Einführung in Datenbanken

Vorlesungsmanuskript

Dr. Josef Templ

Universität Salzburg

SS 2006

http://www.cs.uni-salzburg.at/~josef.templ/

Teil 1

Einführung, Motivation und Überblick, Relationen, Relationenalgebra

Literaturempfehlung

Herman Sauer, *Relationale Datenbanken – Theorie und Praxis*, Addison Wesley.

> Leicht verständliche Einführung in die Welt der relationalen Datenbanken mit guter Beschreibung der verschiedenen SQL-Standards.

A. Kemper / A.Eickler, *Datenbanksysteme – Eine Einführung*, Oldenburg.

> Umfassende, fundierte technische und theoretische Einführung in die Welt der Datenbanken mit zahlreichen annotierten Literaturreferenzen. Gute Grundlage auch für wissenschtliche Beschäftigung mit Datenbanken.

Jim Gray / Andreas Reuter, *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann.

> Alles über Transaction Processing inklusive Konzepte, Techniken, Systeme, Geschichte etc. von einem der Datenbank-Päpste (Jim Gray). Geht sehr in die Tiefe und ist sehr anspruchsvoll.

Motivation:

Was ist eine Datenbank und wozu?

bekannt: Listen, Karteien, Dateien, Textfiles

Liste kann nur Anfügen, kann nur sequentiell Suchen ausser sortierte Liste. Kann dann aber nicht einmal Anfügen.

Karteien sind flexibler, aufwändiger, binäre Suche nach Sortierkriterium, keine Suche nach beliebigen Inhalten, keine garantierte Konsistenz, kein Mehrbenutzerbetrieb.

Dateien: Folge von Bytes, zu primitiv, keine Struktur, keine Konsistenz, Mehrbenutzerzugriff nur lesend.

Textfiles: zeilenorientiert, sehr flexibel, nur für kleine Informationsmengen, sequentielle Suche, keine garantierte Struktur, Mehrbenutzerzugriff nur lesend.

Beispiel für verwaltete Daten: Artikel, Kunden, Lieferanten, Bestellungen.

=>Datenbanken: versuchen Vorteile zu kombinieren:

Konsistenz, Sortierung nach mehreren Kriterien, Mehrbenutzerbetrieb,

effizient auch für große Datenmengen, inkrementelle Änderungen,

basieren meist auf Files.

Verschiedene Möglichkeiten eine Datenbank aufzubauen (hierarchisch=Baum, Netzwerk=Graph, Regelbasiert)

in dieser Vorlesung primär: *Relationale Datenbanken*.

dominiert heute in der Praxis, mathematische Grundlage ist Mengenlehre und Relationenalgebra.

Alternativ: Relationenkalkül (Codd 1972) und später Tupelkalkül. Sind äquivalent zu Relationenalgebra.

Am Ende der Vorlesung werden auch *regelbasierte Datenbanken* behandelt.

Mathematische Grundlagen

Menge = gedankliche Zusammenfassung von wohlunterscheidbaren Begriffen

zB Menge von Zahlen, Menge von Personen,

Meist geschrieben als {element1, element2...}

Kreuzprodukt (kartesisches Produkt) aus Mengen M1, M2 ..., Mn = Menge aller Tupel (A1, A2, ..., An) mit A1 Element aus M1, A2 Element aus M2, ... , An Element aus Mn.

geschrieben als M1 x M2 ... x Mn

Tupel besteht aus einer Folge von Werten (Attribute), die eine Position besitzen.

Relation = Teilmenge aus einem kartesischen Produkt.

n-stellige Relation = Teilmenge aus einem n-stelligen kartesischen Produkt

n = Stelligkeit, Grad, degree

Kardinalität einer Relation wie bei Menge, Anzahl der Elemente, = Tupelanzahl.

=> Relation kann leer sein (leere Menge)

=> alle Elemente (Tupel) haben gleiche Anzahl und Art von Attributen

=> jedes Tupel kommt nur einmal vor, d.h. ist eindeutig

=> Tupel einer Relation haben keine Reihenfolge

Relation alias Prädikat alias Eigenschaft alias Beziehung

Relation = Beziehung (vgl. Volksmund, aber allgemeiner)

z.B. Vorlesung, Student, Student besucht Vorlesung

Relation entspricht Liste, Kartei, Tabelle.

Tupel entspricht Listenzeile, Karteikarte, Datensatz.

Beispiel:

F = {Anna, Maria, Gerda}

M = {Bob, Paul}

kartesisches Produkt aus F und M:

F x M = {(Anna, Bob), (Anna, Paul), (Maria, Bob), (Maria, Paul), (Gerda, Bob), (Gerda, Paul)}

Relation 'ist-verheiratet-mit' = {(Anna, Bob), (Maria, Paul)}

ist eine Teilmenge von F x M

Relationenalgebra

allgemein Algebra = Menge(n) und darauf definierte Operationen

z.B. boolsche Algebra: Operationen auf {true, false} (nach George Bool)

=> *Relationenalgebra* = Operationen auf Relationen (nach E.F.Codd, 1970)

Attribute haben bei Codd keine Position sondern einen (eindeutigen) Namen.

Attribute sind immer atomar, d.h. nicht selbst wieder Tupel oder Mengen.

Menge der möglichen Werte eines Attributs A = Domain von A = dom(A).

Attributwerte sind entweder bekannt (Element aus Domain) oder unbekannt (NULL).

Unterscheidung zwischen Schema (Typ) und Wert (Ausprägung)

Häufige Schemanotation: sch(R): {[A1name: A1typ, ... , ANname: ANtyp]}

Schema von R ist die Menge der Attribute von R.

Mengenoperationen auf Schema erlaubt.

z.B.: sch(ist-verheiratet-mit) : {[frau: F, mann: M]}

Anzahl der Attribute einer Relation R = |sch(R)|

Anzahl der Tupel von R = |R|

Operationen:

- Restriction (=Selection, Tupelauswahl)
- Projection
- Product
- Rename
- Union
- Intersection
- Difference

ableitbare Operationen

- Joins
- Division

RESTR(R, C)

definiert Teilmenge aller Tupel aus Relation R für die Bedingung C erfüllt ist.

Geschrieben auch mit kleinem Sigma mit C als Subscript: σ_C R

$|\sigma_C R| <= |R|$, $|sch(\sigma_C R)| = |sch(R)|$

PROJ(R, <attr1, ... attrN>)

projiziert R auf die selektierten Attribute aus der Relation R.

Kann zu kleinerer Tupelanzahl führen, damit Tupel eindeutig bleiben.

Geschrieben auch mit einem großen Pi mit den Attributen tiefgestellt: $\prod_{\{a1, a2...\}} R$

Die Mengenklammer für die gewählte Attributmenge wird auch oft weggelassen. $\prod_{a1, a2...} R$

$|\prod_{\{a1, a2...\}} R| <= |R|$, $|sch(\prod_{\{a1, a2...\}} R)| <= |sch(R)|$

PRODUCT(R1, R2)

bildet das kartesische Produkt aus R1 und R2. Auch mit Infix-Operator x geschrieben: R1 x R2.

$|R1 \times R2| = |R1| * |R2|$, $|sch(R1 \times R2)| = |sch(R1)| + |sch(R2)|$

Kann zu großer Anzahl von Tupel führen.

RENAME(R, neuerNameVonR)

RENAME(R, Attrx->neuerNamevonAttrx)

Umbenennung einer Relation oder eines Attributs einer Relation.

Kardinalität und Grad bleiben unverändert.

Anwendung zum Beispiel zur Vermeidung von Mehrdeutigkeiten:

Attributnamen in Ergebnis mehrdeutig: PRODUCT(R, R)

Attributnamen in Ergebnis eindeutig durch Qualifizierung mit R bzw. S: PRODUCT(R, RENAME(R, S))

UNION(R1, R2)

bildet die Vereinigung aus R1 mit R2. Auch geschrieben mit Infix Operator: $R1 \cup R2$.

R1 und R2 müssen die gleiche Anzahl und Art von Attributen aufweisen (UNION-kompatibel)

$|R1 \cup R2| <= |R1| + |R2|$, $sch(R1) = sch(R2) = sch(R1 \cup R2)$

INTERSECTION(R1, R2)

bildet die Schnittmenge aus R1 und R2. Auch geschrieben mit Infix Operator: $R1 \cap R2$.

R1 und R2 müssen UNION-kompatibel sein.

$|R1 \cap R2| <= |R1|$, $|R1 \cap R2| <= |R2|$, $sch(R1) = sch(R2) = sch(R1 \cap R2)$

DIFFERENCE(R1, R2)

bildet die Differenzmenge, also alle Tupel aus R1 die nicht in R2 vorkommen.

Auch geschrieben mit Infix Operator: R1 - R2.

Beide müssen UNION-kompatibel sein.

$|R1 - R2| <= |R1|$, $sch(R1) = sch(R2) = sch(R1 - R2)$

Joins

zusammengesetzte Operation aus PROD und RESTR und je nach Join-Art mit PROJ und RENAME.

Allgemeiner (theta) Join(R1, R2, C) = RESTR(PROD(R1, R2), C)

Geschrieben auch als: R1 θ_C R2.

Equi-Join: C nur mit Vergleichsoperator '='

Natural-Join: Equi-Join für alle gleichnamigen Attribute mit Projektion sodass gleiche Attribute im Ergebnis nur einmal aufscheinen.

Auto-Join (=Self Join): Join einer Relation mit sich selbst. RENAME erforderlich.

Outer-Join: obige Joins heissen auch *inner Joins*. Manchmal möchte man alle Tupel einer Relation im Join-Ergebnis, auch jene die gemäß Join-Bedingung kein Partnertupel finden (erweitertes Produkt).

Attribute des fehlenden Partner-Tupel sind unbekannt und daher NULL.

Left-Outer Join: Alle Tupel des linken Operanden bleiben erhalten

Right-Outer Join: Alle Tupel des rechten Operanden bleiben erhalten

Full-Outer Join: Alle Tupel beider Operanden bleiben erhalten

DIVISION

Umkehrung von PROD: R x S / S = R, kaum praktische Bedeutung, nicht in SQL.

Beispiele:

Gegeben seien zwei Relationen 'verheiratet' und 'verlobt', mit Attributdomänen wie vorher (M und F). Es sei die Aufgabe des Standesamtes, zu verwalten, wer mit wem verheiratet beziehungsweise verlobt ist.

sch(verheiratet) : {[mann:M, frau:F]}
sch(verlobt) : {[mann:M, frau:F]}

Aufgabe 1:
Ein Fehler im Standesamt liegt vor, wenn ein Paar sowohl verheiratet als auch verlobt ist. Mit welcher Operation kann man solche Fehler finden?

Lösung 1 durch natural-join:
Standesamtfehler = NATURALJOIN(verheiratet, verlobt)
ergibt Schema {[mann:M, frau:F]}

Lösung 2 durch Schnittmenge:
Standesamtfehler = INTERSECTION(verheiratet, verlobt)
ergibt ebenfalls Schema {[mann:M, frau:F]}

Aufgabe 2:
Ein Heiratsschwindler sei ein Mann, der gleichzeitig verheiratet und verlobt ist, aber mit einer anderen Frau. Wie findet man solche Männer?

Lösung 1 mit Theta-Join:
Heiratsschwindler = JOIN(verheiratet, verlobt, verheiratet.mann = verlobt.mann AND verheiratet.frau != verlobt.frau)
liefert Schema {[verheiratet.mann:M, verheiratet.frau:F, verlobt.mann:M, verlobt.frau:F]}

Lösung 2 (Annahme keine Standesamtfehler) mit Projektion und Schnittmenge:
Heiratsschwindler = INTERSECTION(PROJ(verheiratet, <mann>), PROJ(verlobt, <mann>))
liefert Schema {[mann:M]}

Aufgabe 3:
Unter Umgehung der Verlobung könnte ein Heiratsschwindler auch mehrfach verheiratet sein. Wie findet man solche Männer?

Lösung mit Self-Join und Projektion auf verheiratet.mann:
Heiratsschwindler2 = PROJ(JOIN(verheiratet, RENAME(verheiratet, v2), verheiratet.mann = v2.mann AND verheiratet.frau != v2.frau), <verheiratet.mann>)
Man beachte: Ein Mann, der zB dreimal verheiratet ist, kommt im Ergebnis nur genau 1 mal vor, weil das Ergebnis der Projektion eine Menge vonTupel ist, die nur ein Attribut besitzen. Wie verhält sich die Tupelanzahl ohne Projektion?