ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

GESTIONE DELLE PREFERENZE NEI SISTEMI INFORMATIVI

Elaborata nel corso di: Basi di Dati

Tesi di Laurea di:
BRUNO ROSATI

Relatore:
Prof. STEFANO RIZZI

ANNO ACCADEMICO 2015–2016
SESSIONE III

PAROLE CHIAVE

Preferenze

Database

Winnow

Skyline

Contesto

Dedicato a chi ha il coraggio di restare

Indice

Introduzione

Con il continuo miglioramento delle reti wireless, della larghezza di banda e delle funzionalità dei dispositivi mobile, il numero di servizi disponibili offerti agli utenti mobili necessita di essere continuamente potenziato, fornendo diversi tipi di contenuti, dalla musica alle notizie di servizi basati sulla localizzazione e navigazione. L'enorme aumento della portata di tali servizi è arrivato con l'avvento delle applicazioni web incentrate sull'utilizzo dei database. I più grandi siti di e-commerce, ora come ora, trasportano svariate decine di milioni di offerte per prodotti provenienti da più centinaia di migliaia di punti vendita. La precedente tecnologia per l'elaborazione convenzionale di interrogazioni che restituisce semplicemente l'insieme di tuple che soddisfano il predicato di una query risulta del tutto inadeguata per questo genere di applicazioni poiché richiedono che le tuple siano classificate senza sacrificare le prestazioni delle query. Quindi, gli operatori mobili devono necessariamente perseguire questo obiettivo cercando di offrire agli utenti una quantità sempre maggiore di servizi. Inoltre, al giorno d'oggi, i dispositivi portatili hanno risorse limitate, come potenza di calcolo, la durata della batteria e la quantità di memoria in dotazione, dunque richiedono applicazioni in grado di gestire i dati i più interessanti dall'utente in quel preciso momento. Le informazioni devono quindi essere adeguatamente personalizzate, in modo che l'utente non venga preso alla sprovvista dall'enorme quantità di dati disponibili. D'altra parte, gli utenti vogliono mantenere il controllo delle offerte dei servizi e selezionare singolarmente di cosa usufruire in base alle loro esigenze individuali.
In questo ambito, l'accesso personalizzato ai contenuti è particolarmente importante poiché trovare il servizio adeguato per un utente specifico in una determinata situazione è un problema impegnativo. Effettuare una scelta corretta dei servizi da rendere disponibili per l'utente risulta essere possibile essenzialmente trovando le corrispondenze fra le preferenze dei singoli utenti

e le loro antipatie nei confronti dei servizi offerti in una situazione specifica. La nozione di preferenza è sempre più diffusa nei sistemi informativi attuali. Le preferenze sono utilizzate principalmente per filtrare e personalizzare le informazioni che devono raggiungere gli utenti di tali sistemi.
In questa tesi viene data una definizione formale di preferenza e sono descritte alcune tecniche per l'ottimizzazione semantica delle query di preferenza, con particolare attenzione all'operatore Winnow e all'operatore Skyline. Si cerca di fornire un quadro che attesti la potenza dell'elaborazione logica delle relazioni di preferenza.

Capitolo 1

Fondamenti di preferenze nei sistemi informativi

1.1 Il concetto di preferenza

Nei sistemi informativi attuali, sta sempre più prendendo piede l'esigenza di gestire le preferenze degli utenti in maniera ottimale. La personalizzazione dei sistemi elettronici pone nuove sfide alla tecnologia dei database, cercando una tecnica di modellazione che sia allo stesso tempo potente e flessibile per le preferenze complesse, utilizzate per il filtraggio delle informazioni affinché siano di una qualità più elevata e per ridurre notevolmente il volume dei dati presentati all'utente. Nei sistemi di database, le preferenze di solito sono estrapolate come relazioni privilegiate che vengono utilizzate per creare le query di preferenza. Da un punto di vista formale, le relazioni privilegiate sono relazioni binarie definite sulle risposte di query e possono essere definite usando formule logiche o costruttori di preferenza (è possibile esprimere questi ultimi anche attraverso formule logiche). Questo modello è il punto di partenza sfruttato per studiare le preferenze secondo due differenti approcci, uno di tipo ingegneristico ed un altro di tipo algebrico.
Le preferenze si trovano in ogni aspetto della vita quotidiana di un individuo, sia per questioni lavorative che non. La personalizzazione nei database intende affrontare l'enorme espansione della quantità di dati attualmente a disposizione di uno spettro sempre più ampio di utenti, presentando agli utenti solo gli elementi che sono di loro interesse. Diventa dunque necessario che lo sviluppo della tecnologia riesca a soddisfare in maniera adeguata i

molti aspetti sofisticati delle preferenze.
La personalizzazione ha diverse sfaccettature:

- nel primo scenario, detto "mondo ideale", le richieste avanzate dagli utenti possono essere soddisfatte completamente oppure per niente. In questo scenario l'utente può effettuare la sua scelta all'interno di un set limitato e predefinito. Le query di database in questo contesto sono caratterizzate da vincoli rigidi, fornendo esattamente gli oggetti richiesti se ci sono altrimenti la query dell'utente viene respinta.

- il secondo scenario, detto "mondo reale", è uno scenario piuttosto diverso, dove le preferenze personali sono richieste libere e non vi è alcuna garanzia che vengano essere soddisfatte. In caso di fallimento nella ricerca di un abbinamento perfetto, gli utenti possono concordare compromessi o accettare soluzioni non di loro gradimento in prima istanza.

Al momento non si è ancora arrivati alla teorizzazione di una soluzione completa che possa integrare in modo regolare ed efficiente le preferenze con la tecnologia di database. Si ipotizza che un modello di preferenza praticabile per sistemi di database debba soddisfare il seguente elenco di specifiche:

- Semantica intuitiva: nel processo di modellazione, le preferenze devono avere il valore di elemento fondamentale e non trascurabile. Ciò richiede una comprensione intuitiva e la specifica dichiarativa di preferenze.

- Fondamento matematico: questo requisito deve chiaramente armonizzarsi con la semantica intuitiva.

- Modello di preferenza costruttivo ed estensibile: le preferenze complesse dovrebbero essere costruite induttivamente da quelle di base che utilizzano un repertorio estendibile di costruttori di preferenza.

- Gestione dei conflitti fra preferenze: i conflitti di preferenze non devono causare un errore di sistema, deve essere sostenuta la composizione dinamica di preferenze complesse anche in presenza di eventuali conflitti. Un pratico modello di preferenza dovrebbe essere in grado di

convivere con i conflitti, non fallire se si verificano, né tanto meno vietarli.

- Linguaggi d'interrogazione dichiarativi di preferenza: è necessario assottigliare sempre di più il divario esistente tra richieste e realtà. Bisogna dunque implementare un nuovo modello di ricerca diverso dal modello di corrispondenza esatta dei linguaggi d'interrogazione dichiarativi dei database.

Il modello a preferenze è la chiave verso lo sviluppo di nuove discipline come l'ingegneria delle preferenze e l'algebra delle preferenze.

1.2 Proprietà delle relazioni di preferenza

Le preferenze sono definite utilizzando relazioni di preferenza binarie tra tuple. Le relazioni di preferenza vengono specificate attraverso l'utilizzo di formule di preferenza intrinseche in modo tale che vengano presi in considerazione per le preferenze solo valori che si verificano all'interno delle tuple, non altri come l'appartenenza delle tuple nei rapporti di database.
Una preferenza è formulata su una serie di attributi con un dominio associato di valori. Quando si combinano le preferenze P_1 e P_2, viene stabilito che P_1 e P_2 possono sovrapporsi sui loro attributi, permettendo a più preferenze di coesistere sugli stessi attributi; questo perché, come detto precedentemente, i conflitti fra preferenze devono essere autorizzati.
Si definisce insieme ordinato (oppure ordine) la coppia costituita da un insieme e da una relazione d'ordine su di esso. Un ordine è una relazione binaria "$\leq$" su un insieme R che è allo stesso tempo riflessiva, asimmetrica e transitiva, ovvero che rispetta le seguenti regole:

- $a \leq a$ (riflessività)
- se $a \leq b$ e $b \leq a$, allora $a = b$ (asimmetria)
- se $a \leq b$ e $b \leq c$, allora $a \leq c$ (transitività)

per ogni a, b, c presente in R.
In questo ambito, l'ordine sopra definito è chiamato un non-stretto. In questi contesti, un ordine stretto (o non riflessivo) è una relazione binaria "$<$" che è non riflessiva, transitiva e asimmetrica, ovvero che rispetta le seguenti regole:

- non è ammesso a $<$ a (non riflessività)
- se a $<$ b e b $<$ c, allora a $<$ c (transitività)
- se a $<$ b allora non è ammesso b $<$ a (asimmetria, sottintendendo non riflessività e transitività)

per ogni a, b, c presente in R. Dati due qualsiasi elementi appartenenti all'insieme R, se questi non possono essere sempre confrontati tra loro allora la relazione d'ordine è definita parziale. L'ordinamento parziale stretto permette di mappare direttamente tutte quelle preferenze che ogni individuo nella vita quotidiana può esprimere come "mi piace a più di quanto mi piaccia b". Questo tipo di modellazione delle preferenze è universalmente applicato e intuitivamente comprensibile da tutti.
Dato un insieme A di attributi, una preferenza P è un ordine parziale stretto P = (A, $<$P), dove $<$P $\subseteq$ dom (A) $\times$ dom (A).
Così $<$P è non riflessiva e transitiva (il che implica asimmetria). È importante questa interpretazione prevista:
"x $<$P y" viene interpretato come "y mi piace più di x".
Poiché le preferenze riflettono importanti aspetti del mondo reale, una buona rappresentazione visiva è essenziale.
Dato uno schema di relazione R (A_1 ... A_k) tale che U_i (con $1 \leq i \leq k$) sia il dominio dell'attributo A_i, una relazione $\succ$ è una relazione di preferenza su R se è un sottoinsieme di ($U_1 \times \ldots \times U_k$) $\times$ ($U_1 \times \ldots \times U_k$).
$\succ$ indica una relazione binaria tra tuple appartenenti allo stesso database. Un tupla t_1 domina una tupla t_2 se $t_1 \succ t_2$. Le proprietà tipiche di questa relazione, oltre alle sopracitate non riflessività, transitività e asimmetria, comprendono
transitività negativa: per ogni x, y, z se $x \not\succ y \wedge y \not\succ z$ allora $x \not\succ z$;
connettività: per ogni x, y, $x \succ y \vee y \succ x \vee x = y$.
È doveroso sottolineare che queste proprietà non sono tra loro indipendenti. L'asimmetria implica non riflessività, mentre non riflessività e asimmetria implicano l'asimmetria.
Nella maggior parte delle applicazioni di query di preferenza, le relazioni di preferenza sono ordini parziali stretti. Esistono tuttavia delle situazioni in cui tali relazioni potrebbero non soddisfare qualche proprietà degli ordini parziali. Potrebbero, ad esempio, esserci due tuple t_1 e t_2 tali che $t_1 \succ t_2$ e $t_2 \succ t_1$ semplicemente perché possono esistere validi motivi per cui in

alcuni casi t_1 è preferibile a t_2 mentre in altri si propende per il viceversa; in questo ambito, cade la non riflessività. Allo stesso modo, la transitività non è sempre garantita; ad esempio, t_1 può essere preferita a t_2 e t_2 a sua volta a t_3, ma il divario tra t_1 e t_3 rispetto a qualche proprietà finora ignorata può essere così grande da impedire che t_1 sia preferibile a t_3. Oppure, la transitività potrebbe dover essere abbandonata per evitare cicli nelle preferenze. La composizione di relazioni di preferenza può anche portare a violazioni di asimmetria o transitività.
Ogni relazione di preferenza $\succ_C$ genera una relazione di indifferenza $\sim_C$: due tuple t_1 e t_2 sono indifferenti ($t_1 \sim_C t_2$) se nessuno dei due è preferito all'altro, cioè, $t_1 \not\succ_C t_2$ e $t_2 \not\succ_C t_1$. Se la relazione di preferenza $\succ_C$ è non riflessiva, allora per ogni tupla t, $t \sim_C t$.

1.3 Composizione di relazioni di preferenza

Esistono svariate modalità applicabili per comporre relazioni di preferenza. Generalmente si fa distinzione fra la composizione unidimensionale e quella multidimensionale. Nella composizione unidimensionale, le relazioni di preferenza vengono composte su uno schema di database singolo, ottenendo così un'altra relazione di preferenza all'interno dello schema medesimo. Nella composizione multidimensionale, le relazioni di preferenza vengono definite nell'arco di diversi schemi di database, e il prodotto cartesiano di queste relazioni viene definito come relazione di preferenza.
In questo ambito, le relazioni di preferenza sono definite attraverso formule di preferenza del primo ordine, quindi ogni composizione di relazione di preferenza definibile del primo ordine porta nuovamente ad una formula di preferenza del primo ordine (Teoria del primo ordine).
Vengono ora elencati le principali tipologie di composizione.

Composizione booleana

Per le relazioni di preferenza, l'unione, l'intersezione e la differenza ottemperano chiaramente alle rispettive operazioni dell'algebra booleana.
L'ordine parziale stretto è conservato con l'operazione di intersezione ma non per le operazioni di unione e differenza.
L'ordine debole non viene conservato da alcuna operazione booleana; nell'ambito delle preferenze, infatti, non vengono utilizzati ordini deboli perché

violano la proprietà di transitività negativa.
L'ordine totale non viene conservato da alcuna operazione booleana; prendendo in considerazione l'ordine totale e il suo inverso, si ha che la loro intersezione è vuota e quindi non un ordine totale; in maniera analoga, la differenza tra un ordine totale e se stesso genera nuovamente un insieme vuoto.

Composizione con priorità

Saltuariamente ci si può imbattere in preferenze con priorità, questo accade solitamente in un ambiente multi-agente in cui le preferenze di un agente sostituiscono quelle di un altro agente.
Considerando due relazioni di preferenza $\succ_{C_1}$ e $\succ_{C_2}$ definite sullo stesso schema R, la composizione con priorità $\succ_{C_{1,2}} = \succ_{C_1} \triangleright \succ_{C_2}$ di $\succ_{C_1}$ e $\succ_{C_2}$ è una relazione di preferenza su R definita come

$$t_1 \succ_{C_{1,2}} t_2 \equiv t_1 \succ_{C_1} t_2 \vee (t_1 \sim_{C_1} t_2 \wedge t_1 \succ_{C_2} t_2).$$

Tutto ciò viene letto come "le preferenze vengono stabilite in base a $\succ_{C_2}$ a meno che sia applicabile $\succ_{C_1}$" ($\succ_{C_1} \triangleright \succ_{C_2}$).
Gli ordini deboli e gli ordini totali sono conservati dalla composizione con priorità, cosa che invece non accade per gli ordini parziali stretti.
La costruzione riportata qui sotto può essere generalizzata ad un arbitrario ordine parziale con priorità finito tra relazioni di preferenza.
La composizione con priorità è associativa:
$(\succ_{C_1} \triangleright \succ_{C_2}) \triangleright \succ_{C_3} \equiv \succ_{C_1} \triangleright (\succ_{C_2} \triangleright \succ_{C_3})$
e distributiva per quanto riguarda l'unione:
$\succ_{C_1} \triangleright (\succ_{C_2} \cup \succ_{C_3}) \equiv (\succ_{C_1} \triangleright \succ_{C_2}) \cup (\succ_{C_1} \triangleright \succ_{C_3})$.
Tutto ciò avviene grazie alla proprietà associativa e la proprietà distributiva di $\triangleright$. Questo ordine può essere visto come un grafo in cui i nodi sono costituiti da relazioni di preferenza mentre i rami rappresentano priorità relative (ci sarebbe dunque un ramo $(\succ_1, \succ_{C_2})$ nella situazione appena descritta). Per codificare questo grafo come una singola relazione di preferenza, si potrebbero costruire le definizioni corrispondenti ai percorsi individuali dalle radici alle foglie e, successivamente, prendere una disgiunzione di tutte queste definizioni.

Chiusura transitiva

Da un punto di vista esclusivamente matematico, la chiusura transitiva di una relazione binaria R su un insieme S è la relazione transitiva R^+ sul set S tale che R^+ contiene R. Se la relazione binaria è transitiva già da sé, allora la chiusura transitiva è la relazione binaria stessa; in caso contrario, la chiusura transitiva è una relazione differente.
Ci sono dei casi in cui ci si aspetta che una relazione di preferenza sia chiusura transitiva di un'altra relazione di preferenza che non è transitiva.
In questo ambito, si può esprimere che la relazione di preferenza $\succ_{C^*}$ sia chiusura transitiva di un'altra relazione di preferenza $\succ_C$, utilizzando una formula del primo ordine. Ciò è simile a quanto avviene nei database relazionali, con le query di chiusura transitiva. Attenzione però ad una differenza importante che non deve sfuggire: nei database, viene calcolata la chiusura transitiva di una relazione finita, mentre qui si sta facendo la chiusura transitiva di una relazione infinita definita utilizzando una formula del primo ordine.

Composizione multidimensionale

Sono stati studiati vari modi per definire una relazione di preferenza sul prodotto cartesiano di due relazioni. I due più comuni sono Pareto e la composizione lessicografica.
Dati due schemi relazionali R_1 e R_2 e relazioni di preferenza $\succ_{C_1}$ su R_1 e $\succ_{C_2}$ su R_2, la composizione Pareto P ($\succ_{C_1}$, $\succ_{C_2}$) di $\succ_{C_1}$ e $\succ_{C_2}$ è una relazione di preferenza $\succ_{C_0}$ sul prodotto cartesiano $R_1 \times R_2$ definita come:
$(t_1, t_2) \succ_{C_0} (t'_1, t'_2) \equiv t_1 \succeq_{C_1} t'_1 \wedge t_2 \succeq_{C_2} t'_2 \wedge (t_1 \succ_{C_1} t'_1 \vee t_2 \succ_{C_2} t'_2)$
dove $x \succeq_C y \equiv x \succ_C y \vee x \sim_C y$.
Dati due schemi relazionali R_1 e R_2 e le relazioni di preferenza $\succ_{C_1}$ su R_1 e $\succ_{C_2}$ su R_2, la composizione lessicografica $L(\succ_{C_1}, \succ_{C_2})$ di $\succ_{C_1}$ e $\succ_{C_2}$ è una relazione di preferenza $\succ_{C_0}$ sul prodotto cartesiano $R_1 \times R_2$ definita come:

$(t_1, t_2) \succ_{C_0} (t'_1, t'_2) \equiv t_1 \succeq_{C_1} t'_1 \vee (t_1 \sim_{C_1} \wedge t_2 \succ_{C_2} t'_2)$.
L'ordine parziale stretto non viene conservato né dalla composizione Pareto né dalla composizione lessicografica. L'ordine debole e l'ordine totale sono invece conservati dalla composizione lessicografica, ma non dalla composizione Pareto.

1.4 Ingegneria delle preferenze

Al giorno d'oggi è necessaria la presenza di una struttura efficace che sia in grado di supportare l'implementazione di preferenze complesse utilizzando quelle più semplici; questo perché sono sempre più numerose richieste complesse, relative a diversi attributi. È importante, da questo punto di vista, essere in grado di fornire costruttori delle preferenze di base, che in realtà altro non sono che i modelli stessi delle preferenze, le cui istanze danno la precedenza alle preferenze di base.
Formalmente, un costruttore di preferenze di base ha uno o più argomenti, il primo dei quali caratterizza i nomi degli attributi A e gli altri l'ordinamento parziale stretto <P, facendo riferimento ad A.

1.4.1 Costruttori di preferenze di base

I costruttori delle preferenze di base altro non sono che modelli di preferenza le cui istanze forniscono le preferenze di base. Da un punto di vista ingegneristico è molto importante avere a disposizione questo tipo di costrutto poiché si rivela molto utile nei motori di ricerca personalizzati.
Formalmente, un costruttore di preferenze di base ha uno o più argomenti, il primo per gli attributi mentre i seguenti per l'ordine parziale stretto <P, facente riferimento agli attributi.
Le preferenze di base si distinguono fra quelle numeriche e quelle non numeriche.

Preferenze di base non numeriche

Vengono descritte le principali preferenze di base non numeriche.
Una preferenza POS esprime una condizione secondo la quale un valore desiderato dovrebbe essere all'interno di una data lista di valori. Se il valore richiesto non esiste, saranno presi in considerazione i valori alternativi più "vicini" a quello richiesto.
La preferenza "non dovrebbe esserci" è espressa attraverso la preferenze NEG. Al contrario della preferenza POS, qui l'argomento specifica quale

valore non è desiderato. Anche in questo caso, se la richiesta iniziale non può essere assecondata, è preferibile che non venga restituito un risultato vuoto.
Nelle ultime versioni del linguaggio SQL, sono supportate molte combinazioni di preferenze POS e NEG (ad esempio, POS/NEG e POS/POS) e anche il tipo di preferenza di base CONTAINS per la ricerca nell'intero testo di un documento. Qualsiasi altra preferenza che possa essere espressa come relazione "A è meglio di B" può essere creata come una preferenza di base del tipo EXPLICIT.

Preferenze di base numeriche

Ora vengono descritte le preferenze del tipo P = (A, <P), dove dom(A) è un certo tipo di dati numerici che supporta un operatore di confronto totale '<' e un operatore di differenza '-'.
Le preferenze AROUND e BETWEEN sono preferenze di approssimazione. La preferenza AROUND individua valori prossimi ad un valore numerico indicato. Il suo utilizzo ricorre spesso quando individuare un valore specifico risulta molto difficile oppure di importanza secondaria.
La preferenza BETWEEN[estremo basso, estremo alto] si comporta in maniera analoga; vengono cercati valori che siano all'interno del range identificato dai due estremi specificati in input. Nel caso non fosse possibile, saranno cercati i valori più vicini agli estremi indicati, malgrado siano esterni, sempre per evitare di fornire un risultato vuoto.
Le preferenze LOWEST e HIGHEST sono rispettivamente preferenza di minimizzazione e preferenza di massimizzazione. Capita infatti molto spesso che venga chiesto di individuare il valore più basso o più alto all'interno di un set. Come per tutti gli altri casi, se non fosse possibile individuare quanto richiesto, verrà restituito il valore che più si avvicina.

1.4.2 Costruttori di preferenze complesse

Generalmente, quando si prende una decisione non ci si basa esclusivamente su una singola preferenza, piuttosto su una possibile combinazione complessa di preferenze. Il vero potenziale della modellazione di preferenze scaturisce con l'avvento dei costruttori di preferenze complesse. Le ultime

versioni di SQL offrono i mezzi per assemblare induttivamente preferenze complesse.

Costruttori di preferenze cumulativi

I costruttori di preferenza cumulativi combinano preferenze con provenienza differente.

Accumulazione Pareto
In questo caso, il principio di Pareto viene utilizzato per gestire preferenze di uguale importanza fra loro. L'accumulazione Pareto delle preferenze $P_1, \ldots, P_n$ in un'unica preferenza complessa P è definita nella seguente maniera:
$v = (v_1, \ldots, v_n)$ è migliore di $w = (w_1, \ldots, w_n)$ se e solo se
$\exists$ i tale che v_i è meglio di w_i e
v è uguale o migliore di w in ogni altra dimensione.
L'accumulazione Pareto forma un nuovo ordine parziale stretto, quindi è una preferenza. Il principio di Pareto è stato applicato e ampiamente studiato per problemi di decisione multi-attributo nel campo delle scienze sociali ed economiche.

Preferenze con priorità
Spesso alcune preferenze potrebbero essere considerate più importanti rispetto ad altre. Intuitivamente, se P_1 ha maggiore priorità rispetto a P_2, allora a quest'ultima non è concesso di trovare alcun compromesso con la richiesta di P_1. P_2 viene valutata solo quando P_1 offre più di un'unica alternativa, tutte di pari utilità.
Date $P_1 = (A_1, <P_1)$ e $P_2 = (A_2, <P_2)$, per $x, y \in \text{dom}(A_1) \times \text{dom}(A_2)$ si definisce:
$x <P_1\&P_2\ y$ se e solo se $x_1 <P_1\ y_1 \vee (x_1 = y_1 \wedge x_2 <P_2\ y_2)$.

Costruttori di preferenze aggregativi

I costruttori di preferenze aggregativi assemblano una preferenza complessa P da differenti preferenze di base $P_1, \ldots, P_n$ separate tra loro, che agiscono sullo stesso set di attributi.
Due preferenze $P_1 = (A, <P_1)$ e $P_2 = (A, <P_2)$ sono chiamate disgiunte se $\text{range}(<P_1) \cap \text{range}(<P_2) = \emptyset$, dove per $\text{range}(<P_i)$ di una preferenza si

intende::
range($<P_i$) := $\{x \in dom(A) \mid \exists\ y \in dom(A)$:
$(x, y) \in <P_i \vee (y, x) \in <P_i\}$.
Date le preferenze disgiunte P_1 e P_2, $P = (A, <P_1 + P_2)$ è definita unione di preferenze disgiunte, se:
$x <P_1+P_2\ y$ se e solo se $x <P_1\ y \vee x <P_2\ y$.

Supponendo di avere due preferenze $P_1 = (A_1, <P_1)$ e $P_2 = (A_2, <P_2)$ tali che per i singoli attributi $A_1 \neq A_2$ e che $dom(A_1) \cap dom(A_2) = \emptyset$, si ha quindi che P_1 e P_2 sono preferenze disgiunte.
Sia che, per un nuovo attributo A, $dom(A) := dom(A_1) \cup dom(A_2)$, allora $P = (A, <P_1 \oplus P_2)$ è una preferenza a somma lineare se:
$x<P_1 \oplus P_2\ y$ se e solo se $x <P_1\ y \vee x <P_2\ y \vee$
$(x \in dom(A_2) \wedge y \in dom(A_1))$.

1.5 Algebra delle preferenze

È importante per un sistema di database essere in grado di fornire tutte le risposte che possono rientrare nelle esigenze di un utente. Tuttavia, la maggior parte degli attuali motori di ricerca di database trattano solo vincoli rigidi: una tupla appartiene al risultato della ricerca solo se adempie a tutte le condizioni specificate. Una preferenza consente agli utenti di stabilire un ordine parziale stretto che esprime quali oggetti del database siano migliori di altri. Sulla base di ciò, una query seleziona quegli oggetti che non sono dominati da tutti gli altri nella relazione di preferenza. Per dare all'utente maggiore flessibilità, vengono messi a disposizione molti operatori predefiniti per la costruzione di relazioni di preferenza. Questa struttura algebrica dà un punto di vista del tutto innovativo sulle preferenze.
Per comporre preferenze complesse, sono stati introdotti operatori speciali che si differenziano dalle classiche operazioni standard; queste ultime, infatti, sono anch'esse in grado di fornire supporto nella composizione di alcune tipologie di preferenze complesse ma sono raramente utilizzate nel tipico dominio di applicazione dell'algebra delle preferenze.
Prima dell'introduzione di questa disciplina, non c'era una maniera semplice con cui esprimere teoremi riguardanti la legge distributiva per le preferenze con priorità e le preferenze Pareto. La ragione principale per cui è stato

necessario affrontare questo studio è che in origine l'equivalenza dei termini di preferenza era definita in modo molto implicito; era infatti possibile affermare che: "due termini preferenza sono equivalenti se e solo se le relazioni corrispondenti sono identiche al set di base". Questa definizione non risulta essere molto utile se si cerca di trovare (automaticamente) prove generali di equivalenza. Ora è invece possibile esprimere questo concetto: "due termini di preferenza sono identici, se e solo se le loro rappresentazioni algebriche sono uguali in algebra".
Questo tipo di lavoro comprende anche alcuni aspetti che contribuiscono alla descrizione formale dei problemi relativi al database.

Capitolo 2

L'operatore Winnow

2.1 Proprietà di base

Viene qui analizzato un nuovo operatore dell'algebra relazionale, chiamato Winnow, che è stato introdotto con lo scopo di selezionare dalla sua relazione delle tuple più appetibili secondo la relazione di preferenza data. Nonostante l'operatore Winnow possa essere espresso mediante l'impiego di altri operatori di algebra relazionale, considerandolo singolarmente è possibile sia concentrarsi sulle proprietà astratte di relazioni di preferenza (ad esempio, la transitività), sia studiare speciali tecniche di valutazione e di ottimizzazione per l'operatore Winnow stesso. Per SQL, la scelta da compiere è analoga: il linguaggio query viene opportunamente esteso con un operatore SQL equivalente a Winnow oppure le occorrenze di Winnow sono tradotte in SQL.
L'operatore Winnow ha diverse proprietà che possono offrire un importante contributo nell'ambito del linguaggio SQL, tra cui non vuotezza, monotonia e forza espressiva ma anche alcune proprietà algebriche come quella commutativa e quella distributiva. Analizzare queste proprietà, e come sono state stabilite, è essenziale per la valutazione e l'ottimizzazione delle query di preferenza.
Questo operatore algebrico che preleva l'insieme delle tuple più appetibili da una relazione data, secondo una formula di preferenza, è stato così definito: Se R è uno schema di relazione e C una formula di preferenza che definisce una relazione $\succ_C$ di preferenza su R, allora l'operatore Winnow viene scritto come $\omega_C(R)$, per ogni istanza r di R:

$$\omega_C(r) = \{t \in r \mid \neg\, \exists t' \in r.\ t' \succ_C t\}.$$

Una query di preferenza è una query ad algebra relazionale che contiene almeno un'occorrenza dell'operatore Winnow.

2.2 Non vuotezza

Se una relazione di preferenza $\succ_C$ su R è un ordine parziale stretto, allora per ogni istanza, finita e non vuota, r di R, $\omega_C(r)$ sicuramente non è vuoto. Non è possibile violare questa affermazione appena fatta poiché, così facendo, cadrebbero le ipotesi. Innanzi tutto, si dovrebbe escludere la non riflessività di $\succ_C$; così sarebbe presente una tupla t_0 tale che $t_0 \succ_C t_0$; ma a questo punto $\omega_C(\{t_0\}) = \emptyset$. In secondo luogo, se due tuple sono coinvolte in una violazione dell'asimmetria, si bloccano a vicenda nella visualizzazione del risultato dell'operatore Winnow. Inoltre, senza transitività una relazione di preferenza potrebbe essere asimmetrica però il risultato dell'operatore Winnow sarebbe vuoto. Infine, se la relazione r è infinita, può accadere che $\omega_C(r) = \emptyset$, ad esempio, se r contiene tutti i numeri naturali e la relazione di preferenza è l'ordinamento $>$.

2.3 Monotonia

Se una relazione di preferenza $\succ_C$ su R è un ordine parziale stretto, allora per ogni coppia di istanze r_1 e r_2 di R:

$$r_1 \subseteq r_2 \Rightarrow r_1 \cap \omega_C(r_2) \subseteq \omega_C(r_1).$$

Se $\succ_{C_1}$ e $\succ_{C_2}$ sono relazioni di preferenza su uno schema relazionale R, ed è valida la formula $\forall\, t_1, t_2\ [C_1(t_1, t_2) \Rightarrow C_2(t_1, t_2)]$ allora per tutte le istanze r di R, $\omega_{C_2}(r) \subseteq \omega_{C_1}(r)$. Se $\succ_{C_2}$ è non riflessiva, allora lo è anche il viceversa.

2.4 Forza espressiva

Per algebra relazionale standard, si intendono i seguenti operatori: selezione, proiezione, prodotto cartesiano, unione, differenza (ovvero individuare tutti gli elementi che fanno parte di un insieme ma non di un altro, escludendo quindi dal primo insieme tutti gli elementi facenti parte della sua intersezione con il secondo) e ridenominazione.
L'operatore Winnow può essere espresso in algebra relazionale, senza quindi aggiungere qualsiasi potere espressivo ad essa. Può anche essere utilizzato per simulare la differenza.
La forza espressiva della algebra relazionale standard non cambia se la differenza è sostituita dall'operatore Winnow. Questo significa che tutte le query d'algebra relazionale con l'operatore Winnow possono essere tradotte al calcolo relazionale e, una volta fatto ciò, di nuovo tradotte all'algebra relazionale (senza l'operatore Winnow). Questa informazione ha un'importanza esclusivamente teorica.

2.5 Proprietà commutativa

Viene qui espressa una condizione sufficiente affinché l'operatore Winnow sia commutativo. La proprietà commutativa è fondamentale perché permette di spostare l'operatore Winnow all'interno delle query di preferenza nella posizione che risulta più conveniente. Tuttavia, la proprietà commutativa richiede idempotenza (applicando molteplici volte una data funzione, il risultato ottenuto è uguale a quello derivante dall'applicazione della funzione un'unica volta), cioè, due occorrenze consecutive di Winnow commutano se una può collassare sull'altra.
Se C_1 e C_2 sono formule di preferenza su uno schema R tali che $\succ_{C_1}$ e $\succ_{C_2}$ sono ordini parziali stretti ed è valida la formula
$\forall\ t_1, t_2\ [C_1(t_1, t_2) \Rightarrow C_2(t_1, t_2)]$ allora per tutte le istanze finite r di R:

$$\omega_{C_1}(\omega_{C_2}(r)) = \omega_{C_2}(\omega_{C_1}(r)) = \omega_{C_2}(r).$$

Winnow, per commutare con la proiezione, ha bisogno che la formula di preferenza (cioè il parametro dell'operatore Winnow) venga limitata agli attributi nella proiezione. La proiezione, comunque, non influisce sulla relazione di preferenza in questione.

2.6 Proprietà distributiva

Per applicare la proprietà distributiva dell'operatore Winnow sul prodotto cartesiano di due relazioni, la formula di preferenza (cioè il parametro dell'operatore Winnow) dev'essere scomposta in formule di preferenza che distribuiscono l'argomento della relazione.
Dati due schemi relazionali R_1 e R_2, una relazione di preferenza non riflessiva $\succ_{C_1}$ su R_1 ed un'altra relazione di preferenza (sempre non riflessiva) $\succ_{C_2}$ su R_2, per ogni relazione r_1 e r_2, che sono rispettivamente istanze di R_1 e R_2, vale la seguente proprietà:
$\omega_{C_0}(r_1 \times r_2) = \omega_{C_1}(r_1) \times \omega_{C_2}(r_2)$ dove $C_0 = P(\succ_{C_1}, \succ_{C_2})$.
In questo ambito viene naturale prendere in considerazione quelle relazioni di preferenza che sono costruite attraverso la composizione multidimensionale.
Per completezza, è stato dimostrato che è possibile distribuire l'operatore Winnow anche nell'unione e nella differenza solo nel caso banale in cui la relazione di preferenza sia un insieme vuoto. Due schemi relazionali vengono definiti compatibili se hanno lo stesso numero di attributi e gli attributi corrispondenti hanno gli stessi domini.
Dati due schemi di relazione compatibili R ed S e una relazione di preferenza asimmetrica $\succ_C$ su R, per ogni istanza r di R e ogni istanza di s S si ha che $\omega_C(r \cup s) = \omega_C(r) \cup \omega_C(s)$ e $\omega_C(r - s) = \omega_C(r) - \omega_C(s)$
se e solo se $\succ_C = \emptyset$.

2.7 Preferenze iterate e classificazione

Iterando l'operatore Winnow si possono classificare le tuple di una determinata relazione. La classificazione è definita utilizzando le preferenze iterate. Data una relazione di preferenza $\succ$ definita da una formula di preferenza C, l'ennesima iterazione dell'operatore Winnow ω_C in r è così definita:
$\omega_C^1(r) = \omega_C(r)$
$\omega_C^{n+1}(r) = \omega_C(r - \cup_{1 \leq i \leq n} \omega_C^1(r))$.

Ad esempio, la query $\omega_C^2(R)$ calcola il set di "seconde migliori" tuple.

Se una relazione di preferenza $\succ_C$ su uno schema relazionale R è un ordine parziale stretto, allora per ogni istanza r di R ed ogni tupla $t \in r$,

esiste un i, con i ≥ 1, tale che t $\in \omega_C^i$(r).
Se R è uno schema relazionale e C una formula di preferenza che definisce una relazione di preferenza $\succ_C$ su R, allora l'operatore Ranking è definito come η_C(R) e per ogni istanza r di R: $\eta_C(r) = \{(t, i) \mid t \in \omega_C^i(r)\}$.
L'operatore ranking è definito induttivamente utilizzando la ricorsione ma non può essere espresso in SQL, neppure utilizzando le definizioni ricorsive di SQL. D'altra parte, la classifica può essere calcolata in modo efficiente utilizzando versioni estese dell'algoritmo block-nested-loops.
Le proprietà algebriche dell'operatore Ranking sono parallele a quelle dell'operatore Winnow. Dal punto di vista pratico, la proprietà più importante è quella commutativa.
Dati uno schema relazionale R, una condizione di selezione C_1 su R e la formula di preferenza C_2, se è valida la formula
$\forall\ t_1, t_2\ [(C_1(t_2) \wedge C_2(t_1, t_2)) \Rightarrow C_1(t_1)]$
allora per ogni istanza r di R: $\sigma_{C_1}(\eta_{C_2}(r)) = \eta_{C_2}(\sigma_{C_1}(r))$.

2.8 Altre considerazioni

Come già precisato sopra, l'operatore Winnow ha un funzionamento simile a quello della differenza; questo perché ogni tupla presente nel risultato finale si trova lì perché non è stata "superata", "sovrastata" da alcuna altra tupla presente nel set di partenza. Però, mentre nella differenza, eliminare una tupla significa avere trovato un'altra occorrenza della stessa tupla, per l'operatore Winnow significa aver trovato una tupla dominante. L'operatore Winnow risulta per questo più simile ad un operatore binario, nonostante richieda un solo argomento, perché richiede il confronto di una tupla possibilmente con tutte le altre tuple del set d'ingresso.
Esistono diversi algoritmi con cui è possibile calcolare il risultato dell'operatore Winnow. Il primo di questi è l'algoritmo nested-loops, mentre l'algoritmo block-nested-loops è una sua variante; un'altra possibile soluzione è l'algoritmo sequential-feature-selection.
Tutti gli algoritmi utilizzano una quantità fissa di memoria principale (chiamata finestra). Tuttavia, per l'algoritmo nested-loops, ciò non può essere espresso esplicitamente, dal momento che la memoria a disposizione è irrilevante per le proprietà dell'algoritmo.
L'algoritmo nested-loops è corretto per qualsiasi relazione di preferenza $\succ_C$.

In linea di principio, la relazione di preferenza potrebbe anche essere riflessiva, poiché l'algoritmo confronta esplicitamente una tupla con sé stessa. Gli algoritmi block-nested-loops e sequential-feature-selection richiedono invece che la relazione di preferenza sia un ordine parziale stretto e quindi che sia non riflessiva perché una tupla non si confronta con sé stessa. Neanche questi sono in grado di gestire correttamente la simmetria, ciò si verifica nella situazione in cui ci sono due tuple t_1 e t_2 tali che $t_1 \succ_C$ tped2 e $t_2 \succ_C t_1$; in questo caso, l'algoritmo block-nested-loops romperà il vincolo a seconda dell'ordine in cui compaiono le tuple mentre l'algoritmo sequential-feature-selection non riuscirà ad arrivare al suo termine in quanto non è in grado di effettuare un ordinamento topologico.
L'algoritmo block-nested-loops assume un'importanza di rilievo in quanto è stato studiato come possibile soluzione delle query Skyline nel contesto di database .

Capitolo 3

L'operatore Skyline

3.1 Obiettivi dell'analisi

In questo capitolo, viene preso in analisi un operatore meno generico di quanto non sia l'operatore Winnow; l'esposizione descrive come sia possibile estendere un sistema di database mediante un'operazione di Skyline. Questa operazione effettua una ricerca all'interno di una serie di punti dati, potenzialmente molto vasta, e fornisce una collezione di punti interessanti. Un punto è definito "interessante" se non è dominato da qualsiasi altro punto. Il linguaggio SQL può essere esteso per porre query di Skyline. Alcuni algoritmi alternativi verranno valutati per attuare l'operazione di Skyline, e si analizzerà come combinare le altre operazioni di database (ad esempio, join) con questa operazione.
Le query Skyline permettono di trovare tutte le tuple in una relazione che non sono dominate in tutte le dimensioni da nessun'altra tupla. Più formalmente, la Skyline è definita come l'insieme di quei punti che non sono dominati da qualsiasi altro punto. Un punto domina un altro punto se è migliore in tutte le dimensioni. Dalla Skyline, si può ora prendere la decisione finale, in base alla pesatura delle proprie preferenze personali. La computazione della Skyline è conosciuta come il massimo problema vettoriale.
Possono essere trovate molte applicazioni nel settore del sostegno decisionale in cui l'utilizzo di un'operazione di Skyline è sia efficace che conveniente; un'operazione Skyline può anche essere molto utile per la visualizzazione del contenuto di un database.
Tipicamente, le query Skyline sono formulate nel contesto di multi spazio

euclideo tridimensionale dove la relazione è > oppure <.
Una delle proprietà più interessanti della Skyline è che non importa come si pesano le preferenze personali verso gli attributi della relazione, in ogni caso si troverà la propria preferenza all'interno della Skyline. Formalmente, data una serie M e la sua relativa Skyline, per ogni punto della funzione monotona M $\longrightarrow$ R se p appartiene a M massimizza i punti di tale funzione, quindi p è nella Skyline.
Vengono ora riportate descrizioni di possibili estensioni SQL che implementino le query Skyline. Dopodiché verranno presi in considerazione alcuni algoritmi alternativi per il calcolo della Skyline (e le loro valutazioni); sarà possibile comprendere come questi si siano evoluti e di quanto l'algoritmo originale fosse piuttosto inadeguato se applicato nell'ambito dei database, a causa delle sue prestazioni poco performanti.

3.2 Estensioni SQL ed implementazione dell'operatore Skyline

L'approccio visionato per implementare le query Skyline è quello di estendere un sistema di database già esistente attraverso l'impiego di un nuovo operatore logico a cui si farà riferimento come operatore Skyline. L'operatore Skyline incapsula l'applicazione della clausola SKYLINE OF. L'implementazione di altri operatori (ad esempio, join) non deve essere cambiata.
Secondo la semantica delle query Skyline, l'operatore Skyline è generalmente eseguito dopo le operazioni di join, e group-by ma prima di un operatore di ordinamento finale, se la query ha una clausola ORDER BY (sono possibili eccezioni).
Per analizzare le query Skyline, è stato proposto di estendere la clausola SELECT di SQL attraverso questa nuova clausola opzionale SKYLINE OF come riportato nella figura 3.1.
d1, . . . , dm indicano le dimensioni della Skyline; MIN, MAX, e DIFF specificano se il valore in quella dimensione dovrebbe essere minimizzato, ingrandito, o semplicemente essere diverso. La dimensione x sulla quale si vuole visualizzare la propria preferenza è precisata nella clausola SKYLINE OF della query con un'annotazione DIFF.
La clausola SKYLINE OF seleziona tutte le tuple interessanti, cioè quelle tuple che non sono dominate da qualsiasi altra tupla. Se due tuple hanno

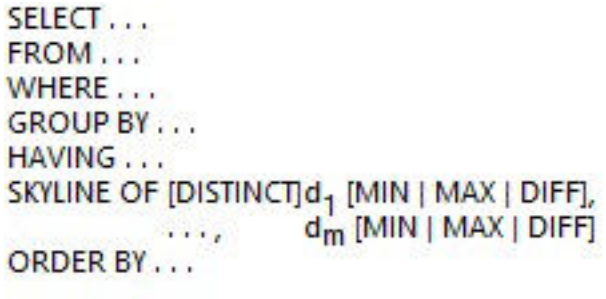

Figura 3.1: Clausola Skyline

gli stessi valori per tutti gli attributi elencati nella clausola SKYLINE OF e non sono dominate da qualsiasi altra tupla, allora sono entrambe parte del risultato. Con DISTINCT, tuttavia, una di esse viene mantenuta (la scelta di quale è imprecisata). È doveroso precisare che è irrilevante quale sia l'ordine con cui le dimensioni sono specificate nella clausola SKYLINE OF. Inoltre, una Skyline unidimensionale è equivalente a una Query SQL di MIN, MAX, o DISTINCT senza clausola SKYLINE OF. Il dominio è una relazione transitiva: se p domina q e q domina T, allora p domina anche T. La proprietà di transitività, precedentemente descritta, è importante per riuscire ad implementare l'operazione di Skyline.
Alla stessa maniera dell'operatore join (e anche per la maggior parte degli altri operatori logici) ci sono molti modi diversi, fisici, per implementare l'operatore Skyline. In questa sezione, vengono descritte alcune varianti.

3.2.1 Traduzione di una query Skyline in una query SQL nidificata

Traducendo la query Skyline in una query SQL nidificata, ogni query Skyline può essere implementata in cima a un sistema di database relazionale. Tutto ciò è rappresentato dalla query in figura 3.2 (le condizioni della query sono complementari fra loro).
Questo approccio offre prestazioni di basso livello per le seguenti ragioni:

- Sostanzialmente, questo approccio corrisponde all'algoritmo "nested-loops" per calcolare la Skyline, perché questa query non può essere non

```
SELECT *
FROM HOTELS h
WHERE h.city = 'Rimini' AND NOT EXIST(
                SELECT *
                FROM HOTELS h1
                WHERE h1.city = 'Rimini' AND
                h1.distance <= h.distance AND
                h1.price <= h.price AND
                (h1.distance < h.distance OR
                h1.price < h.price));
```

Figura 3.2: Query d'esempio

nidificata; come si vedrà nelle seguenti sottosezioni, c'è l'opportunità di fare molto meglio.

- Se la query Skyline implica un join o un group-by, questo join o group-by dovrebbe essere eseguito come parte esterna della query.
- L'operazione Skyline può essere combinata con altre operazioni (ad esempio, join) in alcuni casi, con il risultato di un piccolo costo aggiuntivo nel calcolo della Skyline.

Di conseguenza, si propone di estendere il sistema di database e integrare un operatore Skyline nel sistema.

3.2.2 Operatore Skyline bidimensionale

In precedenza è stato visto che calcolare una Skyline unidimensionale è banale perché equivale a calcolare le funzioni di MIN, MAX o DISTINCT. Il calcolo della Skyline è molto facile anche se la clausola SKYLINE OF coinvolge solo due dimensioni. Una Skyline bidimensionale può essere calcolata attraverso un ordinamento dei dati. Se i dati sono topologicamente ordinati secondo i due attributi specificati nella clausola SKYLINE OF è sufficiente confrontare una tupla con il suo predecessore per verificare se una tupla è parte della Skyline, quindi il costo è molto economico.
Per affrontare Skyline tridimensionali sono stati realizzati anche ulteriori algoritmi speciali, ma non sono stati presi in analisi all'interno di questo elaborato.

3.2.3 Algoritmo Block-nested-loops

Applicando un algoritmo nested-loops e confrontando tutte le tuple con ogni altra tupla, ai ottiene il modo più immediato per calcolare la Skyline. L'algoritmo nested-loops più semplice può essere applicato a qualsiasi query Skyline (non solo bidimensionale), ma ovviamente è molto inefficiente. In questa sezione, viene descritto un algoritmo che è significativamente più veloce; piuttosto che considerare una tupla alla volta, questo algoritmo elabora un blocco di tuple ad ogni sua iterazione.
L'algoritmo Block-nested-loops legge più volte il set di tuple (alla stessa maniera dell'algoritmo nested-loops). Questo algoritmo prevede il mantenimento di tuple fra loro incomparabili all'interno di una finestra nella memoria principale. Quando una tupla p viene letta in input, p viene confrontata con tutte le tuple della finestra e, basandosi su questo confronto, p sarà eliminata, collocata nella finestra oppure in un file temporaneo che sarà esaminato nella prossima iterazione dell'algoritmo. Possono verificarsi tre casi:

- p è dominata da una tupla all'interno della finestra. In questo caso, p viene eliminata e non sarà considerata nelle future iterazioni. Naturalmente, non è necessario confrontare p con tutte le tuple della finestra in questo caso;

- p domina uno o più tuple nella finestra. In questo caso, queste tuple sono eliminate; vale a dire, queste tuple sono rimosse dalla finestra e non saranno considerate in iterazioni future; p è inserita nella finestra;

- p è incomparabile con tutte le tuple nella finestra. Se c'è spazio sufficiente nella finestra, p viene inserita nella finestra. In caso contrario, p viene scritta un file temporaneo. Le tuple del file temporaneo saranno ulteriormente processate nella prossima iterazione dell'algoritmo; quando l'algoritmo inizia, la prima tupla verrà immediatamente inserita nella finestra poiché la finestra è vuota.

Alla fine di ogni iterazione, le tuple della finestra che sono state confrontate con ogni altra tupla stata scritta nel file temporaneo possono essere escluse; queste tuple fanno parte della Skyline, ovvero non sono dominate da altre tuple e a loro volta non dominano alcuna tupla che sia ancora in esame. In particolare, si possono sicuramente escludere ed ignorare per ulteriori

elaborazioni, quelle tuple che sono state inserite nella finestra quando il file temporaneo era vuoto, cioè all'inizio dell'iterazione. Durante la successiva iterazione possono essere escluse quelle altre tuple della finestra che non sono eliminate. Quanto prima una tupla è stata inserita nella finestra, quanto prima la prima tupla potrà essere esclusa durante la successiva iterazione. Per garantire che l'algoritmo arrivi alla sua conclusione e che una coppia di tuple non venga confrontata più di una volta, ad ogni tupla presente nella finestra viene assegnato un contatore che registra l'ordine di inserimento delle tuple nella finestra e nel file temporaneo; leggendo il valore del contatore di una tupla, possono essere escluse dalla finestra tutte le tuple che hanno un contatore con valore inferiore a quello appena letto.
Più la Skyline è di dimensioni ridotte, più questo algoritmo risulta essere efficiente e funzionale. Nel migliore dei casi, non appena il risultato finale, cioè la Skyline, si inserisce nella finestra, l'algoritmo termina in una o al massimo due iterazioni; pertanto, il costo computazione della complessità dell'algoritmo nel caso migliore è dell'ordine di O(n), dove n è il numero di tuple nell'input. Nel caso peggiore, la complessità è dell'ordine di $O(n^2)$. La complessità asintotica è la stessa dell'algoritmo nested-loops primitivo ma l'algoritmo block-nested-loop mostra un utilizzo delle risorse di I/O decisamente migliore rispetto all'algoritmo nested-loops primitivo.

3.2.4 Algoritmo Divide et Impera

L'algoritmo divide et impera è stato proposto affinché il calcolo della Skyline risulti più efficiente nell'ambito dei database, cioè dove solitamente la memoria a disposizione è piuttosto limitata. Questo algoritmo è teoricamente il migliore algoritmo noto per il caso peggiore. Nel caso peggiore, la sua complessità asintotica è dell'ordine di
$O(n * (\log n)^{d-2}) + O(n * \log n)$; n è il numero di tuple in ingresso e d è il numero di dimensioni nella Skyline. Purtroppo, questa è anche la complessità dell'algoritmo nel migliore dei casi; così, ci si aspetta che questo algoritmo funzioni meglio rispetto all'algoritmo block-nested-loops nei casi peggiori, ma allo stesso tempo che sia peggiore nei casi migliori.
L'algoritmo divide et impera di base procede nel seguente modo:

1. Si calcola il mediano m_p (o qualche mediano approssimativo) in input per qualche dimensione d_p. Si divide l'input in due partizioni. P_1

contiene tutte le tuple il cui valore dell'attributo d_p è migliore rispetto a quello di m_p. P_2 contiene tutte le altre tuple.

2. Si calcolano la Skyline S_1 di P_1 e la Skyline S_2 di P_2. Questo calcolo viene ripetuto ricorsivamente, applicando l'intero algoritmo per P_1 e P_2; ossia, P_1 e P_2 vengono a loro volta partizionati. Il partizionamento ricorsivo si interrompe se una partizione contiene solo una o poche tuple. In questo caso, il calcolo della Skyline è banale.

3. Il calcolo della Skyline globale è il risultato del merge fra S_1 e S_2. Cioè, si eliminano quelle tuple di S_2 che sono dominate da una tupla in S_1 (nessuna delle tuple in S_1 può essere dominata da una tupla in S_2 perché ogni tupla presente in S_1 è migliore nella dimensione d_p di ogni altra tupla presente in S_2).

Successivamente, sia S_1 che S_2 vengono nuovamente partizionate utilizzando un mediano m_g per qualche altra dimensione d_g, con $d_g \neq d_p$. Come risultato, si ottengono quattro partizioni: $S_{1,1}$, $S_{1,2}$, $S_{2,1}$, $S_{2,2}$. $S_{1,i}$ è migliore di $S_{2,i}$ in dimensione d_p e $S_{i,1}$ è migliore di $S_{i,2}$ in dimensione d_g (i = 1,2). Adesso, per ottenere il risultato finale, è necessario unire $S_{1,1}$ e $S_{2,1}$, $S_{1,1}$ e $S_{2,2}$, e $S_{1,2}$ e $S_{2,2}$. Non occorre effettuare il merge fra $S_{1,2}$ e $S_{2,1}$ perché le tuple di questi due insiemi è garantito che siano incomparabili. L'unione di $S_{1,1}$ con $S_{2,1}$ (e le altre coppie) viene fatta ricorsivamente applicando la funzione di merge. Si ha quindi che $S_{1,1}$ e $S_{2,1}$ sono di nuovo partizionate.
La ricorsione della funzione di merge termina solo in due condizioni:

- se sono state considerate tutte le dimensioni, oppure
- se una delle partizioni è vuota o contiene solo una tupla

In tutti questi casi la funzione di merge è banale.
L'algoritmo di base ha delle prestazioni terribili se l'input non dovesse essere di dimensioni ottimali per la memoria a disposizione. Fortunatamente, il comportamento di I/O dell'algoritmo di base può essere migliorato abbastanza facilmente. L'idea è quella di dividere in m partizioni in modo tale che ogni partizione possa adattarsi alla memoria disponibile. Se una partizione non rientra nella memoria, deve essere partizionata di nuovo; tutto ciò è analogo al partizionamento ricorsivo se necessario, ad esempio, per l'elaborazione hash join.

Il partizionamento a m vie può essere utilizzato nella prima fase dell'algoritmo di base così come nel terzo passo. Nel primo passo, il partizionamento a m vie viene utilizzato per produrre le partizioni m P_1,..., P_m in modo che ogni P_i si inserisca nella memoria e S_i, la Skyline di P_i, possa essere calcolata in memoria utilizzando l'algoritmo di base. Il risultato finale è prodotto nel terzo passo dalla fusione delle coppie S_i. All'interno della funzione di merge, il partizionamento a m vie viene applicato in modo tale che tutte le sotto-partizioni devono occupare al massimo metà della memoria principale disponibile, questo perché le sotto-partizioni possano essere unite in memoria principale.
In situazioni in cui la memoria principale disponibile è limitata, un'estensione molto semplice all'algoritmo divide et impera prevede che, durante la prima fase in cui i dati sono partizionati in m partizioni:

1. Un grande blocco di tuple venga caricato dall'input; più precisamente, si carica il maggior numero di tuple che può essere inserito nei buffer disponibili della memoria.

2. Si applica l'algoritmo di base divide et impera per questo blocco di tuple per eliminare immediatamente le tuple che sono dominate da altre (ciò viene definito "Early Skyline").

3. Si partizionano le rimanenti tuple in m partizioni.

I compromessi di questo approccio sono abbastanza semplici. Chiaramente, l'applicazione di una Early Skyline precoce comporta costo aggiuntivo di CPU, ma non richiede un elevato consumo di risorse I/O perché meno tuple devono essere scritte e rilette nei passi di partizionamento. In generale, una Early Skyline è interessante se il risultato dell'intera operazione di Skyline è di dimensioni ridotte (ovvero la Skyline è selettiva).

3.3 Algoritmi indicizzati per il calcolo della Skyline

Per eseguire il calcolo della Skyline, è prevista anche l'eventualità dell'utilizzo di indici. Una possibilità è quella di sfruttare degli R-tree in maniera similare al calcolo dei vicini, per calcolare una Skyline. L'idea è quella di trovare buone tuple rapidamente affinché possano essere potati rami interi dell'albero R-tree, che sono dominati da queste tuple.
Possono essere utilizzati indici ordinati monodimensionali (ad esempio, i B-tree) adattando il primo passo dell'algoritmo A_0 di Fagin. Questo ed altri algoritmi possono essere generalizzati per Skyline con più di due dimensioni. Sarà necessario avere un indice per ogni dimensione e l'algoritmo in questione si arresta quando viene trovata una corrispondenza per tutti gli indici. Gli indici sono particolarmente utili per Skyline combinate e query Top N.

3.4 Skyline e Join

L'operatore Skyline è logicamente applicato dopo un join e un group-by. Tuttavia, se un join (o group-by) non è riduttivo, l'operatore Skyline può anche essere applicato prima del join. L'applicazione di un operatore Skyline prima di un join è interessante perché riduce la dimensione del risultato e, quindi, un operatore Skyline inserito prima di un join lo rende più conveniente. Anche i join non riduttivi tendono ad aumentare la dimensione del risultato (le tuple si allargano) cosicché l'operatore Skyline stesso diventi più economico se è effettuato attraverso il join. Inoltre, inserire l'operatore Skyline attraverso un join potrebbe rendere possibile utilizzare un indice per calcolare la Skyline.
Estendere l'ottimizzatore delle query a prendere in considerazione una Skyline push-down risulta essere piuttosto semplice; in sostanza, le stesse misure per query Top N possono essere applicate.
Viene a tutti gli effetti applicata una Early Skyline con quasi nessuna ulteriore spesa generale, come parte di un sort-merge join.

3.5 Calcolo della Skyline mediante l'operazione di Winnow

L'operatore Winnow può essere applicato ad una vista SQL che esprime alcuni costrutti (ad esclusione della Skyline) in una query Skyline. Ciò consente quindi di esprimere una Skyline facendo uso dell'operazione di Winnow.
Per esempio, la query Skyline
SELECT * FROM R
SKYLINE OF A DIFF, B MAX, C MIN
è equivalente a $\omega_C(R)$ dove
$(x, y, z) \succ_C (x', y', z') \equiv x = x' \wedge y \geq y' \wedge z \leq z' \wedge (y > y' \vee z < z')$.
In considerazione di quanto appena esplicitato, le leggi algebriche che caratterizzano le proprietà dell'operatore Winnow sono applicabili anche all'operatore Skyline.

3.6 Considerazioni conclusive

L'operazione Skyline è utile per una serie di applicazioni di database, incluso il supporto decisionale e la visualizzazione. La clausola SKYLINE OF è stata presa in considerazione come una semplice estensione di SELECT di SQL; sono stati presentati e valutati algoritmi alternativi per calcolare la Skyline di un insieme di punti, estendendo un sistema di database già esistente. È stato visto come sia possibile implementare l'operazione Skyline attraverso l'impiego di indici e sono state spiegate le modalità con cui l'operazione Skyline riesce ad interagire con un altro operatore di query, il join.
L'algoritmo nested-loops iniziale è stato esteso affinché si potesse lavorare meglio nell'ambito dei database. Le dimostrazioni teoriche e i calcoli del costo computazionale indicano che, per ottenere un implementazione conveniente, un sistema di database dovrebbe applicare un algoritmo divide-et-impera per calcolare la Skyline solo nei casi peggiori, mentre in tutte le altre situazioni dovrebbe usare l'algoritmo block-nested-loops che, proposto in alternativa all'algoritmo originale, si è rivelato decisamente migliore in parecchi casi, consentendo inoltre di combinare l'operatore Skyline con altre operazioni di database.

Esistono altri algoritmi paralleli, studiati per la risoluzione del massimo problema vettoriale (ovvero il calcolo della Skyline ancheper i casi in cui il numero di dimensioni sia superiore a due); malgrado ciò, nessuno di questi approcci è stato applicato al contesto database.
Nella letteratura "teorica" sono stati studiati alcuni problemi correlati come il calcolo dell'inviluppo convesso di un insieme di dati, ma neppure questo problema è stato affrontato nel contesto dei database, questo perché gli algoritmi noti per la sua risoluzione hanno un costo computazionale sempre superiore a quello degli algoritmi per il calcolo della Skyline.
È presumibile che gli utenti che eseguano un'operazione di Skyline in diverse applicazioni, lo facciano per ottenere un quadro più preciso della situazione in questione e dunque per applicare successivamente una query Top N, con l'intnento di arrivare a risultati più specifici. Per applicazioni differenti, l'operatore Skyline è indispensabile e la ricerca del vicino più prossimo e Top N non aiutano.
Alcune questioni non sono ancora state risolte, una di queste è quella di trovare i migliori algoritmi per calcolare la Skyline con un sistema di database parallelo affinché si possa studiare l'utilizzo di indici in maniera più dettagliata; un'altra è quella di aiutare l'ottimizzatore di query nel decidere se deve essere usato un algoritmo block-nested-loop o un algoritmo divide et impera, per fare ciò è necessario studiare quale tipo di statistiche debbano essere mantenute per stimare le dimensioni di una Skyline.

Capitolo 4

Preferenze contestuali nei database

4.1 Modello a preferenze contestuali

In questo capitolo, l'attenzione si focalizza sul miglioramento delle preferenze attraverso il contesto che può esprimere sia condizioni relative ai dati memorizzati nel database, sia a riguardo di situazioni esterne al database; sono stati delineati modelli per esprimere entrambi i tipi di preferenze. Le preferenze contestuali assumono la forma seguente: l'elemento i_1 è preferito all'elemento i_2 nel contesto di X. Le varie fonti forniscono preferenze in modo indipendente e quindi queste possono contenere cicli e contraddizioni. Vengono dunque prima riconciliate tutte le preferenze accumulate dalle fonti e successivamente utilizzate per creare un ordinamento con priorità delle tuple in una fase di pre-elaborazione. Solo alcuni ordini rappresentativi sono salvati, ciascuno corrispondente ad un insieme di contesti. Questi ordini (e i relativi contesti associati) sono utilizzati in fase d'interrogazione per fornire rapidamente risposte ordinate. Quindi, data una query dell'utente e il contesto circostante, viene analizzato il problema della selezione delle preferenze relative per personalizzare la query.
Come detto in precedenza, il contesto è un termine generico usato in diversi settori. Si distinguono due tipi generali di contesto:

- il contesto interno, che cattura condizioni che coinvolgono gli elementi di dati memorizzati nel database per il quale le preferenze sono espresse;

- il contesto esterno, che riguarda esclusivamente condizioni esterne al database.

Tipicamente, una preferenza contestuale ha due parti: una parte di preferenza che specifica la preferenza stessa e una parte di contesto che specifica le condizioni a cui la preferenza data deve ottemperare. È stato elaborato un modello multidimensionale per esprimere condizioni riguardanti il contesto esterno che permettano l'espressione di preferenze contestuali a vari livelli di dettaglio. Dato un insieme di preferenze contestuali e una query, il modello permette di determinare quali preferenze siano le più rilevanti per la query. Questo problema è chiamato risoluzione del contesto ed il quadro risolutivo per affrontarlo comprende le seguenti procedure:

1. raggruppamento delle preferenze contestuali espresse nella forma $i_1 \succ i_2 \mid X$ che significano che l'elemento i_1 è preferito all'elemento i_2 nel contesto X;

2. tecniche per l'utilizzo di queste preferenze per generare alcuni ordinamenti rappresentativi delle tuple e dei loro contesti associati;

3. tecniche per l'utilizzo di questi pre-ordinamenti per fornire rapidamente risposte ordinate ai quesiti che prendono in considerazione la condizione espressa come parte della query.

Dato uno schema di database R e un insieme di preferenze contestuali P, occorre incorporare la conoscenza di P nel meccanismo di risoluzione delle query. Più nello specifico, per una determinata query, le preferenze contestuali che sono ad essa collegate sono prese in considerazione al fine di fornire risultati ordinati.

4.1.1 Definizione di preferenza contestuale

Le preferenze esprimono interessi degli utenti, simpatie o antipatie. Nella sua forma più generale, una specifica preferenza contestuale CP = (C, P) è una coppia, in cui la parte contestuale C esprime le condizioni alle quali la preferenza specificata da P deve attenersi.
Considerando una relazione di database r con n tuple, $r = \{t_1, \ldots, t_N\}$ con schema $R(A_1, \ldots, A_d)$, $Dom(A_i)$ rappresenta il dominio attivo dell'attributo A_i. Le preferenze contestuali sono della forma $\{A_i = a_{i1} \succ A_i = a_{i2} \mid$

X$\}$, dove X è $\wedge_{j \in l}(A_j = a_j)$, con a_{i1}, $a_{i2} \in Dom(A_i)$, $l \subseteq \{1, \ldots, d\}$ e $a_j \in Dom(A_j)$. Il lato sinistro di una preferenza contestuale specifica la scelta mentre il lato destro è il contesto. La semantica di questa preferenza contestuale in termini di tuple di un database è la seguente: tutte le tuple $t \in r$ tali che $t.A_i = a_{i1}$ e $\forall\, j \in l$: $t'.A_j = a_j$ sono preferibili alle tuple $t' \in r$ per le quali $t'.A_i = a_{i2}$ e $\forall\, j \in l$: $t'.A_j = a_j$. Si può notare che le preferenze contestuali sono orientate all'insieme; una sola preferenza può specificare scelte tra un gran numero di tuple.

4.1.2 Modello multidimensionale

Viene definito ambiente contesto CE, un insieme di parametri di contesto n: CE = $\{C_1, \ldots, C_n\}$, dove ogni parametro C_i, con $1 \leq i \leq n$, acquisisce le informazioni che non fanno parte del database, ad esempio la posizione dell'utente o il tempo corrente.
In particolare, ogni parametro di contesto ha più livelli organizzati in uno schema gerarchico. Sia C un parametro di contesto con $m > 1$ livelli, L_i, con $1 \leq i \leq m$. Il suo schema gerarchico viene così indicato: $L = (L_1, \ldots, L_m)$. L_1 è chiamato livello più basso o più dettagliato dello schema gerarchico mentre L_m è quello superiore o più generale. Tra i livelli di L viene definito un ordine totale tale che $L_1 \prec \ldots \prec L_m$ e tra due differenti livelli L_i, L_j per rappresentare che $L_i \prec L_j$ oppure che $L_i = L_j$, viene utilizzata la notazione $L_i \preceq L_j$. Ogni livello L_j, con $1 \leq j \leq m$, è associato ad un dominio di valori, così definito: $dom_{L_j}(C)$. Per tutti i parametri, il loro livello superiore ha un singolo valore "All", cioè $dom_{L_m}(C) = \{All\}$.
Una gerarchia concettuale è un'istanza di uno schema gerarchico, in cui la gerarchia concettuale di un parametro di contesto C con m livelli è rappresentata da un albero con m livelli; i nodi ad ogni livello j, con $1 \leq j \leq m$, rappresentano valori $dom_{L_j}(C)$.
Uno stato di contesto cs di un ambiente CE = $\{C_1, \ldots, C_n\}$ è una n-upla $(c_1, \ldots. c_n)$, dove $c_i \in dom(C_i)$, con $1 \leq i \leq n$. L'insieme di tutti i possibili stati di contesto, chiamato mondo CW, è il prodotto cartesiano dei domini dei parametri di contesto: $CW = dom(C_1) \times \ldots \times dom(C_n)$.

Descrittore di contesto multi-parametro

La specificazione di contesto C di una preferenza contestuale CP = (C, P) precisa un insieme di stati di contesto o situazioni in cui P va rispettata. C è un descrittore di contesto multi-parametro.
Un descrittore con singolo parametro di contesto scod(C) di un parametro di contesto C è un'espressione della forma $C \in \{v_1, \ldots, v_l\}$, dove $v_k \in$ dom(C), con $1 \leq k \leq l$. Il tutto viene esplicitato con la notazione

$$\text{Context}(\text{scod}(C_i)) = \{v_1, \ldots, v_l\}.$$

Un descrittore di contesto multi-parametro cod è l'espressione della forma $\bigwedge_{j=0}^{k} \text{scod}(C_{i_j})$, con $1 \leq k \leq n$, dove $i_j \in \{1, \ldots, n\}$, $\text{scod}(C_{i_j})$ è un descrittore con singolo parametro di contesto per C_{i_j} e vi è al massimo un solo descrittore con singolo parametro di contesto per ogni C_{i_j}.
Un descrittore di contesto multi-parametro specifica un insieme di stati di contesto calcolati prendendo il prodotto cartesiano dei contesti di tutti i descrittori di contesto con un singolo parametro che appaiono nel descrittore. Se un descrittore di contesto multi-parametro non contiene descrittori per un determinato parametro di contesto, si presume che i valori del parametro di contesto assente siano irrilevanti. Precisamente, se un parametro di contesto C_i manca da un descrittore di contesto multi-parametro, si assume la condizione di default $C_i \in \{\text{All}\}$.Viene chiamato "profilo" l'insieme di tutte le preferenze contestuali (Cod_i, P_i), disponibili per un'applicazione. Il contesto Context(Pr) di un profilo Pr è l'unione dei contesti di tutti i descrittori di contesto che appaiono in Pr,
ovvero $\text{Context}(Pr) = \cup_i \text{Context}(\text{cod}_i)$, dove $(\text{cod}_i, P_i) \in P$.

4.2 Selezione di preferenze contestuali

Con personalizzazione delle query ci si riferisce al processo che prevede l'utilizzo delle preferenze per l'adattamento dei risultati di una query basata sugli interessi degli utenti. L'adattamento può essere realizzato mediante la classificazione dei risultati, la selezione di sottoinsiemi rappresentativi dei risultati o comunque utilizzando diverse presentazioni visive dei risultati. Le preferenze possono essere impiegate per personalizzare i risultati dell'interrogazione sia in una pre-elaborazione che in fase di post-elaborazione. Come passo di pre-elaborazione, le preferenze dell'utente vengono utilizzate

per riscrivere o la query originale precisandone il significato, per esempio con l'aggiunta di condizioni di selezione per incorporare le preferenze. Tali condizioni di selezione sono generalmente considerate vincoli morbidi in contrasto con i vincoli rigidi espressi dalle condizioni di selezione presenti nella query originale. Come passo di post-elaborazione, le preferenze dell'utente vengono utilizzate dopo l'esecuzione della query originale per personalizzare i suoi risultati, ad esempio attraverso una rielaborazione della classifica oppure mediante un filtraggio degli esiti ottenuti.
Indipendentemente da quando le preferenze verranno utilizzate durante l'elaborazione della query, uno dei problemi più comuni è quello di capire quali preferenze nel profilo utente siano da utilizzare per personalizzare ogni query specifica. Il numero di preferenze da utilizzare è fondamentale per il successo della personalizzazione, poiché selezionando troppe preferenze si può causare a un eccesso di specializzazione, mentre la selezione di troppe poche preferenze non può essere efficace. La selezione delle preferenze è basata sulla rilevanza tra preferenze e la query.
Data una query q, per indicare il descrittore di contesto multi-parametro che specifica il contesto associato con q viene usato il codice cod^q dove Context(q) = Context(cod^q). Solitamente, il contesto implicitamente associato ad una query corrisponde al contesto corrente, cioè il contesto dall'utente al momento della presentazione della query. Oltre a questo contesto implicito, è chiaramente previsto ceh le query possano essere ulteriormente precisate in maniera esplicita con descrittori di contesto multi-parametro da parte degli utenti che le hanno poste. Il contesto associato ad una query può corrispondere ad un singolo stato di contesto, in cui ogni parametro di contesto assume un valore specifico dal suo dominio più dettagliato. In altri casi, può essere possibile specificare il contesto della query utilizzando solo valori approssimativi; in queste circostanze, un parametro di contesto può assumere sia un unico valore da un livello superiore della gerarchia, sia anche più di un valore.
Se esiste una preferenza nel profilo il cui contesto comprende uno stato di contesto cs tale che cs = cs^q (ciò è definito "corrispondenza esatta"), allora questa è una preferenza rilevante che può essere utilizzata per personalizzare q.
Dato un profilo Pr e uno stato del contesto cs^1, uno stato $cs^2 \in$ Context(Pr) è una copertura stretta di cs^1 in Pr, se e solo se
cs^2 copre cs^1 e $\neg\exists$ $cs^3 \in$ Context(Pr), $cs^3 \neq cs^2$, tale che cs^2 copra cs^3 e cs^3

copra cs^1.

Risoluzione del contesto

Con risoluzione del contesto si intende il processo di selezione delle preferenze appropriate da un profilo in base al contesto.
Dato un profilo Pr e una query contestuale q, un set di stati di contesto RS, RS $\subseteq$ Context(Pr), è chiamato risoluzione del contesto fissato per q se per ogni stato di contesto $cs^q \in$ Context(q), esiste almeno uno stato di contesto cs in RS tale che cs è una stretta copertura di cs^q in Pr e cs $\in$ RS solo se vi è un $cs^q \in$ Context(q) per cui cs è una stretta copertura in Pr. Ci può essere più di una stretta copertura per una query contestuale.
La definizione della distanza tra due elementi si basa sulla distanza percorsa tra i loro valori nella gerarchia corrispondente e sulla dimensione relativa dei loro domini. Le distanze possono essere utilizzate per selezionare le coperture strette di ciascuna query, creando set di risoluzione di contesto più piccoli. Le distanze possono essere utilizzate anche per includere più di una stretta copertura per ogni query di contesto nella risoluzione di contesto; ciò fornisce un mezzo importante per controllare il grado di personalizzazione. Inoltre è doveroso ricordare che per una query q e profilo Pr, potrebbe non esserci alcuna stretta copertura e quindi nessun set di risoluzione di contesto. In questo caso, le vie percorribili sono due:

- q viene eseguita come una query normale, vale a dire, senza l'utilizzo di alcuna preferenza;

- l'esigenza di rilevanza viene "distesa", cioè si prendono in considerazione gli stati di contesto che sono simili alla query in questione, anche se non dovessero essere necessariamente coperture strette di quest'ultima.

Gli studi di usabilità hanno indicato che, nella maggior parte dei casi, utilizzare esclusivamente una copertura stretta produce risultati leggermente più soddisfacenti rispetto ad utilizzarne più di una, mentre l'uso di preferenze con contesto "rilassato" o "disteso" è comunque migliore rispetto al non usarne nessuno.

4.3 Indici per preferenze contestuali

Viene ora approfondito il calcolo efficiente di insiemi per la risoluzione del contesto. La scansione in rapida sequenza di tutti gli stati di tutte le preferenze in Pr è una valida via da percorrere per raggiungere questo obiettivo. Per migliorare i tempi di risposta, l'indicizzazione delle preferenze in Pr viene calcolata sulla base degli stati di contesto in Context(Pr). A tal fine, sono state introdotte due strutture dati:

- il grafo delle preferenze;
- l'albero del profilo.

4.3.1 Il grafo delle preferenze

Il grafo delle preferenze sfrutta le relazioni di copertura tra gli stati di contesto. Il grafo delle preferenze $PG_{Pr} = (V_{Pr}, E_{Pr})$ di un profilo Pr è un grafo orientato aciclico tale che vi sia un nodo $v \in V_{Pr}$ per ogni stato di contesto $cs \in Context(Pr)$ ed un ramo $(v_i , v_j) \in EPr$, se lo stato di contesto che corrisponde a v_i è una copertura stretta dello stato di contesto che corrisponde a v_j.
Dato un contesto di stato cs e un profilo Pr, gli stati di contesto in Context(Pr) che sono coperture strette di cs sono posizionate attraverso una scansione top-down del grafo di preferenza PG_{Pr} che parte dai nodi V_{Pr} senza archi entranti. La ricerca si ferma a un nodo v del grafo, se v è un nodo foglia o lo stato di contesto v non copre cs.
Lo stato di contesto di un nodo è incluso nel risultato se:

- è un nodo foglia cui contesto stato copre cs oppure
- gli stati di contesto di tutti i suoi figli non coprono cs.

4.3.2 L'albero del profilo

L'albero del profilo esplora prefissi comuni di stati di contesto nel profilo. Un valore $c \in dom(C_i)$ appare in un contesto di stato $cs = (c_1, \ldots, c_i, \ldots, c_n)$ se $c_i = c$. Il prefisso della lunghezza k $(c_1, \ldots, c_k, \ldots, c_n)$ corrisponde a $(c_1, \ldots, c_k)$. Un albero del profilo ha n + 1 livelli. Ognuno dei primi n livelli corrisponde ad uno dei parametri di contesto. La notazione C_{t_i} viene usata

per indicare il parametro mappato a livello i, $t_i \in \{1, \ldots, n\}$. L'ultimo livello, cioè n + 1, include i nodi foglia.
L'albero del profilo T_{Pr} di un profilo Pr è un albero con n + 1 livelli costruito secondo queste disposizioni:

1. Ogni nodo interno al livello k, con $1 \leq k \leq n$, contiene una serie di celle della forma [val; pt] dove val $\in dom(C_{t_k})$ e pt è un puntatore ad un nodo del successivo livello dell'albero, cioè, il livello k + 1.

2. Ogni nodo foglia al livello n + 1 contiene una o più preferenze.

3. Al primo livello dell'albero, c'è un singolo nodo radice che contiene una cella [c, p] per ogni valore di $c \in dom(C_{t_1})$ che appare in uno stato di contesto cs $\in$ Context(Pr).

4. A livello k, con $1 < k \leq n$, c'è un nodo, chiamato nodo v_o, per ciascuna cella $[c_o, p_o]$ di ogni nodo a livello k - 1. Il nodo v_o comprende una cella [c, p] per ogni valore di $c \in C_{t_k}$ che appare in uno stato di contesto cs tale che cs $\in$ Context(Pr) e c_o appare anche in cs. Il corrispondente puntatore p_o punta a v_o.

5. C'è un nodo foglia, chiamato nodo v_l, per ogni cella [c, p] di un nodo al livello n. Il puntatore p punta a questo nodo foglia. Sia $cs = (C_{t_1}, \ldots, c_{t_n})$ uno stato di contesto costituito da valori delle celle nel percorso dal nodo radice al nodo v_l. Il nodo foglia v_l contiene le preferenze associate con lo stato di contesto $cs = (c_1, \ldots, c_n)$.

L'albero del profilo T_{Pr} sostiene l'efficiente controllo di uno stato di contesto $cs = (c_1, \ldots, c_n)$, già ad ogni livello i, così serve solamente seguire il puntatore della cella con il valore di c_{t_i}. Se cs esiste in Pr, allora viene eseguito un unico processo. Se una corrispondenza esatta per cs non esiste in Pr, gli stati di contesto in Context(Pr) che coprono cs sono posizionati mediante un procedimento top-down, in ampiezza trasversale di T_{Pr}. Questi stati di contesto devono essere trattati ulteriormente per identificare quali di questi sono coperture strette. In particolare, a ogni livello i dell'albero, tutti i percorsi di lunghezza i, il cui stato di contesto è uguale o copre il prefisso $(c_{t_1}, \ldots, c_{t_i})$ dello stato di contesto di input cs, sono mantenuti come candidati. L'albero del profilo è tendenzialmente più piccolo del grafo delle preferenze in quanto sfrutta ripetizioni di prefissi di stati di contesto. Con l'albero

del profilo, le corrispondenze esatte si risolvono con un semplice attraversamento radice-foglia, mentre per le corrispondenze non esatte, i percorsi candidati multipli sono mantenuti. Il grafo delle preferenze si sviluppa in modo simile per entrambe le corrispondenze, esatte o meno che siano. È doveroso sottolineare che gli alberi del profilo restituiscono coperture dello stato di contesto, quindi, per calcolare le coperture strette, è richiesto il passaggio aggiuntivo di ordinamento di questi stati di contesto in base alle loro distanze allo stato di contesto della query. È stato mostrato l'impiego del grafo delle preferenze e dell'albero del profilo per individuare le coperture strette di un singolo stato di contesto. Gli algoritmi per localizzare coperture strette di più di un solo stato di contesto possono essere progettati rappresentando stati contesto in Context(q), utilizzando un grafo delle preferenze o un albero del profilo.

4.4 Il problema della selezione del rango

Si prende ora in considerazione il problema della selezione del rango. Data una query selettiva q sulla relazione $r = \{t_1, \ldots, t_n\}$ con schema relazionale $R(A_1, \ldots, A_d)$ e posto che il dominio attivo dell'attributo A_i è $Dom(A_i)$, la query è espressa nella forma $q = \sigma_{\wedge_{j \in \{1, \ldots, u\}}} (A_{i_j} = a_j)$, dove $a_j \in Dom(A_{i_j})$. Sia $q(r) \subseteq r$ il sottoinsieme di tuple in r che sono la risoluzione di q. Il problema della selezione del rango viene definito in questa maniera:
dato un insieme di preferenze $P = \{P_{X_1}, \ldots, P_{X_m}\}$ e una query di selezione q, il problema della selezione rango chiede una permutazione di τ di q(r) tale che
$\tau = \arg \max_{\tau'} \sum_{i=1}^{m} sim(q, X_i) AGREE(\tau', P_{X_i})$,
dove $AGREE(\tau, P_X) = \sum_{(t,t'):\tau(t)<\tau(t')} EFF - P(t, t', P_X)$.

La risoluzione di questo problema sta nel trovare la permutazione τ dell'insieme di tuple presenti in q(r) che armonizzi il maggior numero possibile di preferenze in ingresso. Inoltre, il grado di accordo con una classe di preferenze è ponderato in base alla somiglianza tra i contesti di tali preferenze e la query q.
Per avere una definizione adeguata di questo problema è necessario quantificare la somiglianza tra una query e un contesto ed è perciò doveroso formare le rappresentazioni vettoriali di un contesto e di una query selettiva.

Dato che ciò presume un elevato costo computazionale, occorre pensare ad algoritmi di approssimazione per risolverlo. Ognuno di questi algoritmi richiederebbe di prendere visione almeno una volta di tutte le preferenze per ogni coppia di tuple. Ciò corrisponderebbe ad un tempo quadratico rispetto al numero di tuple presenti nel database. Questa complessità temporale è inaccettabile per le operazioni su grandi basi di dati. La soluzione migliora sta, dunque, nell'adottare un approccio ibrido che consiste in alcune preelaborazioni, eseguite offline, seguite dall'elaborazione online delle query. I risultati dei calcoli offline rendono l'elaborazione on-line delle query estremamente veloce.
La fase offline è sostanzialmente riassumibile in due fasi:

1. Per ogni classe di preferenze, si costruisce un unico ordine delle n tuple in ingresso. Cioè, per ogni preferenza P_{X_i} si trova una permutazione τ_i delle tuple in r tale che
$$\tau_i = \arg \max_{\tau'_i} \text{AGREE}(\tau'_i, P_{X_i}).$$
L'output di questa fase è un insieme m coppie di [contesto, ordine] nella forma $[X_i, \tau_i]$, dove X_i è il contesto delle preferenze in P_{X_i} e τ_i è l'ordinamento delle tuple in r. Fatto l'ordinamento delle tuple, ogni tupla t ha un punteggio associato con la posizione di t in ogni ordine τ_i. Così, il punteggio delle tuple t in τ_i che corrisponde al contesto X_i è:
$$s(t \mid X_i) = n - \tau_i(t) + 1,$$
dove $\tau_i(t)$ rappresenta la posizione di ogni tupla t nell'ordine τ_i. Le tuple in testa all'ordine hanno punteggi più alti, mentre quelli più bassi sono riservati per le tuple che sono verso la coda dell'ordine.

2. Viene ridotto il numero di coppie [contesto, ordine] da mantenere. Cioè, dato il set delle m coppie iniziali $[X_i, \tau_i]$, vengono trovate le permutazioni rappresentative $\overline{\tau}_1, \ldots, \overline{\tau}_l$ Con $l < m$. Queste permutazioni dividono lo spazio delle m coppie iniziali [contesto, ordine] in l gruppi. Ogni gruppo i è caratterizzato dall'ordine $\overline{\tau}_i$ ed una disgiunzione dei contesti $\overline{X}_i \subseteq \{X_1, \ldots, X_m\}$ tale che, per ogni $X_j \in \overline{X}_i$, l'ordine $\overline{\tau}_i$ è una buona rappresentazione per l'ordine iniziale τ_j. Il punteggio di tuple t in $[\overline{X}_i, \overline{\tau}_i]$ ora è dato da:
$$s(t \mid \overline{X}_i) = n - \overline{\tau}_i(t) + 1.$$

Una volta eseguita la fase offline, l'elaborazione delle query avviene in un tempo computazionalmente poco costoso. L'unica mansione da eseguire online è quella di combinare opportunamente le classifiche delle tuple formate a priori da restituire in risposta alla query data.

4.5 Costruzione di ordini dalle preferenze

Il problema dell'ordinamento è difficile quanto il problema del massimo sottografo aciclico, noto per essere NP-difficile (in inglese NP-hard, da nondeterministic polynomial-time hard, problema non deterministico in tempo polinomiale), così definito: dato in ingresso un grafo G pesato e orientato, trovare il sottografo di G con peso massimo che sia aciclico. Il collegamento tra il problema del massimo sottografo aciclico e il problema dell'ordinamento diventa più chiaro tramite la rappresentazione X - grafo delle preferenze. Nel problema dell'ordinamento, l'obiettivo, per un dato X - grafo delle preferenze $G_X(r, E_X)$, è quello di trovare una permutazione τ di $r = \{t_1, t_2, \ldots, t_n\}$ che predisponga un sottografo aciclico $G_\tau(V, E_\tau)$ di G_X, tale che la somma dei pesi dei rami in E_τ sia massimizzata. Questo corrisponde esattamente al problema del massimo sottografo aciclico sul grafo delle preferenze G_X. La conseguenza di ciò è che il problema dell'ordinamento è NP-completo, ovvero rientra nella categoria dei problemi più difficili della classe NP poiché, se si trovasse un algoritmo in grado di risolvere "velocemente" (inteso come tempo polinomiale) un qualsiasi problema NP-completo, allora si potrebbe usarlo per risolvere "velocemente" ogni problema in NP. Ciò implica che la soluzione del problema dell'ordinamento può essere solamente approssimata.
Esistono sostanzialmente tre algoritmi che vengono applicati per il problema dell'ordinamento:

- algoritmo pick-perm;
- algoritmo greedy-order;
- algoritmo MC-order.

I primi due hanno un rapporto di approssimazione piuttosto limitato, mentre il terzo è un procedimento euristico che risulta essere estremamente performante nella pratica.

4.6 Considerazioni conclusive

In questo capitolo, è stato brevemente presentato come sia possibile annotare preferenze con informazioni circostanziali. Il contesto è modellato utilizzando una serie di parametri che prendono valori da domini gerarchici, quindi, consentendo diversi livelli di astrazione per i dati acquisiti. Le preferenze sono precisate con descrittori di contesto che specificano gli stati di contesto in cui una preferenza deve essere rispettata. Analogamente, ogni query è in relazione con un insieme di stati di contesto. È stato preso in considerazione il problema di individuare le preferenze i cui stati di contesto sono maggiormente simili a quelli di una determinata query (risoluzione del contesto). Sono state analizzate le due strutture dati impiegate per risolvere il problema del contesto, vale a dire il grafo delle preferenze e l'albero del profilo, che consentono una rappresentazione compatta di preferenze contestuali.
Nell'approccio quantitativo, le preferenze sono espresse utilizzando funzioni che assegnano un punteggio di interesse per informazioni specifiche. Nel approccio qualitativo, le preferenze tra pezzi di informazione sono specificate direttamente, in genere utilizzando relazioni di preferenza binarie.
È stato mostrato come sia possibile sfruttare le preferenze dell'utente per calcolare alcuni ordinamenti di tuple rappresentativi prima ancora che giunga un'interrogazione, per poi poterli usare per fornire rapidamente risposte ordinate alle query.

Capitolo 5

Interrogazioni con preferenze

5.1 Obiettivi dell'analisi

Sostanzialmente, un'interrogazione ad un database può essere identificata da una condizione che i dati ricercati devono soddisfare. Tale condizione ha spesso una struttura complessa e comprende alcune condizioni atomiche combinate usando differenti operatori logici. Sono state proposte varie estensioni nell'ambito della cosiddetta interrogazione flessibile. Per esempio, una di queste proposte è l'utilizzo di predicati fuzzy che modellano termini linguistici in condizioni; un'altra è l'assegnazione di pesi d'importanza a particolari parti della condizione. L'obiettivo generale è sempre quello di fornire una maggiore flessibilità nell'esprimere le esigenze degli utenti per quanto riguarda i dati richiesti.

5.2 Query bipolari

È stato introdotto il concetto di interrogazione bipolare. Una coppia di condizioni corrisponde rispettivamente all'insieme respinto e all'insieme desiderato di tuple. Questo è il motivo per il cui ci si riferisce a queste query come bipolari. Una definizione equivalente prende in considerazione una coppia di condizioni richieste e preferite. Il primo elemento della coppia corrisponde al complemento del set di tuple respinti. L'interpretazione generale della query bipolare è che le tuple cercate devono soddisfare la condizione richiesta incondizionatamente, mentre la condizione preferita è di

importanza secondaria. L'interrelazione tra questi due tipi di condizioni può essere intesa in vari modi e porta a diverse interpretazioni delle query bipolari.
T = $\{t_j\}$ è un insieme di tuple da interrogare; C(·) e P (·) sono predicati corrispondenti, rispettivamente, alle condizioni richieste e preferite. Questi predicati sono identificati con insiemi fuzzy, ovvero insiemi caratterizzati da una funzione di grado di appartenenza, che mappa gli elementi di un universo in un intervallo reale continuo [0, 1]:

- il valore 0 significa che l'elemento in questione non è per niente incluso nell'insieme fuzzy;
- il valore 1 indica che l'elemento è certamente incluso nell'insieme;
- i valori tra zero e uno indicano quale sia il grado di appartenenza dell'elemento all'insieme fuzzy in questione.

I primi due valori corrispondono alla teoria classica degli insiemi mentre gli altri rientrano in un'estensione della teoria stessa.
C(t) e P(t) indicano i valori della funzione di appartenenza.
È stata adottata un'interpretazione delle query bipolari che porta alla ordinamento lessicografico, $\preceq$, delle tuple, vale a dire,
$t_1 \preceq t_2 \iff (C(t_1) < C(t_2)) \vee ((C(t_1) = C(t_2)) \wedge (P(t_1) \leq P(t_2))$.
Il concetto di query con una simile interpretazione è stato originariamente proposto da Lacroix e Lavency nel 1987. Questa interpretazione può essere formalmente espressa dalla seguente descrizione del set di tuple richieste:
$\{t \in T: C(t) \wedge (\exists s(C(s) \wedge P(s)) \rightarrow P(t))\}$
Al fine di coprire anche predicati fuzzy C e P, viene espressa la caratteristica funzione di appartenenza a questo insieme, γ_1 (C, P, t, T), nella forma seguente:
γ_1 (C, P, t, T) = min(C(t), max(1 - $\underset{s \in T}{max}$ min(C(s), P(s)), P(t))).

5.3 L'operatore Winnow fuzzy e le query bipolari

Il concetto di query bipolare descritto in precedenza può essere anche interpretato come lo sviluppo dell'idea originale di Lacroix e Lavency nell'ottica della logica "fuzzy". Un'altra proposta, con un approccio più generale, prevede di rappresentare le preferenze direttamente mediante relazioni di preferenza, in quanto sono utilizzate nel dominio del processo decisionale.
Per rendere l'operatore Winnow in grado di esprimere query bipolari, con condizioni fuzzy C e P, la sua definizione deve essere estesa. In particolare, i seguenti aspetti devono essere presi in considerazione:

- R dovrebbe essere considerato come una relazione di preferenza fuzzy,
- dev'essere impiegata una controparte fuzzy di un concetto non dominante,
- l'insieme di tuple T dovrebbe essere considerato un insieme fuzzy.

La definizione di operatore Winnow fuzzy è stata fornita utilizzando l'approccio alle funzioni di scelta fuzzy. In questo approccio viene introdotta un'espressione del set di elementi non dominati con la seguente forma
$\mathrm{N(T,\,R)} = \mathrm{T} \cap \bigcap_{\mathrm{s}\in\mathrm{T}} \neg \mathrm{R^{-}(s)}$ dove $\mathrm{R^{-}(s)} = \{\mathrm{u}\colon \mathrm{R(s,\,u)}\}$.
N(T, R) denota l'insieme di elementi non dominati dell'insieme T rispetto alla relazione di preferenza R, mentre $\mathrm{R^{-}(s)}$ è un insieme di elementi dominati dall'elemento s rispetto alla relazione R; ¬A denota invece il complemento dell'insieme A.
Ora l'operatore Winnow viene così definito: $\omega_{\mathrm{R}}(\mathrm{T}) = \mathrm{N(T,\,R)}$.

5.3.1 Relazioni di preferenza fuzzy

Una relazione di preferenza fuzzy su un insieme di tuple T sullo schema $\{\mathrm{A_1}\colon \mathrm{D_1}, \ldots, \mathrm{A_n}\colon \mathrm{D_n}\}$, è una qualsiasi relazione binaria sfocata $\tilde{R}$,
con $\tilde{R} \in \mathrm{F}((\mathrm{D_1} \times \mathrm{D_2} \times \ldots \times \mathrm{D_n}) \times (\mathrm{D_1} \times \mathrm{D_2} \times \ldots \times \mathrm{D_n}))$, che si identifica con la sua funzione di appartenenza $\mu_{\tilde{R}}$.
Un insieme di elementi non dominati rispetto ad una relazione di preferenza fuzzy $\tilde{R}$ viene considerato un insieme fuzzy. Per definire questo insieme, è

stata utilizzata nuovamente l'espressione del set di elementi non dominati, assumendo operazioni standard di intersezione e complemento degli insiemi sfocati, ed una successiva generalizzazione di
$R^{-}(s) = \{u: R(s, u)\}$, ottenendo quindi $\mu_{\tilde{R}^{-}(s)}(u) = \mu_{\tilde{R}}(s, u)$.
Viene quindi definita la funzione di appartenenza dell'insieme fuzzy $\tilde{R}^{-}(s)$ di elementi dominati dall'elemento s rispetto alla relazione $\tilde{R}$.
Infine, per estendere quanto detto in precedenza al caso in cui l'insieme delle tuple T sia un insieme fuzzy, viene riscritta la formula come calcolo dei predicati $N(T, \tilde{R})(t) \Leftrightarrow T(t) \wedge \forall_{s \in T} \neg\tilde{R}^{-}(s)(t)$ dove particolari predicati fuzzy sono indicati con i simboli corrispondenti a quelli degli insiemi fuzzy. Dati un insieme fuzzy $\tilde{T}$ di tuple e una relazione di preferenza fuzzy $\tilde{R}$, entrambe definiti sullo stesso set di tuple T, l'operatore Winnow fuzzy è così definito: $\omega_{\tilde{R}}(\tilde{T})(t) = N(\tilde{T}, \tilde{R})(t)$, con $t \in T$, dove il predicato fuzzy $N(\tilde{T}, \tilde{R})(t) \Leftrightarrow \tilde{T}(t) \wedge \forall_s (\tilde{T}(s) \rightarrow \neg\tilde{R}(s, t))$.

Prendendo ora in considerazione una query bipolare definita da una coppia di condizioni fuzzy (C, P), è possibile identificare queste condizioni con predicati fuzzy, che saranno indicati con gli stessi simboli. Data la relazione di preferenza sfocata $\tilde{R}$, definita come $\tilde{R}(t, s) \Leftrightarrow P(t) \wedge \neg P(s)$, è possibile esprimere una query bipolare come una combinazione della selezione e dell'operatore winnow fuzzy $\omega_{\tilde{R}}(\sigma_C(T)) = N(C(T), \tilde{R})$ dove C(T) è un insieme fuzzy degli elementi di T che soddisfano la condizione C, ovvero, $\mu_{C(T)}(t) = C(t)$.
Utilizzando $N(\tilde{T}, \tilde{R})(t) \Leftrightarrow \tilde{T}(t) \wedge \forall_s (\tilde{T}(s) \rightarrow \neg\tilde{R}(s, t))$,
il predicato $N(C(T), \tilde{R})$ su $\omega_{\tilde{R}}(\sigma_C(T)) = N(C(T)$ viene definito
$N(C(T), \tilde{R})(t) \Leftrightarrow C(t) \wedge \forall_s (C(s) \rightarrow \neg(P(s) \wedge \neg P(t)))$.

5.4 L'operatore Winnow rilassato

Se una relazione di preferenza non è un ordine parziale stretto, potrebbe decadere la proprietà di non vuotezza dell'operatore Winnow oppure potrebbe non essere valida la considerazione che dice che se una relazione di preferenza $\succ_C$ su uno schema relazionale R è un ordine parziale stretto, allora per ogni istanza r di R ed ogni tupla $t \in r$, esiste un i, con $i \geq 1$, tale che $t \in \omega_C^i(r)$. Ciò può verificarsi, per esempio, se la relazione di preferenza è ottenuta tramite composizione. Tuttavia, anche in questo caso vi possono

essere disponibili forme più deboli degli operatori Winnow e Ranking.
Per far fronte a relazioni di preferenza che non sono ordini parziali stretti, è stata definita una nuova forma rilassata dell'operatore Winnow. Viene rilassato il vincolo di non riflessività, ma è preservato quello di transitività. In questa maniera, sono concesse anche violazioni di asimmetria.
Per definire l'operatore Winnow rilassato, è possibile notare che, finché la relazione di preferenza $\succ_C$ è transitiva, la si può utilizzare per definire un'altra relazione di preferenza $\succ_{C_>}$ che è un ordine parziale stretto:
$x \succ_{C_>} y \equiv x \succ_C y \wedge y \not\succ_C x$.
Se R è uno schema relazionale e $\succ_C$ una relazione di preferenza transitiva su R, allora l'operatore Winnow rilassata è scritto come $\psi_C(R)$ e per ogni istanza r di R, $\psi_C(r) = \omega_{C_>}(r)$.
Se R è uno schema relazionale e $\succ_C$ una relazione di preferenza transitiva su R, allora per ogni istanza r di R
$\psi_C(r) = \{t \in r \mid \forall\, t' \in r.\ t \succ_C t' \vee t' \not\succ_C t\}$.
Sempre per le premesse di prima, si ha che:

- per ogni istanza r di R, $\omega_C(r) \subseteq \psi_C(r)$;
- per ogni istanza non vuota r di R, $\psi_C(r)$ non è vuota.

Si può definire l'iterazione dell'operatore Winnow rilassato in modo simile a quella dell'operatore Winnow, che dice che, data una relazione di preferenza $\succ$ definita da una formula di preferenza C, l'ennesima iterazione dell'operatore Winnow ω_C in r è definita come:

- $\omega_C^1(r) = \omega_C(r)$,
- $\omega_C^{n+1}(r) = \omega_C(r \text{ - } \bigcup_{1 \leq i \leq n} \omega_C^i(r))$.

Quindi, se una relazione di preferenza $\succ_C$ su uno schema R è transitiva, allora per ogni istanza r di R e per ogni tupla $t \in r$, esiste un i, con $i \geq 1$, tale che $t \in \psi_{CC}^i(r)$.

5.5 Eliminazione delle occorrenze ridondanti di Winnow

Data un'istanza r di R, l'operatore ω_C è ridondante se $\omega_C(r) = r$. Se si considera la classe di tutti i casi di R, allora tale operatore è ridondante per ogni istanza se e solo se $\succ_C$ è una relazione vuota. Ciò avviene se e solo se C non può essere soddisfatta. Tuttavia, l'interesse è focalizzato solo nei casi che soddisfano un dato insieme di vincoli di integrità (ossia delle proprietà che devono essere soddisfatte dalle istanze di una base di dati), perciò, verrà effettuato il controllo sulla vuotezza della restrizione $\succ_C|_r$ per ogni istanza r che soddisfa il dato insieme di vincoli di integrità.
Dato un insieme di vincoli di integrità F, l'operatore ω_C è ridondante rispetto ad un insieme di vincoli di integrità F se $\forall\, r \in Sat(F)$, $\omega_C(r) = r$. ω_C è ridondante rispetto a un insieme di dipendenze funzionali F se e solo se la formula $\varphi_F(t_1, t_2) \wedge t_1 \succ_C t_2$ non può essere soddisfatta. Ciò dimostra che il controllo per la ridondanza rispetto ad un insieme di dipendenze funzionali F è un problema di soddisfacimento di vincoli.
Per stimare quanto sia difficile verificare la ridondanza rispetto ad un insieme di dipendenze funzionali F, si presume che la dimensione di una formula di preferenza C, su una relazione R, nella forma normale disgiunta sia caratterizzata da due parametri: larghezza(C) (cioè il numero di disgiunzioni in C) e portata(C) (cioè il numero massimo di congiunzioni in una disgiunzione di C); se $C = D_1 \vee \ldots \vee D_m$ e ogni $D_i = C_{i,\,1} \wedge \ldots\ C_{i,\,k_i}$, allora la larghezza(C) = m e la durata(C) = $\max\{k_1, \ldots, K_m\}$.
Partendo dai seguenti presupposti:

- la cardinalità dell'insieme di dipendenze funzionali F è |F| e la sua arietà (ovvero il numero degli argomenti o operandi che la funzione richiede) è al più k;
- la relazione di preferenza data viene definita utilizzando una formula di preferenza C contenente solo i vincoli atomici sullo stesso dominio e tali che la larghezza(C) $\leq$ m e la portata(C) $\leq$ n;
- la complessità computazionale del controllo della soddisfacibilità di una formula di preferenza con n congiunzioni corrisponde a O(T(n))

allora la complessità computazionale del controllo di ω_C per la ridondanza rispetto ad F è di $O(m\ k^{k|F|}\ T(\max(k|F|, n)))$.

5.6 Preferenze più espressive

L'operatore Winnow, quando viene impiegato insieme ad altri operatori dell'algebra relazionale, può esprimere problemi decisionali complessi che coinvolgono le preferenze. In questa sezione viene fatto un approfondimento a riguardo di ciò, analizzando in dettaglio vincoli d'integrità e preferenze estrinseche.

5.6.1 Vincoli d'integrità

Ci sono casi in cui si vuole imporre un vincolo sul risultato dell'operatore Winnow. È importante verificare se i vincoli d'integrità sono imposti prima o dopo l'applicazione dell'operatore Winnow. In generale, è necessario fare distinzione fra i vincoli locali e quelli globali. Un vincolo locale impone una condizione sui componenti di una singola tupla mentre un vincolo globale impone una condizione su un insieme di tuple. Per soddisfare un vincolo globale sul risultato dell'operatore Winnow, si dovrebbe costruire un sottoinsieme massimale del risultato, in modo tale che soddisfi il vincolo. Poiché in generale ci può essere più di uno di questi sottoinsiemi, la costruzione richiesta non può essere descritta mediante una singola query d'algebra relazionale. D'altra parte, i vincoli locali sono facilmente manipolabili, dato che possono essere espressi utilizzando la selezione.
Un veto esprime la proibizione della presenza di uno specifico insieme di valori negli elementi di risposta ad una query di preferenza e quindi può essere considerato come un vincolo locale. Per porre il veto su una tupla specifica $w = (a_1, \ldots, a_n)$ in una relzione S (che può essere definita da una query di preferenza) con n parametri d'ingresso, viene scritta la selezione $\sigma_{A_1 \neq a_1 \vee \ldots \vee A_n \neq a_n}(S)$.

5.6.2 Preferenze intrinseche e preferenze estrinseche a confronto

Le formule di preferenza intrinseche stabiliscono la relazione di preferenza tra due tuple esclusivamente sulla base dei valori che si verificano all'interno di tali tuple. Le formule di preferenza estrinseche, invece, possono riferirsi non solo a predicati di costruzione, ma anche ad altri costrutti, ad esempio, le relazioni dei database. In generale, le preferenze estrinseche possono

utilizzare una serie di criteri: proprietà delle relazioni da cui sono state selezionate le tuple, proprietà di altre relazionii, o confronti di valori aggregati, e non hanno nemmeno bisogno di essere definite utilizzando formule del primo ordine.
È possibile esprimere alcune preferenze estrinseche usando l'operatore Winnow insieme ad altri operatori dell'algebra relazionale, adoperando una strategia a più passi:

1. con l'impiego di una query relazionale, si uniscono tutte le informazioni rilevanti per la preferenza in un'unica relazione,

2. si applica l'operatore Winnow più appropriato per questa relazione,

3. si proiettano le colonne aggiuntive introdotte nel primo passaggio.

Prendendo in considerazione la situazione in cui qualsiasi tupla di una relazione R è preferita rispetto a qualsiasi tupla di una relazione S, si può notare che questa risulta essere effettivamente una preferenza estrinseca, poiché si sulla provenienza delle tuple e non sui loro valori. Tutto ciò può essere gestito contrassegnando le tuple con il nome della relazione appropriata (cosa facilmente implementabile nell'algebra relazionale o SQL) per poi definire la relazione di preferenza utilizzano le etichette che contraddistinguono una relazione dall'altra. Se vi è una tupla che appartiene sia ad R che ad S, allora la relazione di preferenza non sarà riflessiva.

5.7 Query con preferenze

È stata proposta un'estensione del calcolo relazionale in cui possono essere espresse le preferenze per le tuple che soddisfano determinate condizioni logiche. Per esempio, si potrebbe dire che tra le tuple di R che soddisfano Q, sono preferite quelle che soddisfano P_1, tra queste ultime sono preferite quelle che soddisfano P_2. Ciò viene precisato con la seguente procedura: si raccolgono le tuple che soddisfano $Q \wedge P_1 \wedge P_2$; se il risultato dovesse essere vuoto, si scelgono le tuple che soddisfano $Q \wedge P_1 \wedge \neg P_2$; se il risultato è nuovamente vuoto, scegliere le rimanenti tuple di R che soddisfano Q. Nell'algebra relazionale, il tutto viene esplicitato come $\omega_{C^*}(\sigma_Q(R))$ dove C^* è una formula di preferenza definita nel seguente modo:

1. si ottiene la formula C definendo una relazione di preferenza $\succ$
   $\bar{t}_1 \succ \bar{t}_2 \equiv \mathrm{P}_1(\bar{t}_1) \wedge \mathrm{P}_2(\bar{t}_1) \wedge \mathrm{P}_1(\bar{t}_2) \wedge \neg\mathrm{P}_2(\bar{t}_2) \vee \mathrm{P}_1(\bar{t}_1) \wedge \neg\mathrm{P}_2(\bar{t}_1)$
   $\wedge \neg\mathrm{P}_1(\bar{t}_2)$,

2. si trasforma C nella forma normale disgiunta per ottenere una formula di preferenza C' e

3. si applica una chiusura transitiva sul risultato in modo da ottenere una formula di preferenza C^* che definisce una relazione di preferenza transitiva $\succ^*$.

Quanto descritto fin'ora permette di specificare le preferenze utilizzando formule logiche con le loro definizione nell'algebra relazionale. Come risultato, possono essere risolti in modo semplice e pulito le query con preferenze e i problemi decisionali complessi che coinvolgono preferenze.
Chiaramente, tutto ciò si limita alle applicazioni che possono essere interamente descritte all'interno del modello relazionale dei dati. Alcuni casi che non rientrano in questo paradigma sono, ad esempio, le preferenze definite tra più insiemi di elementi, le preferenze eterogenee tra tuple di diverso tipo o con numero differente di parametri in ingresso oppure le preferenze che richiedono scelte non deterministiche.

Capitolo 6

Metodologia per la personalizzazione dei dati contestuali

6.1 Sistema Context-ADDICT

In questo capitolo, l'attenzione si concentra maggiormente su aspetti più pratici nell'utilizzo delle preferenze. L'utilizzo diffuso di dispositivi mobili, con limitazioni in termini di memoria, alimentazione e funzioni di connettività, richiede di minimizzare la quantità di dati da caricare sui dispositivi stessi, al fine di selezionare rapidamente solo le informazioni che veramente sono rilevanti per gli utenti: in un tale scenario, devono essere applicate le metodologie e le tecniche specifiche focalizzate sulla riduzione dei dati.
È stata proposta un'estensione per l'approccio di confezionamento dei dati di Context-ADDICT (ovvero "contesto a conoscenza dei dati di progettazione, integrazione, personalizzazione e confezionamento"), il cui scopo è quello di collegare ed integrare dinamicamente dati eterogenei per essere eventualmente memorizzati su piccoli dispositivi mobili. L'obiettivo principale di questa estensione è quello di personalizzare i dati dipendenti dal contesto ottenuti mediante la metodologia Context-ADDICT, consentendo all'utente di esprimere le preferenze che specificano a quali dati è più interessato (e a quali non) in ogni specifico contesto. Questo passaggio permette di imporre un ordine parziale sui dati e di caricare solo la parte preferita dei blocchi di dati.

L'accesso alle informazioni deve essere opportunamente personalizzato, di modo che l'utente non venga preso alla sprovvista dall'enorme quantità di dati disponibili. A causa di questa ricerca della personalizzazione dei dati, i criteri per il trattamento dei dati (sia off-line o che dinamici) svolgono un ruolo rilevante, in quanto giocano un ruolo centrale nel determinare quali parti di un database (in termini di tuple e attributi) dovrebbero essere mantenute e quali invece dovrebbero essere scartate.
Quando si inizia ad affrontare il problema della personalizzazione dei dati su dispositivi mobili, uno dei metodi più adottati per il confezionamento dei dati si basa sulla nozione di contesto. L'idea è quella di sfruttare la conoscenza della situazione in cui l'utente si trova per ridurre la quantità di informazioni memorizzate appunto su dispositivi mobili.
Uno dei sistemi che affrontano questo problema è, come detto in precedenza, Context-ADDICT, che fornisce un framework di riferimento per la selezione e l'integrazione delle informazioni pertinenti da consegnare sui dispositivi degli utenti, sulla base del loro contesto attuale. Anche se la metodologia usata in precedenza è semplice ed efficace nel filtraggio dei dati, nella riduzione delle ridondanze presenta alcuni limiti dei quali occorre tener conto:

1. Ogni possibile contesto è associato con le porzioni del database rilevanti per quel contesto. Questa associazione viene eseguita in fase di progettazione, fondamentalmente definendo gruppi di viste in termini di selezione, proiezione e operazioni di semi-join su un database relazionale globale.

2. Nessun modello dell'occupazione di memoria viene considerata, tanto meno viene fatta una stima della quantità di dati da memorizzare sul dispositivo dell'utente.

3. L'utente fornisce informazioni parziali sul suo contesto attuale, scegliendo un ruolo ed uno o più interessi definiti in fase di progettazione per l'applicazione di destinazione.

In questa maniera, l'approccio è rivolto a "classi" di utenti che condividono gli stessi contesti, ma non è personalizzato per ospitare classificazioni dei dati, effettuata sulla base dei gusti dei singoli utenti.
I limiti evidenziati indicano la necessità di meccanismi di personalizzazione più potenti e raffinati per quest'ambito. Un approccio semplice ma efficace

proposto in letteratura per eseguire una personalizzazione delle risposte delle query, personalizzato per ciascun utente, sfrutta le preferenze dell'utente. Ci sono molti approcci in letteratura che sfruttano le preferenze, ognuna delle metodologie analizzate è mirata per la personalizzazione delle risposte delle interrogazioni poste, non è stato affrontato il problema in uno scenario in cui le viste del contesto, che specificano insiemi di relazioni, sono definite in fase di progettazione e in cui le preferenze devono essere specificate su di esse.
Anche se la personalizzazione delle query è più impegnativa in quanto deve essere eseguita dinamicamente per ogni interrogazione dell'utente, la personalizzazione delle viste del contesto richiede di considerare più di una relazione e, di conseguenza, i relativi vincoli di integrità (ad esempio vincoli di chiave) tra le relazioni che formano la vista contestuale, sono obbligati ad essere conservati all'interno dei dati memorizzati sul dispositivo dell'utente.
L'estensione del sistema Context-ADDICT, con l'adozione di una prospettiva aggiuntiva per il confezionamento dei dati basati sulle preferenze contestuali, prevede:

- un modello preferenza, che possa eseguire una personalizzazione più avanzata rispetto alla letteratura, per esprimere interessi non solo su tuple ma anche sugli attributi;

- una metodologia che, partendo da viste dipendenti dal contesto (che definiscono insiemi di relazioni), calcola una classifica basata sulle preferenze ed effettua un filtraggio successivo sia sugli attributi che sulle tuple;

- le risultanti preferenze, basate su viste contestuali, si inseriscono nella memoria del dispositivo dell'utente e soddisfano i vincoli di chiave esterna.

6.2 Context Dimension Tree - CDT

Il modello di contesto definito nell'ambito Context-ADDICT è chiamato Context Dimension Tree (CDT) e modella la nozione di contesto come una struttura ad albero; il CDT rappresenta le dimensioni del contesto come figli del nodo radice, ciascuno dei quali cattura una prospettiva diversa del contesto. Un valore di dimensione può essere ulteriormente analizzato secondo diversi punti di vista (detti sotto-dimensioni), generando una sotto-struttura a sua volta. In un CDT, i nodi neri rappresentano le dimensioni e le sotto-dimensioni; i nodi bianchi rappresentano i valori che le dimensioni possono assumere. Quando il numero di possibili valori di una dimensione è grande (ad esempio, quando sono costituiti da una serie di valori numerici) i nodi-attributo (rappresentati da due cerchi concentrici) vengono utilizzati, e le loro istanze sono i valori ammissibili per tale dimensione. Allo stesso modo, gli attributi sono utilizzati anche per selezionare istanze specifiche nel set di valori rappresentati da un nodo bianco. In questo caso, un nodo-attributo connesso ad un nodo bianco esprime un parametro di restrizione che può essere utilizzato per dati distinti relativi all'elemento desiderato. Il parametro può essere un valore costante, un parametro variabile il cui valore viene acquisito dall'applicazione o il risultato di una funzione. In ogni caso, le foglie del CDT non possono essere sia nodi bianchi che nodi-attributo.
Si noti che, dato che la rappresentazione del contesto è strettamente correlata allo scenario dell'applicazione, non può essere definita a priori e solo le dimensioni che sono significative per l'applicazione di destinazione sono incluse nel CDT.
Un'istanza di contesto, denominata configurazione del contesto, è descritta per mezzo di elementi del contesto stesso. Un elemento del contesto può avere due differenti specifiche: *dim_name*: *value* oppure
dim_value: *value*(*param_value*), dove *dim_value* è il nome di una dimensione mentre *value* è il valore per quella dimensione.
Considerando l'organizzazione gerarchica del CDT, è possibile definire una relazione discendente sugli elementi del contesto, per cui un elemento del contesto, ce_i, è discendente di un altro elemento del contesto ce_j, se ce_j è un'istanza di una dimensione nel sotto-albero con radice in ce_i; allo stesso modo può essere definito il rapporto ascendente. Inoltre, per essere coerente con la gerarchia, se un elemento del contesto, ce_i, ha come ascendente l'elemento ce_j con un attributo, l'elemento ce_i eredita l'attributo di ce_j (e i

rapporti di discendenza e di ascendenza sono estesi ai nodi con parametri). L'insieme di tutti gli elementi del contesto che sono discendenti di un elemento ce_i è chiamato $disc(ce_i)$.
In fase di progettazione, una volta che il CDT è stato definito, l'elenco delle sue configurazioni di contesto viene generato in modo combinatorio. Tuttavia, dato uno scenario applicativo e il corrispondente CDT, non necessariamente tutte le possibili combinazioni di elementi del contesto hanno significato. Il modello permette l'espressione dei vincoli tra i valori di un CDT in modo tale da evitare la generazione degli elementi insignificanti. Una volta che le configurazioni del contesto significative sono determinate, ognuna di queste viene associata con una vista corrispondente alla porzione rilevante dello schema di dominio delle informazioni. Questo processo è fatto scrivendo direttamente una query nel linguaggio supportato dal database oppure utilizzando un'interfaccia grafica. La visualizzazione associata con ciascun contesto viene formalizzata come un insieme di espressioni relazionali.

6.3 Fasi della personalizzazione

L'approccio per la personalizzazione basata sulle preferenze contestuali è implementata come un'estensione dell'architettura Context-ADDICT: quando una vista sensibile al contesto è individuata dal sistema, prima del caricamento nel dispositivo dell'utente, la personalizzazione viene effettuata per mezzo di preferenze espresse dall'utente stesso che sono rilevanti per il suo contesto attuale. Il procedimento metodologico è composto da quattro funzioni principali:

1. selezione delle preferenze attive,
2. classificazione degli attributi,
3. classificazione delle tuple,
4. personalizzazione della vista.

La personalizzazione viene eseguita in due fasi distinte: la prima avviene sulle tuple e la seconda sugli attributi. Tuttavia, possono essere introdotte altre alternative: ad esempio la selettività di viste del contesto potrebbe essere usata per veicolare la personalizzazione degli attributi; inoltre,

la personalizzazione automatica degli attributi potrebbe essere considerata quando l'utente non specifica qualsiasi classificazione degli attributi.
Il sistema Context-ADDICT è provvisto di un archivio che contiene, per ciascun utente, l'elenco delle sue preferenze contestuali; questo elenco è chiamato profilo delle preferenze. Quando il dispositivo dell'utente si connette al server dell'applicazione e richiede una sincronizzazione di visualizzazione dei dati secondo il contesto attuale, invia la configurazione del contesto attuale, cioè, il descrittore del contesto. Nel punto 1, il profilo delle preferenze viene analizzato per selezionare quei casi (chiamati preferenze attive) che sono rilevanti per il contesto corrente. In particolare, la preferenza è attiva se la sua configurazione di contesto è uguale (o "più generale") al descrittore di contesto corrente. Questa scelta è motivata dal fatto che un contesto più generale è relativo ad una porzione più ampia dei dati, rispetto alla visualizzazione associata al contesto corrente. Così, possono essere applicate le preferenze specificate in una visione più ampia, con un peso appropriato, anche alla porzione più raffinata dei dati associati con il contesto corrente. Poi, l'insieme di preferenze attive viene applicato sugli attributi (punto 2) e sulle tuple (punto 3) della vista associata al contesto attuale. Il risultato di questi due passaggi è una vista che comprende sia le tuple, sia gli attributi, entrambi con i relativi punteggi associati. Infine, nel punto 4, la vista è decisamente personalizzata e ridotta scartando le tuple e gli attributi meno interessanti.
È doveroso far notare che un altro problema rilevante risulta essere quello legato ai meccanismi previsti per l'utente per la generazione di preferenze.

6.4 Conclusioni

In questo capitolo è stato mostrato un nuovo approccio, proposto per comprendere le preferenze sia sulle tuple che sugli attributi di viste di contesto. L'approccio è un'estensione di Context-ADDICT, un framework per il confezionamento di dati, provenienti da fonti integrate e possibilmente eterogenee, utilizzando la nozione di contesto. Al momento, le ricerche stanno cercando di definire una strategia funzionale per individuare le preferenze degli utenti sulla base della loro precedente interazione con il sistema. Inoltre, è in corso lo sviluppo di un prototipo Java per aumentare la possibilità di specificare le preferenze e di applicarle prima di riporre porzioni di dati del contesto nella memoria del dispositivo di un utente.

Ringraziamenti

Un ringraziamento particolare alla mia famiglia e a tutti coloro che in questi anni mi hanno dato supporto, non facendomi desistere, per riuscire a raggiungere soddisfazioni.

Bibliografia

[1] S. Börzsönyi, D. Kossman , K. Stocker, "*the Skyline Operator*", *Proceedings 17th International Conference on Data Engineering*, 2001

[2] E. Pitoura, K. Stefanidis, P. Vassiliadis, "*Contextual Database Preferences*", *IEEE Data Engineering Bulletin*, 2011

[3] J. Chomicki, "*Preference Formulas in Relational Queries*", *Transactions on Database Systems*, 2003

[4] W. Kießling, "*Foundation of Preferences in Database Systems*", *Proceedings of the 28th international conference on Very Large Data Bases*, 2002

[5] J. Chomicki, P. Ciaccia, N. Meneghetti, "*Skyline Queries, Front and Back*", *Transactions on Database Systems*, 2013

[6] R. Agrawal, R. Rantzau, E. Terzi, "*Context-sensitive ranking*", *A framework for expressing and combining preferences, Proceedings of the 2000*, 2006

[7] S. Zadrożny, J. Kacprzyk, "*Bipolar Queries and Queries with Preferences*", *Handbook of Research on Innovative Database Query Processing Techniques*, 2006

[8] A. Miele, E. Quintarelli, L. Tanca, "*A methodology for preference-based personalization of contextual data*", 2009

[9] J. Chomicki, "*Semantic Optimization of Preference Queries*", 2004

[10] J. Chomicki, "*Queryng withIntrinsic Preferences*", 2002

[11] B. Möller, P. Roocks, "*An Algebra of Layered Complex Preferences*", *Relational and Algebraic Methods in Computer Science*, 2012

[12] W. Kießling, G. Köstler, "*Preference SQL – Design, Implementation, Experience*", 2002.