (Q3.1)
where SHIP.Deadweight > 200
```

R3 : SHIP.Deadweight > 200 → SHIP.Shiptype =’Tanker’

La query Q3.1 può essere trasformata, usando la regola R3 nella seguente query :

```
select SHIP.Shiptype
from SHIP                                                    (Q3.2)
where SHIP.Deadweight > 200
and SHIP.Shiptype = ‘Tanker’ ;
```

Questa nuova query contiene già in un suo vincolo la risposta.
L’esempio sopra può essere incorretto. Se infatti non ci sono Tankers nel database allora la risposta dovrebbe essere null. Sarà compito del sistema di manutenzione delle regole assicurarsi che esiste almeno una tupla che verifichi la correttezza del risultato finale.

1.1.1 Apprendimento delle regole di trasformazione per l’ottimizzazione semantica delle query

In questo paragrafo diamo una serie di definizioni che introducono il concetto di regola semantica. Consideriamo il concetto di formula. Qualsiasi formula può essere considerata come una query. Una formula è chiamata *ground* se essa non contiene variabili. Un importante classe di formule è la *clausola*. Una clausola ha la seguente forma generale :

$$\forall. A_1 \vee \ldots \vee A_k \vee \neg A_{k+1} \vee \ldots \vee \neg A_n$$

dove ogni Ai è una formula atomica, per $i \in \{1, \ldots .,n\}$, e nelle quali le variabili sono intese essere quantificate universalmente (come denota il simbolo ∀). Qualunque clausola può essere riscritta in una forma logicamente equivalente come un implicazione:

$$\forall. A_1 \vee \ldots \vee A_k \leftarrow A_{k+1} \wedge \ldots \wedge A_n.$$

nella quale si sono supposte delle disgiunzioni nella parte sinistra dell’implicazione e delle congiunzioni nella parte destra, e inoltre viene sottintesa la quantificazione universale. Una clausola in questa forma è anche chiamata *regola (rule)*. L’insieme degli elementi atomici sul lato sinistro ($A_{1, \ldots\ldots\ldots, A_k}$) è chiamato *testa* (*head*) della regola mentre la collezione di elementi atomici sul lato destro ($A_{k+1, \ldots\ldots\ldots, A_n}$) *corpo* (*body*). Una *Horn-rule* ha al più un elemento nella head : $k \leq 1$. Una clausola definita ha esattamente un elemento nel head : $k = 1$.
Una ground rule con il body vuoto è chiamata un fatto (*fact*).
Chiameremo la seguente regola una regola per A

$$A \leftarrow B_1, \ldots\ldots, B_n.$$

La tabella di seguito mostra alcune alternative semantiche delle regole.

Classe semantica	Regola	Semantica
Universale	A → C	Tutte le tuple che soddisfano A soddisfano anche C
Esistenziale	A → C	Alcune tuple che soddisfano A soddisfano anche C

Probabilistica	$A \rightarrow C : K$	$Pr.(t \in C \mid t \in A) = K$ La probabilità di validità delle regola è K

Le regole che utilizzeremo per l'ottimizzazione semantica presentano una semantica universale, dunque esse saranno valide per tutte le tuple del database.

1.1.2 Vincoli di integrità e regole di trasformazione delle query

I vincoli di integrità di un database relazionale definiscono la semantica proposta dalle sue relazioni. Molti database hanno un certo numero di vincoli di integrità sui dati che devono essere rafforzati durante la manipolazione. Un semplice esempio di vincolo di integrità è il tipo di dato e il range di valori validi che si ha per ogni dato. Un'altro vincolo di integrità può specificare le associazioni tra records in differenti file.

Alcuni vincoli sono implicitamente rappresentati nel modello dei dati. Un esempio di vincolo implicito nel modello relazionale è la dipendenza funzionale. Altri vincoli sono inerenti al modello dei dati e sono assunti senza che essi siano specificati nello schema. Un esempio di tali vincoli, nel modello relazionale è che i valori del database devono essere atomici e indivisibili. Abbiamo poi vincoli sui dati che vengono espressi esplicitamente. Un esempio di vincolo esplicito può essere il range di valori validi per un attributo. Il vincolo sul salario dell'impiegato che dice che il salario dell'impiegato non può essere più alto del salario del suo capo è un istanza di tali vincoli.

Un'altra classificazione dei tipi di vincoli di integrità è basato sullo stato corrente o passato del database. I vincoli di stato si riferiscono a tutti i vincoli che i dati devono rispettare in ogni stato del database. I vincoli di transizione specificano il modo con cui uno stato del database può essere trasformato in un'altro stato. Una regola che garantisce che il valore del salario può solo aumentare è un vincolo di transizione.

Si deve distinguere tra vincolo di integrità e *regole di trasformazione delle query* che provvedono ad una più accurata descrizione della loro natura. Le regole di trasformazione delle query sono un chiaro stato di vincoli di integrità che risultano utili per l'ottimizzazione semantica per query quantificate universalmente. Queste

regole sono caratterizzate da quantificazioni universali nelle formule che rappresentano quelle query.

1.1.3 Regole di trasformazione definite dall'utente o scoperte

Le *regole di trasformazione definite dall'utente* sono un sottoinsieme dei vincoli di integrità definiti dall'utente. Le trasformazioni definite dall'utente devono essere sempre vere per tutte le tuple di ogni stato valido del database. Qualunque aggiornamento che viola tali regole verrà rifiutato.

Le *regole di trasformazione scoperte* in un certo stato del database possono o meno rappresentare una proprietà del database indipendentemente dal tempo. Per esempio, nel database shipping possiamo avere che, in uno specifico stato del database, c'è solo una nave di tipo 'Tanker' che risulta essere assicurata da 'Lloyds'. Sarà possibile scoprire e fare uso della seguente regola di trasformazione :

Reg2 : (ShipType = 'Tanker') and (Issuer = 'Lloyds') → (ShipName = 'Queensea')

Tutte le query che fanno riferimento alla classe 'Tanker' possono ora essere esaminate considerando la tupla con ShipName = 'Queensea' nel corrente stato del database. In generale una regola di trasformazione scoperta può essere invalidata da futuri aggiornamenti del database.

Durante l'ottimizzazione semantica una query Q può essere trasformata nella query (Q and (A → C)). Se Q è una query congiuntiva e ha una clausola A, allora le query Q e (Q and C) sono equivalenti. L'implicazione A → C è data dall'equivalenza logica di (A and (A → C)) e (A and C). Allora la correttezza della regola di trasformazione A → C può essere determinata dall'equivalenza tra l'espressioni delle query A e (A and C). Una regola di trasformazione

A → C è corretta nel corrente stato del database se la query logica A e (A and C) sono query equivalenti nel corrente stato del database.

Una regola di trasformazione delle query sarà utile per la risposta ad una query Q se essa apporta un sostanziale riduzione del costo di esecuzione della query Q durante l'ottimizzazione semantica. Nel prossimo paragrafo si evidenziano le principali caratteristiche delle regole semantiche.

1.2 Definizione delle caratteristiche delle regole

Non tutte le regole sono utili per l'ottimizzazione semantica. Il primo criterio di una regola semantica utile è che deve essere sempre consistente con il database, altrimenti l'ottimizzatore potrebbe dare un risultato scorretto. Un'altro criterio base è la funzionalità, cioè che una regola semantica deve essere in una forma pronta per essere usata.

Consistenza e funzionalità, non sono comunque sufficienti per delineare una classe abbastanza ristretta di regole semantiche derivate. Una regola che presenti un alto grado di utilità per il processo di ottimizzazione semantica deve essere in grado di produrre un alto risparmio per un gran numero di query, mentre deve richiedere un costo minimo dovuto al suo utilizzo. Il costo dell'uso delle regole semantiche includono lo spazio di memorizzazione per le regole, il tempo di computazione dovuto alla loro scelta e applicazione delle regole durante l'ottimizzazione, e al costo di manutenzione delle regole in presenza di cambiamenti del database. Se si generassero delle regole semantiche invarianti che sono consistenti con tutti i possibili stati del database malgrado qualsiasi mutamento del database, allora il costo di manutenzione delle regole potrebbe essere eliminato. Comunque, non è possibile garantire che tutte le regole trovate siano invarianti, poiché è impossibile per il sistema di apprendimento delle regole semantiche verificare se una regola è invariante senza una completa conoscenza del dominio di applicazione del database. Perciò il sistema deve cercare di minimizzare il costo di manutenzione in presenza di cambiamenti di stato del database.

Ci sono diversi possibili approcci a questo problema. Quando il sistema scopre che una regola è diventata inconsistente dopo una transazione che ha modificato i dati del database, esso può cancellare la regola o alterarla. Eliminare le regole inconsistenti è semplice ed efficace, comunque, se le regole derivate dal sistema di apprendimento non sono robuste contro i cambiamenti del database, molte delle regole diverranno inconsistenti e le regole rimanenti potranno essere insufficienti a supportare l'ottimizzatore semantico. Di conseguenza il sistema dovrà rigenerare le

regole semantiche frequentemente rischiando di aumentare i costi dell'ottimizzazione. Alternativamente il sistema può scegliere di aggiustare le regole inconsistenti.

1.3 Alcuni approcci al problema del SQO

1.3.1 Acquisizione di conoscenza nel database per l'ottimizzazione semantica

L'apprendimento di regole per la trasformazione di query può essere *query-driven* o *data-driven*. Nell'approccio query-driven la ricerca delle nuove regole di trasformazione è guidata dall'insieme delle query che arrivano al database attraverso la comparazione di query e la generazione di ipotesi e testing. Nella comparazione l'insieme di query, che arrivano dopo l'ultimo aggiornamento, sono analizzate attraverso il confronto dell'insieme delle tuple ritrovate per rispondere alle varie query. Se l'insieme delle tuple che sono state ritrovate per le due query sono identiche, allora possiamo generare delle regole di trasformazione di query attraverso le restrizioni delle due query. Nell'approccio con la generazione di query e testing le query sono usate per generare una regola di trasformazione candidata. Una regola candidata consiste di una restrizione come antecedente e di un insieme di variabili come conseguente. La restrizione antecedente è generata dalle restrizioni delle query che arrivano al sistema. L'insieme delle variabili libere per il conseguente è generato da euristiche. La restrizione antecedente e valutata sullo stato corrente del database per ritrovare e classificare tutti i possibili valori delle variabili libere per formare il conseguente.

Nell'approccio data-driven la ricerca è diretta dai dati, la quale conduce l'apprendimento di regole caratterizzanti particolari strutture dei dati che rappresentano le regole di trasformazione di query.

Approcci data-driven possono basarsi su algoritmi di scoperta sviluppati nell'Intelligenza Artificiale (AI). Molti di questi algoritmi di ricerca sono stati rappresentati in linguaggi simili al First Order Predicate Logic (FOLP), e queste regole possono essere usate per rappresentare vincoli di integrità e regole di trasformazione.

Uno svantaggio dell'approccio query-driven è che possiamo ottenere un risparmio

del costo di esecuzione di una query solo se query simili sono ripetute. Quindi questo metodo può incorrere a un più alto costo quando molte query nuove arrivano al sistema. Il vantaggio nell'approccio data-driven è che le regole sono apprese a priori, e possono essere applicate a molte query che non sono mai state generate prima. D'altra parte dobbiamo osservare che con quest'ultimo approccio è possibile generare un numero estremamente elevato di regole di trasformazione molte delle quali non saranno mai utilizzate ma che comporteranno un costo di manutenzione e di matching che potrebbe diventare molto dispendioso.

1.3.2 Tecniche di apprendimento e di scoperta nell'AI

Gli algoritmi AI di apprendimento sono basati sul concetto di controllo dell'apprendimento e scoperta non controllata. L'algoritmo di apprendimento controllato usa un controllore esterno e un insieme di esempi di prova per imparare un insieme di concetti predeterminati. Dopo la fase di training questi algoritmi sono in grado di classificare correttamente semplici dati in un particolare concetto. Esempio di un algoritmo di apprendimento controllato e l'ID3. Il concetto di algoritmo controllato non può essere direttamente applicato al problema di apprendimento di regole di trasformazione. In questo dominio non ci possono essere concetti conosciuti a priori per trovare un insieme di esempi di prova.

La scoperta non controllata riguarda l'apprendimento senza l'aiuto di controllore esterno o esempi. Algoritmi per la scoperta non controllata sono stati applicati a problemi di formazione di tassonomie e scoperta di leggi empiriche. La formazione di tassonomie riguarda la scoperta di regole di classificazione usando raggruppamenti numerici e raggruppamenti concettuali. I metodi di raggruppamento numerico suddividono i dati in gruppi basati sulla loro distanza in uno spazio n-dimensionale Metodi di raggruppamento numerico non sono interessanti poiché essi non generano formule FOLP necessarie per la rappresentazione delle regole di trasformazione. Raggruppamenti concettuali provvedono ad una partizione dei dati basata su certe classi concettuali. La teoria e un l'algoritmo per il raggruppamento concettuale sono

stati sviluppati in [(24) R. S. Michalski (1980)]. Il raggruppamento concettuale usa la conoscenza di background della funzionalità dei raggruppamenti per guidare il processo verso la formazione di raggruppamenti più utili [(16) R. S. Michalski (1983), (27) S. J. Hanson (1990)].

La tecnica di scoperta per l'apprendimento empirico di leggi possono scoprire sia leggi quantitative che leggi qualitative. Le leggi quantitative usano variabili numeric-valued e funzioni matematiche che riassumono i dati e possono essere scoperte attraverso tecniche come l'analisi di regressione [(28) S. P. Ghosh (1987)]. Precedenti lavori sulla scoperta automatica di regole quantitative sono Bacon system [(29) P. Langley (1983)] e Forty-Niner [(30) J. Zytkow (1991)]. L'approccio quantitativo non può essere direttamente applicato al dominio di apprendimento dei vincoli di integrità come non può essere applicato per produrre regole FOLP. Le leggi qualitative rappresentano associazioni logiche tra i dati. Tali associazioni o consistono nel ritrovamento dei confini del dominio di un attributo o consistono in associazioni tra attributi come X >Y, dove X e Y sono attributi. Alcuni degli algoritmi per la scoperta di regole qualitative sono AM [(31) D. B. Lenat (1983)] e Glaubert [(32) P. Langley (1986)]. Entrambi i metodi usano domini euristici per effettuare la ricerca dei concetti di interesse.

Le leggi empiriche qualitative sono vicine alle regole di trasformazione delle query. Per esempio i vincoli di integrità rappresentano il dominio degli attributi e possono essere rappresentati in un linguaggio di tipo FOLP.

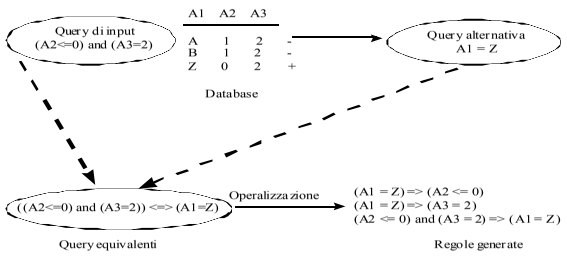


Figura 3

1.3.3 Acquisizione delle regole di trasformazione basato su un algoritmo di generazione di query equivalenti

La figura 3 mostra un semplice scenario per l'apprendimento semantico di regole di trasformazioni di query. Il sistema di apprendimento consiste di due componenti, un componente per l'apprendimento induttivo e un componente di sfoltimento.

Supponiamo che una query con un alto costo di esecuzione sia data in input al sistema di apprendimento delle regole di trasformazione per SQO. Ricevuta la query il sistema applicherà un algoritmo di apprendimento induttivo per generare una query equivalente alternativa alla query data con un basso costo di esecuzione. Il componente di sfoltimento allora prende la query iniziale e la query alternativa generata dall'algoritmo per derivare un insieme di regole semantiche. Nella figura le tuple del database considerato sono etichettate come positive (+) se soddisfano la query iniziale o come negative (-) se non soddisfano la query iniziale. La query alternativa generata deve considerare tutte le tuple positive e nessuna delle tuple negative ciò assicurerà quindi l'equivalenza delle query iniziale e la query alternativa generata dal sistema di apprendimento delle regole di trasformazione. Dato un insieme di tuple classificate positive o negative il problema di derivare una descrizione che consideri tutte le tuple positive ma non quelle negative è conosciuto come apprendimento induttivo controllato [(53) Shavlik and Dietterich, (1990)].

Nell'esempio il sistema genera una query alternativa avente un solo predicato (A1 = 'Z') il quale copre tutte le tuple positive ed esclude quelle negative. Questa query è meno costosa della query di input.

Il componente di sfoltimento genera regole semantiche operative che permettono all'ottimizzatore di dedurre una query equivalente alla query in input e alla query alternativa. Questo lo si raggiunge attraverso due passi. Il primo passo comporta di trasformare l'equivalenza di due query in un insieme di regole Horn-clause. Questo passo è chiamato operalizzazione poiché esso produce regole operative che sono cioè pronte per essere usate dall'ottimizzatore. Il secondo passo è di sfoltire l'espressioni presenti negli antecedenti delle regole prodotte. Tale processo permette di generare delle regole semantiche più robuste, cioè regole che presentono una più facile manutenzione essendo meno soggette a rischi di diventare inconsistenti con i futuri stati del database, inoltre lo sfoltimento permette di aumentare l'applicabilità delle regola.

Torniamo a considerare l'esempio di figura. Nella fase di operazionalizzazione delle regole prodotte l'equivalenza della query di input e la query alternativa prodotta dal sistema e trasformata in due seguenti regole

1) $(A2 \leq 0)$ and $(A3 = 2) \Rightarrow (A1 = 'Z')$

2) $(A1 = 'Z') \Rightarrow (A2 \leq 0)$ and $(A3 = 2)$

La regole (2) può essere espansa per soddisfare la sintassi Horn-clause richiesta:

3) $(A1 = 'Z') \Rightarrow (A2 \leq 0)$

4) $(A1 = 'Z') \Rightarrow (A3 = 2)$

Dopo la trasformazione il sistema ha generato le regole (1), (3) e (4) che sono operative per il sistema. Nel secondo passo il sistema prova a sfoltire gli antecedenti delle regole così prodotte. Nel nostro esempio le regole (3) e (4) hanno solo un espressione come antecedente, dunque il sistema non effettuerà tale passo. E' possibile invece eseguire questo secondo passo sulla regola (1) attraverso lo sfoltimento della prima espressione $(A2 \leq 0)$ o della seconda espressione $(A3 = 2)$ basandosi su opportuni indicatori. Durante la stima il sistema dovrà verificare, prima

di eliminare un espressione contenuta nell'antecedente, se la nuova regola risulta essere ancora consistente con lo stato del database. La sola regola consistente con il database prodotta dalla fase due risulta essere la seguente

5) $(A2 \leq 0) \Rightarrow (A1 = 'Z')$

Si osservi come le regole così ottenuto risultano essere maggiormente applicabili poiché presentano una sola espressione nell'antecedente. E' infatti più probabile che un predicato di una query da ottimizzare sia implicato da un antecedente di questo tipo. Le regole così sfoltite presentano, in generale, una probabilità minore che un cambiamento dello stato del database rendano queste regole inconsistenti.

1.3.4 Apprendimento di regole per SQO basato sulla distribuzione dei dati

Viene proposto un sistema per la scoperta di leggi qualitative per il problema dell'apprendimento di regole per la trasformazione di query. L'approccio è del tipo data-driven e si basa sull'assunzione che regole utili possono essere derivate da una non uniforme distribuzione dei valori degli attributi.

1.3.4.1 Distribuzione dei dati

Una distribuzione dei dati è una rappresentazione dello stato di un dato database. Una distribuzione dei dati è una funzione che mappa una griglia e un database al numero di tuple in ogni cella della griglia.

Una griglia è specificata da un insieme di coordinate che sono funzione degli attributi del database. La dimensione di una griglia rappresentano il numero di coordinate della griglia, assumendo che le coordinate sono linearmente dipendenti. Una griglia è ortogonale agli attributi di un database se ogni asse-coordinata rappresenta un attributo unico del database. Una griglia è obliqua se una coordinata rappresenta una funzione di due o più attributi. Un esempio di griglie per il database SHIPPING sono <Capacity, Quantity> e <Capacity + Quantity, Capacity - Quantity>. Il primo esempio è una griglia ortogonale mentre il secondo è una griglia obliqua.

Le celle della griglia dividono i valori possibili per ogni coordinata in un insieme finito

di ranges. Poniamo essere Si (i=1...n) l'insieme di ranges per le n coordinate di una griglia n-dimensionale. L'insieme delle celle per questa griglia può essere rappresentato dal prodotto cartesiano S1 x S2 x .....Sn.

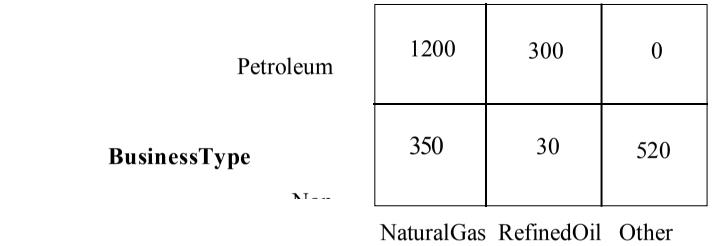


Figura 4

Esempio: Consideriamo la griglia di figura 4 con un insieme coordinate <BusinessType, CargoType>. L'insieme di range per la prima coordinata può essere (Petroleum, Non-Petroleum). Per la seconda coordinata può essere (NaturalGas, RefinedOil, Other). Il numero delle celle della griglia saranno dunque sei.

Una distribuzione dei dati delle tuple nel database SHIPPING può essere descritto ponendo dei numeri interi in ogni cella, rappresentanti il numero di tuple che sono descritte da quella cella.

1.3.4.2 Scoperta delle regole di trasformazione delle query

I vincoli di integrità danno origine a speciali modelli di distribuzione dei dati formati da una griglia attraverso l'insieme di attributi presenti nel vincolo di integrità. Per esempio il vincolo di integrità (Bussines = 'Petroleum') → (CargoType ∈{'NaturalGas','RefinedOil'}) sul database SHIPPING risulta in una distribuzione con nessuna tupla nella cella ('Petroleum', 'Other'), come mostrato in figura 3. In generale un vincolo di integrità del tipo (Attributo1 = 'a') → (Attributo2 = 'b') fa si che nessuna tupla di una cella soddisfi ((Attributo1 = 'a') and (Attributo2 <> 'b')).

Patterns di una distribuzione di dati per il corrente stato di un database possono essere usati per scoprire regole di trasformazione delle query. Un semplice insieme di patterns sono mostrati nella tabella 1 insieme con le regole di trasformazione che

possono essere derivate sulla base dei patterns.

Data-distribution Pattern	Regole di trasformazione
La colonna (A1=’a’) ha tutte le tuple nella cella corrispondente a (A2=’b’) nella griglia ortogonale <A1,A2>	(A1=’a’) → (A2=’b’)
La colonna (A1=’a’) ha tutte le tuple nelle celle corrispondente a (A2=’b’) o (A2=’c’)	(A1=’a’) → (A2 ∈ {b,c})
La colonna ((A3+A4)=’e’) non ha tuple nella griglia obliqua <(A3 + A4),(A3 - A4)>	(A3 + A4 ≠’e’)
La colonna ((A1=’a’) e (A2=’b’)) ha tutte le tuple nella cella corrispondente a (A3≠’c’) nella griglia <A1,A2,A3>	(A1=’a’) and (A2=’b’) → (A3≠’c’)

1.4 Definizioni per il trattamento delle regole

In un sistema rule-based per l'ottimizzazione semantica risulta essere importante il trattamento delle regole che esprimono la conoscenza semantica che viene prima generata dal sistema di generazione automatico delle regole e poi vengono utilizzate per generare le query equivalenti a basso costo di esecuzione. Le operazioni principali che vengono eseguite sulle regole sono quelle di confronto. A secondo del tipo di confronto che si va a fare e possibile gestire insiemi di regole (rule set) che vengono utilizzati dall'SQO. Si potrà cosi aggiungere, eliminare, valutare le diverse regole presenti nel rule set per rendere più efficienti le diverse procedure che vengono eseguite dall'SQO.

Di seguito verranno proposti alcune definizioni e regole che riguardano il trattamento di regole semantiche.

Definizione di inconsistenza di una regola : Dato il database D nello stato x diremo che la regola $A \rightarrow B$ è inconsistente con lo stato x del database se la query $Q(\neg B \wedge A)$ ritorna una risposta non vuota, cioè

$$Rx(Q) \neq \varnothing .$$

Definizione di regole equivalenti : Date due regole R1: $A \rightarrow B$, R2 : $C \rightarrow D$ diremo che R1 e R2 sono equivalenti se

$$A = C \quad e \quad Rx(B) = Rx(D).$$

E' necessario soffermarci sul concetto di equivalenza che si è voluto esprimere con quest'ultima definizione. L'equivalenza di una regola vuole cogliere l'equivalenza delle regole in considerazione della loro utilità nel processo di ottimizzazione semantica. Quindi l'equivalenza che si vuole esprimere riguarda l'utilità che le regole hanno per il sistema di SQO. Dato che l'utilità di una regola derivata risulterà essere finzione del costo risparmiato dall'applicazione di tale regola e dal suo grado di

applicabilità (anche stimato), cioè dalla frequenza di applicazione di tale regola, possiamo dare la seguente definizione di equivalenza. Due regole sono equivalentemente utili se producono un uguale costo risparmiato e se presentano un grado di applicazione uguale.

Esempio. Si considerino le due regole

R1 : $A1 \rightarrow B1$, con $A = (a > 10)$ e $B = (b > 200)$

R2 : $A2 \rightarrow B2$, con $C = (a > 5)$ e $B = (b > 150)$

Vediamo come e quando possiamo applicare le due regole considerando tre predicati differenti che possono essere ipoteticamente presenti all'interno di una query Q.

1) a>12 : è possiamo applicare a questo predicato sia la regola R1 che la regola R2. L'applicazione della regola R1 ci permette di considerare il vincolo (b > 200) mentre l'applicazione della regola R2 ci permette di considerare il vincolo (b >150).
2) a>7 : è possiamo applicare a questo predicato solo la regola 2 e quindi considerare il vincolo (b> 150).
3) a>3 : non è possibile applicare nessuna delle due regole.

L'esempio ci fa vedere come al vincolo A1, presente nell'antecedente della regola R1, è possibile associare un fattore di selettività più alto del vincolo A2, presente nell'antecedente della regola R2. Ciò indica che sarà più probabile che venga utilizzata la regola R2 rispetto alla regola R1. Si osserva inoltre che il vincolo B1, presente nell'antecedente della regola R1, copre un range più ristretto del range coperto dal vincolo B2, presente nell'antecedente della regola R2. Avremo dunque una maggiore probabilità che l'introduzione del vincolo B1, nella query da ottimizzare Q, porti un costo risparmiato maggiore del costo risparmiato dovuto all'introduzione del vincolo B2.

Definizione di matching tra una regola proposta e una regola derivata : Consideriamo le seguenti regole: la regola proposta RP1 : $A1 \rightarrow b1$, dove b1 rappresenta solo il nome di un attributo, la regola proposta RP2 : $A2 \rightarrow B2$ e la regola derivata RD : $C \rightarrow D$.

Diremo che la regola proposta RP1 *match* la regola derivata RD se

$$A1 \rightarrow C$$

e se l'attributo b1 è uguale all'attributo che compare nell'espressione C.
Diremo che la regola proposta RP2 match la regola derivata RD se

$$A2 \rightarrow C \quad e \quad D \rightarrow B2.$$

Consideriamo ora le regole che riguardano le regole di tipo range-rule. Le regole che verranno prese in considerazione sono basate su operazioni logiche di implicazione, unione, intersezione e negazione. Si suppone che siano date le seguenti due regole R1 : A1 $\rightarrow$ C1 e R2 : A2 $\rightarrow$ C2.

Regola 1: (Regola transitiva). Date R1 e R2 se abbiamo che C1 $\rightarrow$ A2 allora è possibile aggiungere al rule set la regola A1 $\rightarrow$ C2.

Dimostrazione : Consideriamo la tupla t $\in$ A1. Con t $\in$ A1 si intende una tupla per cui i valori degli attributi soddisfano le clausole della formula logica A. se t $\in$ A1 allora dalla R1 possiamo dedurre che t $\in$ C1. Analogamente t $\in$ C1 implica , a causa di C1 $\rightarrow$ A2, che t $\in$ A2 che a sua volta implica, a causa di A2 $\rightarrow$ C2 che t $\in$ C2. Dunque possiamo affermare che (for all t) : (t $\in$ A1) $\rightarrow$ (t $\in$ C2). Allora A1 $\rightarrow$ C2.

Regola 2 : (regola di unione). Date le regole R1 e R2 se A1 e A2 sono restrizioni sugli stessi attributi, per esempio l'attributo 'a', e C1 e C2 sono restrizioni sugli stessi attributi, per esempio 'c', allora è possibile derivare la regola Unione(A1,A2) $\rightarrow$ Unione(C1,C2).

Dimostrazione : la dimostrazione è simile a quella della regola di chiusura 1.

Regola 3 : (regola di intersezione). Date le regole R1 e R2 possiamo considerare anche la regola Intersezione (A1,A2) $\rightarrow$ Intersezione(C1,C2).

Regola 4 : (regola del complemento).data la regola R1 possiamo considerare allora anche la regola B $\rightarrow$ D, dove B = Not(C1) e D = Not(A1).

CAPITOLO 2

2.1 Acquisizione automatica delle regole semantiche per l'ottimizzazione semantica

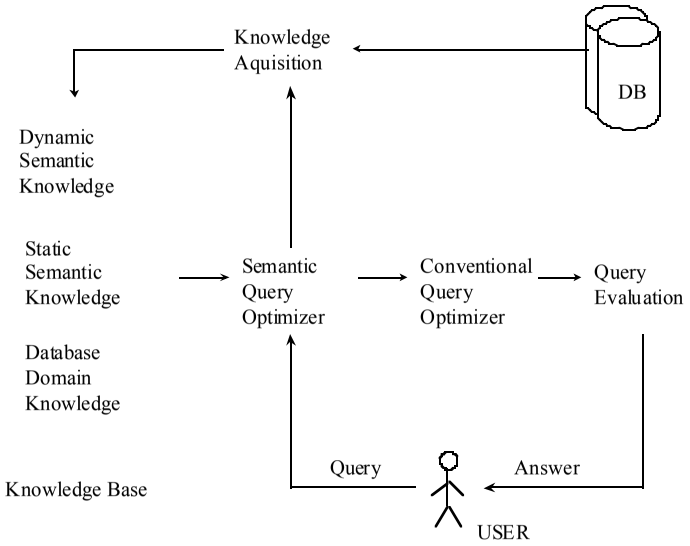


Figura 1

Il sistema che si vuole proporre in questa tesi, intende estrarre da un database, in modo automatico, la conoscenza semantica necessaria all'SQO per un suo efficace funzionamento. Il processo di derivazione automatica è noto come Knowledge Acquisition (KA). Il successo dell'ottimizzazione semantica delle query dipende dal poter disporre di una buona conoscenza semantica. Come presentato nel primo capitolo, la conoscenza semantica è rappresenta attraverso delle regole di trasformazione. Il numero di regole semantiche che si può estrarre da un database è molto grande, quindi la principale difficoltà nella KA, sarà quella di identificare tutte le regole semantiche utili per il funzionamento dell'SQO.

Consideriamo la figura 1 che mostra un sistema generico per l'ottimizzazione semantica delle query illustrando come i moduli interagiscono tra di loro e il tipo di conoscenza che occorre prendere in considerazione. L'utente immette una query che arriva all'ottimizzatore semantico. Si noti come l'uscita dell'SQO è l'input dell'ottimizzatore classico, che risulta essere sempre un modulo indispensabile per DBMS. L'ottimizzatore semantico risulta essere complementare all'ottimizzatore classico e non sostitutivo. L'ottimizzatore semantico di query farà uso della conoscenza presente nel database per aggiungere o eliminare restrizioni nella qualificazione della query. La conoscenza include la conoscenza semantica statica, la quale è vera per tutti gli stati del database, e la conoscenza semantica dinamica che è vera per il corrente stato del database. Il nostro sistema di acquisizione intente identificare, estrarre e gestire in modo automatico, la conoscenza dinamica

Nella figura 2 è mostrata l'architettura per l'acquisizione della conoscenza in un database.

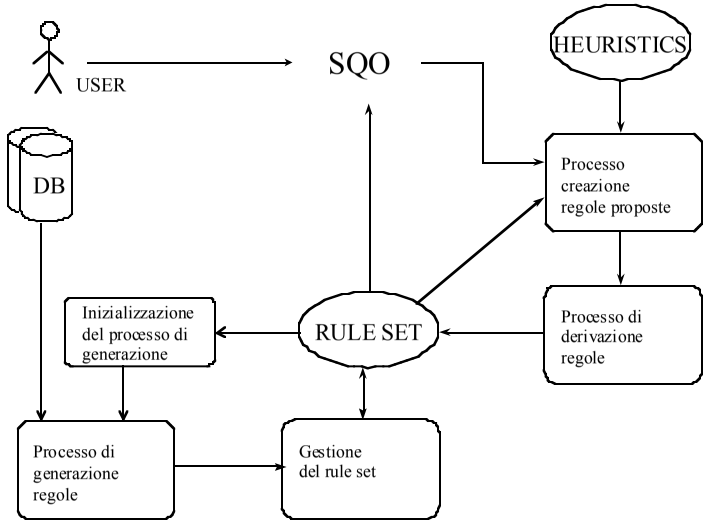


Figura 2

Le regole semantiche scoperte sono memorizzate nell'insieme rule set e sono del tipo ($JC \wedge P \rightarrow RestC$), dove JC è un insieme di equi-join che definiscono una relazione R; P è un insieme di restrizioni sulla relazione R e RestC è una restrizione su R che rappresenta il conseguente della regola. Quindi l'antecedente delle regole può essere composto da più restrizioni congiunte, mentre il conseguente è formato da una sola restrizione. In particolare, nel rule set troviamo solo la conoscenza dinamica del database, che è quella che a noi interessa generare. Per creare le regole semantiche si utilizzano due processi: *processo di derivazione (PD)* e *processo di generazione (PG)*. Il processo di derivazione delle regole si basa su un approccio query-driven, con l'utilizzo di *euristiche* per una creazione mirata delle regole semantiche di cui il sistema necessita. Le regole sono ottenute analizzando le risposte delle query generate nel processo di derivazione. Il processo di generazione delle regole si basa invece su un approccio di tipo data-driven. Le regole sono ottenute analizzando direttamente i dati del database.

Uno degli intenti che si propone il nostro sistema, è proprio quello di cercare di integrare questi due modi di estrarre delle regole dal database, eliminando i punti deboli di un tipo di approccio con i pregi dell'altro.

Il modulo "*Gestione del rule set*" si occupa della manutenzione della conoscenza dell'insieme delle regole. In particolare assicura la validità delle regole per ogni stato del database e fa in modo che non vi sia una ridondanza di conoscenza all'interno del rule set. Le regole create dal processo di generazione non vanno direttamente al rule set, ma vengono prima controllate dal modulo "Gestione del rule set". Questo controllo serve per impedire di inserire nel rule set regole ridondanti. Le regole uscite dal PD possono essere inserite direttamente nel rule set, visto che il controllo di non ridondanza è stato effettuato prima che iniziasse il processo di derivazione.

Elemento cruciale per il processo di derivazione sono le *regole proposte*. Le regole proposte indicano al sistema quali regole dovrebbero essere derivate.

Esse si presentano nella seguente forma:

RP1	***attributo***	***vincolo***
antecedente	LIBRO.Collocazione	(> 25)
conseguente	LIBRO.Codice	

Nella quale viene espressa la/le restrizioni dell'antecedente, mentre del conseguente viene riportato solo il nome dell'attributo sul quale occorre ricercare una restrizione. Nel caso che la regola riguardi attributi appartenenti a relazioni diverse, nell'antecedente verrà specificato anche il join che collega le diverse relazioni.

RP2	***attributo***	***vincolo***
Antecedente	LIBRO.Collocazione	(> 25)
And	LIBRO.Soggetto	(= RIVISTA.Soggetto)
Conseguente	RIVISTA.Codice	

Nell'approccio query-driven vengono analizzate le query che sono immesse nell'ottimizzatore. Il modulo "*Processo creazione regole proposte*" controlla se è possibile applicare l'euristiche, alla qualificazione della query, attraverso la corrente conoscenza presente nel rule set. Ad esempio, consideriamo la seguente query:

```
select LIBRO.Titolo
from LIBRO
where LIBRO.Collocazione > 25;
```

Un euristica di tipo index cercherebbe di introdurre nella query una restrizione sull'attributo indicizzato LIBRO.Codice, attraverso una regola del rule set. Regola che dovrebbe avere come antecedente una restrizione implicata da (LIBRO.Collocazione = 25), e come conseguente una restrizione su LIBRO.Codice. Se nel rule set non è presente una regola di questo tipo, viene generata la regola proposta RP1. Dato che

le regole proposte che possono essere generate sono numerose, mentre solo alcune sono effettivamente utili, e che per esaminarle tutte occorrerebbe un tempo troppo grande, si rende necessario una selezione delle regole proposte da prendere in considerazione per il processo di derivazione. Nel PD le regole sono create attraverso l'esecuzione di apposite query che vengono generate dal sistema stesso. Non tutte le regole scoperte sono adatte per l'SQO, quindi alcune verranno scartate mentre le rimanenti saranno inserite nel rule set.

Ad esempio, dalla regola proposta RP1 si costruisce la seguente query:

```
select LIBRO.Codice
from LIBRO
where LIBRO.Collocazione>25;
```

Se ad esempio il risultato della query comprende dei valori superiori a 100 si può generare una regola derivata che esprime che tutti i libri con Collocazione maggiore di 25 hanno un codice superiore a 100. Tale regola viene espressa nel seguente modo:

RP1	***attributo***	***vincolo***
antecedente	LIBRO.Collocazione	(> 25)
conseguente	LIBRO.Codice	(> 100)

Il PG crea le regole da inserire nel rule set prendendo in considerazione direttamente i dati presenti nel database. Il modulo "*Inizializzazione PG*" decide su quali relazioni occorre effettuare la ricerca, e quali sono gli attributi dell'antecedente e del conseguente delle regole che si devono generare nel PG.

Ad esempio, consideriamo le due relazioni del database Biblioteca: LIBRO(Codice, Titolo, Collocazione, Anno) e RIVISTA(Titolo, Numero, Anno). Il modulo di inizializzazione potrebbe decidere che occorre effettuare una ricerca delle regole, sulla relazione definita dal join LIBRO.Anno=RIVISTA.Anno, inoltre le regole dovranno

avere come antecedente una restrizione sull'attributo LIBRO.Collocazione e come conseguente una restrizione sull'attributo RIVISTA.Titolo.

Aumentiamo il dettaglio della descrizione del sistema KA considerando la figura 3.

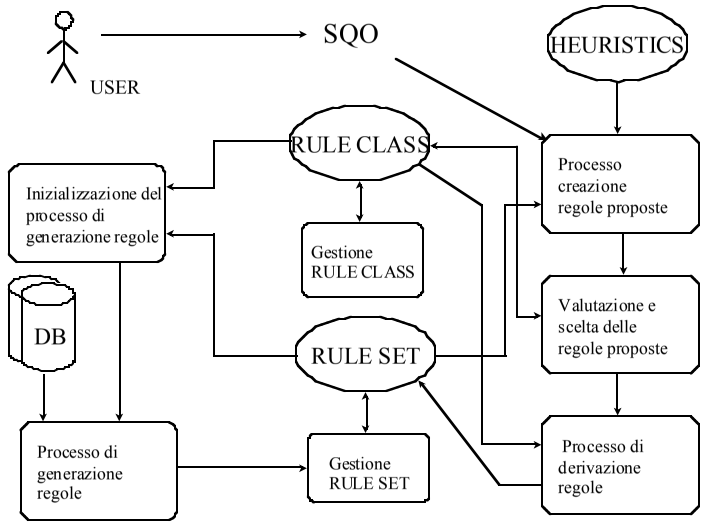
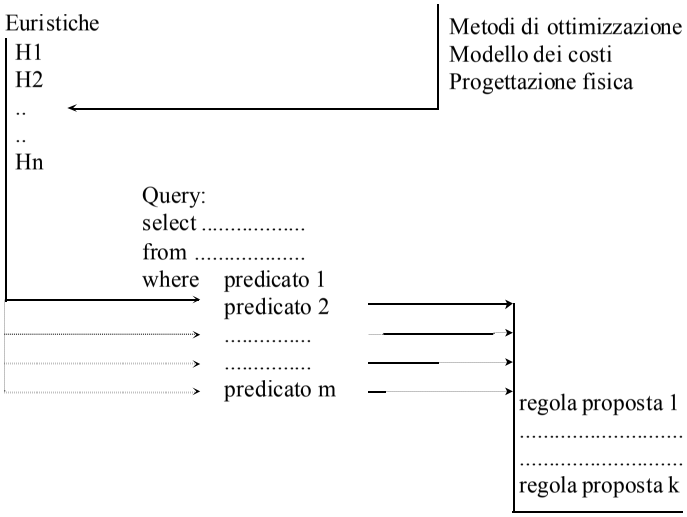


Figura 3

Dallo schema precedente è stata estratta dal modulo "Processo creazione regole proposte" la fase di selezionamento delle regole proposte per il processo di derivazione. Questa fase, che è di fondamentale importanza per il buon funzionamento di tutto il sistema, è eseguita nel modulo "*Valutazione e scelta delle regole proposte*". Il nuovo insieme "*RULE CLASS*" contiene una serie di informazioni che tengono conto dei tentativi passati di derivazione delle regole proposte. Queste informazioni vengono raccolte durante la fase di selezione delle RP e durante il processo di derivazione. Nello schema viene ora evidenziato come l'informazione, per dare inizio al processo di generazione delle regole, arrivi dai due insiemi rule set e rule class. I dati contenuti nel rule class hanno un ruolo molto importate sia per il processo di derivazione, sia per il processo di generazione. Nel PD le rule class ci permettono di valutare le regole proposte più promettenti per la derivazione. Nel PG le rule class ci informano su quali relazioni e su quali attributi conviene cercare di estrarre delle regole.

Nella sezione che segue si inizia a descrivere il processo di creazione delle regole proposte. Verranno inizialmente descritte le euristiche che il sistema usa per generare le opportune RP. Poi verrà presentato l'algoritmo che ci permette di controllare quale informazioni, non presenti nel rule set, occorre cercare di estrarre dal database.



2.2 Definizione delle euristiche

2.2.1 Sguardo d'insieme

Come accennato l'acquisizione della conoscenza durante il processo di derivazione delle query, utilizza l'applicazione di euristiche. Tale approccio è stato proposto da diversi autori [(1)King J.J.,1981], [(10)Siegel, 1988], e prevede l'uso di un insieme di regole predefinite da applicare su query in modo da apprendere un certo numero di regole efficienti per l'ottimizzazione. Scopo delle euristiche è quello di creare regole semantiche che racchiudano parte della conoscenza intrinseca del database (conoscenza intensionale) e tali da potere essere utili nel generare query alternative a quelle originali. Queste query vengono successivamente passate all'ottimizzatore classico che ha la possibilità di generare migliori piani di accesso rispetto alla query iniziale. È quindi logico che per generare le euristiche vengono considerati concetti tipicamente propri dell'ottimizzazione classica, come l'accesso alle pagine dati attraverso un indice su un attributo o la risoluzione di un join utilizzando un particolare algoritmo, ecc. Un problema che emerge nell'identificare la conoscenza intrinseca è che le quantità di informazioni che possiamo estrarre dai dati sono

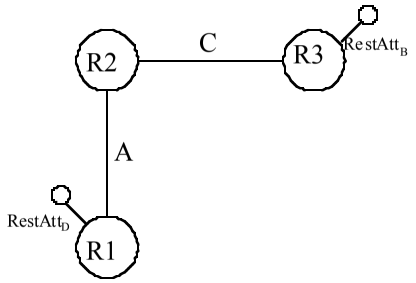
innumerevoli, e la scelta di quante e quali informazioni risultino essere utili al nostro scopo non è facile. L'approccio di tipo query-driven che verrà considerato è stato proposto in [(10) Siegel (1988)] e sfrutta le euristiche per identificare le situazioni più promettenti che possono generare quelle informazioni utili alla risoluzione di query. Cercare regole su attributi indicizzati, riconoscere vincoli ridondanti, diminuire il costo computazionale dell'esecuzione di un join inserendo degli opportuni vincoli, ed eliminare join dalla qualificazione di una query sono i principali criteri che sono espressi dall'euristiche.

È importante fare la seguente osservazione. L'applicazione delle euristiche non ha lo scopo di eseguire il processo di ottimizzazione semantica ma quello di generare il numero maggiore di regole che possono servire all'SQO e sarà l'SQO a decidere quali delle regole scoperte, devono essere utilizzate.

Consideriamo, ad esempio, che l'ottimizzatore debba ottimizzare la seguente query:

```
select R1.E
from R1, R2, R3
where R1.A = R2.A
and R2.C = R3.C
and R3.B = Y
and R1.D = Z;
```

Riportiamo anche il grafo della query per visualizzare meglio i collegamenti tra le diverse relazioni. Nel grafo le relazioni sono espresse dai nodi, mentre le etichette associate ai lati rappresentano gli attributi di join.



Vediamo ora come l'ottimizzatore può procedere per trasformare semanticamente la query. La prima strada che può seguire l'ottimizzatore è quella di cercare di eliminare dei join ridondanti, e quindi delle relazioni che non influiscono sul risultato della query. Un attributo della relazione R1 compare nella clausola select, quindi R1 non può essere eliminata. Si cerca di eliminare R3, ma questo è possibile solo se la restrizione $RestAtt_B$ è ridondante e si ha che $R2.C \subseteq R3.C$ (la dimostrazione è riportata più avanti). Se non vi sono regole semantiche per eliminare la relazione R3 si procede in altro modo. Se l'attributo R3.B non è indicizzato, si cerca di introdurre una restrizione su un attributo indicizzato della relazione R3. Si vede come è possibile applicare sia trasformazioni semantiche che eliminano restrizioni, sia trasformazioni semantiche che aggiungono restrizioni, su una stessa relazione.

Il sistema KA non sapendo a priori quali regole potranno essere scoperte, cerca di estrarre tutte le regole possibili che l'ottimizzatore semantico può applicare. É poi l'ottimizzatore a decidere quali regole applicare e quali no a secondo del caso.

Si ricordi anche che le regole semantiche, che sono state generate a partire dalle restrizioni di una query immessa da un utente, debbono essere utilizzate per ottimizzare query che verranno immesse nel sistema in futuro, e non per ottimizzare la query corrente.

2.2.2 Il linguaggio delle query

Facciamo alcune considerazioni sulle query che vengono considerate dal nostro sistema. In particolare vediamo il tipo di predicati che possiamo prendere in

considerazione. La query che un utente immette può presentare nella clausola where delle espressioni in cui abbiamo dei predicati P_i che possono essere collegati dagli operatori logici and e or. Eventualmente è possibile introdurre anche delle parentesi per cambiare la precedenza di applicazione degli operatori logici.

Nel proseguo le query potranno essere espresse attraverso il formato SQL o in modo più compatto nella seguente forma :

$$Q[T: p]$$

T è l'insieme degli attributi di output, corrispondenti agli attributi che compaiono nella clausola select nelle query espresse in SQL.

L'espressione p è del tipo

JP and JR

dove JP, join predicates, è una congiunzione di predicati di equijoin :

$$(R_1.A_1=R_2.A_2 \text{ and } \ldots \text{ and } R_{n-1}.A_{n-1}=R_n.A_n)$$

e JR, join restrictions, è un espressione del tipo

$$P_1 \text{ OL } P_2 \text{ OL } \ldots \text{ OL } P_n ,$$

con OL= and || or.

I predicati P_i possono essere del tipo :

Attributo Op v_1, con Op $\in \{>, \geq, <, \leq, =, \neq\}$

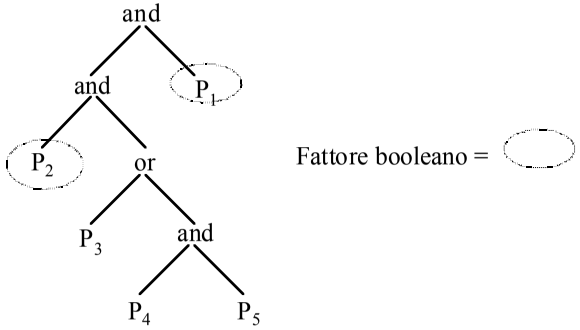
Attributo in $(v_1, \ldots\ldots, v_n)$

Attributo between v_1 and v_2

Consideriamo come esempio la seguente query:

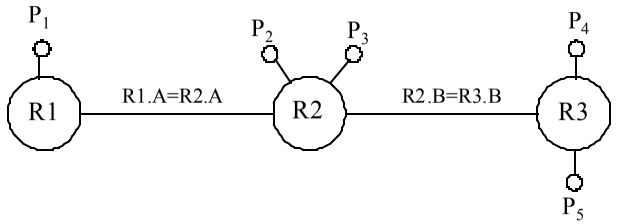
Q[T: (R1.A=R2.A) and (R2.B=R3.B) and P_1 and P_2 and (P_3 or (P_4 and P_5))]

Sotto è riportato l'albero dei predicati, dove è possibile vedere i fattori booleani della query. I predicati che nell'albero hanno un operatore or come antenato non sono fattori booleani.



Fattore booleano =

Riportiamo anche il query-graph nel quale i nodi rappresentano le relazioni coinvolte dalla query, e i lati i join colleganti le relazioni. Su ogni relazione sono riportati i predicati di selezione.



I fattori booleani sono i predicati P_1 e P_2.

Tra le euristiche che presenteremo, solo l'euristica H3 viene applicata sia ai predicati che sono fattori booleani che ai predicati che non lo sono. Le altre euristiche, H1, H2, H4, H5, considerano solo i predicati che sono fattori booleani. Anche l'algoritmo CLOSURE che useremo per calcolare la chiusura dei predicati della query, considera solo i predicati che sono fattori booleani. Quindi immessa una query generica del tipo Q[T: p], il sistema identificherà quali sono nell'espressione p i predicati fattori booleani e quelli che non lo sono. Poi viene generata una query Q[T:q], dove q è una qualificazione congiuntiva, fatta da tutti i predicati che sono fattori booleani dell'espressione p. La qualificazione q comprende anche i predicati di join.

2.2.3 Il linguaggio delle regole

Considereremo delle regole nella forma

$$A \rightarrow p_n$$

dove A è l'antecedente della regola, ed è un espressione del tipo $p_1 \wedge p_2 \wedge . . . . . \wedge p_{n-1}$. P_n è il conseguente della regola.

Dove i p_i, con $1 \leq i \leq n$, sono dei predicati del tipo

1) Attributo Op v_1, con Op $\in \{>, \geq, <, \leq, =, \neq\}$
2) Attributo in $(v_1, \ldots\ldots, v_n)$
3) Attributo between v_1 and v_2
4) $\text{Attributo}_i = \text{attributo}_j$

Una regola può essere espressa anche nella seguente forma:

$$(\text{JP and C}) \rightarrow b$$

dove nell'antecedente sono stati separati i predicati di join dai predicati si selezione. JP riunisce i join, in forma congiuntiva, dell'antecedente della regola

$$(R_1.A_1=R_2.A_2 \text{ and } \ldots \text{ and } R_{n-1}.A_{n-1}=R_n.A_n)$$

e C riunisce i predicati di selezione, del tipo 1), 2), 3) dell'antecedente, sempre in forma congiuntiva.

2.2.4 Euristica H1*

Se in una query una relazione R ha un attributo ristretto A e un attributo con indice clustered non ristretto B, allora cerca una regola dove la restrizione su A implica una restrizione su B

-Euristica H1-

L'euristica H1 è detta 'Euristica di introduzione' poiché cerca di inserire in una query un predicato su un attributo indicizzato. La query equivalente così ottenuta potrà essere risolta con un costo minore dato che adesso è possibile accedere alla relazione R con l'indice clustered sull'attributo B. L'euristica di introduzione è una delle più importanti poiché può apportare un risparmio considerevole nell'esecuzione della query.

Per tutti i predicati che sono anche fattori booleani nella query applichiamo l'euristica H1. Tale proposizione prevede di cercare, per ogni relazione che compare nella query, un attributo che sia ristretto, quindi presente all'interno dell'albero dei predicati. Si dovrà in seguito cercare quali sono gli attributi con indice di tale relazione. Se esistono tali attributi indicizzati potremo generare la regola proposta. La regola proposta viene costruita nel seguente modo. L'antecedente della regola sarà dato dal predicato corrente, mentre il conseguente è dato solo dall'attributo sul quale abbiamo l'indice.

Come esempio consideriamo la seguente query :

- Nel seguito gli esempi verranno riferiti al seguente schema relazionale: LIBRO(Cod_lib, Titolo, Autore, Soggetto, Anno, Collocazione), RIVISTA(Titolo, Numero, Soggetto, Anno, Collocazione). Con indici costruiti su LIBRO.Cod_lib, RIVISTA.Titolo, RIVISTA.Numero

```
select LIBRO.Titolo
from LIBRO                                    (Q1)
where LIBRO.Collocazione = 20 ;
```


La regola proposta generata è di questo tipo :

RP1	*Attributo*	*Vincolo*
Antecedente	LIBRO.Collocazione	(= 20)
Conseguente	LIBRO.Cod_lib	

Abbiamo dunque che nel conseguente non c'è nessuna restrizione ma solo l'attributo che bisogna 'sondare' per stabilire se vi è una relazione favorevole tra la restrizione nell'antecedente e l'attributo del conseguente.
Un esempio di regola derivata che incontra la regola proposta RP1 è la regola RD1.

RD1	*Attributo*	*Vincolo*
Antecedente	LIBRO.Collocazione	(= 20)
Conseguente	LIBRO.Cod_lib	(> 310)

La query iniziale potrà così essere trasformata nella seguente query :

```
select LIBRO.Titolo
from LIBRO                                    (QE1)
where LIBRO.Collocazione = 20
and LIBRO.Cod_lib > 310 ;
```

Indipendentemente dal modello dei costi che il sistema database utilizza per l'ottimizzazione delle interrogazioni, è possibile stabilire che il costo di esecuzione di Q1, Ca(Q1) è sicuramente superiore a quello di QE1.

2.2.5 Euristica H2

Se la sola restrizione sulla relazione R è una condizione di join, e R non ha un indice sull'attributo di join, allora cerca una restrizione su questa relazione. La nuova condizione deve essere dedotta dalla relazione di join.

-Euristica H2-

Tale euristica si basa sul fatto che il costo di esecuzione di un join risulta essere inferiore se esistono restrizioni su attributi appartenenti alla relazione coinvolta nel join, in quanto esse permettono generalmente di considerare un numero inferiore di tuple di R nell'esecuzione dell'algoritmo di join. L'euristica H2 viene applicata ai predicati semplici del query-tree che sono anche fattori booleani.
Le query che dunque interesseranno l'euristica H2 saranno di questo tipo :

```
select  RIVISTA.Titolo,LIBRO.Titolo
from RIVISTA,LIBRO                                          (Q2)
where LIBRO.Autore = 'Neruda Pablo'
and RIVISTA.Soggetto = LIBRO.Soggetto ;
```

L'euristica H2 può essere applicata a questa query perché, a parte il predicato di join, non c'è un'altra restrizione sulla relazione RIVISTA e inoltre sull'attributo di join considerato non esiste un indice.
Quando l'euristica H2 è applicata alla query restituisce 5 regole proposte, una per ogni attributo della relazione RIVISTA. Una di queste regole proposte è la seguente :

RP2	*Attributo*	*Vincolo*
Antecedente	LIBRO.Autore	(= 'Neruda Pablo')
Conseguente	RIVISTA.Numero	

Le altre regole proposte sono simili, eccetto che per l'attributo Numero il quale è rimpiazzato dagli attributi Soggetto, Titolo, Collocazione, Anno.
Anche in questo caso abbiamo che nel conseguente non c'è nessuna qualificazione, ma solo l'attributo che bisogna 'sondare' per stabilire se vi è una relazione favorevole tra la qualificazione nell'antecedente e l'attributo del conseguente.
La regola proposta RP2 potrebbe incontrare la regola derivata RD2 :

RD2	*attributo*	*vincolo*
antecedente	LIBRO.Autore	(= 'Neruda Pablo')
conseguente	RIVISTA.Numero	(> 300)

e quindi inserire nella query iniziale il conseguente di tale regola. Si viene così a generare la seguente query :

```
select  RIVISTA.Titolo,LIBRO.Titolo
from RIVISTA,LIBRO
where LIBRO.Autore = 'Neruda Pablo'                         (QE2)
and RIVISTA.Soggetto = LIBRO.Soggetto
and RIVISTA.Numero > 300 ;
```

Possiamo in generale ritenere che Ca(QE2)<Ca(Q2).

2.2.6 Euristica H3

Se esiste una restrizione su un attributo con indice che non è fattore booleano, allora cerca di rendere tala restrizione un fattore booleano.

- Euristica H3 -

H3 è un *vincolo euristico di rimozione* che tenta di rimuovere particolari condizioni di selezione ridondanti. Vediamo come questo tipo di trasformazione può comportare un miglioramento. Si suppone di considerare un algoritmo di accesso ai dati di una interrogazione che utilizza al più un indice. Con l'algoritmo che usa al più un indice, si sceglie un indice tra quelli costruiti su attributi che compaiono in predicati di ricerca ; successivamente si decide se è preferibile utilizzare l'indice o se procedere con la scansione sequenziale. I predicati di ricerca presi in considerazione devono però soddisfare l'ulteriore condizione di essere fattori booleani. Quindi se in una query compare un espressione del tipo "where $Rest_1$ and...and (R1.A=v1 or R1.B=v2)", le due condizioni di selezione sull'attributo R1.A e R1.B non verrebbero prese in considerazione dall'algoritmo di ricerca. Eliminando però una delle due condizioni avremo che l'algoritmo può prendere in considerazione la condizione rimanente.

L'euristica H3 restituisce una regola proposta con l'antecedente e il conseguente vincolati entrambi. Il campo di applicazione di tale regola comprende sia i predicati che non sono fattori booleani e sia i predicati che lo sono. I predicati che verranno inseriti come antecedenti sono per la query fattori booleani, mentre i predicati inseriti come conseguenti non lo sono.

Consideriamo come esempio la seguente query :

```
select RIVISTA.Soggetto
from RIVISTA                                        (Q3)
where RIVISTA.Anno > 1995
```



```
and ( RIVISTA.Numero < 6  or  RIVISTA.Collocazione > 123 );
```

L'applicazione dell'euristica H3 restituisce la seguente regola proposta :

RP3	*Attributo*	*Vincolo*
Antecedente	RIVISTA.Anno	(> 1995)
Conseguente	RIVISTA.Collocazione	(> 123)

Supponiamo che si abbia una regola RD3.

RD3	*attributo*	*vincolo*
antecedente	RIVISTA.Anno	(> 1994)
conseguente	RIVISTA.Collocazione	(> 150)

Avremo che la regola proposta RP3 match la regola RD3. Allora la restrizione che compare come conseguente della regola PR3 può essere rimossa dalla query.
La query equivalente che risulta da questa trasformazione è la query QE3.

```
select RIVISTA.Soggetto
from RIVISTA                                    (QE3)
where RIVISTA.Anno > 1995
and RIVISTA.Numero < 6 ;
```

Avendo un indice sull'attributo Numero avremo un predicato di ricerca che è anche fattore booleano per la query QE3. Ciò provocherà un significativo riduzione del costo stimato di accesso della query QE3, rispetto al costo stimato per la query Q3, per la quale non era possibile considerare l'accesso con l'indice su Numero.
In generale possiamo considerare Ca(QE3) << Ca(Q3).

2.2.7 Euristica H4

Se una relazione R1 è presente con un join (R1.A = R2.A), e se nessuno degli attributi di R1 compare nella clausola select della query, ad eccezione di A, allora cerca di verificare che R2.A $\subseteq$ R1.A.

-Euristica H4-

Questa euristica si propone di generare una regola per eliminare i join che non sono necessari alla risoluzione della query. L'eliminazione di un join comporta generalmente un elevato risparmio, poiché esso risulta essere una delle operazioni più costose da eseguire nella risoluzione di una query. Vediamo le condizioni che ci permettono di eliminare un singolo join (R1 join R2). Se il predicato di join eliminato era l'unico predicato nel quale compariva la relazione R1 nella query, allora anche la relazione R1 viene eliminata.
Consideriamo la seguente proposizione.
Proposizione : Supponiamo che R2 sia la sola relazione con cui è collegata R1 nella query Q[T :q] ; dove T è l'insieme degli attributi *target,* cioè gli attributi che compaiono nella clausola select, e q è una qualificazione congiuntiva.
Q[T : q] è semanticamente equivalente a Q′[T′ : q - $Rest_{R1}$(q) - {R1.A=R2.A}], se e solo se

1. R1.A $\supseteq$ R2.A
2. $Rest_{R1}$(q) = $\varnothing$ o $Rest_{R1}$(q) sono ridondanti
3. Attributi(R1) $\cap$ T $\subseteq$ {R1.A}

dove T′ è T se R1.A $\notin$ T ; altrimenti T′ e (T - {R1.A}) $\cup$ {R2.A}, e $Rest_{R\ 1.A}$ è l'insieme di restrizioni specificate su R1.

Queste tre condizioni risultano essere condizioni necessarie e sufficienti per l'eliminazione di un join [(45)Yu. 1994]. Proviamo solo la condizione sufficiente.

La condizione 3. attributi(R1) $\cap$ T(q) $\subseteq$ {R1.A} implica T(q) $\cap$ Attributi(R1) = $\varnothing$ o T(q) $\cap$ Attributi(R1) = {R1.A}. Nel secondo caso, l'attributo R1.A in Target(q) può essere shifted con R2.A sfruttando l'equijoin. Se la condizione 2. è soddisfatta, allora $Rest_{R1}$ possono essere eliminate mantenendo l'equivalenza della query Q. Conseguentemente, dalla condizione 1, il join (R1.A=R2.A) è chiaramente non necessario, e può così essere rimosso.

Si osservi che la condizione R1.A$\supseteq$R2.A è sempre verificata se R2.A è foreign key di R1.A.

L'eliminazione del join comporta la rimozione della relazione R1 dalla query. Questo è il caso nel quale è stata scoperta la *ridondanza della relazione R1*, che si verifica quando nessuno dei suoi attributi o restrizioni contribuiscono alla risposta della query.

Si noti che uno dei requisiti per eliminare una relazione da una query, è che le restrizioni in cui compare un attributo di tale relazione, devono essere ridondanti, e sia quindi possibile eliminarle. Si dovrà quindi successivamente applicare l'euristica H5 che cerca di generare delle regole che eliminino tali attributi.

Consideriamo il seguente esempio.

```
select RIVISTA.Titolo
from LIBRO, RIVISTA                                    (Q4)
where LIBRO.Soggetto = RIVISTA.Soggetto
and RIVISTA.Collocazione = 231 ;
```

La query Q4 coinvolge le due relazioni LIBRO e RIVISTA, le quali sono collegate dall'equijoin sull'attributo Soggetto. Sono inoltre verificate le ipotesi di applicazione dell'euristica H4 che risultano essere le condizioni 2) e 3) della proposizione vista. È

possibile applicare l'euristica in modo da verificare anche l'ipotesi 1), cosicché si possa eliminare il join tra le relazioni LIBRO e RIVISTA.

La conseguente regola proposta generata è la seguente.

RP4	*Attributo*	*Vincolo*
Antecedente	RIVISTA.Soggetto	----------
Conseguente	LIBRO.Soggetto	----------

Nella regola proposta RP4 compaiono solo i nomi degli attributi di join, senza che vi sia alcun riferimento ad un qualche vincolo. Tale regola afferma che si deve cercare di verificare la condizione RIVISTA.Soggetto $\subseteq$ LIBRO.Soggetto. Se la condizione viene verificata verrà generata la seguente regola derivata.

RD4	*Attributo*	*vincolo*
antecedente	RIVISTA.Soggetto	----------
conseguente	LIBRO.Soggetto	----------

Si osservi come la regola derivata RD4 sia identica alla regola proposta RP4. Se la regola RP4 sta' ad indicare che occorre verificare la condizione di superset dell'insieme dei valori di LIBRO.Soggetto sull'insieme dei valori RIVISTA.Soggetto; la creazione della regola RD4 indica che tale verifica ha avuto esito positivo, e quindi che la regola RP4 ha ora dignità di regola derivata, cioè utilizzabile nel processo di ottimizzazione.
La query Q4 può essere trasformata nella query equivalente QE4.

```
select RIVISTA.Titolo
from RIVISTA                                        (QE4)
and RIVISTA.Collocazione = 231 ;
```

Evidentemente avremo che Ca(QE4) < Ca(Q4).

2.2.8 Euristica H5

Se una relazione R1 è presente in una query con un equijoin e delle restrizioni $Rest_i(R1)$, e R1 non ha attributi target che non appartengono a equijoin , allora cerca delle regole che rendano ridondanti le restrizioni $Rest_i(R1)$.

-Euristica H5-

Con l'euristica H4 si cercava di trovare delle regole che verificassero una delle condizioni necessarie per l'eliminazione di un join. Prendendo come riferimento il join (R1.A = R2.A) la condizione è espressa con la relazione (R1.A$\supseteq$R2.A). Eliminando il join viene anche eliminata la relazione R1 dalla query, quindi è necessario che se la relazione R1 ha degli attributi ristretti $Rest_{R1}$, questi debbano essere ridondanti rispetto alla chiusura della qualificazione della query. L'euristica H5 si propone di trovare delle regole che permettano di accertare la ridondanza delle restrizioni $Rest_{R1}$ in modo da poterle eliminare e verificare così tutte le condizioni necessarie per l'eliminazione della relazione R1, e dunque anche del join (R1.A = R2.A) dalla query.
Le due euristiche H4 e H5 contribuiscono entrambe alla trasformazione semantica equivalente che permetta di eliminare un join. È stato scelto di non considerare un unica euristica perché può accadere che, una regola del tipo H4, cioè che verifichi (R1.A $\supseteq$ R2.A) sia già presente nell'insieme del rule set, mentre non sono presenti delle regole del tipo H5, cioè quelle che accertano la ridondanza delle restrizioni $Rest_{R1}$, o viceversa. Avendo a disposizione le due euristiche H4 e H5 sarà possibile, durante la fase di SQO, richiamare l'euristica specifica che generi la regola di cui si ha bisogno. Come esempio si consideri la query Q5, simile alla query Q4 vista precedentemente, a cui è stata aggiunta una restrizione sulla relazione LIBRO ; relazione che si vuole provare ad eliminare. Si supponga che alla query Q4 sia già stata applicata l'euristica H4 e che essa abbia generato una regola che afferma che

RIVISTA.Soggetto ⊆ LIBRO.Soggetto. Sono così verificate tutte le ipotesi che permettono l'applicazione dell'euristica H5.

```
select RIVISTA.Titolo
from LIBRO,RIVISTA                                    (Q5)
where LIBRO.Soggetto = RIVISTA.Soggetto
and RIVISTA.Collocazione = 231
and LIBRO.Anno = 1995;
```

la regola proposta generata è la RP5.

RP5	***Attributo***	***Vincolo***
Antecedente	RIVISTA.Collocazione	(= 231)
and	LIBRO.Soggetto	(= RIVISTA.Soggetto)
Conseguente	LIBRO.Anno	(= 1955)

Se questa regola genera una regola derivata che verifica la ridondanza della restrizione (LIBRO.Anno = 1995) sarà possibile trasformare la query Q5 nella query semanticamente equivalente QE5.

```
select RIVISTA.Titolo
from LIBRO,RIVISTA                                    (QE5)
where LIBRO.Soggetto = RIVISTA.Soggetto
and RIVISTA.Collocazione = 231 ;
```

La query QE5 può essere trasformata nella query QE5' eliminando da essa la relazione LIBRO, e quindi il join (LIBRO.Soggetto = RIVISTA.Soggetto).


```
select RIVISTA.Titolo
from RIVISTA                                          (QE5')
where  RIVISTA.Collocazione = 231 ;
```

Se nella clausola *select* invece dell'attributo RIVISTA.Titolo, ci fosse stato l'attributo LIBRO.Soggetto, l'eliminazione del join era comunque possibile.

```
select LIBRO.Soggetto
from LIBRO,RIVISTA
where LIBRO.Soggetto = RIVISTA.Soggetto
and RIVISTA.Collocazione = 231 ;
```

Occorre solo sostituire l'attributo target LIBRO.Soggetto con l'attributo RIVISTA.Soggetto, senza che ciò alteri la risposta dell'interrogazione.

```
select RIVISTA.Soggetto
from RIVISTA
where RIVISTA.Collocazione = 231 ;
```

2.3 Processo per la creazione delle regole proposte

Introduzione

Nella sezione 2.3 si vuole proporre un estensione del metodo di [(56)Siegel M. (1992)] che può essere così riassunto. Prima di attivare il processo di derivazione automatica delle regole, vengono aggiunte alla query Q in questione, tutte quelle restrizioni non esplicitamente date, ma implicate logicamente da Q e dalle regole presenti nello schema (insieme RULE SET). Vediamo con un semplice esempio, quali sono i vantaggi di questo nuovo approccio.

Si consideri le relazioni R1($\underline{A}_1$, A_2, A_3, A_4), R2(B_1, B_2, B_3,), dove l'attributo sottolineato $\underline{A}_1$ è un attributo indicizzato, e la seguente query:

select R1.A_3
from R1, R2
where R1.A_2=10
and R1.A_3=R2.B_3
and R2.B_1<40;

Supponiamo che nel sistema sia presente la seguente regola:

$$\text{Reg1)}\ (R1.A_2>9) \rightarrow (R1.A_3=8),$$

Applicando le euristiche[1] alle restrizioni di Q otterremo l'insieme S costituito dalle seguenti regole proposte:

RP1(Q)) $(R1.A_2=10) \rightarrow \underline{A}_1)$

RP2(Q)) $(R1.A_2=10) \wedge (R1.A_3=R2.B_3) \rightarrow (R2.B_1<40)$

[1] Per semplificare l'esempio si sono consideratwe solo l'euristiche H1, H2, H5

RP3(Q)) $(R2.B_1<40) \wedge (R1.A_3=R2.B_3) \rightarrow (R1.A_2=10)$

Le restrizioni di Q implicano la regola reg1, è possibile così considerare l'insieme S* formato dalle restrizioni di Q più il conseguente della regola reg1.

Se applichiamo all'insieme S* le euristiche[1] otteniamo le seguenti regole proposte:

RP1(Q)) $(R1.A_2=10) \rightarrow \underline{A}_1)$

RP2(Q)) $(R1.A_2=10) \wedge (R1.A_3=R2.B_3) \rightarrow (R2.B_1<40)$

RP3(Q)) $(R2.B_1<40) \wedge (R1.A_3=R2.B_3) \rightarrow (R1.A_2=10)$

RP4(Q)) $(R1.A_3=8) \rightarrow A_1$

RP5(Q)) $(R1.A_3=8) \wedge (R1.A_3=R2.B_3) \rightarrow (R2.B_1<40)$

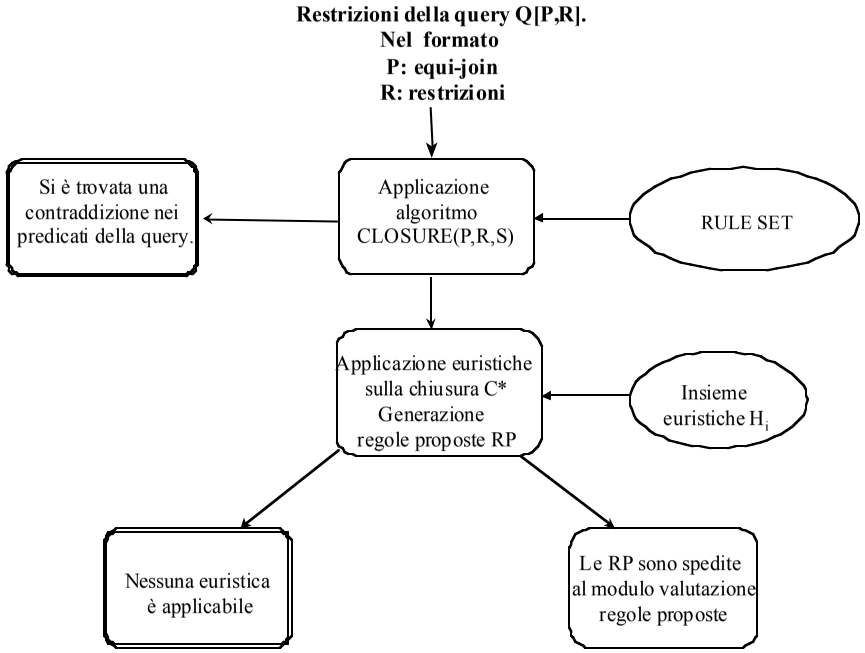
I vantaggi dell'utilizzo di questa metodologia per la generazione delle regole proposte rispetto a quella seguita da Siegel sono due:

1) Con Siegel verrebbero generate solo le regole proposte dell'insieme S, mentre con questa metodologia si generano le regole proposte dell'insieme S*.
2) L'ottimizatore di Siegel è organizzato in modo tale da effettuare l'applicazione dell'euristiche e la successiva fase di match, più volte per estrarre l'intero insieme di regole proposte. Nel nostro sistema le regole proposte vengono applicate una sola volta.

[1] Visto che la restrizione (R2.B_1<40) è implicata dalle restrizioni di Q, e quindi è semanticamente ridondante rispetto a Q, non è stata generata nessuna regola proposta per la sua eliminazione.

2.3.1 Processo per la creazione delle regole proposte

Le query immesse nel sistema dall'utente, vengono spedite dall'SQO al modulo "Creazione regole proposte". Le query vengono rappresentate nella seguente forma: Q[P, R], dove P è l'insieme dei join della query e R è l'insieme delle restrizioni. Nella figura sotto sono rappresentate le principali fasi che vengono eseguite nel modulo. La prima fase consiste nel calcolare la chiusura delle restrizioni che compaiono nella query, rispetto all'insieme delle regole rule set. Per comodità, chiameremo il rule set, insieme S.



La chiusura di un insieme di restrizioni R, rispetto ad un insieme di regole S e un insieme di equi-join P, è composto da tutte le possibili restrizioni implicate o dedotte da R attraverso S e P. La chiusura verrà denotata con C*(R, P, S). Nella chiusura

C*(R, P, S) troviamo dunque tutte le possibili restrizioni di S che possiamo applicare alla query Q[P, R]. La chiusura viene calcolata dall'algoritmo CLOSURE(R, P, S). Quest'algoritmo, oltre alla chiusura di R, è in grado di verificare se esistono delle contraddizioni nella qualificazione della query rispetto alla conoscenza in S. Se viene scoperta una contraddizione significa che la risposta dell'interrogazione è nulla, quindi il processo di ottimizzazione si ferma. La fase successiva all'applicazione dell'algoritmo CLOSURE, è di verificare quali euristiche è possibile applicare alle restrizioni R.

2.3.2 Algoritmo per il calcolo della chiusura di un insieme di restrizioni

L'obbiettivo che si ci propone, è quello di generare un algoritmo che possa calcolare la chiusura di un insieme di restrizioni R, rispetto ad un certo insieme di regole S. Nella chiusura si troveranno così tutte le restrizioni che possiamo dedurre da R, attraverso la conoscenza posta in S.

Supporremo che tutti i predicati di join e i predicati di selezione di una qualificazione congiuntiva q siano raggruppati, rispettivamente, nell'insieme JP(q) e Res(q). Data la regola (JC and A) $\rightarrow$ b, chiameremo JC l'insieme delle *precondizioni di join*.

Consideriamo alcune caratteristiche dei predicati di join. Due differenti insieme di predicati di join possono essere equivalenti se essi conducono allo stesso risultato. Per esempio, ($R_1.X = R_2.Y$ and $R_2.Y = R_3.Z$) è equivalente a ($R_1.X = R_3.Z$ and $R_2.Y = R_3.Z$). Perciò, dato un qualsiasi insieme di predicati di join possiamo suddividerlo in classi di equivalenzaindotte dalle relazioni di equivalenza.

Definizione : La *chiusura dei join* di un insieme di predicati di join P, chiamata JP*(P), è la partizione di tutti gli attributi di join, dedotti da P, all'interno di classi di equivalenza indotte dalle relazioni di equijoin. JP*(P) = {$Equi\_Set_1$, $Equi\_Set_2$, . . . ,$Equi\_Set_k$) dove ogni $Equi\_Set_i$, $1 \leq i \leq k$, è una *classe di equivalenza*. Cioè, per ogni R.A, S.Y $\in$ $Equi\_Set_i$, con $1 \leq i \leq k$, R.X = S.Y appartiene a P o è implicato da P ; se

$R.X \in Equi\_Set_i$, $S.Y \in Equi\_Set_j$, e $i \neq j$, allora $R.X = S.Y$ non è ne in P e nemmeno è implicato da P.

Possiamo immediatamente osservare il seguente lemma

Lemma : Due insiemi di predicati di join JP1 e JP2 specificano le stesse operazioni di join se e solo se JP*(JP1) = JP*(JP2).

Esempio :

1) $JP^*(\{R_1.A = R_2.B, R_2.B = R_3.C, R_1.E = R_4.F\}) = \{\{R_1.A, R_2.B, R_3.C\}, \{R_1.E, R_4.F\}\}$.
2) Supponendo che $JP_1 = \{R_1.X = R_2.Y, R_2.Y = R_3.Z\}$ e $JP_2 = \{R_1.X = R_3.Z, R_2.Y = R_3.Z\}$ allora $JP^*(JP_1) = JP^*(JP_2) = \{\{R_1.X, R_2.Y, R_3.Z\}\}$. Quindi esse specificano la stessa operazione di join, sebbene in forma differente.

Generazione della chiusura di un insieme di restrizioni

Dato un insieme di predicati di join P e un insieme di restrizioni R (esse possono essere dedotte da una query la cui qualificazione è q, $q = R \cup P$) e sotto una conoscenza base S, si vuole dedurre tutte le possibili restrizioni da P e R rispetto ad S. Possiamo considerare due tipi di restrizioni che possono essere dedotte.

1) *Restrizioni propagate.* Queste sono restrizioni deducibili dalle restrizioni conosciute, usando solo i predicati di equijoin P. Per esempio la restrizione propagata $R_i.A < 30$ deriva dalla restrizione $R_j.A < 30$ e dalla condizione di equijoin $R_i.A=R_j.A$.
2) *Restrizioni dedotte.*. Queste sono le restrizioni derivate attraverso l'applicazione di qualche regola. Per esempio $R_i.Z < 30$ può essere dedotta dal predicato di join $R_i.X=R_j.Y$ e dalla restrizione $R_j.Y < 40$ e applicando la regola $\{ R_i.X=R_j.Y$ and $R_j.Y < 50 \rightarrow R_i.Z < 30\}$. La condizione $R_j.Y < 50$ è soddisfatta da $R_j.Y < 40$.

Di seguito si discuterà sulla deduzione di restrizioni propagate e sulla soddisfacimento dei predicati di join delle regole.

Consideriamo il seguente lemma che ci permette di verificare quando un predicato di

precondizione è soddisfatto da un insieme di predicati di equijoin.

Lemma : la precondizione di join JC di una regola (JC → A) → B è *soddisfatta* da un insieme di predicati di equijoin P sse per ogni $Equi\_Set_i \in JP^*(JC)$, esiste un $Equi\_Set_j \in JP^*(P)$ tale che $Equi\_Set_i \subseteq Equi\_Set_j$.

Questo lemma afferma che, dopo aver calcolato la chiusura di join di JC e P, se ogni Equi_Set nella chiusura di JC risulta essere un sottoinsieme di una qualche classe della chiusura di join di P, allora la precondizione di join di JC della regola è soddisfatta dai predicati di equijoin di P.

Basandoci su questo lemma, possiamo costruire l'algoritmo JOIN_PRE(P*, JC), il quale accetta un insieme di predicati di join P (questi possono essere l'insieme dei predicati di join della qualificazione di una query), e i join di precondizione JC di una regola (JC → A) → B, e ritorna TRUE se la condizione di join di questa regola è soddisfatta da P ; altrimenti ritorna FALSE.

Algoritmo JOIN_PRE(P*, JC)

```
begin
   P* = {Equi_Set_P1, Equi_Set_P2, . . . , Equi_Set_Pk}
   JP*(JC) = EQUI_CLASS(JC) = {Equi_Set_JC1, Equi_Set_JC2, . . . , Equi_Set_JCn}
   for i = 1 to n
        if there exists {Equi_Set_Pj | Equi_Set_JCi ⊆ Equi_Set_Pj, Equi_Set_Pj ∈ P* }
           then continue ;
        else return FALSE ;
   return TRUE ;
end
```

Consideriamo le deduzioni delle restrizioni propagate per un dato insieme di restrizioni R sotto un dato insieme di predicati di join P. Per ogni restrizione X, X∈R nella forma (T.A op c), l'insieme delle restrizioni propagate da X sotto P è dato dall'insieme delle restrizioni (S.B op c) tale che T.A, $S.B \in Equi\_Set_k \in JP^*(P)$. L'insieme delle restrizioni propagate per ogni X, X∈R, costituisce le restrizioni

propagate per R, sotto P e p, denotato con PROPAGATE(R, P). Dunque qualsiasi restrizione specificata su T.A, sarà propagata a tutti gli attributi all'interno della stessa classe di equivalenza contenete l'attributo T.A nella chiusura di join. Sarà sufficiente, quindi, mantenere una sola copia di tutte le restrizioni della classe di equivalenza contenete T.A.

Esempio. Si consideri la seguente qualificazione q :

$R_1.A_1 = R_2.A_2$ and $R_3.A_3 = R_2.A_2$ and $R_1.A_4 = R_4.A_5$ and $R_1.A_1 \leq 22$ and $R_3.A_3 > 10$ and $R_1.A_6 \neq 0$.

La sua chiusura di join può essere calcolata come $JP^*(JP(Q)) = \{\{R_1.A_1, R_2.A_2, R_3.A_3\}, \{R_1.A_4, R_4.A_5\}\}$. Se abbiamo una regola $R_1.A_1 = R_3.A_3$ and A → B (con JC = $(R_1.A_1 = R_3.A_3)$), la sua chiusura di join è $JP^*(JC) = \{\{R_1.A_1, R_3.A_3\}\}$. Dato che $\{R_1.A_1, R_3.A_3\} \subset \{R_1.A_1, R_2.A_2, R_3.A_3\}$, la precondizione di join di questa regola è soddisfatta nonostante che JC $\not\subset$ JP(q). Dalla restrizione $R_1.A_1 \leq 22$ in q, verrebbero prodotte le restrizioni $R_2.A_2 \leq 22$ e $R_3.A_3 \leq 22$ dato che $R_2.A_2$, $R_3.A_3$ sono nella stessa classe di equivalenza ; mentre non possiamo ottenere nessuna restrizione propagata da $R_1.A_6 \neq 0$. In pratica noi possiamo mantenere, per la classe di equivalenza $\{R_1.A_1, R_2.A_2, R_3.A_3\}$ solo le restrizioni generiche W.A ≤ 22, W.A < 10, dove W.A può essere rimpiazzata da un qualsiasi attributo all'interno della classe di equivalenza.

2.3.2.1 Conoscenza del dominio di un database

Nell'analisi della conoscenza e della sua rappresentazione all'interno di un database, è importante considerare anche la conoscenza del dominio del database. Effettuiamo alcune considerazioni su tale argomento, in modo da poter introdurre particolari rappresentazioni, e algoritmi, che possono variare a seconda del tipo di dominio dei valori, a cui queste strutture fanno riferimento.

In generale possiamo considerare due tipi di dominii in un sistema database.

1) *Dominii non ordinati* : Se D è un dominio non ordinato e x, y sono elementi di D, allora avremo che o $x = y$ o $x \neq y$. Per esempio, in un dominio concernete colori, noi siamo interessati a sapere solo se due colori sono uguali o no. Non esiste un ordinamento tra i colori. Restrizioni specificate su dominii non ordinati utilizzano solo operatori di uguaglianza e disuguaglianza ($=, \neq$).

2) *Dominii ordinati*: Se D è un dominio ordinato e x, y sono elementi di D allora uno e solo uno delle seguenti comparazioni può essere vera, $x < y$, $x = y$, o $x > y$. Un tipico dominio di questo tipo coinvolge i numeri interi e i numeri reali. Il primo è un *dominio ordinato discreto*, e il secondo è un *dominio ordinato continuo*. Restrizioni specificate su dominii ordinati possono usare i seguenti operatori ($<, \leq, >, \geq, =, \neq$).

Supponiamo che R sia un insieme di restrizioni. Alcune restrizioni possono essere ridondanti. Per esempio, se entrambe $W.A < 3$ e $W.A < 9$ sono presenti in R, allora $W.A < 9$ è ridondante. Un'altro esempio è che se $W.A < 3$, $W.A > 0$, $W.A \neq 1$, e $W.A = 2$ sono presenti in R, e il dominio di W.A è un dominio di interi (un dominio discreto), allora $W.A=2$ è ridondante rispetto alle altre restrizioni (e viceversa). In questo ultimo esempio, se il dominio fosse stato un insieme di numeri reali (dominio continuo), allora $W.A = 2$ non sarebbe stato ridondante.

Come mostrato sopra il dominio ordinato coinvolge più operatori del dominio non ordinato. Perciò sarà sufficiente considerare le caratteristiche dei domini ordinati. In particolare consideriamo un dominio ordinato discreto come un dominio di numeri interi.

All'interno del dominio intero, le restrizioni $W.A \geq c$, $W.A \leq c$, and $W.A = c$ sono rispettivamente equivalenti a $W.A > c - 1$, $W.A < c + 1$ e ($W.A < c + 1$, $W.A > c - 1$). Dunque è sufficiente considerare l'insieme degli operatori $\{<, >, \neq\}$. Le considerazioni che si faranno ora , riguardano le restrizioni, appartenenti ad un insieme R che sono specificate nella classe di equivalenza contenete W.A. L'insieme di restrizioni R, su una classe di equivalenza contenente W.A, può essere rappresentato da un intervallo (c_{low}, c_{high}), che chiameremo *intervallo minimo*, ed eventualmente da una lista di disuguaglianze $W.A \neq c$, dove c è interno all'intervallo. Per tutte le

restrizioni di R aventi l'operatore <, come ad esempio : $W.A < c_1$, $W.A_2 < c_2$, . . . ,$W.A > c_k$, definiamo il valore $c_{high} = \min\{c_1, c_2, \ldots, c_k\}$. Analogamente definiamo, rispetto a tutte le restrizioni in R, aventi l'operatore <: $W.A < v_1$, $W.A < v_2$, . . . , $W.A < v_h$, il valore $c_{low} = \max\{v_1, v_2, \ldots, v_h\}$. Naturalmente le disuguaglianze in R, del tipo $W.A \neq c$, con $c \leq c_{low}$ o $c \geq c_{high}$ sono ridondanti rispetto all'intervallo (c_{low}, c_{high}), quindi possono essere eliminate. Osserviamo inoltre che, se abbiamo una disuguaglianza $W.A \neq c$, con $c = c_{low} + 1$, allora possiamo ridurre il nostro intervallo prendendo come valore inferiore $c_{low} = c_{low} + 1$. E' possibile procedere in modo analogo con il valore superiore c_{high}. Chiameremo MINRest(W.A, R) il minimo insieme di restrizioni di disuguaglianza e l'intervallo minimo per W.A in R. Possiamo considerare una procedura MINInterval(R) che, se non si rilevano contraddizioni tra i vincoli di R, ritorna MINRest(W.A, R) delle restrizioni presenti in R.

Vediamo ora quando una restrizione K della forma (W.A op c), op $\in \{ <, >, =\}$ è implicata da un insieme di restrizioni T, usando la conoscenza del dominio. È chiaro che questo tipo di implicazione coinvolge solo restrizioni specificate sugli attributi di una stessa classe di equivalenza. Usando la rappresentazione minima descritta sopra, noi possiamo determinare se K è implicato da T nel seguente modo:

1) se K è uguale a $W.A \neq c$: K è implicato solo se una delle seguenti tre condizioni è vera :
   a) $W.C \neq c \in$ MINRest(W.A, T),
   b) $c_{minlow} \geq c$,
   c) $c_{minhigh \leq c}$
2) K è uguale a $W.A < c$: K è implicato sse $c \leq c_{minlow}$,o
3) K è uguale a $W.A > c$: K è implicato sse $c \geq c_{minhigh}$.

Possiamo definire una funzione IMPLY(T, K) che ritorna un valore vero se K è implicata da T usando solo la conoscenza del dominio ; altrimenti falso. Si osservi come l'implicazione espressa dalla funzione IMPLY sia di carattere logico, indipendente dallo stato del database.

Diamo una definizione di *chiusura delle restrizioni* per un dato insieme di restrizioni

rispetto ad un insieme di regole S.

Definizione: La chiusura delle restrizioni C*(R, P, S) è definita nel seguente modo.

1) $R \subseteq C^*(R, P, S)$.

2) $PROPAGATE(C^*(R, P, S), P) \subseteq C^*(R, P, S)$.

3) $\{p| JP \rightarrow (p_1 \text{ and } \ldots \text{and } p_k \rightarrow p) \in S, JOIN\_PRE(P^*, JC)$, e per ogni p_i, $1 \leq i \leq k$, $IMPLY(C^*(R, P, S), p_i)\} \subseteq C^*(R, P, S)$.

4) Solo le restrizioni espresse in 1), 2), e/o 3) sono in $C^*(R, P, S)$.

Le prime due condizioni assicurano che la chiusura contiene tutte le restrizioni date e le restrizioni ottenute dalla propagazione usando i predicati di join. La terza condizione afferma che se tutte le condizioni a sinistra di un vincolo sono soddisfatte, allora la restrizione sulla destra del vincolo è contenuta nella chiusura. Ogni volta che una nuova restrizione è ottenuta, le condizioni 2) e 3) devono essere applicate per dedurre nuove restrizioni. La chiusura di R e P è definita rispetto ad una data conoscenza espressa dall'insieme di regole S.

Il seguente algoritmo CLOSURE(R, P, S) accetta un insieme di predicati di join P, un insieme di restrizioni R, e un insieme di vincoli S.

Algoritmo CLOSURE(R, P, S)

```
begin
   P* = JP(P) ; /*Calcolo la chiusura del join */
   C° = MINInterval(R) ; /* calcola il minimo intervallo delle classi equivalenti in R*/
   if C° = FALSE then return FALSE ;/* scoperta contraddizione */
   k = 0 ;
   RULES = {E → F| JC → E → F ∈ S, JOIN_PRE(P*, JC)}/*il vincolo soddisfa la
      precondizione di join */
   while (there exists X, X and G → y ∈ RULES and IMPLY(C^K, X) )
      if G = 0 { RULES = RULES – {X and G → y} ;/*elimina la regola */
                 C^(k+1) = MINInterval(C^k ∪ {y}) ;
                 if C^(k+1) = FALSE then return FALSE ;/*scoperta contraddizione */
                 k = k +1;}
      else RULES = RULES ∪ {G → y} – {X and G → y} ;/*elimino X dall'LHS del
           vincolo*/
   return C^k ;/*ritorna la chiusura finale */
```



```
end
```

L'uscita di tale algoritmo può essere, o un insieme di restrizioni C equivalente alla chiusura delle restrizioni C*(R, P, S) in un numero finito di passi, o da in uscita un valore FALSE se si ha una contraddizione in questa chiusura. Inizialmente C^0 contiene le rappresentazioni minime delle restrizioni di tutte le classi di equivalenza associate a R. Tutte le regole in S le cui precondizioni di join sono soddisfatte (JOIN_PRE(P*, JC) = TRUE) sono poste in RULES. Questo algoritmo genera una sequenza C^k, k = 1, 2, ... di insiemi di restrizioni dedotte. L'insieme finale C^k è equivalente alla chiusura C*(R, P, S) se non viene scoperta nessuna contraddizione. C^{k+1} è costruito a partire da C^k nel seguente modo. Considero la restrizione X nell'antecedente di una regola. Se X è implicato da C^k, allora cancello X dall'antecedente della regola. Dopo la cancellazione se nell'antecedente non vi sono più restrizioni (tutti i predicati dell'antecedente della regola sono stati soddisfatti) allora la regola può essere applicata e il suo conseguente y può essere dedotto. C^k è espanso in C^{k+1} attraverso l'applicazione della procedura MINInterval che aggiunge a C^k la restrizione y mantenendo la rappresentazione minima in C^{k+1}. Se la procedura MINInterval rileva una contraddizione allora l'algoritmo ritornerà un valore FALSE.

2.3.2.2 Applicazione della chiusura delle restrizioni alla qualificazione di una query

Vediamo ora come possiamo utilizzare la chiusura delle restrizioni di una qualificazione di una query. Data la query Q[T:P,R], applicando l'algoritmo CLOSURE(R, P, S) alla query Q rispetto ad un insieme di regole S, otteniamo l'insieme di tutte le restrizioni deducibili e propagate dalla qualificazione di Q, rispetto all'insieme S. Solo alcune di queste restrizioni sono però utili al processo di SQO, e in particola sono utili quelle restrizioni che verificano le ipotesi delle euristiche H_i definite precedentemente.

Esempio:

Si consideri le relazioni R1($\underline{A}_1$, A_2, A_3), R2($\underline{B}_1$, B_2, B_3, B_4) e R3(C_1, C_2), dove gli attributi sottolineati $\underline{A}_1$ e $\underline{B}_1$ sono attributi indicizzati, e la query:

$Q[T : (R1.A_2=R2.B_2) \wedge (R3.C_1=R2.B_1) \wedge (R1.A_3=R2.B_4) \wedge (R1.A_2>10) \wedge (R3.C_2=210)]$.

Si supponga che nell'insieme S siano presenti solo le seguenti due regole :

Reg1)$(R1.A_2 > 2) \rightarrow (R1.A_3 = 400)$,

Reg2)$(R3.C_1 = R2.B_1) \wedge (R1.A_3=R2.B_4) \wedge (R3.C_2>200) \wedge (R1.A_3 = 400) \rightarrow (R2.B_3< 100)$,

Reg3)$(R1.A_2 = R3.C_1) \wedge (R3.C_2 > 200) \rightarrow (R1.A_1 = 50)$.

Applichiamo l'algoritmo CLOSURE({$(R1.A_2>10)$, $(R3.C_2=210)$}, {$(R1.A_2=R2.B_2)$, $(R3.C_1=R2.B_1)$, $(R1.A_3=R2.B_4)$ } , {S}).

Nell'esecuzione dell'algoritmo vengono generate le seguenti strutture :

(a) $P^* = JP^*(P) = \{\{W1; R1.A_2, R2.B_2\}, \{W2; R2.B_1, R3.C_1\}, \{W3; R1.A_3, R2.B_4\}\}$

$C^0 = \{\{W1, (c_{minlow} > 10, c_{minhigh} < \infty)\}, \{R3.C_2, (c_{minlow}>209, c_{minhigh}<211)\}\}$

(b) $JP^*(JC_{reg1})=\{ø\}$

RULES = $\{(R1.A_2 > 2) \rightarrow (R1.A_3 = 400)\}$

(b1) $JP^*(JC_{reg2})=\{\{R3.C_1, R2.B_1\}, \{R1.A_3, R2.B_4\}\}$

RULES = $\{(R1.A_2>2) \rightarrow (R1.A_3=400), (R3.C_1 = R2.B_1) \wedge (R1.A_3=R2.B_4) \wedge ($ $R3.C_2 >200) \wedge (R1.A_3 = 400) \rightarrow (R2.B_3< 100), \}$

(b2) $JP^*(JC_{reg3})=\{\{R1.A_2, R3.C_1\}\}$

RULES = $\{(R1.A_2>2) \rightarrow (R1.A_3=400), (R3.C_1 = R2.B_1) \wedge (R1.A_3=R2.B_4) \wedge ($ $R3.C_2 >200) \wedge (R1.A_3 = 400) \rightarrow (R2.B_3< 100)\}$

(c) $C^1 = \{\{W1, (c_{minlow} > 10, c_{minhigh} < \infty)\}, \{R3.C_2, (c_{minlow}>209, c_{minhigh}<211)\}, \{W3, (c_{minlow} > 399, c_{minhigh} < 401)\}\}$

RULES = $\{(R3.C_1 = R2.B_1) \wedge (R1.A_3=R2.B_4) \wedge (R3.C_2 > 200) \wedge (R1.A_3 = 400) \rightarrow (R2.B_3<100)\}$

(c1) $C^2 = \{\{W1, (c_{minlow} >10, c_{minhigh}<\infty)\}, \{R3.C_2, (c_{minlow}>209, c_{minhigh}<211)\}, \{W3, (c_{minlow} > 399, c_{minhigh} < 401)\}\}$

RULES = $\{(R3.C_1 = R2.B_1) \wedge (R1.A_3=R2.B_4) \wedge (R1.A_3 = 400) \rightarrow (R2.B_3<100)\}$

(c2) $C^2 = \{\{W1, (c_{minlow} >10, c_{minhigh}<\infty)\}, \{ R3.C_2, (c_{minlow}>209, c_{minhigh}<211)\}, \{W3,$

($c_{minlow} > 399$, $c_{minhigh} < 401$)}, {$R2.B_3$, ($c_{minlow} > -\infty$, $c_{minhigh} < 100$)} }

RULES = {ø}

(d) $C^* = C^2$.

Vediamo i vari passi che segue l'algoritmo nel generare la chiusura C*.

Al punto (a) l'algoritmo calcola la chiusura dei join e il minimo intervallo delle restrizioni della query Q. In (b) inizia la costruzione dell'insieme RULES attraverso l'applicazione della funzione JOIN_PRE alle tre regole dell'insieme S [(b), (b1), (b2)]. Costruito l'insieme RULES inizia il ciclo nel quale si verifica quando l'insieme C^k implica una delle restrizioni di una regola posta in RULES (c). Si osservi come ogni ciclo consideri una restrizione alla volta delle regole di RULES, e ogni volta che si verifica l'implicazione, la regola viene aggiornata eliminando la restrizione implicata [(c1), (c2)]. Quando in RULES non ci sono più regole implicate da C^k, l'algoritmo si ferma (d). C^2 risulta essere la chiusura delle restrizioni della qualificazione della query Q. Si osservi come la regola 3) non è stata inserita in RULES in quanto la precondizione di join di questa regola non è soddisfatta dalla chiusura dei join di Q. Le regole utilizzate per la costruzione della chiusura C^2 sono la 1) e 2).

L'utilizzo delle regole 1) e 2) può essere fatto corrispondere all'applicazione dell'euristica H2 (scan reduction) ; che prevede di introdurre delle restrizioni, nella qualificazione di una interrogazione, su relazioni coinvolte in join. L'applicazione dell'euristica H1 invece non è stata possibile visto che nella chiusura C* non compare nessuna restrizione su un attributo indicizzato.

L'esempio visto offre lo spunto per potere generare delle funzioni che diano il seguente output :

- l'insieme delle restrizione che è possibile dedurre da S, che soddisfano alle ipotesi delle euristiche predefinite H_i,
- l'insieme delle regole proposte corrispondenti a quelle euristiche, a cui non è stato possibile far corrispondere nessuna restrizione di C*.

Ad esempio, nella applicazione dell'algoritmo CLOSURE visto prima, non si è trovato nessuna restrizione in C* su un attributo indicizzato di R1. Quindi sarebbe auspicabile generare delle regole che avessero, un antecedente implicato dalle restrizioni di Q, e come conseguente una restrizione su $R1.\underline{A_1}$.

Ricapitolando, le funzioni che si vogliono costruire, devono indicare, attraverso la generazione di regole proposte, quale conoscenza, che non è stato possibile trovare in S, sarebbe utile creare per l'SQO.

2.3.3 Generazione delle regole proposte

Dopo che abbiamo trovato la chiusura C* delle qualificazioni della query, dobbiamo verificare quali regole proposte occorre generare. A tale scopo si considerano le funzioni $GENRP\_H_i(C^*, P^*)$. Ognuna di queste funzioni, applica all'insieme C* una diversa euristica H_i. Le funzioni avranno come output delle regole proposte che verranno inviate al modulo "Selezione regole proposte". Le funzioni prendono come parametri la chiusura delle restrizioni generate dall'algoritmo CLOSURE, e la chiusura dei join della query P*. La generazione delle regole proposte non viene fatta rispetto alle restrizioni della query originaria Q, ma rispetto alla sua chiusura C*. In questo modo tutta la conoscenza semantica presente nel sistema (regole in S), e applicabile a Q, è posta in C*, cosicché la funzione $GENRP\_H_i(C^*, P^*)$ può operare nel migliore dei modi. Questo ci consente di :

1) generare il maggiore numero di regole proposte. Infatti in C* vi sono un numero di restrizioni maggiore o uguale alle restrizioni presenti nella qualificazione della query Q, quindi sarà possibile considerare un numero maggiore di regole proposte.
2) generare regole non ridondanti. Essendo tutta la conoscenza semantica applicabile riunita in C*, la funzione $GENRP\_H_i(C^*, P^*)$ genererà solo regole proposte non ridondanti rispetto ad S.

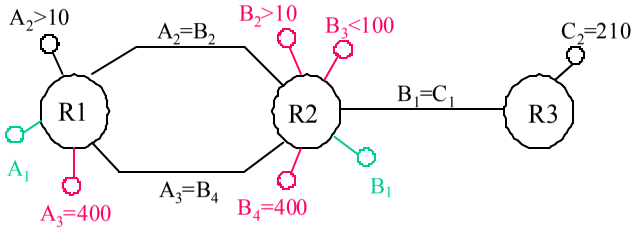
Esempio :

Consideriamo la chiusura C* che abbiamo calcolato precedentemente per la query Q, e l'euristica H1.

$C^* = \{\{W1, (c_{minlow} > 10, c_{minhigh} < \infty)\}, \{R3.C_2, (c_{minlow} > 209, c_{minhigh} < 211)\}, \{W3, (c_{minlow} > 399, c_{minhigh} < 401)\}, \{R2.B_3, (c_{minlow} > -\infty, c_{minhigh} < 100)\}\}$

$P^* = JP^*(P) = \{\{W1; R1.A_2, R2.B_2\}, \{W2; R2.B_1, R3.C_1\}, \{W3; R1.A_3, R2.B_4\}\}$

Riportiamo il grafo della query ,disegno in nero, in cui è stato aggiunto l'insieme delle restrizioni che sono state estratte da S, disegno in rosso, e gli attributi sui quali le euristiche cercano di generare delle restrizioni (disegno in verde).



L'euristica H1 può essere applicata alle relazioni R1 e R2, per le quali non esiste una restrizione, rispettivamente, sugli attributi indicizzati A_1 e B_2. La relazione R3 non presenta attributi con indice.

Per generare la prima regole proposta, l'euristica H1 inserisce tutte le restrizioni sulla relazione R1, che si trovano in C*, nell'antecedente della regola.

Si ottiene la regola proposta RP1.

RP1	*Attributo*	*Vincolo*
Antecedente	$R1.A_2$	(> 10)
and	$R1.A_3$	(= 400)
Conseguente	$R1.A_1$	

In modo analogo, H1 genera la seconda regola proposta sulla relazione R2.

RP2	*Attributo*	*Vincolo*
Antecedente	$R2.B_2$	(> 10)
and	$R2.B_3$	(< 100)
and	$R2.B_4$	(= 400)
Conseguente	$R1.B_1$	

Si osservi come si possono generare delle ridondanze negli antecedenti delle regole proposte. Ad esempio, in RP1, dalla regola Reg1, sappiamo che la prima restrizione implica la seconda. In RP2 non vi è questa ridondanza tra le restrizioni degli antecedenti.

L'euristica H2 non può venire applicata, visto che sono già presenti restrizioni su attributi di join, su ogni relazione.

Le euristiche H4 e H5 devono verificare innanzitutto quali relazioni hanno la possibilità di essere eliminate, indipendentemente dalle regole presenti nel sistema. Queste sono le relazioni che non presentano attributi nella clausola select; esclusi gli attributi di join della query. Supponiamo che la query Q[T: q], abbia $T=\{R1.A_2\}$. Quindi solo nella relazione R1 è presente un attributo target. Le relazioni R2 e R3 possono essere eliminate, se si riesce a derivare delle regole che permettano di verificare le rimanenti due condizioni per l'eliminazione di un join.

H4 genera una regola proposta per verificare se $R2.B_1 \subseteq R3.C_1$.

RP3	*Attributo*	*Vincolo*
Antecedente	$R2.B_1$	----------

Conseguente	$R3.C_1$	----------

Questa regola proposta serve per l'eliminazione della relazione R3. Nel caso, durante il processo di SQO, si riuscisse ad eliminare la relazione R3, H4 genera anche una regola proposta per l'eliminazione della relazione R2. Si crea la regole proposta RP2 per verificare se $R1(A_2,A_3) \subseteq R2(B_2,B_4)$.

RP4	*Attributo*	*Vincolo*
Antecedente	$R1(A_2, A_3)$	----------
Conseguente	$R2(B_2, B_4)$	----------

L'euristica H5 genera delle regole proposte che cercano di eliminare le restrizioni sulle relazioni R2 e R3. Le restrizioni che si considerano sono quelle che sono risultate non ridondanti dopo aver calcolato la chiusura C*. Nel nostro caso sono le restrizioni che nel grafo, presentato precedentemente, sono segnate in nero e che appartengono alla query Q. Si noti, che anche queste restrizioni, sarebbero potute divenire ridondanti. Ad esempio, se in C* fosse stata introdotta una restrizione che implica una restrizione che appartiene alla query Q. La relazione R2 non ha restrizioni non ha restrizioni quindi l'euristica H5 non è applicabile. La relazione R3 presenta la restrizione sull'attributo C_2. Le regole proposte generate da H5 sono le seguenti :

RP5	*Attributo*	*Vincolo*
Antecedente	$R1.A_2$	(> 10)
and	$R1.A_3$	(= 400)
and	$R1.A_2$	(= $R2.B_2$)
and	$R2.B_1$	(= $R3.C_1$)
Conseguente	$R3.C_1$	

RP6	*Attributo*	*Vincolo*

Antecedente	$R2.B_2$	(> 10)
and	$R2.B_4$	(= 400)
and	$R2.B_3$	(< 100)
and	$R2.B_1$	(= $R3.C_1$)
Conseguente	$R3.C_1$	

2.4 Valutazione e scelta delle regole proposte per il processo di derivazione

Le regole proposte, generate nel modulo "Processo di creazione regole proposte", sono generalmente numerose. Se cercassimo di derivare da ognuna di esse delle regole da inserire nel rule set, è probabile che molte di queste regole risulterebbero inadeguate per il processo di ottimizzazione semantica. Il numero di regole del rule set è limitato dal valore S_{reg}, quindi solo le regole più utili all'SQO vengono inserite nel rule set, le altre sono scartate. Occorre dunque considerare delle regole proposte dalle quali si pensa di derivare delle regole che abbiano un utilità sufficientemente grande da poter essere inserite nel rule set. Inoltre si può verificare il caso in cui regole proposte, dalle quali in passato non si è ricavata nessuna regola utile, vengano generate di nuovo dal sistema. In questo caso si devono scartare queste regole proposte. E' necessario quindi potere valutare le diverse regole proposte per decidere se da una regola dobbiamo cercare di derivare regole o se deve essere scartata. Le regole proposte verranno valutate attraverso un parametro che chiameremo utilità potenziale di una regola.

Prima di iniziare a descrivere il processo di valutazione delle regole proposte, si introduce il concetto di utilità di una regola partendo dal problema dell'utilità che può sorgere durante l'SQO. Successivamente definiremo l'insieme rule class, che è l'insieme che tiene traccia dei tentativi passati di derivazione di regole proposte.

2.5 Problema dell'utilità e Utilità di una regola

Diamo la seguente definizione:

Definizione: Definiamo *risparmio* di una regola semantica, rispetto ad una sua applicazione su una query Q da parte dell'ottimizzatore semantico, come la differenza tra il costo di esecuzione della query prima dell'applicazione della regola semantica e il costo di esecuzione della query dopo l'applicazione della regola semantica.

Il problema dell'utilità può sorgere quando consideriamo gli effetti collettivi delle regole scoperte nel processo di ottimizzazione. Il problema dell'utilità, originariamente identificato da [(44) Minton, (1988)], fa riferimento alla situazione in cui le regole scoperte degradano le prestazioni del sistema. In un sistema rule-based, il problema dell'utilità può sorgere perché l'utilità di una regola è determinata non solo dal risparmio apportato dalla regola e dalla sua applicabilità, ma anche dal costo per il sistema ad individuare ed applicare la regola per l'SQO, *match_cost*. Le prestazioni del sistema dunque possono degradare, se diminuiscono il risparmio medio e la frequenza di applicazione o se aumenta il match_cost.

Il match_cost può aumentare, o perché ci sono troppe regole nel rule-set, e quindi il numero di confronti, per verificare l'applicabilità di ognuna delle regole alla query di input, risulta essere troppo elevato; o perché regole individuali sono troppo complesse per essere *matched* efficientemente. Regole complesse sono quelle che presentano un numero di qualificazioni congiuntive elevato nell'antecedente e nel conseguente della regola. Regole di questo tipo comportano dei problemi in fase di applicazione ; un numero elevato di restrizioni nell'antecedente della regola diminuisce la probabilità che una regola sia applicabile e quindi considerata per una possibile trasformazione di una query. Mentre un numero elevato di restrizioni nel conseguente diminuisce la funzionalità delle regola, nel senso che alcuni vincoli del conseguente se introdotti in una query possono portare una riduzione del costo mentre altri vincoli del conseguente, della stessa regola, possono portare invece dei peggioramenti del costo della query, compromettendo così il valore del costo risparmiato che è possibile associare a tale regola.

Nel sistema che presentiamo, si cerca di limitare il problema dell'utilità ricercando regole che probabilmente presenteranno dei risparmi alti, grazie all'utilizzo delle euristiche. Si cerca di derivare delle regole con degli antecedenti che coprano un range di valori abbastanza ampio, in modo che possano essere applicate frequentemente. Inoltre si cerca di avere un basso match_cost, derivando delle regole con un solo conseguente, e quindi di più facile applicabilità, e limitando il numero di

regole che compongono il rule set. Viene eseguita la manutenzione del rule set, in modo da eliminare da esso le regole ridondanti e le regole con valori bassi del risparmio.

Partendo dai parametri visti per il problema dell'utilità, andiamo a definire una funzione utilità che ci consentirà di assegnare ad ogni regola un valore utilità. Tale utilità vuole misurare l'effetto complessivo che la regola ha sul sistema, comprendendo sia i risparmi apportati che i costi di manutenzione. I parametri considerati sono il valore medio del risparmio apportato da una regola, la frequenza di applicazione della regola, e il costo per mantenere la coerenza della regola con lo stato corrente del database.

La funzione utilizzata è la seguente:

$$\text{Utilità}(R)=\text{Risparmio\_medio} \times \text{Frequenza\_di\_applicazione} - \text{Costo\_di manutenzione}$$

Questa funzione utilità viene applicata alle regole derivate alle quali occorre associare un valore utilità. Questo valore viene usato o per decidere se inserire una regola derivata nel rule set, o per valutare quando una regola del rule set non è più utile per il sistema.

Per quanto riguarda la valutazione di una regola proposta, si considera un valore utilità che definiremo *potenziale*, in quanto stima la potenzialità della regola proposta di derivare una regola utile. La stima verrà eseguita con l'aiuto di rule class che tengono conto della derivazione passata di regole proposte simili a quella da stimare. A tale scopo viene introdotto la nozione di rule class.

2.6 RULE CLASS

Si vuole definire un sistema capace di tenere traccia dei tentativi passati di derivare regole. Si vuole inoltre poter effettuare una selezione delle regole proposte per potere confrontare, e poi scegliere, le regole proposte che meglio si prestano per la fase di derivazione delle regole. A tale scopo vengono definite due tipi di *rule class*. Una per le regole proposte create da euristiche H1 e H2, una per le regole proposte create dall'euristiche H3, H5.

RC(JC , ant, con, fsma, um, cont, #reg, rp set, H1/H2).

RC(JC , ant, con, δfsm*, um, cont, #reg, rp set, H3/H5).*

- *ant*, *con* e *JC* indicano rispettivamente gli attributi delle restrizioni dell'antecedente e l'attributo delle restrizione del conseguente, delle regole proposte costruite sulla relazione R(JC) che appartengono a questa classe. Ad esempio, la regole proposta $(R1.A=R2.A)\wedge(R1.B>10)\rightarrow(R2.C)$, appartiene alla classe RC({R1.A=R2.A},{ R1.B}, {R2.C}, fsma, um, cont, #reg, rp set, H_i). Sarà così possibile associare ad un insieme di regole proposte, aventi caratteristiche comuni, una serie di valori che andiamo ora a vedere.
- *fsma* è il *fattore di selettività medio degli antecedenti*. Fsma è il valore medio del fattore di selettività degli antecedenti delle regole proposte, che appartengono ad una rule class, e dalle quali è stato possibile ottenere in passato, dal processo di derivazione, una regola da inserire rule set.
- *δfsm* è la *differenza media dei fattori di selettività dell'antecedente e il conseguente delle regole proposte*. Data una regola proposta generata da un euristica di tipo H3 o H4, definiamo la differenza dei fattori di selettività della regola δfs(RP)= fsa-fsc. Il δfsm è la media dei δfs(RP) delle regole proposte che appartengono ad una rule class, e dalle quali è stato possibile ottenere in passato,

dal processo di derivazione, una regola da inserire rule set.

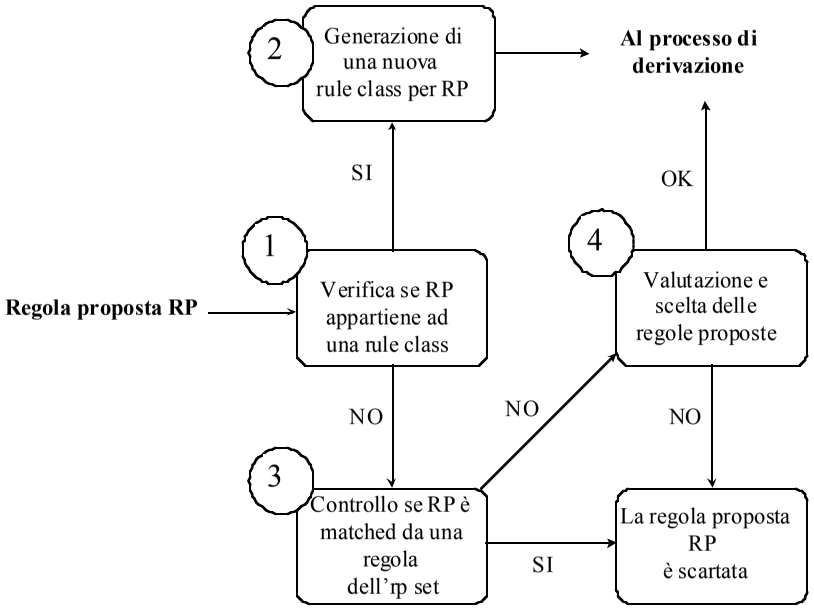
- *um* è l'*utilità potenziale media* della rule class, ottenuta dalla media dei valori di utilità delle stesse regole derivate descritte per il calcolo del valore fsma.
- *cont* è il numero di regole proposte da cui sono stati calcolati i valori fsma e um.
- *#reg* è il numero di regole proposte totali, appartenenti alla rule class, che sono state generate dal sistema. Esso comprende sia le regole dalle quali si è derivata una regola, sia le regole proposte che sono state scartate.
- *rp set* è una lista di coppie di valori {ant(RP), U(RP)}se la regola proposta è stata creata da un euristica H1 o H2 ; per regole proposte generate da euristiche H3 o H5 abbiamo delle terne di valori {ant(RP), con(RP), U(RD)}. In questa lista vengono memorizzate gli antecedenti (nell'ultimo caso visto anche i conseguenti) delle regole proposte che, sono state selezionate per il processo di derivazione, ma dalle quali non si è derivata una regola da inserire nel rule set. Insieme ad ant(RP) si memorizza anche il valore dell'utilità della regola derivata scartata. Rp set serve per tenere traccia di tentativi passati di derivazione di una regola proposta non andati a buon fine.
- H_i indica il tipo di euristica a cui è associata la rule class. Ogni rule class farà riferimento alle regole proposte generate da una stessa euristica H_i.

I dati delle rule class vengono acquisiti in modo automatico durante il processo di selezione delle regole proposte e il processo di derivazione. Le informazioni contenute nelle RC sono utilizzate nella fase di selezione delle regole proposte e per la fase di attivazione del processo di generazione delle regole. Nel primo caso si utilizzano il *um, fsma, rp set*. Il um e fsma permettono di stimare un *up(RP)*, *utilità potenziale* per una regola proposta. Questo valore rende possibile confrontare regole proposte eterogenee, cioè appartenenti a classi diverse o create dall'applicazione di euristiche differenti. Attraverso il valore up di ogni regola possiamo selezionare la RP migliore. Rp set, al contrario, individua le regole proposte dalle quali siamo certi che non potremo derivare delle regole utili appartenenti ad una data RC.

Per dare inizio al processo di generazione utilizziamo invece *cont*, *#reg*, *ant* , *con* e *JC(RC)*. Un valore grande di cont significa che sono state inserite molte regole nel rule set, tutte appartenenti alla stessa rule class a cui appartiene cont. Da ciò, possiamo dedurre che queste regole non hanno una grande campo di applicabilità ed è per questo che il sistema continua ad derivarne. Un valore molto grande di (#reg - cont) significa che sono state generate molte regole proposte, appartenenti ad una stessa rule class RC, dalle quali non è stato possibile derivare delle regole utili, e quindi sono state scartate. La grande insistenza del sistema di generare tali regole proposte portano a pensare che sia conveniente effettuare una analisi più mirata e accurata, per la ricerca di regole appartenenti alla rule class RC(JC, ant, con, fsma, um, cont, #reg, rp set). La ricerca verrà effettuata sulla relazione definita da JC, e le regole estratte dovranno essere del tipo Rest(ant)→Rest(con).

2.7 Processo di valutazione delle regole proposte

Si consideri la seguente figura che descrive il processo di valutazione delle regole proposte. Ad ogni regola proposta, che proviene dal modulo "Creazione regole proposte", si dovrà associare una rule class, per poter calcolare la sua utilità potenziale up. A seconda del valore dell'utilità potenziale della regola si decide se scartare la regola o se mandarla al processo di derivazione.



Consideriamo le singole fasi che portano alla scelta di una regola proposta.

(1) Verifica se esiste una RC per la regola proposta RP

E' possibile pensare ad un flusso di regole proposte che arrivano al modulo "Valutazione e scelta regole proposte". Ad ognuna di queste regole proposte dovrà essere associata una rule class affinché si possa effettuare una stima della sua utilità potenziale up.

Definizione :Data la regola proposta

$$RP :(JC(RP)\wedge Rest_1(A_1)\wedge \ldots\ldots\wedge Rest_{n-1}(A_{n-1})\rightarrow Rest_n(A_n),$$

generata da una euristica H_i, e la rule class RC(JC, ant, con, fsma, um, cont, #reg, rp set, H_i), diremo che la regola proposta RP appartiene alla classe RC se :

1) JC*(RP)=JC*(RC)[1],

2) $\{A_1, A_2, \ldots A_{n-1}\}=\{ant\}$, A_n=con e

3) $H_i(RC)=H_i(RP)$

Dove con JC*(RP) e JC*(RC) si è indicata la chiusura dei join rispettivamente della rule class RC e della regola proposta RP.

Se esiste una rule class a cui appartiene la regola proposta allora inviamo la regola proposta al modulo (3). Altrimenti la regola è inviata al modulo (2) nel quale viene generata una nuova rule class per la regola.

(2) Creazione di una nuova rule class

Se la regola proposta RP non appartiene a nessuna rule class, occorre generarne una nuova. I parametri relativi a JC(RC), ant, con, fsma sono ricavati direttamente dalla regola proposta. Se non esiste una rule class da associare alla RP, significa che non è mai stata effettuata una derivazione di una regola proposta simile ad RP. Non avendo nessun riscontro di tentativi passati di derivazione, si lascia in sospeso l'assegnazione del valore um, sostituendolo con un " ?", e si manda la regola proposta RP direttamente al modulo di derivazione delle regole proposte. All'uscita del modulo si potrà sostituire a " ?" il valore U, corrispondente al valore utilità, che è

[1] JC* rappresenta lachiusura di un unsieme di restrizioni. La definizione di chiusura è stata data nella sezione 2.3.1 a pag 60

possibile associare alla regola che è stata derivata da RP. Nel caso che da RP non si possa derivare nessuna regola, lasceremo nella rule class "?", e la regola proposta verrà inserita nel rp set. Il rp set sarà vuoto mentre #reg è posto uguale ad 1, e cont=0.

Esempio : Se non esiste una RC per la RP :$(JC(RP) \wedge Rest_1(A_1) \wedge \ldots \wedge Rest_{n-1}(A_{n-1})) \rightarrow Rest_n(A_n)$ si genera la rule class $RC(JC(RP),\{A_1, \ldots, A_{n-1}\}, A_n, fs(RP), ?, 0,1,\{\})$.

(3) Verifica se esiste una regola proposta, appartenente al rp set, che match la RP

Alla RP é stata associata una rule class, nella quale è presente la lista rp set, delle regole proposte che in passato non hanno dato un buon esito. Se una regola proposta appartenente al rp set, match la RP, andremo a scartare la RP. Se una regola dell'rp set match RP, significa che l'antecedente della regola RP ha una copertura superiore dell'antecedente della RP_i del rp set. Non è possibile che, aumentando il range dell'antecedente di una regola proposta che ha avuto un cattivo esito nel processo di derivazione, si possa derivare una regola utile. Questo vale sia per le regole proposte di introduzione, generate dall'euristiche H1 e H2, che per le regole proposte che vogliono verificare la ridondanza di una restrizione.

Come esempio consideriamo la regola proposta RP, generata da un euristica H1, e la rule class RC1. Per regole proposte generate da un euristica H2, si procede in modo analogo.

$$RP1 : (A>100) \rightarrow (B)$$

RC1(R1, A, B, fsma, um, cont, #reg, rp set, H1) ;

dove rp set = {(A<50)→B, (A>90)→B}.

La prima restrizione RP_1, dell'insieme rp set, non implica l'antecedente di RP. Mentre abbiamo che $RP_2 \rightarrow (A>100)$. La regola proposta RP viene scartata.

Vediamo il caso di una regola proposta generata da un euristica H3 o H5.

Prendiamo ad esempio la regola proposta RP2 e la rule class RC2.

$$RP2 : (A>100) \rightarrow (B>30)$$

RC1(R1, A, B, fsma, um, cont, #reg, rp set, H3) :

dove rp set={(A>120)→(B>20), (A=10)→(B>20)}, e R1 è la relazione di A e B.

La prima restrizione match la regola proposta RP2. La regola proposta RP2 viene scartata.

Se non troviamo nessuna regola RP_i che implica la RP, possiamo continuare il processo di selezione delle regole proposte.

(4) Valutazione e scelta delle regole proposte

Dopo il controllo del modulo (3) la regola proposta e la corrispondente rule class vengono mandati al modulo "Valutazione e scelta regole proposte". La prima operazione che viene eseguita è di stimare che la regola proposta in input possa generare una regola derivata avente un valore dell'utilità sufficientemente alto da poter essere inserita nel rule set. Questa stima viene eseguita attraverso il parametro *utilità potenziale* (up). Il parametro up della regola proposta viene poi confrontato con *il valore soglia dell'insieme rule set* S_{reg} che rappresenta l'utilità che una regola derivata deve avere per essere inserita nel rule set. Ad ogni regola del rule set è associato un valore U(RD), dove U(RD) è l'utilità della regola nel sistema. Il sistema che si occupa della manutenzione del rule set fa si che vi sia un numero massimo M di regole che si possa avere nell'insieme. Vi è inoltre una soglia minima U_{min} del valore dell'utilità che una regola scoperta deve avere per essere inserita nel rule set. Se il numero di regole nel rule set è inferiore alla soglia M, il valore minimo di U, che una regola dovrà avere per essere inserita nel rule set, sarà uguale a U_{min}. Se invece si è

raggiunto il numero di regole massimo che è possibile avere nel rule set, allora il valore U_{min} è uguale al valore dell'utilità U(RD) più basso delle regole poste all'interno del rule set.

Una regola proposta viene selezionata per il processo di derivazione se ha un valore del up superiore del valore soglia $S_{reg.}$ In questo confronto, per calcolare up(RP), si prende in considerazione anche il fattore di selettività dell'antecedente della regola fsa(RP). Viene utilizzato il fattore di selettività delle regole proposte perché così è possibile diversificare la stima delle regole proposte appartenenti ad una stessa rule class.

Esempio : Consideriamo le seguenti due regole proposte :

RP1 : (A>30)→(B)

RP2 : (A=8) →(B)

Sia RP1 che RP2 appartengono alla stessa rule class RC(R1, A, B, fsma, um, cont, #reg, rp set, H1). Si potrebbe assegnare ad up(RP1) e up(RP2) il valore um della classe a cui appartengono. Questo in virtù del fatto che, regole che sono state poste nel rule set in passato, con gli stessi attributi di RP1 e RP2, presentavano un utilità media uguale um. Ma RP1 presenta un antecedente che copre un range, superiore al range coperto dall'antecedente di RP2, quindi l'antecedente di RP1 interesserà un numero di tuple superiore dell'antecedente di RP2. Sarà più probabile derivare regole con delle restrizioni più selettive sull'attributo B, da RP1 piuttosto che da RP2. Dato che H1 e H2 sono euristiche di introduzione, restrizioni più selettive nel conseguente di una regola derivata, comporta avere un risparmio maggiore dall'applicazione della regola.

Per il calcolo della up(RP) si utilizzeranno due formule differenti a seconda del tipo di regola proposta. Per le regole proposte generate dalle euristiche H1 e H2 si utilizzerà la formula (f1).

$$up(RP)=\lceil (fsma(RC) - fsa(RP)) \times um(RC) + um(RC) \rceil \qquad (f1)$$

Il calcolo di up(RP) viene eseguito considerando inizialmente il valore um(RC) della rule class associata alla regola proposta RC. Poiché si vuole tenere conto anche della selettività dell'antecedente di RP, il valore che verrà assegnato ad up(RP) sarà una funzione di fsa(RP). In particolare se fsa(RP) è più grande di fsma(RC), la minore selettività dell'antecedente di RP comporterà una stima di up(RP) inferiore a um(RC). Al contrario se fsa(RP) è più piccolo di fsma(RC), la maggiore selettività dell'antecedente di RP comporterà un up(RP) maggiore di um(RC). Con una selettività fsa(RP)≈fsma(RC) avremo una up(RP) circa uguale a um(RC).

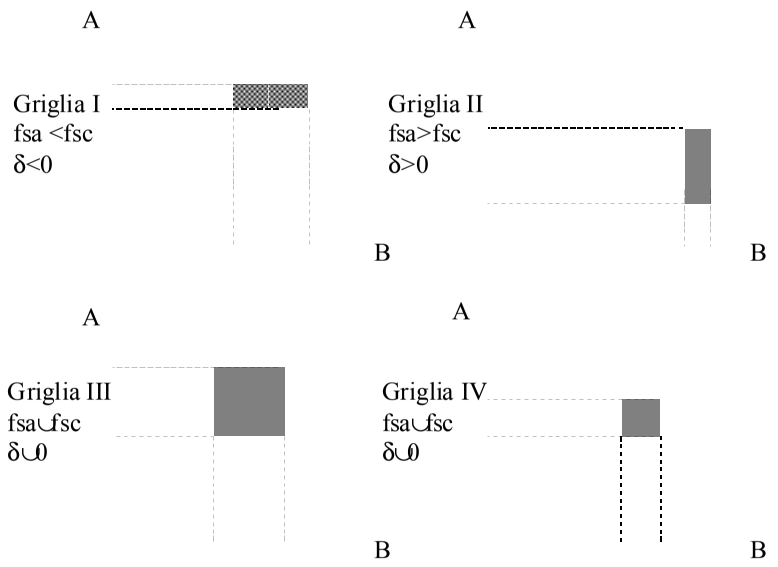
Per calcolare up(RP) delle regole proposte generate dalle euristiche H3 e H5 si utilizzerà la formula (f2).

$$up(RP)=\lceil -1/2 \mid \delta fsm(RC) - \delta fs(RP) \mid \times um(RC) + um(RC) \rceil \qquad (f2)$$

con $\delta fs(RP)= fsa(RP) - fsc(RP)$; con $\mid \delta fsm(RC) - \delta fs(RP) \mid$ il valore assoluto della differenza dei valori di selettività.

Il valore $\delta fsm(RC)$ ci dice quale tipologia di regole sono state estratte con successo in passato.

Consideriamo la figura che mostra 4 griglie. Ogni griglia rappresenta una distribuzione diversa delle tuple, della relazione R(A, B), che verificano delle regole del tipo Rest(A)→Rest(B). Nella griglia I il rettangolo scuro mostra le tuple che verificano una regola in cui il fattore di selettività dell'antecedente è minore del fattore di selettività del conseguente. Questa specie di regola nella griglia è mostrata con un rettangolo orizzontale. Nella griglia II, il rettangolo verticale, mostra una regola in cui il fattore di selettività dell'antecedente è maggiore del fattore di selettività del conseguente. Nelle griglie III e IV sono mostrare due regole in cui i fattori di selettività sono circa uguali. L'ipotesi da cui parte la formula (f2) è che, se in passato sono state derivate con successo delle regole proposte di un certo genere (come quelle di figura I, II, III, IV), allora questo tipo di regole proposte hanno più probabilità di derivare delle regole. La formula (f2) calcolerà, per una regola proposta, un up circa uguale ad um(RC), se $\delta fs(RP) \approx \delta fsm(RC)$. Se invece $\delta fs(RP) > \delta fsm(RC)$ o $\delta fs(RP) < \delta fsm(RC)$, il valore di up sarà tanto minore di um quanto maggiore è la differenza tra $\delta fs(RP)$ e $\delta fsm(RC)$.



con un $\delta fsm(RC)>0$

Vediamo un esempio della applicazione della formula f1. Supponiamo di avere la seguente regola proposta e la corrispondente rule class associata:

$$RP : (A>60) \rightarrow B$$

RC(R1, A, B, 0.3, 8.7, 2, 1, rp set, H1)

$$up(RP)=\lceil (fsma(RC) - fsa(RP)) \times um(RC) + um(RC) \rceil \qquad (f1)$$

Consideriamo il seguente valore soglia affinché una regola derivata possa entrare nel rule set.

$$S_{reg}=9$$

Per il calcolo del fattore di selettività della regola RP vengono utilizzati i seguenti valori che indicano il valore massimo e il valore minimo che l'attributo A può assumere:

$$VM=100$$

$$Vm=0$$

Calcoliamo la selettività dell'antecedente della regola proposta RP3[1].

$$fsa(RP3)=\frac{V_M-40}{V_M-V_m}=\frac{100-60}{100}=0.4$$

Perché la regola proposta RP3 venga scelta occorrerà che il suo up(RP3) stimato sia superiore al valore S_{reg} calcolato con la formula f1.

$$up(RP3)=\lceil (fsma(RC3)-fsa(RP3))\times um(RC3)+um(RC3)\rceil$$

$$up(RP3)=\lceil (0.3-0.4)\times 8.7+8.7\rceil=8$$

Dato che abbiamo che up(RP3) < Sreg la regola proposta RP3 non può essere selezionata per il processo di derivazione di una regola da inserire nel rule set.
Supponiamo che il fattore di selettività della regola RP3 non sia più di 0.4 ma sia 0.2. Diminuendo il fattore di selettività, secondo le ipotesi fatte precedentemente, si ha una maggiore probabilità di derivare una regola con una utilità alta. Quindi il up(RP3) della regola proposta che si sta' valutando dovrà essere superiore al valore calcolato precedentemente. In questo modo pur mantenendo uno stesso um(RC3) la regola RP3 può essere scelta.

[1] La stima del fattore di selettività è eseguita, in mancanza di strumenti di stima più precisi, considerando l'ipotesi di uniforme distribuzione dei valori delle relazioni.

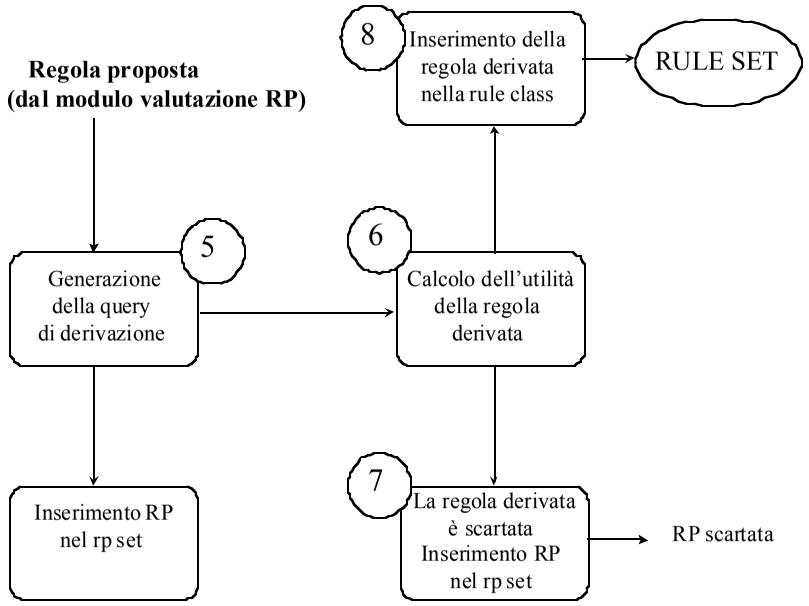
Il nuovo up(RP3) è uguale a:

$$up(RP3)=\lceil (0.3-0.2)\times 8.7+8.7\rceil=10$$

abbiamo che up(RP3) > S_{reg}, quindi la regola proposta RP3 può essere inviata al processo di derivazione.

2.8 Derivazione delle regole proposte

Il processo di derivazione di una regola presenta come input la regola proposta che è stata selezionata nel modulo "Valutazione e scelta regole proposte". Il processo di derivazione consiste, a partire da una regola proposta, nel generare una query dalla cui risposta si potrà costruire una regola derivata. Successivamente si calcolerà per tale regola un valore utilità U(RD), e se tale valore supera la soglia S_{reg}, la regola derivata verrà immessa nell'insieme rule set. Nel caso si avesse $U(RD)<S_{reg}$ la regola derivata viene scartata.



(5) Generazione della query di derivazione e generazione della regola derivata

A secondo del tipo di euristica, che ha generato la regola proposta, viene costruita una query dalla cui risposta sarà possibile ricavare una regola derivata. La query costruita viene chiamata *query di derivazione*.

In questo paragrafo si descriveranno le varie query di derivazione che possono

essere costruite, per ogni tipo di euristica, che ha generato la regola proposta. A partire dal risultato della query di generazione viene descritto come viene creata la regola derivata.

L'antecedente della regola derivata sarà uguale all'antecedente della regola proposta da cui deriva e che ha dato inizio al processo di derivazione. Il conseguente della regola generata dipende dal risultato, dal tipo di euristica che ha generato la regola proposta e dal dominio dell'attributo del conseguente della regola derivata. Per attributi ordinati del conseguente, è possibile descrivere un range di valori utilizzando operatori quali >, <, ≥, ≤, *between,* oppure se il numero di valori restituito dalla query è limitato si utilizza l'operatore set che descrive un insieme di valori limitato. Ad esempio si può scegliere che il numero massimo di valori presenti nell'insieme sia minore o uguale a 20, altrimenti si utilizzano gli operatori prima visti. Se l'attributo è non ordinabile allora è possibile usare solo l'operatore di set, o se il risultato della query è costituito da un unico valore l'operatore di uguaglianza.

Si consideri la regola proposta RP generata dall'euristica H1.

RP1	Attributo	Vincolo
Antecedente	LIBRO.Collocazione	(= 20)
Conseguente	LIBRO.Cod_lib	

Alla regola proposta RP1 viene associata la query Q1.

```
select  LIBRO.Cod_lib
from  LIBRO                                          (Q1)
where LIBRO.Collocazione=20 ;
```

Il conseguente della regola proposta determina gli attributi da selezionare nella query e l'antecedente determina la parte *where* della query. Se la risposta della query Q1

non è nulla possiamo creare una regola derivata. Ad esempio se la query Q1 presenta valori superiori a 1000, possiamo generare la regola RD1.

RD1	Attributo	Vincolo
Antecedente	LIBRO.Collocazione	(= 20)
Conseguente	LIBRO.Code	(> 1000)

In caso di risposta nulla della query la regola proposta viene scartata.
Se la regola proposta è stata prodotta da un vincolo euristico di rimozione (euristica H3), allora ci sarà anche un vincolo associato con il suo conseguente. Per esempio consideriamo la regola proposta RP2 generata dall'euristica H3.

RP2	Attributo	Vincolo
Antecedente	LIBRO.Collocazione	(> 1200)
Conseguente	LIBRO.Anno	(> 1995)

Se questa regola proposta è stata selezionata per la derivazione, allora sarà generata la seguente query.

```
Select LIBRO.Anno
from LIBRO                                          (Q2)
where LIBRO.Collocazione > 1200;
```

Se la risposta alla query Q2 presenta dei valori di LIBRO.Anno tutti superiori ad 1995, allora è possibile generare una regola. Altrimenti la regola proposta RP2 sarà scartata.

RD2	Attributo	Vincolo
Antecedente	LIBRO.Collocazione	(> 1200)

Conseguente	LIBRO.Anno	(> 1995)

Se la regola proposta coinvolge attributi di relazioni differenti, come nel caso delle euristiche H2, allora la query generata deve includere una condizione nella clausola *where* collegante queste due relazioni. Un esempio è dato considerando la regola proposta PR2.

RP2	Attributo	Vincolo
Antecedente	LIBRO.Autore	(= 'Neruda Pablo')
and	LIBRO.Soggetto	(=RIVISTA.Soggetto)
Conseguente	RIVISTA.Titolo	

La query generata per questa regola proposta è :

```
select RIVISTA.Titolo
from LIBRO,RIVISTA                                  (Q3)
where LIBRO.Autore = 'Neruda Pablo'
and LIBRO.Soggetto = RIVISTA.Soggetto ;
```

Nell'euristica H4 si deve verificare se l'insieme dei valori di un attributo sono o no un sottoinsieme dei valori di un altro attributo. Supponiamo di avere una regola proposta per la quale si deve verificare che RIVISTA.Soggetto $\subseteq$ LIBRO.Soggetto. Se abbiamo che l'attributo Soggetto è una foreign key di RIVISTA e una primary key di LIBRO, allora la condizione di superset è sempre verificata[1]. In tal caso possiamo generare una regola senza dover eseguire nessuna query. Altrimenti generiamo la seguente query:

```
select RIVISTA.Soggetto
from RIVISTA                                        (Q4)
```

[1] Si suppone che nel database siano verificati i referential integrity

```
where not exists
        (select LIBRO.Soggetto
        from LIBRO
         where RIVISTA.Soggetto = LIBRO.Soggetto) ;
```

Se la query Q4 da un risultato nullo allora la condizione RIVISTA.Soggetto $\subseteq$ LIBRO.Soggetto è verificata, altrimenti la condizione non è vera.

(6) Generazione della regola derivata e sua valutazione

Non tutte le regole generate dal processo di derivazione vengono poste nel rule set. Solo le regole derivate che presenteranno un valore dell'utilità superiore alla soglia minima S_{reg} verranno inserite. La fase successiva alla generazione di una regola sarà quindi quella di effettuare una sua valutazione. La valutazione di una regola viene fatta calcolando il suo valore utilità U(RD). La formula per il calcolo dell'utilità di una regola è uguale a :

$$U(RD) = Risp(RD) \times Freq - Man$$

Al momento in cui si crea una nuova regola il valore della frequenza (Freq)viene posto uguale a 1, e il costo di manutenzione (Man) uguale a 0. Dunque la sua l'utilità viene posta uguale a:

$$U = Risp(RD)$$

Il risparmio Risp(RD) della regola derivata è calcolato rispetto alla query iniziale dalla quale è stata ricavata la regola proposta da cui si è derivata RD. Quindi Risp(RD) è la differenza tra il costo di esecuzione della query prima dell'applicazione della regola derivata RD, e il costo di esecuzione della query dopo l'applicazione della regola derivata. Quando ad una regola derivata RD è stato assegnato il rispettivo valore di utilità U(RD), bisogna verificare se tale valore e' superiore al valore limite S_{reg}.

(7) La regola derivata non supera la soglia minima S_{reg}

Se la regola derivata non supera la soglia S_{reg} significa che all'interno del rule set vi sono delle regole che presentano un valore di utilità superiore a quella della regola appena creata. In questo caso per registrare, che da una regola proposta RP, si è derivato una regola RD con un valore dell'utilità tale da non permettere alla regola RD di essere inserita nel rule set, si inserisce l'antecedente (per le regole proposte generate da H3 e H5 inseriamo anche il conseguente) di RP nell'insieme rp set e ad essa viene associato il suo valore dell'utilità U(RD).

(8) La regola derivata supera la soglia minima S_{reg}

Le regole derivate possono essere inserite nel rule set senza effettuare nessun controllo sulla ridondanza della regola rispetto all'insieme delle regole.

Quando una regola derivata supera la soglia S_{reg} vengono aggiornati alcuni valori della rule class alla quale appartiene la regola.

$$cont = cont + 1$$

Si aggiorna il fattore di selettività medio degli antecedenti delle regole fsa.

$$fsma = \frac{fsma \leftarrow (cont - 1)}{cont} + \frac{fsa(RD)}{cont}$$

Il valore upm della rule class viene aggiornato in modo da considerare anche il valore U(RD). Così, se è upm il valore del costo risparmiato potenziale medio della rule class, il suo nuovo valore sarà:

$$upm = \frac{upm \leftarrow (cont - 1)}{cont} + \frac{U(RD)}{cont}$$

La regola proposta che ha generato la regola RD viene cancellata dal sistema.

Ora la regola RD può essere inserita nell'insieme rule set. Se il numero delle regole dell'insieme ha già raggiunto il suo valore massimo, si dovrà eliminare dall'insieme la regola avente il valore utilità più basso in modo da non superare il valore limite del numero di regole massimo dell'insieme rule set.

CAPITOLO 3

3.1 Generazione regole con un approccio data-driven

Generalità

Il processo per l'estrazione di regole, può essere condotto attraverso un approccio di tipo data-driven. Come accade spesso ogni tipo di soluzione ad un problema ha i suoi difetti e i suoi pregi ; dunque la soluzione ottimale va ricercata confrontando le diverse soluzioni cercando di eliminare i difetti di un metodo attraverso i pregi di un'altro. Nella ricerca delle regole attraverso un approccio query-driven troviamo un ottimo metodo per individuare regole su attributi che hanno buone probabilità di comparire nelle interrogazioni al database. Queste regole non vengono utilizzate per l'ottimizzazione di tale query, ma serviranno ad ottimizzare interrogazioni future aventi delle qualificazioni simili alla query corrente. L'ipotesi principale su cui si basa questo procedimento è che in un database vengono inserite query aventi caratteristiche che si ripetono, ad esempio interrogazioni su uno stesso gruppo di attributi. Dunque regole utili all'ottimizzazione di una query possono essere utili anche per l'ottimizzazione di query future. Ma l'informazione che ci viene fornita dall'approccio query-driven, che risulterà essere importante nell'analisi che seguirà in questo capitolo, è che esso individua le relazioni e gli attributi sui quali è conveniente estrarre il maggior numero di regole. La limitazione di questa strategia è che possibile generare regole che non verranno mai usate. Ad esempio questo può accadere quando la regola generata presenta un antecedente con delle restrizioni molto selettive, e quindi con un grado di applicabilità basso. Oppure quando vengono introdotte una successione di query che effettuano delle ricerche diverse, quindi le regole generate per una interrogazione non saranno utili per le query successive.

Nell'approccio data-driven, che è basato su una analisi diretta dei dati per l'estrazione delle regole, è possibile generare tutte le regole che si ritengono utili, per una qualsiasi relazione. Il tipo e il numero di regole che è possibile generare è molto ampio e rappresenta il difetto di tale strategia

Per definire il tipo di regola si fa riferimento principalmente a due elementi : gli insiemi degli attributi che compongono l'antecedente e il conseguente della regola; il tipo di restrizioni generate (ad esempio restrizioni di uguaglianza, o che esprimo range di appartenenza più o meno ampi).

In questo capitolo verrà presentata una strategia di apprendimento di regole tipiche degli approcci data-driven, ma che utilizza informazioni generate dall'applicazione delle euristiche, che permettono di scegliere su quali attributi compiere l'analisi e quali regole risultano essere utili all'SQO.

Nella figura 1 di pagina 97 sono mostrati i due moduli principali che costituiscono il sistema per la generazione delle regole. Il modulo "Inizializzazione processo generazione regole" analizza le informazioni presenti nell'insieme RULE CLASS. Le rule class permettono di tener traccia di tutti i tentativi passati, sia quelli riusciti che quelli non riusciti, di derivare regole attraverso il sistema query-driven. È possibile definire alcuni criteri (vedi pag 75/76) per selezionare alcune rule class dalle quali ricavare i seguenti parametri da spedire al modulo "Generazione regole proposte".

1) una congiunzione di equijoin, $R_1.B_1=R_2.B_2 \wedge \ldots \wedge R_{m-1}.B_{m-1}=R_m.B_m$, che definiscono una relazione R contenete le tuple dalle quali si vogliono estrarre delle regole,
2) un espressione della forma $A_1 \wedge \ldots \wedge A_{n-1} \rightarrow A_n$, dove gli A_i, $1 \leq i \leq n-1$, sono gli attributi della relazione R che dovranno essere ristretti nell'antecedente delle regole generate, mentre A_n è l'attributo del conseguente delle regole generate.

Occorrerà definire dei valori che permettano di stabilire la selettività, sia dell'antecedente che del conseguente, delle regole che si dovranno generare.

3) il fattore di selettività minimo dell'antecedente delle regole S_{fsa},
4) il fattore di selettività massimo del conseguente delle regole S_{fsc}.

Nelle sezioni che seguono verrà descritto il modulo "Generazione regole proposte". In tale modulo si procede ad estrarre delle regole da una relazione R, dove R può anche non essere una relazione base del database.

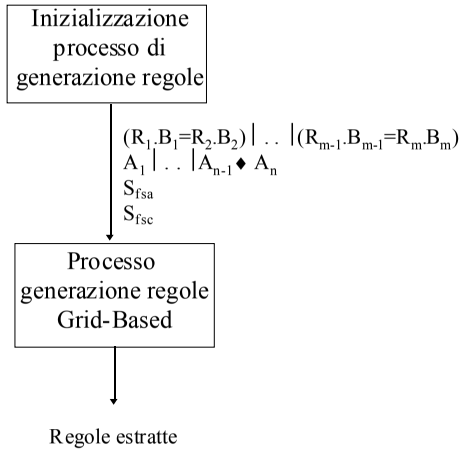


Figura 4

3.2 Generalità sul sistema di distribuzione dei dati grid-based

Una distribuzione dei dati è una rappresentazione formale dello stato di un dato database, una funzione che mappa una griglia e un database al numero di tuple di ogni cella della griglia. Una griglia è una decomposizione regolare dello spazio di tutte le possibili tuple nel database in una collezione di celle disgiunte. Una griglia è specificata su un insieme di coordinate, e queste coordinate sono funzioni degli attributi del database. La dimensione della griglia rappresenta il numero di coordinate se si assume che le coordinate siano linearmente indipendenti. Le celle della griglia sono date dalla divisione dei possibili valori di ogni coordinata in un insieme finito di range. Denotiamo con S_i (i = 1, .. ,n) l'insieme dei range per le n coordinate di una griglia di dimensione n. L'insieme delle celle per questa griglia può essere rappresentata dal prodotto cartesiano degli insiemi dei range, $S_1\ S_2 \ldots S_n$.

Uno dei problemi maggiori che si presentano per la costruzione di una griglia è la scelta dell'ampiezza dei range. Griglie su attributi a valori discreti possono essere creati contando le occorrenze di ogni differente valore come in un istogramma, e perciò la scelta del range non è un problema in questo contesto. La scelta del range

per attributi a valori continui è un compito più difficile. Il problema che si vuole discutere è quello di decomporre il dominio, di un attributo a valori continui, in un insieme finito di range. I più importanti fattori che occorre tenere presente per decidere la migliore decomposizione di un attributo sono il tipo di regole che si vuole estrarre, il costo di memorizzazione della griglia, il costo di generazione delle regole dalla griglia. Una decomposizione grossolana di un attributo riduce il costo di memorizzazione e il costo di scansione della griglia. Con tale decomposizione è probabile generare una struttura che non permetta all'algoritmo di ricerca di estrarre delle regole utili.

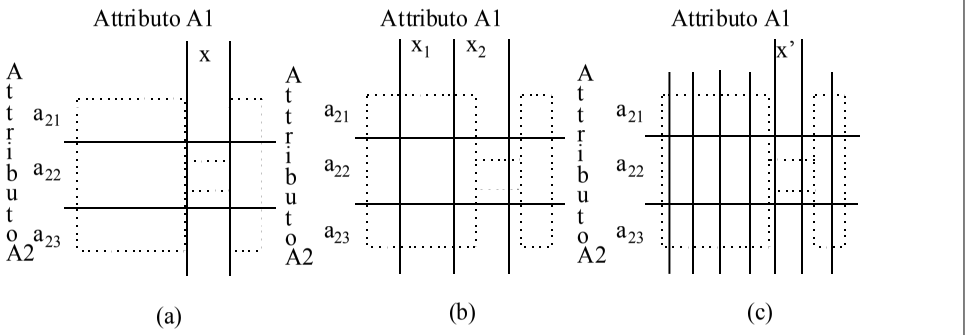


Figura 5

Una decomposizione fine dell'attributo incrementerà i costi di immagazzinamento e di scansione. Con tale decomposizione si possono acquisire molte regole che presentano però un aspetto negativo. Con la decomposizione fine le regole possono coprire molti piccoli sub-range dell'attributo, questo rende la regola applicabile ad un piccolo insieme di dati. D'altra parte la decomposizione fine permette di trovare regole che altrimenti verrebbero trascurate. Nella figura sopra è mostrato un esempio di griglia con attributo a valori continui il quale è stato decomposto, rispettivamente nelle figure (a), (b), (c), in modo ideale, grossolano, e fine. La regione tratteggiata indica la distribuzione dei dati nella griglia. Nel caso ideale, l'algoritmo scoprirà la

seguente regola : (A1 = x) --> (A2 = a22) (reg1). Questa regola sarà trascurata nel caso di decomposizione grossolana, perché i dati sono distribuiti su tutti i valori dell'attributo

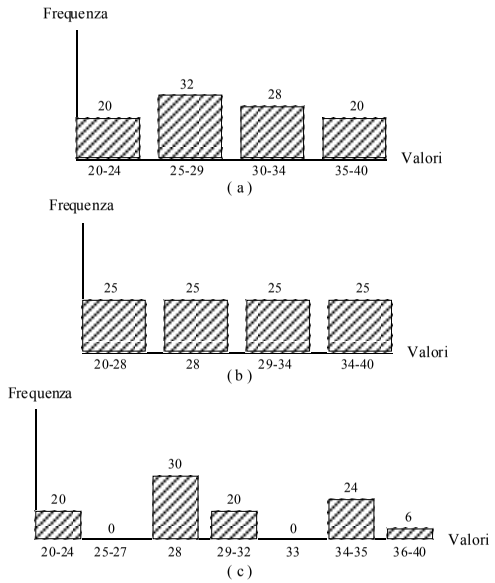


Figura 6

Età	Numero	Somma
20	2	2
21	3	5
22	5	10
23	8	18
24	2	20
25	0	20
26	0	20
27	0	20
28	30	50
29	2	52
30	8	60
31	5	65
32	5	70
33	0	70
34	10	80
35	14	94
36	2	96
37	1	97
38	1	98
39	1	99
40	1	100

A2 quando A1=x1 e A1=x2. Nel caso fine può essere estratte la seguente regola: (A1=x') → (A2=a22) (reg2). Questa regola è più debole della reg1 dato che x', nella figura 2.c, è un sub-range di x nella figura 2.a. Possiamo affermare che la decomposizione fine, in questo caso, non sfrutta a pieno le caratteristiche implicate dai dati. Ci sono diversi metodi statistici che aiutano nella decomposizione del

dominio in un insieme finito di range per una griglia. Vediamo tre metodi che costruiscono in maniera diversa i range della griglia [(40) Mannino M. V., (1988)]. In Mannino (1988) sono proposti i tre tipi di istogrammi che andiamo ora a descrivere. Questi metodi sono stati proposti per realizzare un sistema per la stima dei costi di esecuzione di query. Alcuni di questi criteri verranno utilizzati per generare un algoritmo che sia in grado di estrarre delle regole da un database.

Nella figura 3 sono riportati tre grafici a barre in cui sull'asse orizzontale vengono riportati i range di suddivisione del dominio dell'attributo, e sull'asse verticale la frequenza delle tuple appartenenti al dato range. I grafici fanno riferimento ai dati posti nella tabella alla destra del grafico. Nella tabella è riportato la distribuzione di frequenza dell'attributo "età" su 100 studenti. La forma del grafico cambia a secondo del criterio scelto nel generare i range. Nella figura 3 sono rappresentati i seguenti tre criteri :

a)equal-whith

b)equal-height

c)variable-whith

Il criterio equal-whith divide i valori dell'attributo in range di uguale ampiezza. Il criterio equal-height definisce i range in modo che il numero di elementi all'interno di ogni range sia lo stesso. Il criterio variable-width definisce i range in modo che la frequenza all'interno di ogni range rispetti un certo criterio, ad esempio che i valori siano uniformemente distribuiti.

Prima di vedere quali sono i fattori rilevanti che occorre tenere presenti durante la generazione delle regole diamo la seguente definizione.

Definizione: Chiamiamo fattore di selettività f_s dell'antecedente (conseguente) di una regola, rispetto ad una relazione R, il rapporto tra il numero di tuple che verificano le restrizioni dell'antecedente (conseguente) e il numero di tuple della relazione R

$$f_s = \frac{NT(Ante)}{NT(R)}$$

Le regole che si vogliono ottenere devono avere le seguenti due caratteristiche principali:

1) un antecedente con un fattore di selettività basso in modo che il grado di applicabilità della regola sia alto.

2) un conseguente con un fattore di selettività sufficientemente piccolo in modo da garantire, in seguito ad un suo inserimento in una query da ottimizzare semanticamente, un risparmio.

Il punto 1) è giustificato dal fatto che, se il f_s dell'antecedente è molto alto, la probabilità che tale regola venga utilizzata è bassa. Ad esempio con una restrizione nell'antecedente del tipo (Imp.Codice = 100), con una relazione Imp che conta 10000 tuple, presenta un f_s=1/10000. Questa regola può essere usata solo in una query che cerchi un impiegato con codice uguale a 100.

Il punto 2) intende assicurare che l'applicazione della regola comporti effettivamente un risparmio nell'esecuzione di una query. Se ad esempio, una regola ha un conseguente, su un attributo indicizzato, poco selettivo, anche se introduciamo in una query il conseguente di tale regola non avremo nessun miglioramento nell'esecuzione della query. Si deve inoltre tenere conto che il risparmio apportato da una regola deve essere superiore al costo aggiuntivo dovuto alla fase di ottimizzazione semantica. Si dovrà stabilire una soglia massima del f_s. La scelta può essere condotta considerando le formule per il calcolo del costo di accesso sequenziale , con indice clustered e con indice unclustered.

$$C(Scan) = NP$$

$$C(IX_{CLUSTERED}) = \lceil f_s \times NP \rceil$$

$$C(IX_{UNCLUSTERED}) = \lceil \Phi(NT \times f_s, NP) \rceil$$

Ad esempio, per le regole che hanno un conseguente su un indice, possiamo scegliere il valore massimo del f_s in modo che verifichi la seguente diseguaglianza

$$\frac{C(IX)}{C(Scan)} > M$$

dove M è un valore prefissato.

Le regole che si vogliono generare sono composte da restrizioni con operatori del tipo (=, >,<, ≥, ≤) ; inoltre , visto che si ha a che fare con intervalli verrà usato anche l'operatore between. Si genereranno delle regole del tipo ($JC \wedge P \rightarrow Rest$), dove JC è un insieme di equi-join che definiscono una relazione R; P è un insieme di restrizioni sulla relazione R e Rest è una restrizione su R che rappresenta il conseguente della regola. Quindi l'antecedente delle regole può essere composto da più restrizioni congiunte, mentre il conseguente è formato da una sola restrizione. Dunque in una griglia di dimensione D avremo una restrizione come conseguente e un numero massimo di D –1 restrizioni che formano l'antecedente.

3.3 Generazione griglia

Prima di iniziare a descrivere l'algoritmo per la generazione dell'istogramma multidimensionale, vediamo quale forma può assumere un istogramma per descrivere una distribuzione di tuple. Come esempio prenderemo un istogramma a due dimensioni della figura 4, che descrive la distribuzione di tuple di una relazione R con attributi x e y a valori discreti. La regione 1 e 2 rappresentano lo spazio delle tuple della relazione R. I punti all'interno delle regioni rappresentano le tuple.

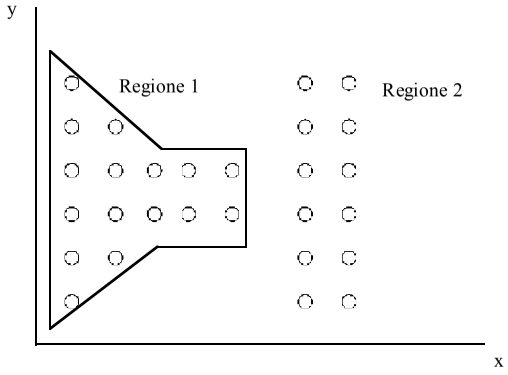


Figura 7

Il problema di generare un istogramma equi-weight è equivalente a coprire tutte le tuple all'interno dello spazio delle tuple con rettangoli che contengono uno stesso numero di tuple. Un esempio è mostrato in figura 5 nel quale i rettangoli contengono ognuno *t* tuple. I rettangoli sono chiamati *istogrammi equi-weight* o *buckets equi-weight*.

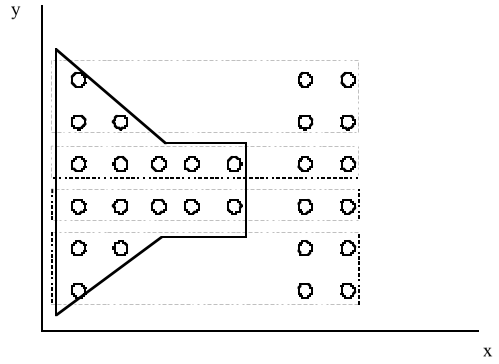


Figura 8

Nella figura 5 è mostrata una copertura dello spazio delle tuple attraverso la partizione dell'asse x in intervalli di uguale dimensione. La suddivisione permette di creare dei rettangoli di uguale area che coprono l'intero spazio delle tuple. Ogni rettangolo contiene un numero di tuple che può differire da un rettangolo e l'atro. I rettangoli sono chiamati *istogrammi equi-width o buckets equi-width*.

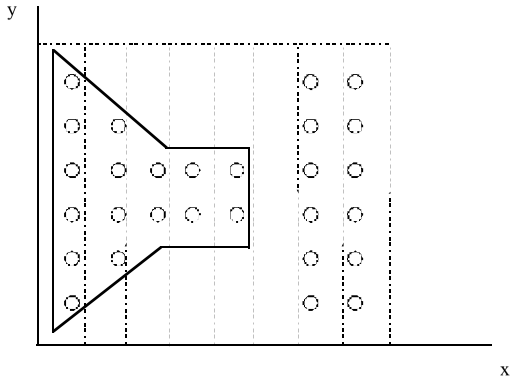


Figura 9

La costruzione della griglia prevede di utilizzare il criterio equal-width per la partizione degli attributi che andranno a formare l'antecedente della regola, e il criterio equal-

height per partizionare l'attributo del conseguente della regola che si vuole generare. È stato scelto il criterio equal-width per gli attributi dell'antecedente perché esso è in grado di individuare range vuoti (range con distribuzione uniforme con densità zero) e per la sua facilità di implementazione. Per la partizione dell'attributo del conseguente si è scelto il criterio equal-height perché esso permette di effettuare una stima accurata del numero di tuple soddisfacenti il conseguente.

Iniziamo a considerare la costruzione di una griglia a due dimensioni, descrivendo le caratteristiche principali del metodo adottato per la sua costruzione. Successivamente verrà proposto un esempio con una griglia a tre dimensioni. Concluderemo questa sezione con alcune note sull'implementazione del metodo utilizzato.

3.3.1 Generazione di una griglia a 2 dimensioni

Si consideri una relazione R(A, B) in cui A e B sono due attributi a valori numerici interi. Indicheremo con NT il numero di tuple della relazione R. Le regole che si vogliono estrarre sono del tipo (Rest(A) $\rightarrow$ Rest(B)), aventi una

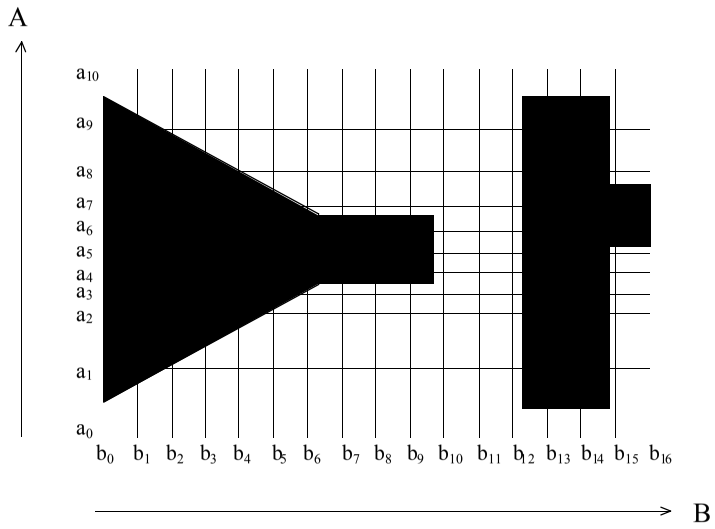


Figura 10

restrizione sull'attributo A come antecedente e una restrizione sull'attributo B come conseguente. Consideriamo la figura 7 in cui è stata rappresentata lo spazio delle tuple della relazione R. Sull'asse orizzontale sono riportati i valori dell'attributo B mentre sull'asse verticale sono disposti i valori dell'attributo A. Si consideri una routine di ordinamento chiamata SORT(P_1,P_2,P_3) che ha tre parametri. Il primo parametro indica l'attributo sul quale la relazione deve essere ordinata in modo ascendente. Il secondo e il terzo parametro indicano gli estremi dell'intervallo di valori dell'attributo P_1 nel quale effettuiamo l'ordinamento. Ad esempio SORT(B, 100, 200) ordina le tuple con valore compreso tra 100 e 200 dell'attributo B in modo ascendente. Indicheremo con $bucket_i$, con $i \in \{A, B\}$, il numero di divisioni per ogni attributo i-esimo (dimensione). Il dominio dell'attributo B viene suddiviso secondo il criterio equal-width, mentre l'attributo A viene suddiviso secondo il criterio equal-height.

Dapprima la relazione R viene ordinata sull'attributo A invocando SORT(A, 0, A_M) dove si è indicato con A_M il valore massimo dell'attributo A. Avendo stabilito che il

numero di divisioni dell'attributo A è uguale a $bucket_A$, il numero di tuple per ogni partizione di A è uguale a NT(R) / $bucket_A$. Si effettua poi un secondo ordinamento della relazione R sull'attributo B invocando SORT(B, 0, B_M). Vengono poi ottenute le varie partizioni di uguale taglia. Alla prima partizione fanno parte la prima tupla fino alla tupla ($\frac{NP(R)}{bucket_A}+1$); alla seconda partizione fanno parte le tuple da ($\frac{NP(R)}{bucket_A}+1$) fino (2* $\frac{NP(R)}{bucket_A}$), etc. Nella figura 7 le partizioni sull'asse A sono indicati con gli intervalli $[a_i, a_{i+1}[$. Durante il partizionamento vengono quindi stabiliti i valori a_i che delimitano gli intervalli di A.

Il dominio di B viene diviso in un numero di range uguale a $bucket_B$ e con un numero di tuple per partizione che varia. Nella figura le partizioni dell'asse B sono indicate con gli intervalli $[b_i, b_{i+1}[$. Durante il partizionamento di B vengono eseguite tre operazioni. La prima è quella di memorizzare per ogni $range_B$ il numero di tuple di ogni intervallo di B. La seconda è di memorizzare, per ogni $range_B$, i $range_A$ a cui appartengono le tuple che sono coperte dal $range_B$. Facciamo un esempio utilizzando la figura 7. Per il $range_{b1} = [b_0, b_1[$ verrà memorizzato il numero di tuple NT_{b1} che appartengono al primo intervallo di B. Sarà memorizzato anche l'intervallo $[a_0, a_{10}]$ dato che esso rappresenta l'intervallo minimo che copre tutte le tuple appartenenti al $range_{b1}$. Per il $range_{b7} = [b_6, b_7[$ si memorizzerà NT_{b7} e l'intervallo $[a_3, a_7[$. La terza operazione prevede l'ottimizzazione degli intervalli di B.

L'ottimizzazione degli intervalli comporta tre tipi di trasformazioni dei range:

1) Sia $[b_i, b_{i+1}[$ un intervallo di B e $[a_j, a_y[$ il range di A ad esso associato. Se l'intervallo precedente $[b_{i-1}, b_i[$ ha associato il range $[a_j, a_y[$, allora possiamo eliminare il taglio b_i e considerare il nuovo range $[b_{i-1}, b_{i+1}[$ avente un numero di tuple uguale a $NT_{b'h} = NT_{bi-1} + NT_{bi}$ con un range di A uguale a $[a_j, a_y[$.
2) Sia $[b_i, b_{i+1}[$ un intervallo di B e $[a_j, a_y[$ il $range_A$ ad esso associato (il $range_A$ può essere anche nullo). Se l'intervallo precedente $[b_{i-1}, b_i[$ presenta un $range_A$ $[a_m, a_n[$ tale che $a_m < a_j$ o $a_n > a_y$, allora scorriamo le tuple, a partire da quelle con valore $B=b_i$, in ordine inverso fino a trovare una tupla $T(A = a_w, B = b_w)$, con o $a_w < a_j$ o $a_w \geq a_y$. Se l'intervallo ha un $range_A$ nullo si considera la prima tupla che si incontra. Il nuovo valore di b_i sarà uguale a b_w. Avremo un ampliamento dell'intervallo $[b_i, b_{i+1}[$ che diventerà $[b_w, b_{i+1}[$.
3) Sia $[b_i, b_{i+1}[$ l'intervallo di B corrente e $[a_j, a_y[$ il range di A ad esso associato. Se l'intervallo precedente $[b_{i-1}, b_i[$ presenta un $range_A$ $[a_m, a_n[$ tale che $a_m > a_j$ o $a_n < a_y$ (oppure un $range_A$ nullo), allora scorriamo le tuple, a partire da quelle con valore $B=b_i$, in ordine ascendente fino a trovare una tupla $T(A = a_w, B = b_w)$, con o $a_w < a_m$ o $a_w \geq a_n$. Se l'intervallo precedente era nullo si considera la prima tupla che si incontra. Il nuovo valore di b_i sarà uguale a b_w. Avremo un ampliamento dell'intervallo $[b_{i-1}, b_i[$ che diventerà $[b_{i-1}, b_w[$.

La figura 8 mostra la griglia ottenuta applicando le trasformazioni 1), 2) e 3). I nuovi intervalli di B sono delimitati da linee continue, mentre le linee tratteggiate delimitano i vecchi intervalli.

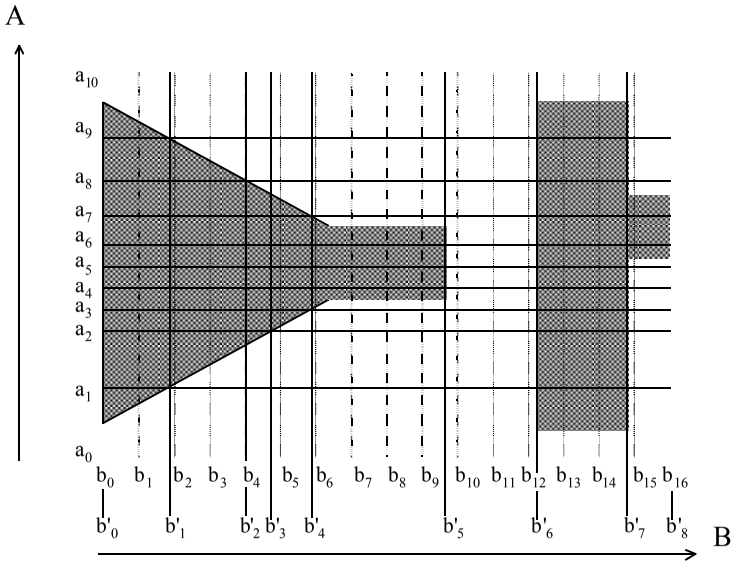


Figura 11

Gli ultimi dati che occorre calcolare per valutare le regole che si andranno ad estrarre, sono quelli che ci permettono di calcolare il numero di tuple di ogni intervallo di A, e quindi il fattore di selettività del conseguente delle regole. Supponiamo di considerare l'intervallo $[b'_4, b'_5[$ con $range_A$ $[a_3, a_7[$. Il numero di tuple del $range_A$ sarà uguale a

$$NTA_{b'5} = (7-3) \leftrightarrow \frac{NT(R)}{bucket_A}$$

Allo stesso modo è possibile trovare il numero di tuple di ogni intervallo di A associato ad un particolare range di B.

A questo punto i dati in nostro possesso sono quelli rappresentati nella seguente tabella.

$Range_B$	N° tuple del $range_B$	$Range_A$	N° tuple del $range_A$
$[b'_0, b'_1[$	$NT_{b'1}$	$[a_0, a_{10}]$	$10 \times TB$
$[b'_1, b'_2[$	$NT_{b'2}$	$[a_1, a_9[$	$8 \times TB$
$[b'_2, b'_3[$	$NT_{b'3}$	$[a_1, a_8[$	$7 \times TB$
$[b'_3, b'_4[$	$NT_{b'4}$	$[a_2, a_8[$	$6 \times TB$
$[b'_4, b'_5[$	$NT_{b'5}$	$[a_3, a_7[$	$4 \times TB$
$[b'_5, b'_6[$	$NT_{b'6}$	Null	
$[b'_6, b'_7[$	$NT_{b'7}$	$[a_0, a_{10}]$	$10 \times TB$
$[b'_7, b'_8]$	$NT_{b'8}$	$[a_5, a_8[$	$3 \times TB$

dove si è indicato con TB il numero di tuple appartenenti ad ogni $range_A$.

OSSERVAZIONI

Nella figura 7 si vede come la distanza tra le linee orizzontali, che delimitano i $range_A$, diminuisca nelle zone in cui vi è una maggiore densità di tuple. Questo fa si che la fase di generazione del conseguente di una regola risulti più sensibile la dove vi è una popolazione (tuple) a maggior densità, mentre è più grossolana nelle zone di minor densità. La scelta dell'ampiezza del $range_B$ e cruciale per la qualità del risultato e dunque delle regole che si ottengono. Più l'ampiezza è minore e più fine sarà l'analisi della distribuzione dei dati. Per scoprire intervalli con densità di tuple nulla, l'ampiezza del range scelto deve essere minore di delta / 2, dove delta è l'ampiezza minima dei range che si vogliono scoprire.

3.3.2 Generazione di una griglia a 3 dimensioni

Generiamo ora una griglia sulla relazione R(A, B, C), in cui si suppone che tutti gli attributi sono a valori interi. La griglia avrà quindi dimensione tre in cui avremo sempre A come attributo conseguente, mentre B e C saranno gli attributi antecedenti. Le operazioni iniziali da effettuare sono le medesime di quelle viste nel caso della griglia a 2 dimensioni. Si sceglie il numero di divisioni che si vuole operare su ogni attributo. Siano $bucket_A$, $bucket_B$ e $bucket_C$ rispettivamente il numero di divisioni sull'asse A, B e C. Ordiniamo la relazione R sull'attributo A invocando la funzione SORT(A, 0, A_M). Si effettua una partizione dell'attributo A secondo il criterio equi-height. Ordiniamo la relazione sull'attributo B ed effettuiamo il partizionamento seguendo il criterio equi-width. Otteniamo dunque $bucket_B$ partizioni di ampiezza uguale. Chiamiamo queste partizioni : *partizioni primarie*. Si esegue la fase di ottimizzazione dei range ricavati su B, ottenendo $bucket_{B'}$ intervalli $[b'_{i-1}, b'_i[$. Con $bucket_{B'}$ si è indicato il numero di partizioni dell'attributo B dopo la fase di ottimizzazione. Poi si ordina ognuna di queste partizioni sull'attributo C invocando SORT(C, b'_{i-1}, b'_i), con $0 \le i \le bucket_{B'}$. Dunque dividiamo ogni partizione primaria in $bucket_C$ *partizioni secondarie*. Avremo che le partizioni secondarie che sono state ricavate da una singola partizione primaria, sono tutte associate alla partizione primaria genitrice. Anche sul $range_C$ eseguiamo la fase di ottimizzazione degl'intervalli.

Il nuovo numero di intervalli dell'attributo C sarà uguale ad $bucket_{C'}$. il numero totale delle partizioni secondarie sarà uguale a NB = ($bucket_{B'}$ * $bucket_{C'}$). NB è anche il numero di regole che l'algoritmo è riuscito ad estrarre.

Le fasi principali dell'algoritmo descritto sono rappresentate nelle figure che seguono.

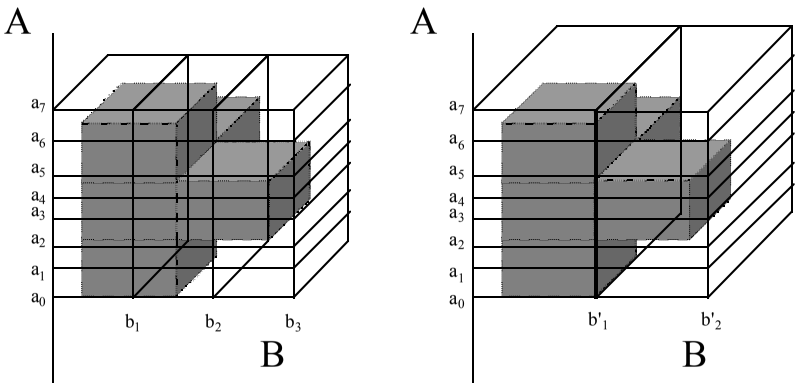


Figura 9

Nella figura 9 vediamo la griglia tridimensionale con in primo piano gli attributi A e B. Come numero di intervalli di ogni asse si sono scelti i seguenti valori: $bucket_A$=7, $bucket_B$=3, $buket_C$=3. Il partizionamento di A è espresso attraverso i valori a_i, $0 \le i \le 7$. La suddivisione dell'asse B è espressa con i valori b_i, $0 \le i \le 3$. La griglia a sinistra mostra la suddivisione dell'intervallo di B prima dell'ottimizzazione. La figura di destra mostra la griglia dopo l'ottimizzazione. Alla fine del **p**artizionamento **p**rimario abbiamo i seguenti dati:

PP1) $[b'_0, b'_1[, [a_0, a_7[$;

PP2) $[b'_0, b'_1[, [a_2, a_5[$.

Nella figura 10 sottostante è mostrato il **p**artizionamento **s**econdario sulla prima partizione primaria che è espressa dall'intervallo $[b'_0, b'_1[$.

Dopo la fase di ottimizzazione (figura di destra) otteniamo i seguenti dati:

PP1-**PS**1) $[b'_0, b'_1[, [c'_0, c'_1[, [a_0, a_7[$;

PP1-**PS**2) $[b'_0, b'_1[, [c'_1, c'_2[, [a_2, a_5[$.

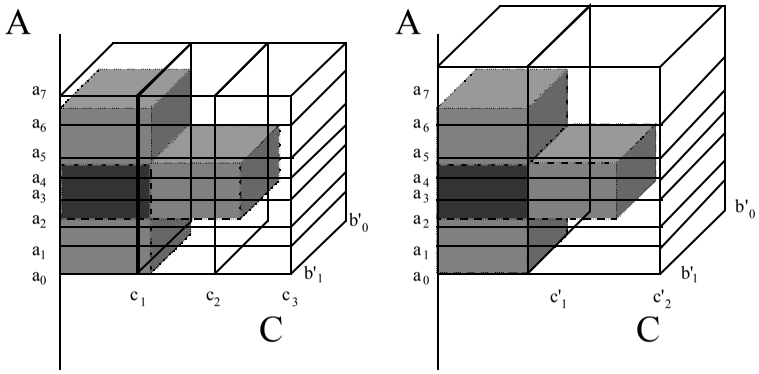


Figura 10

Nella figura 11 abbiamo il partizionamento secondario della seconda partizione di B che è espressa dall'intervallo $[b'_1, b'_2]$. Dopo la fase di ottimizzazione (figura di destra) otteniamo i seguenti dati:

PP2-**PS**1) $[b'_1, b'_2[, [c'_0, c'_1[, [a_2, a_5[$;

PP2-**PS**2) $[b'_1, b'_2[, [c'_1, c'_2[,$ null.

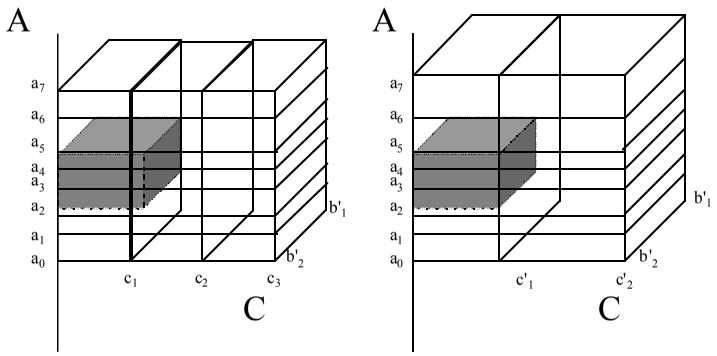


Figura 11

Le quattro partizioni trovate corrispondono alle seguenti 4 regole:

Reg1)(B between b'_0 and b'_1) $\wedge$ (C between c'_0 and c'_1) $\rightarrow$ (A between a_0 and a_7)

Reg2)(B between b'_0 and b'_1) $\wedge$ (C between c'_1 and c'_2) $\rightarrow$ (A between a_2 and a_5)

Reg3)(B between b'_1 and b'_2) $\wedge$ (C between c'_0 and c'_1) $\rightarrow$ (A between a_2 and a_5)

Reg4)(B between b'_1 and b'_2) $\wedge$ (C between c'_0 and c'_1) $\rightarrow$ NULL

Per stimare l'utilità di queste regole, ad esse sono associati i seguenti dati:

Regola	***N° tuple antecedente***	***N° tuple conseguente***
Reg1	$NT_{p1\text{-}1}$	7 _ TB
Reg2	$NT_{p1\text{-}2}$	3 _ TB
Reg3	$NT_{p2\text{-}1}$	3 _ TB
Reg4	$NT_{p2\text{-}2}$	0

L'algoritmo proposto può facilmente essere esteso per generare griglie con dimensioni maggiori. Per esempio, una griglia a 4 dimensioni costruita sulla relazione R(A,B,C,D), dovremmo ordinare ognuna delle ($bucket_{B'}$ * $bucket_{C'}$) partizioni secondarie sull'attributo D e dividere ogni partizione secondaria in $bucket_D$ *partizioni terziarie*. Si esegue l'ottimizzazione sulle partizioni terziarie ottenendo $bucket_{D'}$ nuove partizioni terziarie. Il numero totale delle partizioni che si vengono ad ottenere è uguale a ($bucket_{B'}$ * $bucket_{C'}$ * $bucket_{D'}$).

L'ultimo passo dell'algoritmo sarà quello di scegliere le regole che presentano una selettività del conseguente inferiore alla soglia S_{fsc}, e un fattore di selettività dell'antecedente superiore alla soglia S_{fsa}.

3.3.3 Costo di costruzione della griglia

Si supponga di dover calcolare il costo della costruzione di una griglia a 4 dimensioni. Si prende in considerazione una relazione R(A, B, C, D) con un numero di pagine dati uguale a Z. Si assume che tutte le tuple della relazione abbiano la stessa dimensione. Il costo per ordinare l'intera relazione R è

$$C=C'*Z*\log(Z)$$

Dove C' è una costante. Inizialmente vengono eseguiti due ordinamenti sull'intera relazione, uno sull'attributo conseguente (A) e uno su un attributo che farà parte dell'antecedente di una regola (volendo seguire l'esempio precedente prendiamo B).

$$C_1=C_2= C'*Z*\log(Z)$$

L'ordinamento di B ci consente di ottenere le partizioni primarie. Ognuna di queste partizioni primarie è ordinata per poter ottenere $bucket_C$ partizioni secondarie. Il costo di ordinamento per quest'ultima operazione è dato da

$$C_3 = C' * \sum_{1}^{bucket_B} \frac{NT_i}{TB} * \log \frac{NT_i}{TP}$$

Analogamente ,per la partizione terziaria, il costo di ordinamento è

$$C_4 = C' * \sum_{1}^{bucket_B * bucket_C} \frac{NT_j}{TB} * \log \frac{NT_j}{TP}$$

Il costo totale risulta essere quindi

$$C_{TOT} = C_1 + C_2 + C_3 + C_4 =$$

$$= C' * \overline{2Z} * \log Z + \sum_{1}^{bucket_B} \frac{NT_i}{TB} * \log \frac{NT_i}{TP} + \sum_{1}^{bucket_B * bucket_C} \frac{NT_j}{TB} * \log \frac{NT_j}{TP}$$

Supponiamo che ogni partizione ottenuta su una stessa dimensione D_i contenga lo stesso numero di tuple. Quindi le $bucket_B$ partizioni primarie conteranno ognuna $\frac{NT(R)}{bucket_B}$ tuple, le $bucket_B * bucket_C$ partizioni secondarie conterranno ognuna $\frac{NT(R)}{bucket_B * bucket_C}$ tuple. Il costo totale diventa uguale a:

$$C_{TOT} = C_1 + C_2 + C_3 + C_4 =$$

$$= C' * Z\overline{2} \log Z + \log \frac{Z}{bucket_B} + \log \frac{Z}{bucket_B * bucket_C}$$

Generalizzando ad un sistema con dimensione D, e assumendo lo stesso numero p di divisioni per ogni partizionamento, il costo di ordinamento totale è

$$C_{TOT} = C' * Z * \log \frac{Z^D}{b^{\frac{(D+1)(D-2)}{2}}}$$

Si osservi come il costo di ordinamento per un partizionamento diminuisca all'aumentare della dimensione della griglia.

CAPITOLO 4

Introduzione

In questo capitolo prenderemo in esame il problema della manutenzione delle regole semantiche dinamiche. Esso si compone di due sezioni: Manutenzione del rule set e Controllo e ripristino della validità delle regole del rule set. Nella prima sezione si descrive il problema generale della manutenzione delle regole, riportando alcune soluzioni che sono state proposte in letteratura; poi si considera un esempio pratico per la manutenzione riprendendo alcuni concetti ed esempi considerati nei capitoli precedenti. Nella seconda sezione si approfondisce il discorso del controllo e del ripristini della validità delle regole dinamiche in seguito alle transazioni del database.

4.1 Manutenzione del rule set

Il rule set contiene l'insieme delle regole che l'ottimizzatore semantico utilizza per effettuare le trasformazioni semantiche delle query immesse nel sistema dagli utenti. In esso troviamo la conoscenza semantica dinamica del sistema necessaria all'SQO. Il processo di SQO comprende un *matching cost* dovuto alla scansione di tutte le regole a sua disposizione per vedere quale di queste regole convenga applicare. Avere un numero elevato di regole comporta quindi un aumento del matching cost con un possibile deperimento delle prestazioni dell'ottimizzatore. Se il tempo impiegato per l'ottimizzazione semantica è troppo elevato si potrebbe verificare la condizione per cui, il tempo di esecuzione di una query risparmiato grazie al processo di SQO, è minore del tempo che ha impiegato il sistema per effettuare tale ottimizzazione. Occorre quindi che nel rule set vi siano un numero non troppo elevato di regole ma con un grande valore di utilità. La politica del sistema di manutenzione del rule set è quella di tentare di diminuire la quantità e aumentare la qualità delle regole. La qualità di una regola viene misurata dalla sua utilità. Ogni regola del rule

set ha associato un valore utilità che dovrà essere tenuto aggiornato. Il valore utilità risulta essere calcolato attraverso tre parametri; il risparmio medio *risp*, la frequenza di utilizzo *f* e il costo di manutenzione *man*. Ogni regola del rule set viene associata a questi tre parametri i cui valori vengono aggiornati dal sistema di gestione a secondo dell'utilizzo della regola o del tempo che tale regola trascorre nel rule set.

Le operazioni principali di cui si deve occupare il gestore del rule set sono:

1) mantenimento della consistenza delle regole

2) aggiornamento del valore utilità delle regole

3) mantenimento del rule set come insieme minimale di regole

4) controllo del numero e della qualità delle regole

1) Mantenimento della consistenza delle regole

Differentemente dagli aggiornamenti che violano i vincoli di integrità (regole specificate dall'utente), gli aggiornamenti che violano le regole derivate sono accettate dal sistema ; dovremo quindi modificare tali regole. L'impatto di un aggiornamento su una regola derivata è simile all'impatto di un aggiornamento su una vista derivata o su un indice: risulta necessario determinare il costo di aggiornamento delle regole derivate.

Scoperta delle violazioni

Un aggiornamento che causa una violazione di una regola derivata definisce un *eccezione* a quella regola. Una volta che un eccezione è scoperta la regola violata deve essere cancellata o riscritta. I metodi usati per determinare e gestire eccezioni hanno associato un costo di manutenzione.

Il più semplice approccio per processare una violazione è di cancellare la regola violata. Se la regola cancellata era significativa, è probabile che una regola simile sarà derivata per lo stato del database che include l'aggiornamento che ha generato la violazione.

Per esempio consideriamo l'eccezione EX1 alla regola R1.La regola R1 dice che tutti i libro con soggetto uguale ad astrologia hanno una collocazione ugaule a 100. Se viene inserito un libro di astrologia con collocazione uguale a 104 si genera l'eccezione EX1.

R1	*Attributo*	*Vincolo*
Antecedente	LIBRO.Soggetto	(= 'Astrologia')
Conseguente	LIBRO.Collocazione	(= 100)

EX1	*Attributo*	*Vincolo*
Antecedente	LIBRO.Soggetto	(= 'Astrologia')
Conseguente	LIBRO. Collocazione	(= 104)

Si vede come il metodo che comporta la cancellazione della regola R1, risulta essere molto inefficiente poiché si cancella una regola molto utile a causa di una singola eccezione.

Alternativamente regole derivate possono essere modificate cosicché la nuova regola è valida per lo stato del database che include l'eccezione. Per esempio la regola R1 può essere modificata per conformarsi all'eccezione EX1. La nuova regola R9 che si genera è ora valida per lo stato del database dopo l'aggiornamento.

R2	*Attributo*	*Vincolo*
Antecedente	LIBRO.Soggetto	(= 'Astrologia')
Conseguente	LIBRO. Collocazione	(set (100, 104))

Ovviamente ci sono dei limiti sul numero di volte che possiamo riscrivere una regola. Se il numero di disgiunzioni nel conseguente aumenta, allora la selettività del conseguente può essere usata per determinare se la nuova regola deve essere cancellata.

Un'altro approccio è di non modificare la regola violata, ma far si che la regola abbia un riferimento all'eccezione. Occorrerà quindi, quando si considera una certa regola, prendere in considerazione anche i suoi riferimenti ad eccezioni. Questo metodo è consigliabile in sistemi con regole semplici. Come esempio consideriamo la query Q1. L'introduzione del conseguente della regola derivata R1 dovrebbe normalmente

portare a una contraddizione e generare una soluzione nulla. Ma se R1 include un riferimento al record dell'eccezione EX1, allora l'eccezione sarà usata nella trasformazione.

```
Select LIBRO.Titolo
from LIBRO                                          (Q1)
where LIBRO.Soggetto = 'Astrologia'
and LIBRO. Collocazione > 100;
```

Il sistema dovrà inoltre preoccuparsi di decidere quando una regola si è sovraccaricata di eccezioni e merita quindi di essere cancellata.

Costi di manutenzione C_{man}

La manutenzione di un insieme di regole utili fa parte del costo di un sistema database. Il *costo di manutenzione*, che noi indicheremo con C_{man} include il *costo della scoperta di violazioni* e il costo di manutenzione delle regole dopo che si è verificata la violazione. Includendo il valore C_{man} nel calcolo dell'utilità di una regola derivata, il sistema di manutenzione può distinguere tra regole che sono costantemente violate e regole che più probabilmente rimangono valide dopo un aggiornamento.

Il costo del controllo di integrità dipenderà dall'implementazione dell'insieme di regole e dalla progettazione fisica del database. La stima del costo del controllo di integrità è simile alla funzione della stima del costo per la manutenzione degl'indici.

Il costo del mantenimento di una regola una volta che è stata violata dipenderà dalla struttura del database e dal metodo scelto per la manutenzione dell'insieme delle regole. La tecnica scelta per la manutenzione dell'insieme delle regole dipenderà dall'applicazione; per esempio, in applicazioni dove si verificano pochi aggiornamenti, ci saranno poche violazioni di regole violate. Perciò usando il più semplice approccio della cancellazione delle regole violate ci aiuterà a ridurre la complessità della procedura di manutenzione.

Indipendentemente dal metodo scelto per la manutenzione delle regole derivate, il costo di manutenzione influisce sulla gestione dell'insieme rule set. L'efficacia del metodo di manutenzione influirà sulla qualità delle regole presenti nel rule set e a sua volta influirà sulle prestazioni dell'ottimizzatore semantico delle interrogazioni.

Nel nostro sistema il costo della manutenzione viene espresso attraverso il parametro C_{man}, dato dalla somma del costo di scoperta delle violazioni e del costo di ripristino della validità della regola. Come unità di misura del costo di manutenzione si adotta il numero di pagine che occorre accedere per eseguire le rispettive operazioni.

2) Aggiornamento dei parametri delle regole del rule set

Ogni volta che il processo di ottimizzazione semantica utilizza una regola del rule set occorre aggiornare i parametri associati alla regola utilizzata. I parametri da aggiornare sono il risparmio medio che l'applicazione della regola ha apportato nel processo di SQO , e la frequenza di utilizzo della regola.

Esempio. Supponiamo che in un processo di ottimizzazione sia stata usata la regola RD1.

$$RD1 : (A=20) \rightarrow (B=1000)$$

il risparmio apportato da RD1 è

$$risp(RD1) = 20$$

Gli aggiornamenti dei parametri associati alla regola RD1(rispm, freq, man) sono:

$$freq = freq + 1$$

$$rispm(RD1) = \frac{rispm(RD1) \leftarrow (freq - 1) + risp(RD1)}{freq}$$

Un altro tipo di aggiornamento, che occorre effettuare sui parametri che definiscono l'utilità, servono per tenere conto anche del valore tempo. Se una regola trascorre molto tempo nel rule set, senza essere mai utilizzata dall'SQO, vedrà il suo valore utilità diminuire. La diminuzione dell'utilità di una regola si ottiene diminuendo periodicamente il valore freq di ogni regola. Regole che hanno una frequenza di applicazione alta, e quindi utili al sistema per ottimizzare semanticamente molte query, vedranno il loro valore utilità rimanere a valori alti; mentre regole poco utilizzate vedranno il loro valore utilità decrescere nel tempo fino a che la loro utilità scenderà al disotto del valore soglia S_{reg}. Queste regole verranno eliminate dal rule set in modo da consentire l'introduzione di nuove regole.

3) mantenimento del rule set come insieme minimale di regole

Le regole da inserire nel rule set possono essere state prodotte o dal processo di derivazione delle regole proposte, o dal processo di generazione delle regole.

Nel primo caso siamo sicuri che, l'informazione semantica contenuta nella regola derivata, non è presente nel rule set. Infatti nella fase di creazione delle regole proposte, si effettua un controllo della conoscenza semantica presente nel rule set; e solo se si riscontra, attraverso l'applicazione di una euristica, che sarebbe utile aggiungere una certa conoscenza semantica nel rule set, si genera una regola proposta. La regola derivata dalla regola proposta vuole sempre aggiungere un informazione mancante all'insieme delle regole.

Nel secondo caso, vengono generate delle regole senza tenere conto dell'informazione semantica contenuta in rule set. Una volta che sono state generate le regole, e che è stata fatta una selezione delle regole più promettenti per l'ottimizzazione semantica, prima di inserirle nel rule set occorre verificare che non contengano informazioni ridondanti con quelle contenute nell'insieme.

Utilizzando l'algoritmo CLOSURE, vediamo come è possibile mantenere il rule set un insieme minimale di regole. Osserviamo innanzitutto come una regola può essere

dedotta dalle regole attualmente presenti nel rule set. Per esempio se abbiamo:

rule set = { (R1.A = 'a') → (R1.C > 40), (R1.B = 'b') → (R1.A = 'a') }

e la regola Reg1:(R1.B='b') → (R1.C > 25), vediamo come, applicando la proprietà transitiva alle regole del rule set, sia possibile dedurre la regola Reg1. Un suo inserimento nel rule set, comporterebbe un aumento del costo del sistema, dovuto al mantenimento della coerenza della regola ed all'aumento del matching cost. D'altro canto il sistema non trae nessun beneficio aggiuntivo da tale regola, visto che l'informazione ad essa associata era già presente all'interno del rule set. Diamo ora la definizione di ridondanza di una regola.

Definizione : Una regola A → B è *ridondante* rispetto ad un insieme di regole S, nello stato x del database, se può essere dedotta dalla conoscenza esistente in S.

Si rende necessario definire un algoritmo per testare la ridondanza di una regola rispetto ad una data conoscenza S (nel caso che si stà esaminando S = rule set). Possiamo considerare il seguente algoritmo TESTING(A, JC, B, S) che accetta in input i seguenti valori: la regola Reg = ((JC → A) → B), avente come antecedente il/i predicati di join JC e il/i predicati di selezione A, e come conseguente il vincolo B; e l'insieme delle regole S rappresentante la conoscenza rispetto alla quale si vuole verificare la ridondanza della regola Reg. Se la regola Reg può essere dedotta da S l'algoritmo restituisce un valore TRUE, altrimenti restituisce un valore FALSE. L'algoritmo inoltre è capace di scoprire anche eventuali contraddizioni tra la regola Reg e l'informazione contenuta in S.

Algoritmo TESTING(A, JC, B, S)

```
begin
   closure_set = CLOSURE(A, JC, S) ;
   if (closure_set = FALSE) return (contraddizione) ;
   else  if (IMPLY(closure_set, B) return TRUE ;
         else return FALSE ;
end
```

Vediamo ora come è possibile costruire un algoritmo per generare un insieme minimale (Yu, 1989). Per ogni regola C in S, viene applicato l'algoritmo SIMPLIFY(C,

S). Esso ritorna un valore NULL se C è ridondante rispetto all'insieme S - {C} (in questo caso scarteremo la regola C), oppure ritorna una regola semplificata C' di C. Dove C' implica C attraverso S, e C' non ha predicati ridondanti nell'antecedente. L'insieme minimo può non essere unico.

Algoritmo SIMPLIFY(JC $\rightarrow p_1$ and p_2 and . . . . . and $p_n \rightarrow$ B, S)
begin
if TESTING({$p_1, p_2, \ldots, p_n$}, JC, S, B) then return NULL;
else if there exists p_i, TESTING({$p_1, p_2, \ldots p_{i-1}, p_{i+1}, \ldots, p_n$}, JC, S, p_i)
then SIMPLY(JC $\rightarrow p_{i+1} \ldots$ and $p_n \rightarrow$ B, S)
else return (JC $\rightarrow p_1$ and p_2 and . . . and $p_n \rightarrow$ B)
end

La prima condizione, se soddisfatta, scarta le regole ridondanti, cioè quelle che possono essere dedotte dalle regole presenti in S. La seconda condizione ricorsivamente eliminerà i predicati semplici ridondanti posti nell'antecedente delle regole, l'algoritmo restituirà così le regole semplificate.

Tutte le regole che provengono dal processo di generazione delle regole vengono controllate dall'algoritmo SIMPLIFY e, se non sono ridondanti, inserite nell'insieme rule set.

4) controllo del numero e della qualità delle regole

Il numero delle regola all'interno del rule set non può superare il valore massimo M. Questo per evitare che vi sia una degradazione delle prestazioni dell'SQO, a causa di un numero eccessivo di confronti dell'ottimizzatore semantico con le regole dell'insieme. Si ha anche un valore soglia S_{reg} dell'utilità che una regola deve avere per essere inclusa nel rule set. Questo valore può variare a secondo del numero di regole all'interno del rule set e dal valore dell'utilità delle regole. Se il numero di regole del rule set è inferiore a M, si cercherà di inserire nell'insieme il numero maggiore di regole, che però dovranno avere un utilità che sia almeno sufficiente a garantire all'SQO delle prestazioni minime, in termini di risparmio, accettabili. Quindi viene fissato un valore U_{min} dell'utilità che le regole devono avere per essere prese in

considerazione dal sistema durante la fase di generazione delle query equivalenti. Nel caso che stiamo prendendo in esame, per cui l'insieme rule set non è ancora pieno, il valore S_{reg} sarà uguale a U_{min}. Quando il numero delle regole del rule set è uguale a M, si assegnerà a S_{reg} il valore utilità più piccolo delle regole presententi nel rule set. Il valore S_{reg} viene considerato durante la fase di selezione di una regola proposta per il processo di derivazione e durante la fase di selezione delle regole prodotte dal processo di generazione.

Si è visto come i parametri, da cui dipende l'utilità di una regola, possano variare nel tempo. Il risparmio medio di una regola e la frequenza di utilizzo, cambiano ogni volta che la regola è utilizzata dall'SQO; la frequenza di utilizzo della regola viene decrementata se la regola non è utilizzata; il costo di manutenzione di una regola cambia ogni volta che la regola subisce un eccezione. Queste variazioni possono portare il valore dell'utilità di una regola al di sotto della soglia minima U_{min}. Il sistema di gestione dovrà quindi controllare il valore utilità di ogni regola; nel caso in cui tale valore è minore di U_{min}, la regola viene eliminata dall'insieme rule set.

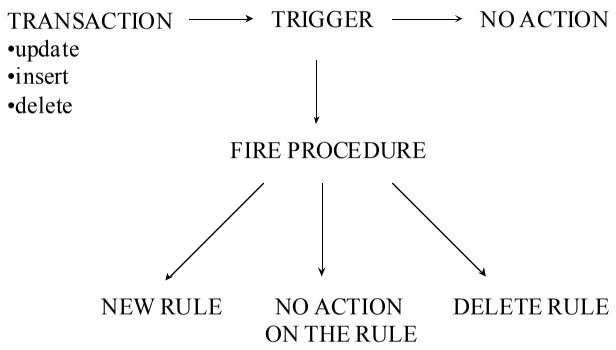


Figura 1

4.2 Controllo e ripristino della validità delle regole del rule set

Una volta che si sono create un insieme di regole semantiche dinamiche occorre accertarsi che tali regole rimangano valide per ogni stato futuro del database. Come già detto tale problema si divide in due parti: la fase che *controlla* che le regole sono verificate nello stato corrente del database, e l'eventuale *ripristino* della validità delle regole. In letteratura un problema similare è stato approfondito in particolare modo per regole semantiche statiche, rappresentanti vincoli definiti in fase di progetto del database (integrity constraint, referential integrity, ecc) e a cui devono sottostare tutte le istanze del database. I vincoli di integrità nei database system sono mantenuti validi o attraverso il rolling back di qualsiasi transizione che violi tali vincoli o attraverso la modifica delle operazioni che possono violare i vincoli. Nel caso di regole semantiche dinamiche si deve intervenire diversamente poiché se una transizione risulta essere valida, cioè l'esecuzione della transizione non porta il database in uno stato inconsistente, le operazioni che definiscono la transizione non possono essere non eseguite. Possiamo notare invece che le operazioni di controllo che individuano tutte le possibili operazioni che possono invalidare una regola (tali

operazioni definiscono la fase di *triggering*), possono essere utilizzate in generale sia per il controllo di regole statiche che quelle dinamiche.

Nella figura 1 sono mostrate le fasi principali per garantire la validità delle regole semantiche in seguito a cambiamenti dello stato del database dovuti ad operazioni di aggiornamento, cancellazione ed inserimento di tuple.

Nella sezione che segue si prenderà in considerazione come sia possibile scoprire le operazioni che possono invalidare una regola attraverso un analisi sintattica del vincolo [(54) Ceri (1990)]. Successivamente si vedrà un caso pratico su come possiamo controllare delle regole con il DBMS INGRES.

```
1.  Constraint  ::=  Table.List  Predicate
2.  Tab-List    ::=  T1 [V1], . . . T[Vn]
3.  Predicate   ::=  exists Select-Exp
4.              |    Item-Exp Connector Select-Exp
5.              |    Item-Exp1 Comp-Op Item-Exp2
6.              |    Predicate1 and Predicate2
7.              |    Predicate1 or Predicate2
8.              |    not Predicate1
9.              |    (Predicate1)
10.                  Select-Exp ::=
                     from From.List [where Predicate]
11.             |    se1ect-Exp1 Set-Op Select-Exp2
12.             |    (Select-Exp1)
13. Val-Exps    ::=  *
14.             |    Val-Exp1, ...,Val-Expn
15. Val-Exp     ::=  Col-Name
16.             |    constant
17.             |    Fn(Val-Exp1, ...,Val-Expn)
18. From-List   ::=  T1 [V1], ... . Tn [Vn]
19. Item-Exp    ::=  Va1-Exp
20.             |    <Val-Exp1, ...,Val-Expn>
21.             |    (select
                     Agg-Fn( [distinct] Col-Name)
                     from From-List
                     [where Predicate])
22. Col-Name    ::=  [T.] C
23. Connector   ::=  in | not in
                |    Comp-Op any | Comp-Op all
24. Set-Op      ::=  Union | intersect
25. Comp-Op     ::=  = | != | < | > | <= | >=
26. Agg-Fn      ::=  sum | min | max | avg | count
                |    user-defined aggregate
                     function
27. Fn          ::=  + | - | * | /
```

select [distinct] Val.Exp

Figura 2

4.2.1 Controllo validità delle regole attraverso un analisi sintattica delle regole

In [(54) Ceri, (1990)] viene descritto un metodo per derivare automaticamente l'insieme di operazioni che possono causare la violazione di un vincolo. Tale obiettivo è raggiunto attraverso un analisi statica del vincolo, il quale viene espresso attraverso un linguaggio precedentemente definito. Il linguaggio definito dall'autore, figura 2, è una variazione dell'usuale sintassi SQL per la definizione di predicati. Per un dato vincolo viene derivato un insieme di *operazioni invalidanti*. Questo insieme include ogni operazione di manipolazione dei dati la cui esecuzione può portare il database in uno stato che viola il vincolo ; queste diventano le operazioni di triggering per il controllo della validità delle regole. L'insieme delle operazioni invalidanti contengono gli elementi **insert into t**, **delete from t** e **update t.c** , dove t è il nome di una relazione e c è un attributo della relazione t. Per un dato vincolo, l'insieme è generato attraverso l'analisi della struttura sintattica del vincolo.

Ad esempio consideriamo l'operazione di update che possono invalidare un vincolo. Per trovare le operazioni di update invalidanti, si deve determinare gli attributi per i quali, se il valore dell'attributo è cambiato in una o più tuple, allora il vincolo non è più rispettato. Si considerano due tipi di attributi:

1) Attributi i cui valori individuali sono usati direttamente nella valutazione del vincolo. Prendendo in considerazione la grammatica di fig. 2, tali attributi appartengono all'elemento Item-Exp: come singolo valore (19, 15), nelle tuple complesse (20), o come oggetti di funzioni o funzioni aggregate (17, 21).
2) Attributi che sono parti di un insieme, questi attributi si trovano in Val-Exps all'interno dell'espressione select-Exp(10).

Ad esempio, se l'attributo t.c appare nel vincolo come parte di Item-Exp, allora l'aggiornamento di t.c è un operazione invalidante (caso 1).

In [(54) Ceri, (1990)] la ricerca delle operazioni di insert e delete risulta essere più complessa, quindi per un approfondimento del metodo utilizzato si rimanda all'articolo.

4.2.2 Considerazioni sul controllo e ripristino della validità di regole derivate

Per stabilire quali controlli occorre eseguire per verificare la validità di una regola, individuiamo quali sono le transazioni che possono invalidare una regola. Per il tipo di regole che si è considerato ci sono 4 tipi di transazioni che possono invalidarle. Una regola viene invalidata se:

1) aggiorniamo una tupla che non soddisfa l'antecedente della regola, in modo tale che la tupla risultante verifichi tutte le restrizioni dell'antecedente ma non la restrizione del conseguente
2) aggiorniamo una tupla che verifica la regola, in modo che il nuovo valore dell'attributo del conseguente non verifica più la restrizione del conseguente,
3) inseriamo una nuova tupla che soddisfa l'antecedente della regola ma non il suo conseguente

4) cancelliamo l'ultima tupla che verifica la regola.

Ad esempio, consideriamo la relazione R1(A, B) e la regola

$$Reg1 : (A>10) \rightarrow (B=5)$$

La tabella 1 mostra le tuple di R1. Nella tabella si è raggruppato all'inizio le tuple che verificano la regola Reg1.

N°Tupla	A	B
1	12	5
2	13	5
3	15	5
4	7	5
5	6	2

Tabella 1

Se aggiorniamo la tupla 5 della tabella, sostituendo il valore A=6 con A=11, allora viene eseguito un aggiornamento di tipo 1. Il risultato dell'aggiornamento sulla relazione R1 è quello di generare un eccezione alla regola Reg1, dato dalla nuova tupla (A=11,B=2).

Se aggiorniamo la tupla 1, sostituendo al valore B=5 con B=6, allora viene eseguito un aggiornamento di tipo 2. Il risultato dell'aggiornamento è quello di generare un eccezione alla regola Reg1, dato dalla tupla (A=12,B=6).

Se inseriamo nella relazione R1 la tupla (A=13,B=6) si genera un eccezione alla regola Reg1.

La cancellazione dell'ultima tupla che verifica la regola di per se non invalida la regola. Però si può fare la seguente osservazione. Se non esistono più tuple che

verificano una regola, l'ottimizzatore semantico non potrà più risolvere correttamente le tautologie.

Ad esempio consideriamo la query Q1 e la regola Reg1.

```
Select R1.B
from R1                                    (Q1)
where R1.A>10 ;
```

Applicando Reg1 a Q1 otteniamo la query Q2.

```
Select R1.B
from R1                                    (Q2)
where R1.A>10
and R1.B=5
```

Abbiamo una tautologia nella query Q2, per la quale la risposta alla query è già contenuta nel predicato R1.B=5. Ma se non si garantisce che esiste almeno una tupla di R1 con valori A>10 e B=5, anche se nel database non vi sono tuple che invalidano la regola R1, non è possibile dedurre dai vincoli della query la risposta. Questo perché non sappiamo se la risposta corretta è '5' o nulla.

Per scoprire una eccezione ad una regola occorre definire un trigger capace di riconoscere il verificarsi di uno dei quattro eventi descritti precedentemente. Inoltre si dovrà attivare una procedura capace di risolvere l'eccezione. La risoluzione dell'eccezione consiste nel ripristinare la validità della regola e poi valutare se è conveniente mantenere la nuova regola nel rule set o se deve essere eliminata.

Per mostrare la costruzione di un sistema di controllo delle regole, si considera il DBMS INGRES. In questo sistema è possibile creare delle *rule* capaci di eseguire una

specificata procedura del database ogni volta una data espressione logica è true. Utilizzando questi strumenti del DBMS INGRES proviamo a vedere come è possibile scoprire e ripristinare una regola violata.

4.2.2.1 Controllo validità delle regole costruite su una sola relazione base

Il controllo della regola Reg1 dell'esempio precedente, dove Reg1 è costruita su una sola relazione base del database, risulta essere agevolmente risolto definendo alcune rule e una procedura.

```
create rule update_reg1 after insert, update of R1
where new.A>10 and new.B≠5
execute procedure proc1_reg1 ( b=new.B) ;
```

La rule *update_reg1* manda in esecuzione la procedura *proc1_reg1* ogni volta che viene inserita o aggiornata una tupla che invalida la regola Reg1.

```
create procedure proc1_reg1 ( b integer) as
declare
  newReg1 varchar(60) ;
begin
 /*creo la nuova regola newReg1 cambiando il conseguente della regola Reg1*/
  newReg1= '(A>10)→(B in (5,'+ :b +'))' ;
  elimino Reg1 dal rule set;
  /* Calcolo la selettività del conseguente della regola fs ; se fs è inferiore
    alla soglia prefissata S, newReg2 è inserita nel rule set */
 if fs < S then inserisco newReg1 nel rule set ;
end ;
```

La procedura *proc1_reg1* crea la nuova regola newReg1 aggiungendo al conseguente di Reg1 il valore dell'attributo B della tupla che ha generato l'eccezione. NewReg1 è dunque una regola valida per il nuovo stato del database. La regola Reg1 viene eliminata dal rule set. Si calcola poi la selettività f_s del conseguente di newReg1. Se f_s è inferiore ad una soglia S prefissata, la nuova regola viene inserita nel rule set,

altrimenti è scartata. La verifica della selettività del conseguente della regola newReg1, serve ad impedire che nel rule set vengano inserite delle regole poco utili, poiché il loro conseguente è verificato da un numero elevato di tuple. Si suppone di considerare delle regole generate da euristiche di introduzione H1, per le quali risulta fondamentale che il loro conseguente abbia una selettività non elevata. Nel proseguo di questa sezione si manterrà tale ipotesi.

```
create rule delete_reg1 after delete from R1
where old.A>10 and old.B=5
execute procedure proc2_reg1 ;
```

La rule *delete_reg1* manda in esecuzione la procedura *proc2_reg1* ogni volta che viene eliminata da R1 una tupla che verifica la regola Reg1. La procedura *proc2_reg1* controlla che vi sia almeno una tupla di R1 che verifichi Reg1. Se nel database non vi sono più tuple che verificano la regola Reg1, allora la regola viene eliminata dal rule set.

```
create procedure proc2_reg1 as
declare
  I integer ;
begin
  select count(*) into :I from R1 where A>10 and B=5 ;
  if I=0 then elimina regola dal rule set ;
end
```

Nella sezione che segue viene studiato il controllo della validità delle regole costruite su differenti relazioni base. La sezione è divisa in tre paragrafi, in cui nei primi due si tratta il controllo delle regole in caso di update delle relazioni, nel terzo si vede il controllo delle regole in caso di insert e delete.

4.2.2.2 Controllo validità delle regole costruite su più di una relazione base

Consideriamo ora il caso più generico in cui abbiamo una regola che riguarda più relazioni base del database. Il sistema INGRES supporta rule costruite solo su singole relazioni base. Per scoprire il verificarsi di una eccezione ad una regola che riguarda più di una relazione, occorrerà generare delle rule per ognuna di tali relazioni.

Per il controllo della validità di una regola vengono utilizzate delle procedure in cui è eseguita una select, che esegue il join tra la tupla che è stata aggiornata, e le tuple delle altre relazioni che compaiono nella regola, che verificano le restrizioni dell'antecedente. La select è necessaria per la verifica della validità di una regola perché in essa compaiono attributi che appartengono a relazioni differenti, quindi il controllo deve essere fatto attraverso un join. Ogni volta che si manda in esecuzione una procedura si deve considerare un costo dovuto all'esecuzione della select. È conveniente quindi limitare l'esecuzione di una procedura, generando delle rule che lancino una procedura solo se si verifichino opportune condizioni. Essendo la rule costruita su una sola relazione, le condizioni verificate riguarderanno solo tale relazione. Tali condizioni sono necessarie, ma non sufficienti, affinché una regola possa essere stata invalidata da un operazione di aggiornamento, solo dopo l'esecuzione della select possiamo stabilire se abbiamo un eccezione alla regola.

Verranno presi in considerazione due gruppi di rule e procedure a seconda si considerino controlli su relazioni che presentano attributi che compaiono solo nell'antecedente di una regola, o relazioni che presentano attributi che compaiono sia nel conseguente che nell'antecedente di una regola.

Date le relazioni R1(A, B) e R2(B, C, D), consideriamo la regola Reg2.

$$\text{Reg2} : (R1.B=R2.B)\wedge(R1.A>5)\wedge(R2.D=7)\rightarrow(R2.C=8)$$

Si osservi come la relazione R2 presenta degli attributi che compaiono sia nell'antecedente che nel conseguente della regola Reg2. La relazione R1 ha attributi che compaiono solo nell'antecedente della regola. Entrambe le relazioni hanno sia

attributi di selezione (R1.A, R2.C, R2.D), sia attributi di join (R1.B, R2.B) nella regola Reg2.

4.2.2.3 Aggiornamento su relazione antecedente della regola

Consideriamo il caso che venga aggiornata una tupla di una relazione che presenta attributi solo nell'antecedente di una regola.

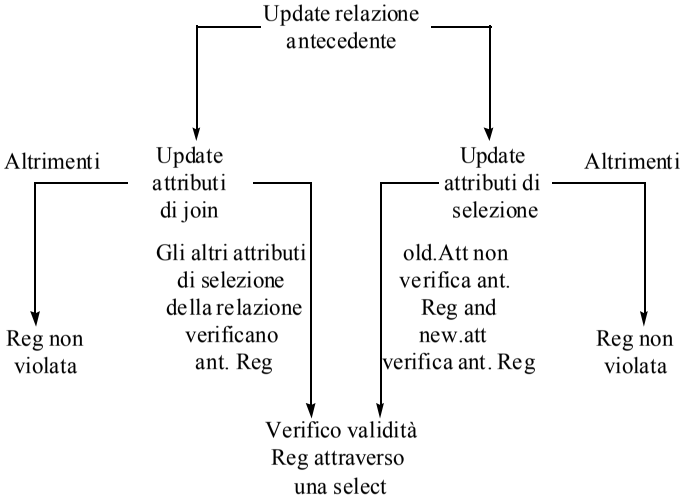


Figura 3

Nella figura 3 è mostrato come gli aggiornamenti possono riguardare un attributo di join e un attributo di selezione. Nel primo caso, se è stato aggiornato il valore di un attributo di join della regola, si testano gli altri valori degli attributi della relazione, che eventualmente possono essere ristretti nella regola, e se tali valori verificano le restrizioni della regola, allora è possibile che l'aggiornamento possa invalidarla.

Vediamo alcuni esempi considerando la tabella 2. Se si aggiorna la tupla n°3 cambiando il valore di B da 4 a 9, la regola Reg2 non sarà più valida. Infatti la tupla n°3 di R1 ,congiunta secondo l'equi-join (R1.B=R2.B), con la tupla n°5 di R2 verifica le restrizioni dell'antecedente di Reg2 ma non il suo conseguente.

N°tupla	R1.A	R1.B
1	6	3
2	8	2
3	12	4
4	4	5
5	10	12
6	2	14
7	8	7

R2.B	R2.C	R2.D	N°tupla
3	8	7	1
2	8	7	2
1	8	7	3
5	3	7	4
9	6	4	5
3	8	7	6
7	4	6	7

Tabella 2

Nel caso si aggiorni un attributo di selezione della regola, se si verifica che il vecchio valore dell'attributo ristretto (o degli attributi ristretti, se la relazione ha più attributi ristretti nell'antecedente) non verificava la restrizione della regola, mentre il nuovo valore verifica la restrizione, allora la regola può essere stata invalidata dall'aggiornamento. Esempio, se la tupla n°4 di R1 è aggiornata in (A=14, B=5) la Reg1 non è più valida a causa della tupla n°4 di R2 che presenta un valore di R2.C che non verifica il conseguente della regola.

Definiamo le rule e la procedura per garantire la validità della regola Reg2 rispetto ad aggiornamenti sulla relazione R1. Le considerazioni fatte sulla relazione R1 possono essere estese in generale ad ogni relazione che presenta attributi solo nell'antecedente di una regola.

create rule *update1_reg2* **after update of** R1
where (old.B≠new.B and new.A>5)
or (old.A≤5 and new.A>5)
execute procedure *proc1_reg2* (a=new.A, b=new.B) ;

Nella clausola where della rule *update1_reg2* sono presenti due espressioni logiche collegate da un *or*. La prima espressione controlla se è stato aggiornato l'attributo di join R1.B e se il valore dell'attributo di selezione R1.A verifica la restrizione di Reg2. La seconda espressione controlla se c'è stato un aggiornamento del valore dell'attributo di selezione R1.A per il quale abbiamo che il vecchio valore non verificava la restrizione di Reg2 mentre il nuovo la verifica. Solo in uno di questi due casi sarà necessario chiamare la procedura *proc1_reg2* per verificare se l'aggiornamento ha invalidato Reg2.

```
create procedure proc1_reg2 ( a integer, b integer) as
declare
   newReg2 varchar(60) ;
   c integer ;
begin
 select R2.C[1] into :c from R1, R2
where R2.D=7 and R1.A=:a
and R2.B= :b;
if ( :c≠8) then
   {
   elimino Reg2  dal rule set;
   /*creo la nuova regola newReg2 cambiando il conseguente della regola Reg2*/
   newReg2= '(R1.A>5)∧(R2.D=7)∧(R1.B=R2.B)→(R1.C in (5,' +  :c  + '))' ;
   /* Calcolo la selettività del conseguente della regola fs, se fs è inferiore
      alla soglia prefissata S, la regola newReg2 è inserita nel rule set */
   if fs<S  then  inserisco newReg2 nel rule set ;
   }
end ;
```

In *proc1_reg2* viene eseguita una select che ha come attributo target l'attributo presente nel conseguente delle regola. La select congiunge la tupla di R1 che è stata aggiornata con le tuple di R2 che verificano l'antecedente di Reg2. Se la risposta della select sono dei valori che non verificano il conseguente della regola Reg2, allora l'aggiornamento della tupla di R1 ha invalidato la regola Reg2. In questo caso si

elimina la regola non più valida Reg2 e si genera la nuova regola newReg2 creata aggiungendo al valore '5' del conseguente di Reg2, il valore presente nella risposta della select diverso dal valore '5'. Se la nuova regola presenta una selettività del conseguente che è al di sotto del valore soglia S, allora la regola viene inserita nel rule set.

[1] In INGRES, all'interno di una procedura possiamo considerare select che ritornano solo singole tuple di dati. Si ipotizza quindi che il risultato della select di una procedura riguardi una singola tupla, anche se in realtà la risposta può essere composta da più tuple.

4.2.2.4 Aggiornamento su relazione antecedente-conseguente della regola

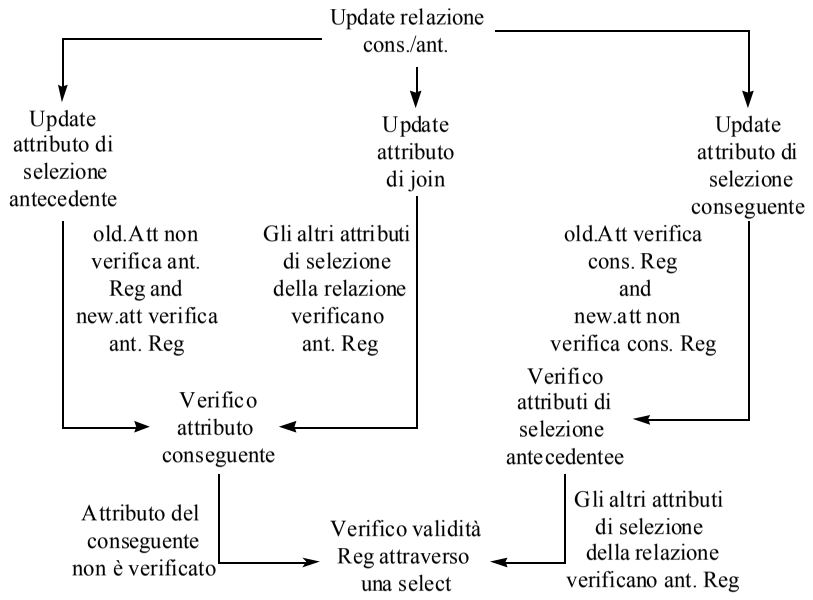


Figura 4

Nella figura 4 è mostrato le verifiche che occorre eseguire nel caso dell'aggiornamento della relazione a cui appartiene l'attributo ristretto del conseguente di una regola. In questa relazione compare anche un attributo di join, ed eventualmente altri attributi ristretti nell'antecedente della regola. Si considerano quindi tre tipi di aggiornamenti.

1) Se viene aggiornato il valore dell'attributo ristretto nell'antecedente di una regola, è possibile che una regola sia invalidata se sono verificate due condizioni. La prima condizione è che il vecchio valore dell'attributo non verificava la restrizione, mentre il nuovo valore lo verifica. La seconda è che, nella stessa tupla aggiornata, il valore dell'attributo conseguente non verifica la restrizione del conseguente. Ad esempio se viene aggiornata la tupla n°7 della relazione R2 con (B=7, C=4, D=7), Reg2 è invalidata dall'aggiornamento.
2) Se l'aggiornamento riguarda il valore di un attributo di join, si dovrà verificare che il valore dell'attributo ristretto (o degli attributi ristretti) dell'antecedente della regola verifichi la restrizione. Inoltre se il valore dell'attributo del conseguente verifica la restrizione del conseguente, allora è possibile che l'aggiornamento abbia creato un eccezione alla regola. Un esempio di un aggiornamento di questo tipo che invalida la regola Reg2, è cambiare la tupla n° 5 con (B=4, C=6, D=4).
3) L'ultimo caso che troviamo nella figura 4, riguarda l'aggiornamento del valore dell'attributo ristretto nel conseguente della regola. Se il vecchio valore verificava la restrizione, mentre il nuovo valore non verifica la restrizione del conseguente, e se gli eventuali valori presenti nella stessa tupla, degli attributi ristretti nell'antecedente della regola, verificano le restrizioni, allora occorre verificare la validità della regola. Ad esempio se aggiorniamo la tupla n° 6 di R2 con (B=3,C=4, D=7), la Reg2 viene invalidata.

Per la relazione R2 abbiamo le seguenti rule.

create rule *update2_reg2* **after update of** R2
where (((old.D≠7 and new.D=7)
or (old.B≠new.B and new.D=7)) and (new.C≠8))
or (old.C=8 and new.C≠8 and D=7)
execute procedure *proc2_reg2* (d=new.D, b=new.B) ;

La rule *update2_reg2* presenta nella clausola where le condizioni espresse dallo schema di figura 4. Ripresentiamo tale schema sostituendo alle condizioni generiche, le espressioni logiche della rule *update2_reg2*.

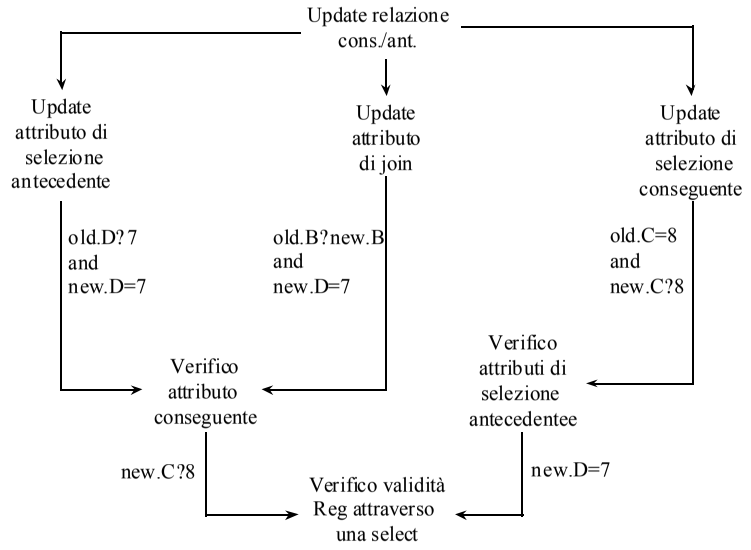


Figura 5

Le condizioni poste nella figura 5 esprimono tre percorsi logici, corrispondenti a verifiche rispettivamente su, attributi di selezione dell'antecedente, su attributi di join e sull'attributo di selezione del conseguente.

```
create procedure proc2_reg2 ( d integer, b integer) as
declare
   newReg2 varchar(60) ;
   c integer ;
begin
  select R2.C into :c from R1, R2
where  R2.D=:d and R1.A>5
and R1.B= :b ;
if ( :c≠8) then
   { elimino Reg2  dal rule set;
  /*creo la nuova regola newReg2 cambiando il conseguente della regola Reg1*/
  newReg2= '(R1.A>5)∧(R2.D=7)∧(R1.B=R2.B)→(R1.C in (5,' +  :c  + '))' ;
  /* Calcolo la selettività del conseguente della regola fs, se
     fs è inferiore alla soglia prefissata S, la regola è inserita nel rule set */
  if fs<S  then  inserisco newReg2 nel rule set ;
   }
end ;
```


Solo se tutte le condizioni poste su uno dei tre percorsi sono verificate, sarà possibile che vi sia stato un aggiornamento che abbia invalidato una regola. In tal caso la rule manderà in esecuzione la procedura *proc2_reg2*.Nella procedura *proc2_reg2* viene eseguita una select che congiunge la tupla di R2 modificata con le tuple della relazione R1 che verificano l'antecedente di Reg2. Se nella risposta della select abbiamo valori di R2.C che non verificano il conseguente della regola Reg2, allora viene cancellata la Reg2 dal rule set e si genera la nuova regola newReg2. NewReg2 è creata aggiungendo al valore '5' presente nel conseguente di Reg2, il valore presente nella risposta della select che è diverso da '5'.

4.2.2.5 Inserimento e cancellazione

Definiamo le rule che controllano la regola Reg2 in caso di inserimento di una nuova tupla nelle relazioni R1 e R2 , e in caso di cancellazione.

```
create rule insert1_reg2 after  insert  into R1
where new.A>5
execute procedure proc1_reg2 (a=new.A, b=new.B) ;

create rule insert2_reg2 after insert  into R2
where new.D=7 and new.C≠8
execute procedure proc2_reg2 (d=new.A, b=new.B) ;
```

La rule *insert1_reg2* esegue la procedura *proc1_reg2*, già vista per il caso di update, ogni volta che viene inserita una tupla in R1 che verifica la restrizione dell'antecedente della regola sull'attributo A. Solo le tuple di R1 che verificano la restrizione R1.A>5 possono invalidare Reg2. Ad esempio se inseriamo in R1 la tupla (A=10, B= 5), la congiunzione tra la tupla inserita di R1 e la tupla n°4 di R2 genera un eccezione alla regola Reg2. La procedura *proc1_reg2* sostituirà a Reg2 una nuova regola newReg2 che tenga conto dell'eccezione che si è generata.

La rule *insert2_reg2* esegue la procedura *proc2_reg2* ogni volta che viene inserita una tupla in R2 che verifica l'antecedente della regola ma non il conseguente. Ad esempio inseriamo in R2 della tupla (B=4, C=4, D=7), la congiunzione tra la tupla inserita in R2 e la tupla n°3 di R1 genera un eccezione alla regola Reg2.

Le rule che riguardano la cancellazione eseguono la procedura *proc3_reg2* quando viene eliminata una tupla che verifica le restrizioni della regola Reg2. La procedura *proc3_reg2* controlla che vi siano, dopo l'aggiornamento, ancora delle tuple che verificano la regola Reg2.

```
create rule delete1_reg2 after delete in R1
where new.A>5
execute procedure proc3_reg2 (d=old.A, b=old.B) ;

create rule delete2_reg2 after delete in R2
where old.D=7 and old.C=8
execute procedure proc3_reg2 (d=old.A, b=old.B) ;

create procedure proc3_reg2 as
declare
  I integer;
begin
  select count(*) into :I from R1,R2
  where R1.A>5 and R2.D=7
  and R1.B=R2.B and R2.C=8;
  if (I=0) then elimina Reg2 dal rule set;
end;
```


CAPITOLO 5

5.1 Conclusioni e lavori futuri

L'ottimizzazione semantica di interrogazioni può ridurre significativamente il tempo di risposta del database attraverso la riformulazione delle interrogazioni. Il successo nel ridurre il costo di esecuzione di un interrogazione è fortemente condizionato dai metodi di apprendimento e di manutenzione della conoscenza semantica per fornire al sistema un insieme utile di regole di trasformazione dell'interrogazione.

In questa tesi sono state discusse le problematiche di acquisizione e manutenzione di conoscenza nell'ambito dell'ottimizzazione semantica delle interrogazioni. Questi due aspetti sono fortemente correlate tra loro per quanto riguarda il processo di ottimizzazione.

Per quanto riguarda l'acquisizione di conoscenza, e' stato presentato un sistema che integra un approccio di tipo query-driven con un approccio di tipo data-driven in modo da compensare le lacune di una metodologia con i pregi dell'altra. Attraverso un approccio query-driven si è voluto acquisire le informazioni necessarie da utilizzare per la ricerca delle regole semantiche con un approccio data-driven. Infatti l'approccio data-driven permette di generare un insieme di regole più completo dell'altro ma non è in grado di limitare la ricerca alle regole utili alla ottimizzazione semantica delle interrogazioni. In altri termini, l'approccio data-driven è più potente, nel senso che permette di estrarre un elevato numero di regole ma è meno selettivo di quello query-driven.

Un primo obbiettivo che si ci è proposti è stato quello di integrare i due approcci. Il procedimento utilizzato è stato quello di generare delle regole proposte attraverso un approccio query-driven. L'utilizzazione di tali regole e la conseguente raccolta di informazioni attraverso la definizione delle rule class, ci consente di dedurre su quali relazioni è possibile effettuare una ricerca più efficace con un approccio data-driven.

Un secondo obbiettivo è stato quello di proporre un sistema per la generazione di regole con un approccio data-driven. In particolare e' stato proposto un algoritmo per la scoperta di regole attraverso la costruzione di una griglia.

Per manutenzione delle regole scoperte si intende il processo di controllo e di ripristino della validità delle regole. Nella tesi, dopo l'analisi di alcuni approcci presentati in letteratura, sono stati descritti alcuni metodi per la manutenzione delle regole come la riscrittura delle regole in caso di una loro violazione, o l'utilizzo di riferimenti alle eccezioni che si sono verificate per quella regola. Inoltre è stato considerato il problema del controllo della validità delle regole in seguito ad aggiornamenti del database. Infine e' stato proposto un tipo di manutenzione attraverso la generazione di procedure.

5.1.1 Ricapitolazione degli argomenti trattati

In questa sezione viene effettuato un riassunto dettagliato e una valutazione finale dei principali temi trattati in questa dissertazione.

5.1.1.1 Valutazione e scelta delle regole semantiche attraverso un processo query-driven

Nel capitolo 2 è stato descritto il processo di acquisizione delle regole attraverso un approccio query-driven. Tale approccio prevede l'utilizzo di euristiche [(1)King(1981)], da applicare alle interrogazioni generate dagli utenti, per estrarre la conoscenza semantica necessaria per l'SQO. In particolare è stata proposta una euristica (euristica 3) che può essere applicata a fattori booleani di un albero dei predicati di una query, formato da or di predicati semplici. L'approccio query-driven utilizzato si avvale del concetto di regola proposta [(56) Siegel (1992)] utilizzata nella fase di scelta dell'informazione semantica che conviene cercare. Al fine di creare un sistema capace di tenere traccia dei tentativi passati di derivare regole, e di poter

effettuare una selezione delle regole proposte che meglio si prestano per la fase di derivazione delle regole, è stato introdotto l'insieme rule class.

Il processo per l'acquisizione delle regole è stato diviso in tre rappresentazioni. La prima riguarda la generazione delle regole proposte attraverso l'algoritmo Closure [(45) Wei S.(1994)]. La seconda rappresentazione mostra come una regola proposta viene scelta per il processo di derivazione. La terza rappresentazione mostra il processo di derivazione delle regole.

5.1.1.2 Generazione regole con un approccio data-driven

Il capitolo 3 presenta un metodo data-driven per la generazione delle regole. Esso si basa sul concetto di griglia vista come una regolare decomposizione dello spazio di tutte le possibili tuple di una data relazione in una collezione di celle disgiunte. Dopo aver evidenziato le caratteristiche principali di un sistema di acquisizione grid-based, è stato proposto un algoritmo per la scoperta di regole attraverso la costruzione di una griglia multidimensionale. Al fine di potere selezionare le regole più promettenti per l'ottimizzazione, l'algoritmo è in grado di effettuare una stima della selettività dell'antecedente e del conseguente di una regola.

Si è inoltre visto come, per effettuare una ricerca più mirata delle regole, sono state utilizzate le informazioni contenute nell'insieme RULE CLASS. Ciò ha permesso di stabilire su quali relazioni effettuare la ricerca delle regole, e quali attributi dovevano comparire nell'antecedente e nel conseguente delle regole. Tali informazioni sono state raccolte durante il processo di derivazione delle regole attraverso l'approccio query-driven.

Un osservazione che emerge al termine di questo capitolo è che si sente la necessità di ulteriori indagini per utilizzare altre risorse di informazioni, al fine di indirizzare la ricerca alla scoperta delle sole regole utili per l'SQO. Infatti, anche se l'utilizzo di un approccio query-driven, e in particolare dell'applicazione di euristiche, ci consentono di individuare un insieme di attributi su cui effettuare la ricerca data-driven, occorre caratterizzare maggiormente la ricerca delle regole. Ad esempio utilizzando

informazioni quali la conoscenza del dominio fornita dall'utente, tipo di distribuzione delle query, tecniche di data-mining.

5.1.1.3 Manutenzione delle regole scoperte

Una volta che una regola è stata scoperta dal database occorre effettuare un monitoraggio continuo della sua utilità e della sua validità. Nel capitolo 4 si è parlato della manutenzione delle regole scoperte, inteso come il processo per il controllo e il ripristino della validità delle regole. Nel capitolo sono stati descritti alcuni metodi per la manutenzione delle regole come la riscrittura delle regole in caso di una loro violazione, o l'utilizzo di riferimenti alle varie eccezioni che si sono verificate per quella regola. Inoltre è stato considerato il problema del controllo della validità delle regole in seguito ad aggiornamenti del database. Si è visto come una possibile strada sia quella di individuare le transazioni che possono rendere una regola non più valida. Quindi solo in seguito all'esecuzione di queste particolari transazioni si dovrà accertare che esse non abbiano causato l'invalidamento di una regola.

5.1.2 Conclusioni e lavori futuri

L'ottimizzazione semantica di interrogazioni può ridurre significativamente la velocità di risposta del database attraverso la riformulazione della conoscenza intensiva. Il successo dell'SQO nel ridurre il costo di esecuzione di un interrogazione, è fortemente condizionato dai metodi di apprendimento e di manutenzione della conoscenza semantica per fornire al sistema un insieme utile di regole.

In questa dissertazione si sono affrontate le principali problematiche inerenti all'acquisizione di regole semanatiche per l'SQO. E' stato proposto un algoritmo per la scoperta di regole attraverso la costruzione di una griglia. È stata formulata una euristica che può essere applicata anche a fattori booleani formati da or di predicati semplici. Per finire si è visto una possibile strada per la manutenzione delle regole dinamiche attraverso la definizione di rule e procedure.

Lavori futuri riguardano la possibilità di poter genere delle regole semantiche capaci di esprimere una più completa conoscenza intensionale per migliorare ulteriormente il risparmio ottenuto dall'SQO. È possibile estendere alcune considerazioni che si sono fatte anche ai database ad oggetti. Per quanto riguarda il processo di acquisizione di tipo data-driven possiamo considerare l'utilizzo di tecniche di data-mining per generare delle regole utili all'ottimizzazione semantica

BIBLIOGRAFIA

1. J. J. King, QUIST : A system for semantic query optimization in relational databases, *Proc. 7th VLDB Conf,* (1981).

2. Hammer and Zdonik, Knowledge Based query processing, *Proc. 6th Conf on VLDB,* (1980).

3. S .T. Sbenoy and Z. M. Ozsoyoglu, Design and Implementation of a Semantic Query Optimizer, *IEEE Transactions on Knowledge and Data Engineering, pp. 362-375* (1989).

4. H. H. Pang, H. J. Lu, and B. C. Ooi, *An* Efficient Semantic Query Optimization Algorithm, *Proc. Data Engineering Conference,* IEEE, (1991).

5. G. M. Lohman, Panei Discussion on Semantic Query Optimization, *Proc. Data Engineering Conf,* (1985).

6. Shekhar, J. Srivastava, and S. Dutta, A Formai Modei of Trade-offs between Execution and Optimization Costs in Semantic Query Optimization, *IntL Conf on Very Large Databases (VLDB) (reprinted in N.H. J.D.K.E. 1992),* (1988).

7. C. T. Yu and W. Sun, Automatjc Knowledge Acquisition and Maintenance for Semantic Query Optimization, *IEEE Transactions on Knowledge* and *Data Engineenng,* pp. *362-375,* (1989).

8. S. T. Shenoy and Z.M.Ozsoyoglu, A System for Semantic Query Optimization, *Proc. ACM-SJGMOD, PP.*181-195, (1987).

9. A. V. Aho, J. E. Hopcroft, and J. D. Uliman, *The Design and Analysis of Computer Algorithms,* Beh Telephone Laboratories, (1974).

10. M. Siegel, *Automatic Rule Derivationfor Semantic Queiy Optimization. Ph.D. diss.,* Boston Univ. (1988).

11. M. Siegel, Automatic Rule Derivation for Semantic Query Optimization, *Proc. of the Second Internationai Conference on Expert Database Systems, pp.* 371-385 George Mason Foundation, (1988).

12. M. Siegel, E. Sciore, and S. Salveter, Rule Discovery for Query Optimization, *Knowledge Discovery in Databases,* The AAAI Press, (1991).

13. C. L. Chang, DEDUCE 2: Further Investigations of Deduction in Relational Data bases, pp.201-236 in *Logic and Data Bases,* ed. J. Minker, Plenum Press, New York (1978).

14. J. M. Nicolas, Logic for Improving Integrity Checking in Relational Databases, *Acta Informatica* 18 pp.227- *253* Springer Verlag, (1982).

15. R. S. Michalski, A Theory and Methodology of Inductive Learning, in *Machine Learning: An Artiticial Intelligence Approach,* ed. T. M. Mitchell, Morgan Kaufinann Publishers, Inc., Los Altos, California (1986).

16. R. S. Michalski and R. E. Stepp, Learning from Observation: Conceptual Clustering, pp.331-363 in *Machine Learning: An Artificial Intelligence Approach,* ed. T. M. Mitchell, Tioga, Palo Alto, California (1983).

17. J. R. Quinlan, Probabilistic Decision Trees, pp.140-152 in *Machine Learning: An*

Artificial Intelligence Approach, ed. Yves Kodratoff, Morgan Kaufmann Publishers, Inc., San Mateo, California (1990).

18. S.C. Shapiro, *Encyclopedia ofArtificial Intelligence,* A Wiley-Interscience Publication (1990).

19. R. R. Sokal and R. H. Sneath, *Principles ofNumerical Taxonomy,* W. H. Freeman, San Francisco (1963).

20. R. M. Cormark, A review of classification, pp. l34-32l in J. *Roy. Stat. Soc., Series A,* (1971).

21. M. R. Anderberg, *Clustering analysis,* Academic Press, New York (1973).

22. J. C. Gower, A comparison of some methods of cluster analysis, *Biometrics 23,* pp.623-637 (1967).

23. E. Diday and J. C. Simon, Clustering analysis, *Communication* and *Cybernetics,* Springer-Verlag, New York, (1976).

24. R. S. Michalski, Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts, J. *PoL Anal. Inform. Sys* 4, pp.219-244 (1980).

25. P. Langley and S. Sage, Conceptual Clustering as Discrimination Learning, *Proceedings of the Fifth Biennial Conference of the Canadian Society for Computational Studies of Intelligence,* pp.95-98 (1984).

26. D. Fisher, *A Hierarchical Conceptual Clustering Algorithm, Technical Report,*

Department of Information and Computer Science, University of California, Irvine (1984).

27. S. J. Hanson, Conceptual Clustering and Categorization: Bridging the Gap between Induction and Causal Models, pp.235-268 in *Machine Learning: An Artificial Intelligence Approach, ed.* Yves Kodratoff, Morgan Kaufmann Publishers, Inc., San Mateo, California (1990).

28. S. P. Ghosh, Statictics Metadata: Linear Regression Analysis, pp.3-17 in *Fouiujc~tions of Data Organization,* ed. Katsumi Tanaka, Plenum Press, New York (1987).

29. P. Langley, J. Zytkow, H. Simon, and 0. Bradshaw, Rediscovering Chemistry with the BACON System, pp. 307-330 in *Machine Learning: An Artificial Intelligence Approach,* ed. T. M. Mitchell, Tioga, Palo Alto, California (1983).

30. J. Zytkow and J. Baker, Interactive Mining of Regularities in Databases, *Knowledge Discovery in Databases,* The AAAI Press, (1991).

31 D. B. Lenat, The Role of Heuristics in Learning by Discovery: Three Case Studies, in *Machine Learning: An Artificial Intelligence Approach,* ed. T. M. Mitchell, Tioga, Palo Alto, California (1983).

32. P. Langley, J. Zytkow, H. Simon, and G. Bradshaw, The Search for Regularity: Four Aspects of Scientific Discovery, pp.425-469 in *Machine Learning: An Artificial Intelligence Approach,* ed. T. M. Mitchell, Morgan Kaufmann Publishers, Inc., Los Altos, California (1986).

33. R. Elmasri and G. Wiederhold, Data Model Integration Using the Structural

Model, *Proc. International Conference on Management of Data,* ACM SIGMOD, (1979).

34. M. Hammer and D. McLeod, Semantic Integrity in a Relational Data Base System, *VLDB, (1975).*

35. K. Eswaran and D. D. Chamberlin, Functional Specifications of a Subsystem for Database Integrity, *VLDB, (1975).*

36. E. Codd, Extending the Database Relational Model to Capture More Meaning, *TODS* 4:4(December 1979).

37. P. Chen, The Entity Relationship Mode-Toward a Unified View of Data, *TODS* 1:1(March 1976).

38. J. Schmidt and J. Swenson, On the Semantics of the Relational Model, *SIGMOD,* (1975).

39. K. Y. Whang and R. Krishnamurthy, The Multilevel Grid File - A Dynamic Hierarchical Multidimentional File Structure, *International Symposium on Database Systems for Advanced Applications,* (Tokyo, Japan, April, 1991).

40. M. V. Mannino, P. Chu, and T. Sager, Statistical Profile Estimation in Database Systems, *Computing Surveys* 20, No. 3ACM, (September, 1988).

41. K.Whang, S. Kim, and G. Wiederhold, *Dynamic Maintenance of Data Distributionfor Selectivity Estimation,* Dept. of Computer Science, Stanford University (September, 1991).

42. L. F. Mackert and G.M. Lohman, R* Optimizer Validation and Performance Evaluation for Local Queries, *Proc. ACM-SJGMOD, pp.84-95* ACM, (1986).

43. W. Ziarko. The special issue on rough sets and Knowledge discovery. Computational Intelligence,11(2), (1995).

44. S. Minton. Learning effective Search Control Knowledge: An Explanation-Based Approch. PHD thesis, Carnegie Mellon University, Schoolof Computer Science, (1988).

45. W. Sun and Clement T. Yu IEEE "Transactions on knowledge and data engineering", Vol. 6 NO. I, (February 1994).

46. Chun-Nan Hsu and Craig A. Knoblock. Reformulating query plans for multidatabase systems. In Proceedings of the Second International Conference on Information and Knowledge Management (CIA7M-93), Washington, D.C., (1993).

47 Chun-Nan Hsu and Craig A. Knoblock. Rule induction for semantic query optimization. In Machine Learning, Proceedings of the 11th International Conference (ML-94), San Mateo, CA, 1994. Morgan Kaufmann (1994).

48. Chun-Nan Hsu and Craig A. Knoblock. Estimating the robustness of discovered knowledge. In Proceedings of the First International Conference on Knowledge Discovery and Data Mining(KDD-95), Menlo Park, CA, 1995. AAAI Press (1995).

49. Chun-Nan Hsu and Craig A. Knoblock. Discovering robust knowledge from

dynamic closed-world data. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), Portland, Oregon, 1996. AAA1 Press (1996).

50. Chun-Nan Hsu and Craig A. Knoblock. Using inductive learning to generate rules for semantic query optimization. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, chapter 17. AAAI Press/MIT Press, (1996).

51. Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering Algorithms for information agents. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAA 1-96), Portland, OR, (1996).

52. Yigal Arens, Craig A. Knoblok, and Wei-Min Shen. Query refomulation for dynamic information integration. *Journal of the Intelligent Informational Systems, Special issue on Intelligent Information Integration*, (1996).

52. J. Shavlik and T. A. Dietterich. *Reading in machine Learning.* Morgan Kaufmann, San Matteo, CA, (1990).

54. Ceri S. and Widom J. *Deriving production rules for constraint maintenance. In Proceeding of the 16th Internetional Conference on VLDB Endowment (1990).*

55. Ceri S. *Automatic generation of production rules for integrity Maintenance. In ACM Transaction on Database System, Vol 19, No 3, (1994).*

56 Siegel M. and Sciore E." *A method for automatic rule derivation to support*

semantic query optimizazion" on Transaction on Database Systems. ACM (1992).