Universität Rostock

Traditio et Innovatio

Verteidigung Bachelorarbeit

Eignung von Query Containment-Techniken zur Reformulierung von Anfragen bei der datenschutzfreundlichen Gestaltung von Informationssystemen

Johann Kluth

Motivation

- Query Containment
  - Datenbankanfragen beschleunigen
  - Datenbankanfragen verteilen
  - Anfrageergebnisse in Assistenzsystemen anonymisieren
    - → Datenschutz einhalten
    - → benötigte Daten bleiben erhalten

Übersicht

- Zwischenstand — Zusammenfassung
  - Techniken
  - MiniCon
- Umsetzung
  - Implementierung
  - Ergebnisse

Query Containment

Test zweier Anfragen auf Enthaltensein

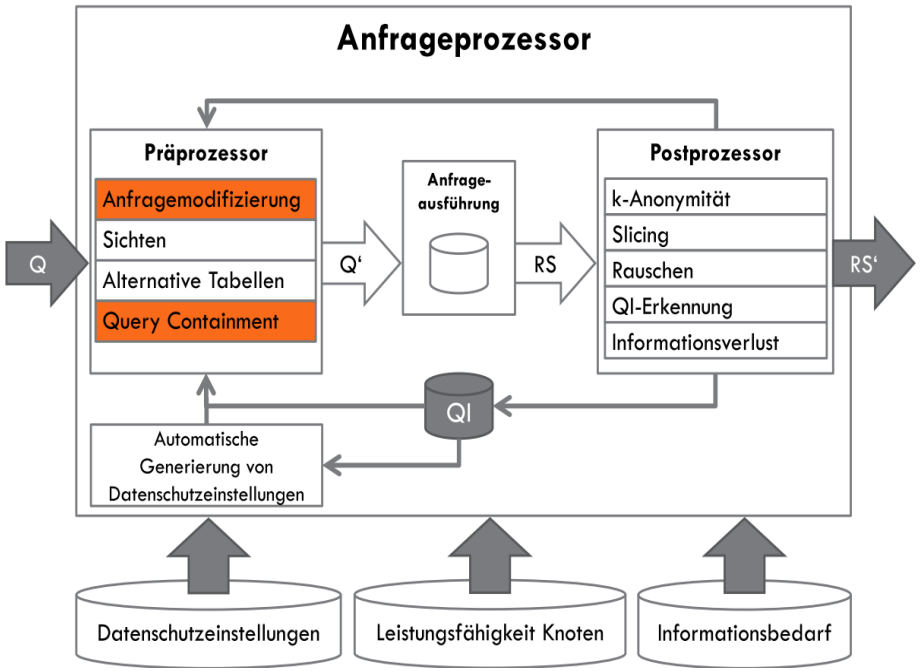


Abbildung: Die Einordnung von QC im Anfrageprozessor des PArADISE-Projektes.

Answering Queries using Views

Gegeben: Anfrage Q in KNF, Menge von Sichten V, auf Datenbank D.
Gesucht: Rewriting $r(V) = Q_1$ von Q

Anforderungen:

- möglichst vollständig die Sichten nutzen,
- möglichst wenig unterschiedliche Sichten nutzen (weniger Joins) und
- semantisch äquivalent zur originalen Anfrage sein.

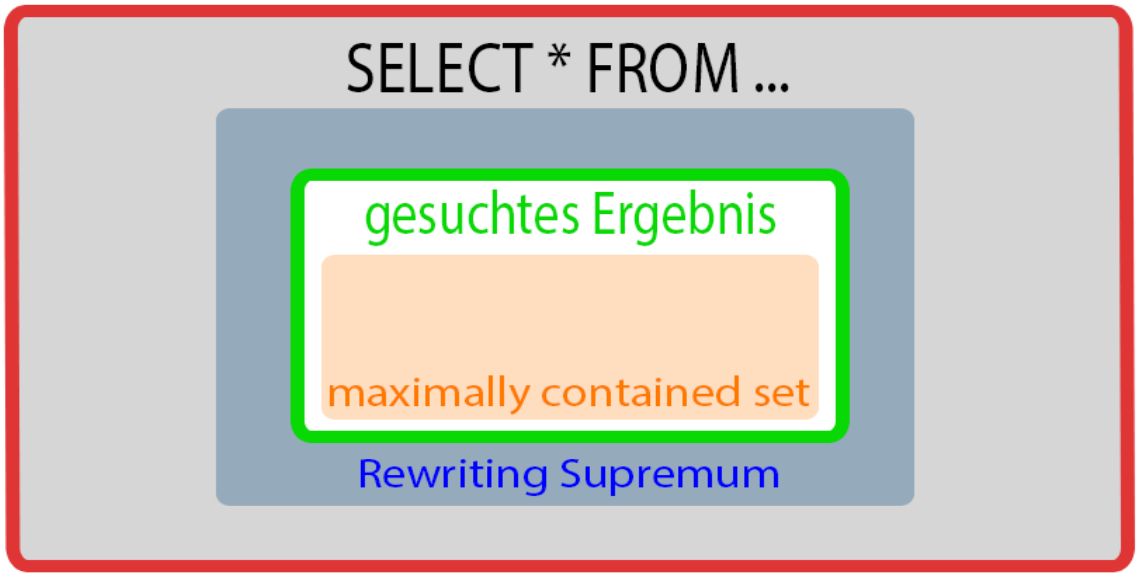
Answering Queries using Operators

Operatoren anstelle von Sichten nutzen: Anwendung in Assistenzsystemen

Gegeben: O = Menge möglicher Operatoren

Gesucht: Rewriting, welches eine Teilmenge der Operatoren aus O nutzt. Anfragen liefern dann ein gefiltertes Ergebnis an den Server.

Query Containment Problem



AQuV: $\nexists\, Q' : Q_1(D) \sqsubset Q'(D) \sqsubseteq Q(D)$

AQuO: $\nexists\, Q' : Q_1(D) \sqsupset Q'(D) \sqsupseteq Q(D)$

Vergleich der Algorithmen

Algorithmen:

- Bucket
- MiniCon
- Inverse Rules

Kriterien:

- Allgemein
- Einsatzmöglichkeiten
- Laufzeiteffizienz

Vergleich der Algorithmen

Algorithmen:

- Bucket
- MiniCon
- Inverse Rules

Kriterien:

- Allgemein
- Einsatzmöglichkeiten
- Laufzeiteffizienz

MiniCon-Algorithmus

Ist eine Verbesserung des Bucket-Algorithmus, basiert daher auf diesem.

$$Q(x) = p_1(x, y) \wedge p_1(y, x) \wedge p_2(x, y)$$
$$V_1(a) = p_1(a, b) \wedge p_1(b, a)$$
$$V_2(c, d) = p_2(c, d)$$
$$V_3(f, h) = p_1(f, g) \wedge p_1(g, h) \wedge p_2(f, g)$$

Teilziele:

1. $p_1(x, y)$
2. $p_1(y, x)$
3. $p_2(x, y)$

MiniCon-Algorithmus

MiniCon Description

$h(v_i)$	h	φ	G
$V_2(c,d)$	$c \to c, d \to d$	$x \to c, y \to d$	3
$V_3(f,f)$	$f \to f, h \to f$	$x \to f, y \to g$	1, 2, 3

MiniCon-Algorithmus

1. Teilziel: $p_1(x, y)$

Gefundene Sicht: $V_1(a) = p_1(a, b) \wedge p_1(b, a)$

V_1 widerspricht folgenden Bedingungen für eine MCD:

1. *Die Sicht enthält in ihrem Kopf mindestens eine Kopfvariable der Anfrage.*
2. *Wenn eine Variable der Anfrage in einem Verbund als Prädikat vorkommt, dann:*
   - 2.1 *muss derselbe Verbund auch in der Sicht vorkommen, oder*
   - 2.2 *die Variable MUSS im Kopf der Sicht stehen.*

MiniCon-Algorithmus

1. Teilziel: $p_1(x, y)$

Gefundene Sicht: $V_3(f, h) = p_1(f, g) \wedge p_1(g, h) \wedge p_2(f, g)$

- V_3 kann auch das **2.** und **3. Teilziel** abdecken (G = 1, 2, 3)
- Bedingungen für MCDs sind erfüllt
- Algorithmus läuft weiter, bis alle MCDs gefunden sind

Bemerkung: **2. Teilziel** trivial.

MiniCon-Algorithmus

3. Teilziel: $p_2(x, y)$

Gefundene Sicht: $V_2(c, d) = p_2(c, d)$

- V_2 deckt das Teilziel komplett ab + erfüllt Regeln

$\rightarrow$ MCD wird übernommen

$\rightarrow$ Algorithmus bricht ab, da alle MCDs gefunden sind

MiniCon-Algorithmus

Die gefundenen MCDs zu der Anfrage Q.

$h(v_i)$	h	φ	G
$V_2(c,d)$	$c \to c, d \to d$	$x \to c, y \to d$	3
$V_3(f,f)$	$f \to f, h \to f$	$x \to f, y \to g$	1, 2, 3

Vorteil: Kein extra Test auf Containment notwendig (MiniCon garantiert das von sich aus)

Wichtige Klassen

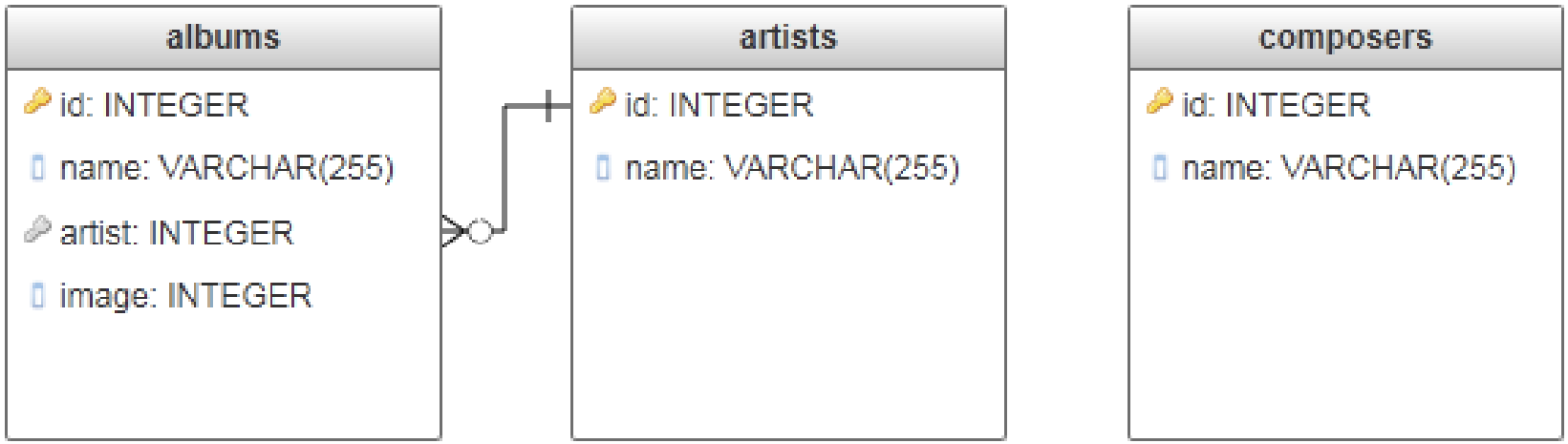
- **MiniCon:** Ausführung des Algorithmus
- **MCD:** Repräsentation einer *MiniCon Description*
- **Query:** Analyse einer SQL-Anfrage
- **View:** Analyse einer Sicht

Wichtige Funktionen

- **checkFirstMiniConProperty:**
  - Prüfe erste Bedingung für eine MCD
- **checkSecondMiniConProperty:**
  - Prüfe zweite Bedingung für eine MCD
- **formingMCDs:**
  - Erstellt die MCDs aus Informationen der Sichtdefinition
- **combiningMCD:**
  - Kombiniert die MCDs zu einem Rewriting der Anfrage

Testfälle

Datenbankschema (relevanter Ausschnitt):



Alle vorhandenen Sichten stehen auf S. 33 meiner Arbeit.

Testfall 1 — Einfacher Fall

1. SELECT * FROM artists

Passende Sicht:

artview: SELECT id, name FROM artists

Rewriting:

- **SELECT id, name FROM artview**

Testfall 2 — Unmöglicher Fall

2. SELECT * FROM years

Kein Rewriting, da keine passende Sicht existiert, welche auf die Relation *”years”* zugreift.

Testfall 3 — Teilmenge

3. SELECT * FROM composers

Passende Sicht:

containedset: SELECT id, name FROM composers WHERE id < 500

Rewriting:

- **SELECT id, name FROM containedset**

ACHTUNG: Ergebnismenge des Rewritings ist kleiner als die der originalen Anfrage.

Anforderungen

- Bei jedem Attribut muss der Relationenname mit angegeben werden.
- Anfrage muss eine SPJ-Anfrage in SQL-Notation sein. Vergleiche auf Äquivalenz im WHERE-Teil erlaubt.
- Alle Sichten müssen komplett klein benannt werden und dürfen keine kompletten Relationennamen enthalten.
- Datenbankmanagementsystem: PostgreSQL

Zusammenfassung der Arbeit

- Bucket-, MiniCon- und Inverse-Rules-Algorithmus
  1. analysiert
  2. vorgestellt
  3. verglichen
- MiniCon implementiert und getestet

Ausblick

1. Chase & Backchase
2. Christian Langmacher: *Vertikale Verteilung von SQL-Anfragen auf DBMS-Servern abnehmender Leistungsfähigkeit.*
3. Tanja Auge: *Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen.*

ENDE

Fragen? Anmerkungen? Feedback?