„Geschichte“ der Datenbank Retrieval

Anna-Lena Berndt

Überblick

- **Einleitung**
  - “Nomale” DB-Abfragen
  - “Präferenz” DB-Abfragen
- **M. Lacroix & P. Lavency (Philips)**
  - Ansatz
  - Das System
  - Rückblick
- **A. Motro (University of Southern California)**
  - Ansatz
  - Das System
  - Rückblick
- **R. Agrawal & E. Wimmers (IBM)**
  - Ansatz
  - Das Framework
  - Rückblick
- **Zusammenfassung**

Einleitung

- **„Normale“ DB-Abfrage :**
  - Exakte Angabe des Gesuchten
  - Nur exakte Treffer in der Antwortmenge
    - Problem: keine Treffer
  - Beispiel: Autosuche im Internet
    - Dies ist die moderne Sprichwörtliche „Suche nach der Nadel im Heuhaufen“
  - Lösung:
    - Viele “normale” DB-Abfragen mit kleinen Änderungen
    - „Präferenz“ DB-Abfragen

Einleitung

- **„Präferenz“ DB-Abfrage**
  - Was sind Präferenzen
    - Porsche ist besser als VW
    - Umso günstiger umso besser
  - (Schwierige) Festlegung der Ähnlichkeit
    - Umkreissuche
    - Farben
  - Treffer in der Antwortmenge sind exakt oder nähern sich dem Ziel der Anfrage
  - Kein Verlust der Popularität und Einfachheit
  - Problem: DB-Systeme können nicht direkt mit Präferenzen umgehen
  - Lösung: „aufsetzen“ auf das bestehende DB-System

Ansatz

- **Idee und Problemstellung**
  - Experimentelle Programmierumgebung
  - Abfragen der Versionen von Objekten in Programmen
  - Objekte anhand ihrer Eigenschaften identifizieren und nicht an den Namen
- **Lösung**
  - Präferenzklausel wird in *Domain Relational Calculus family (DRC*) eingebunden

Das System

- **Spezifiziertes Constraint zur Angabe von Präferenzen**
  - *Select* Name *from* PKW *from which* ***prefer*** *those having*…
- **Ablauf:**
  - Abfrage wird ohne ***prefer*** ausgeführt
  - ***prefer*** wird auf die Antwortmenge angewendet
  - Ist die Antwortmenge leer, wird ***prefer*** ignoriert
  - Sonst reduziert sich die Antwortmenge

Das System

- **Prioritäten werden durch die Reihenfolge der Präferenzen festgelegt**
  - *select Name from PKW*
    *from which **prefer** those having* Farbe = Blau
    *from which **prefer** those having* Typ = Limousine
    *from which **prefer** those having* minimum Preis…
- **Bei gleichwichtigen Präferenzen wird nur die Preferklausel wiederholt**
  - *select Name from PKW*
    *from which **prefer** those having* Farbe = Blau
    ***prefer** those having* maximum Preis…

Rückblick

- **Fazit**
  - Einfaches ausdrücken von mehreren Präferenzen
  - Mit DRC sehr mühsam
  - Nicht sehr effizient, da DRC- Abfragen kombinatorische Funktionen über die # der Präferenzen sind => sehr komplex

Ansatz

- **Idee und Problemstellung**
  - Kein direkter Umgang mit Präferenzen
- **Lösung**
  - Direkte Verarbeitung von Präferenzen (*VAGU*E)
  - Ergänzung des relationalen Datenmodells mit Datenmetriken (Definition von Distanzen zwischen 2 Werten derselben Domäne)
  - Neues Problem: Verschiedene Benutzer habe verschiedene Vorstellungen von Metriken
  - *VAGUE* stellt individuelle Sichten und Prioritäten zur Zerfügung

Das System

- **VAGUE Aufbau**
  - Jede Domäne ist mit Datenmetriken ausgestattet
  - Vergleichsoperator *similar-to*
  - Durchmesser (Obergrenze von allen Distanzen einer Metrik) zum Schätzen von nicht definierten Distanzen
  - Radius (ermittelt die Werte die in einem bestimmten „Umkreis“ liegen) der zum skalieren von Distanzen benutzt wird
  - Radius = 0 => Abfrage ohne Präferenzen
  - Durchmesser = Radius => alle DS berücksichtigt

Das System

- **VAGUE Aufbau**
  - Berechenbare Metriken
  - Tabellarische Metriken (Die Metrik durchsucht die Tabelle)
  - Referentielle Metriken (Distanzen zwischen den Schlüsselattributen)
  - Mehrere Metriken für dieselbe Domäne
  - Ist die Antwortmenge leer wird diese auch zurückgegeben
  - Der Benutzer kann den Radius verdoppeln

Das System

- **Beispiel für Metriken**

Domäne	Metrik	Typ	Durchmesser	Radius
Farbe	String	Tab	10	0,5
Preis	Int	Com	50.000	5.000
# Sitzplätze	Int	Com	5	0

String: 0 = Identität , 1 = völlig verschieden

- Zum ranken wird die Reihenfolge anhand der Distanzen festgelegt wobei die kleinste Gesamtdistanz den ersten Platz erhält usw…
- **(** (1*Farbe)^2 + (2*Preis) ^{2} + (3* #Sitzplätze) ^{2} **)** ^{½}

  wobei 1,2 und 3 Gewichtungen des Benutzers sind
- DB-Designer legt Metriken, Durchmesser und Radien fest

Das System

- **Beispiel mit QUEL- Abfragesprache**
  - *Similar-to* = „ ?= “
  - *range of f is* Farbe *range of p is* Preis *range of m is* PKW *retrieve* (p.PKW)
    where m.Farbnr = f.Farbnr *and* m.Name = p.Name
    and f.Farbe ?= Blau

- **Interaktion mit dem Benutzer**
  - Abfrage wird vom Benutzer eingegeben
  - Benutzer wählt Metrik und Gewichtung
  - Komplette Abfrage wird auf die DB angewendet
  - Liefert die Antwortmenge zurück
  - „verranken“ des Systems möglich

Rückblick

- **Fazit**
  - Sehr effizient, da Metriken nicht erst während der Abfrage berechnet werden
  - Schnittstellenrealisierung möglich
  - Sehr flexibel durch mehrfache Metriken und Angaben von Gewichtungen
  - Kein erstellen von Metriken
  - Radius darf erhöht werden
  - Ein *Tupel* muss alle Präferenzen erfüllen

Ansatz

- **Idee und Problemstellung**
  - Explosion von Informationen im WWW
  - Langes und aufwendiges Suchen
  - Schwierig wenn man gar nicht weiß was man will

- **Lösung**
  - Framework für Präferenzabfragen
  - Vielseitig im Anwendungsgebiet
  - Ein „Alleskönner“

Das Framework

- **Aufbau**
  - Präferenz wird durch ein *Entity* ausgedrückt
  - *Entity* enthält eine Menge von benannten Feldern
  - Jedes Feld kann Werte eines bestimmten Typs annehmen
  - * ist ein *wild card*
  - Eine Menge von Präferenzen wir durch einen Kombinations-operator kombiniert
  - Dieser ist mit einer Wertefunktion instanziiert
  - Gleiche Präferenzen können in mehreren Wegen kombiniert werden
  - Eine Kombination von zwei Präferenzen ergibt eine neue Präferenz
  - Diese kann wieder kombiniert werden

Das Framework

- **Präferenzfunktion**
  - Beinhaltet eine Menge von Basis Typen (*int, string, boolean*…)
  - Datentyp *score* repräsentiert die Benutzerpräferenz
  - *score* liegt in [0,1] ; 1 die höchste und 0 die niedrigste Präferenz
  - Keine Präferenzangabe „⊥“. Veto „#“
  - *record type* ist eine Menge von Paaren (PKW_Name : *string,*Preis: *int,* … ,Farbe : *string*)
  - Ein *record* enthält in jedem Feld ein Wert von dessen Typ r(Farbe) ist ein Element vom Typ *string*
  - Eine Präferenzfunktion p ist eine Funktion die zu einem gegebenen *record type* und *score* die passenden *records* findet

Das Framework

- **Kombinieren von Präferenzfunktionen**
  - Funktion *combine* die mit einer Wertefunktion einen neuen *score* berechnet
  - Erzeugt eine neue Präferenzfunktion *combine* (f) (p1…pn)(r)
  - Ergebnisse werden nach dem *score* geordnet

- **Beispiel**
  - Max M und Tina T wollen ein Auto kaufen, haben aber unterschiedliche Vorstellungen
  - Streitpunkte sind Autotyp, Preis und Farbe
  - *record* (Autotyp,Preis,Farbe)

Das Framework

- **Tinas Präferenzen**
  - T (Kombi, * , *) = 0,8
  - T (Limousine, * , *) = 0,5
  - T (Kombi, * , Blau) = 1
  - T (Limousine, <30.000 , *) = 0,9
- **Maxs Präferenzen**
  - M (Limousine, * , *) = 1
  - M (Limousine, * , Rot) = #
  - M (Limousine, <50.000 , Weiß) = 0,8
- **Diese Präferenzen werden zu einer Endpräferenz kombiniert**

Rückblick

- **Fazit**
  - Sehr leistungsstark
  - Benutzer muss mit dem System sehr vertraut sein
  - *Framework* => kein bereits realisiertes System
  - Bibliothek von Wertefunktionen genügt einer großen Anzahl an Benutzern

Zusammenfassung

	Lacriox	**Motro**	**Rakesh**
SQL nah	Ja	Ja	Nein
Vorkenntnisse des Benutzers	Nein	Ja (Metriken)	Ja
Abfrageart	Teilmenge	Ganze DB	Ganze DB
Flexibel	Nein	Ja (Metriken)	Ja
Effizient	Nein	Ja	Ja
Zuverlässig	Ja	Nein (ranken)	Ja

Zusammenfassung

- Je mehr die Systeme können desto schlechter wird die Benutzerfreundlichkeit

Vielen Dank
für eure
Aufmerksamkeit!

Noch Fragen ???