Präfix-Bäume als Indexstruktur für String-Attribute in relationalen Datenbanken

Exposé zur Diplomarbeit
Nicky Hochmuth

29. November 2005

Betreuer: Prof. Ulf Leser
(HU Berlin, Inst. für Informatik)
Zeitrahmen: Oktober - Juni 2006

Motivation

Die Anwendungsbeispiele für String-Attribute in Datenbanken sind reichhaltig und vielseitig. Ob einfache Adressverwaltung, Familienregister für genealogische Archive, linguistische Korpora, Keywortlisten im Textmining oder Gensequenzen in der Bioinformatik - überall ist eine Informationsverarbeitung und Speicherung in Form von Zeichenketten unabdingbar. Diese großen Datenmengen in Datenbanken zu verwalten, ermöglicht nicht nur ein schnelles und gezieltes Wiederauffinden einzelner Informationen, sondern vor allem die effiziente und universelle Nutzung aller Daten zur explorativen Datenanalyse.

Zeichenketten sind zwar in ihrer Ausprägung und Nutzung sehr vielseitig und flexibel, haben aber den Nachteil, dass sie im Allgemeinen nur sequenziell verarbeitet werden können. Schon die Bestimmung der gebräuchlichsten Relationen wie der *Äquivalenz*, *Größer* und *Kleiner* über String-Attribute erreicht daher zumeist höhere Komplexitäten als über Attribute mit primitiven Datentypen. Bei der explorativen Datenanalyse ist eine schnelle Verarbeitung meist wichtiger als eine minimierte Speicherung und daher bietet sich die Nutzung von Indexstrukturen an. Die verbreiteten Datenbank-Managementsysteme unterstützen vor allem die Methodiken des B*-Baumes sowie der Hashbasierten-Indexierung [2, 7, 9].

Beide Methodiken besitzen jedoch effizienzbezogene oder funktionale Nachteile. Der Nachteil der Hashindexierung ist im Besonderen, dass keine effiziente lexikografische Ordnung über das Attribut hergestellt werden kann und daher nur ein Equi-Join

möglich ist [2]. B*-Bäume nutzen genau diese lexikografische Ordnung zwischen den Entitäten des Attributes um boolesche Funktionen wie *Größer* oder *Kleiner* effizient zu berechnen und den Equi-Join mit Hilfe eines Sort-Merge-Join zu realisieren. Die Baumstruktur dieses Indexes ermöglicht eine sequenzielle Ordnung bei etwa gleichen Zugriffszeiten auf die einzelnen Entitäten. Der entscheidende Nachteil der B*-Baum-Indexierung für Strings besteht darin, dass die Kosten zur Errechnung der Join-Funktion (in den meisten Fällen die Äquivalenz) eines einzelnen Tupels im WorstCase linear von dessen Wortlänge abhängt, da beide Strings zeichenweise sequenziell verglichen werden. Durch die lexikografische Ordnung der Tupel wird die Menge der zu vergleichenden Tupelkombinationen zwar auf ein lineares Maß beschränkt. Beim nächsten ausstehenden Tupel-Vergleich werden jedoch wieder der Reihe nach von vorn beginnend die beiden Wortpaare zeichenweise verglichen. Erkenntnisse aus vorangegangenen Tupelvergleichen werden nicht einbezogen. Die Komplexität eines Sort-Merge-Joins über String-Attribute ergibt sich somit aus dem Produkt der Entitätenanzahl und deren Wortlänge. Versuche, deshalb den Index oder die Schlüsselwerte des Baumes auf einen Präfix der Worte mit bestimmter Länge zu beschränken, erhöhen zwar die Performanz beim Zugriff auf einzelne Daten - aber nicht bei einem Join [6, 1]. Diese ineffiziente Besonderheit von Strings kann solange nicht verhindert werden, wie die String-Entität in ihrer Gänze als atomare Einheit betrachtet und auch so in Indexstrukturen abgebildet wird.

Zielsetzung

Ziel dieser Diplomarbeit ist die Evaluation von Präfix-Bäumen als Indexstruktur für String-Attribute in relationalen Datenbanken und anderen Anwendungen. Dieser Index soll eine kompakte Speicherung, schnelle Wartung und Berechnung von String-Äquivalenzen und eine lexikografische Sortierung ermöglichen. Die durch die konkrete Datenstruktur und ihre Realisierung zu erwartenden Komplexitäten sollen analysiert und experimentell evaluiert werden.

Herangehensweise

Präfix-Bäume werden bisher effektiv beim Pattern Matching in Form von Keyword-Trees [4], zum Frequent Itemset Mining im Datamining [5, 3], sowie zum Berechnen von Set-Joins [8] und zur kompakten Speicherung von Wortmengen benutzt. In der Arbeit soll versucht werden, Präfix-Bäume als eine Indexierungsstruktur für String-Attribute in einem RDBMS zu nutzen, um die obig genannten Forderungen zu realisieren. Die Zeichenketten sollen nicht mehr in ihrer Gänze indexiert werden, sondern Zeichen für Zeichen. Die lexikographisch sortierte Indexierung soll weiterhin die Anzahl der mög-

lichen Join-Kanditaten einschränken (Pruning) und darüberhinaus sollen gemeinsame Präfixe die Anzahl der Zeichenvergleiche minimieren und somit die Berechnung von Equi-Joins beschleunigen.

Zuerst soll die Modellierung in einer nativen Implementierung umgesetzt und getestet und anschließend in das RDBMS Oracle integriert werden. Dazu werden die Schnittstellen des *Oracle Data Cartrige Interface* (ODCI) und insbesondere die des *ODCIIndex* genutzt. Dieses Interface ermöglicht es dem Index-Designer, eigens implementierte domainspezifische Indexe so in das RDBMS zu integrieren, dass für den Nutzer die gewohnte Funktionalität erhalten bleibt. Vergleichend mit anderen Indexierungsmethoden in Oracle soll vor allem die Performance folgender Methoden und Eigenschaften untersucht werden:

- Index-Bau
- Insertion / Deletion
- Selection / Rang-Selection mit/ohne Sortierung
- Equi Join
- Skalierbarkeit der Indexgröße

Außerdem wird versucht, die Suche mit Wildcards (LIKE Operator) zu unterstützen aber es wird darauf verzichtet alle Funktionalitäten des Transaktionsmanagsments zu realsieren.

Literatur

[1] J. Gallenbacher A. Buchmann. Skript: Mehrwegbäume. URL, http://www.dvs1.informatik.tu-darmstadt.de/DVS1/lectures/ws01-02/inf3/folien/inf3-folien-by6-teil7.pdf.

[2] S. Sudarshan Abraham Silberschatz, Henry Korth. *Database Systems Concepts.* McGraw-Hill Science/Engineering/Math, 4 edition, 2001.

[3] Bart Goethals Francesco Bonchi. Fp-bonsai: The art of growing and pruning small fp-trees. *PAKDD 2004, LNAI 3056*, pages 155–160, 2004.

[4] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1 edition, 1997.

[5] Y. Yin J. Han, J. Pei. Mining frequent patterns without candidate generation. *In Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, 2000.

[6] MySQL. 7.4.5. how mysql uses indexes. URL, http://dev.mysql.com/doc/refman/5.0/en/create-index.html.

[7] Oracle. *Oracle9i Data Cartridge Developer's Guide Building Domain Indexes*, release 2 (9.2) edition.

[8] V. Pudi R. Jampani. Using prefix-trees for efficiently computing set joins. *In Proc. of Intl. Conf. on Database Systems for Advanced Applications*, 2005.

[9] Sundara Agarwal DeFazio Srinivasan, Murthy. Extensible indexing: A framework for integrating domain-specific indexing schemes into oracle8*i*. 16th Int. Conf. on Data Engineering (ICDE)*, pages 91–100, 2000.*