Eine universelle Datenbankschnittstelle zur Nutzung austauschbarer und erweiterbarer Zugriffsmethoden

Dipl.-Inf. Ralph Acker
Transaction Software GmbH
Thomas-Dehler-Straße 18
81737 München
ralph.acker@transaction.de

Zusammenfassung

Es wird eine Programmierschnittstelle vorgestellt, die ein Datenbanksystems in einem definierten Maße öffnet und die es erlaubt, Zugriffsoperationen und Datenorganisation mit möglichst geringem Aufwand austauschbar und vergleichbar zu implementieren. Zusätzlich soll die Kombination von modularen Implementierungen von Zugriffsmethoden und Datenorganisation möglich sein. Auf der Ebene der Anfragesprache wird diese Erweiterungen möglichst transparent sein. Die beschriebene Schnittstelle wird in das relationale Datenbanksystem Transbase der Firma Transaction Software GmbH integriert werden.

Schlüsselwörter: user-defined indexing, advanced relational analysis operators, API, optimizer

Einleitung

Zugriffspfade und Datenhaltung sind immer noch ein zentrales Thema aktueller Forschungsarbeiten der Datenbankgemeinde. Es werden Forschungsvorhaben und Projekte auf den Gebieten XML-orientierte Datenhaltung und Indexierung, Indexstrukturen für Datawarehouses und zugriffsorientierte bzw. rechnerarchitekturoptimierte Datenhaltung diskutiert. Meist stützen sich diese Arbeiten auf Experimente mit einer Vielzahl von Prototypimplementierungen mit unterschiedlichstem Funktionsumfang ab, wodurch ein Vergleich der Ergebnisse erheblich erschwert wird.
Inhalt des Beitrags ist die Beschreibung einer universellen Programmierschnittstelle, die die unteren Schichten eines robusten, leistungsfähigen und vollständigen Datenbanksystems öffnet, um zu ermöglichen Zugriffsoperationen und Datenorganisation mit möglichst geringem Aufwand austauschbar und vergleichbar zu implementieren. Zusätzlich soll es möglich sein, modulare Implementierungen von Zugriffsmethoden und Datenorganisation zu kombinieren, um so bei Weiterentwicklungen von möglichst vielen existierenden Komponenten Gebrauch machen zu können. Auf der Ebene der Anfragesprache werden diese Erweiterungen möglichst transparent sein und der Optimierer wird selbständig über die Verwendung eines Zugriffspfads entscheiden können. Somit können alle Aspekte des Leistungsverhaltens eines Datenbanksystems in die Bewertung einfließen, wie etwa das Verhalten bei beliebigen SQL Anfragen, Mehrbenutzerbetrieb auf verschiedenen Isolationsebenen und unter Transaktionsschutz, sowie Logging und Recovery.
Obwohl es bereits von mehreren kommerziellen Datenbankherstellern Entwicklungsbestrebungen zu einer offenen Programmierschnittstelle für Zugriffspfade gibt, scheinen diese Unternehmungen andere Zielsetzungen zu haben. Dadurch ergeben sich bedeutende Defizite für den Einsatzmöglichkeiten dieser Schnittstellen. Freie Datenbanksysteme hingegen bieten jede Möglichkeit Änderungen an der Zugriffschicht vorzunehmen, allerdings gibt es hierfür keine klar definierten Schnittstellen, die auch Verwaltung und Kombination von Zugriffmethoden erleichtern könnten.

Anwendungsgebiete

Die Anwendungsmöglichkeiten eines solchen Systems scheinen vielfältig. Der nächste Abschnitt soll einen Denkanstoß zu dessen Verwendungsmöglichkeiten geben.

- **Funktionale Indexstruktuen:** Abgegrenzte Anwendungsgebiete stellen oft sehr spezielle Anforderungen an die verwendeten Indexstrukturen des darunter liegenden Datenbanksystems. So ist es beispielsweise im Bereich der Mobilfunk-Messdatenerfassung notwendig, beim Prozessieren eines

speziellen Joins gegenüber einem Equi-Join eine gewisse Fehlertoleranz bezüglich der Messdaten zuzulassen. Zusätzlich soll eine zeitliche Korrelation der Messdaten berücksichtigt werden. Ein solcher spezialisierter „Fuzzy Join“ könnte durch eine geeignete, speziell zugeschnittene Indexstruktur unterstützt werden. Weitere Anwendungsgebiete sind lokalisierte Indexstrukturen wie etwa der Soundex-Index.

Unstrukturierte Daten: Eine neue Indexstruktur wäre in der Lage unstrukturierte Daten, die nicht mit der relationalen Tabellenstruktur vereinbar sind, effizient zu speichern und zu durchsuchen.

Multidimenisionale Indexstrukturen: Im Bereich der Datawarehouse-Anwendungen werden permanent neue oder bewährte, für spezielle Anwendungen modifizierte, Indexmethoden untersucht. Inzwischen gibt es eine Vielzahl von Universal- und Nischenlösungen, die alle ihre spezifischen Vor- und Nachteile haben. Die Möglichkeit für eine spezielle Anwendung gezielt den optimalen Index auszuwählen wäre ein vielversprechendes Novum.

Physisches Datenlayout: Schon seit längerer Zeit wird untersucht, welchen Einfluss die räumliche Aufteilung der Daten auf Seitenebene auf die Leistungsfähigkeit eines Datenbanksystems hat, das ein ganzes Schema oder einzelne Tabellen (z.B. durch Caching) im Hauptspeicher prozessiert. Es wurde gezeigt, dass eine geeignete Umstrukturierung des Seitenlayouts im Hinblick auf moderne Rechnerarchitekturen einen spürbaren Leistungszuwachs bringen kann. Hier wird durch erhöhte Lokalität der benötigten Daten innerhalb einer Seite die Effizienz der Speicherhierarchie verbessert und die parallele Abarbeitung der Prädikatauswertung durch Pipelining in modernen Prozessoren erhöht.

Datengateway: Eine weitere Eigenschaft erweiterbarer Indexe ist, dass die Daten auch außerhalb des Datenbanksystems liegen können. Damit stellt der Index einen Datengateway dar, der z.B. zu einem RDBMS, ORDBMS oder XML DMBS eines anderen Herstellers führen kann, Zugriff auf ein Flatfile erlauben kann oder Daten aus dem Netz zur Verfügung stellt. Dies kann nützlich sein für seltene Zugriffe auf legacy Systeme oder um die externen Daten mittels des vollen funktionalen Umfangs von SQL zu konvertieren und in eine lokale Relation zu importieren. Das Vorhandensein von standardisierten Treibern (z.B. ODBC, OLE DB, JDBC, C-ISAM) für eine Vielzahl solcher Zugriffe erlaubt die Bereitstellung einer großen Palette von Zugriffmöglichkeiten durch Implementierung jeweils eines einzigen Wrappers für jede dieser Schnittstellen.

XML: Die Unterstützung der effizienten Suche in XML-Dokumenten und deren Datenhaltung in einem relationalen Datenbanksystem stellt eine neue Herausforderung an die bestehenden Systeme dar. Eine universell erweiterbare Zugriffsstruktur und Datenorganisation würde eine Integration sicherlich erleichtern und eine an das Datenbanksystem gekoppelte Anwendung auch auf zukünftige Anforderungen vorbereiten.

Spezifikation

Die Architektur einer solchen Schnittstelle soll an das bekannte COM-Schnittstellen-System angelehnt sein. Dynamische Bibliotheken (Zugriffspfad-Treiber) exportieren eine Untermenge vordefinierter Schnittstellen. Ein neuer Treiber muss sich bei dem Zugriffstreiber-Manager des

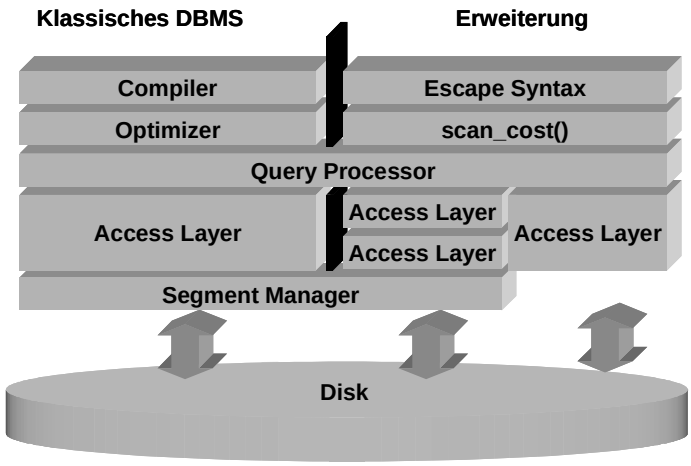

Abbildung 1: Gegenüberstellung des klassischen Schichtmodells eines DBMS und verschiedener Erweiterungen bei Kombination von Zugriffsmethoden und bei interner sowie externer Datenhaltung.

Datenbanksystems registrieren und wird auf Kompatibilität und Funktionsumfang geprüft. Somit wird eine Menge solcher Treiber dann über den Treibermanager verwaltbar.
Alternativ kann ein solcher Treiber auch in Java implementiert und als Java Archiv zur Verfügung gestellt werden. Hierbei kann es zwar einerseits zu Performanceeinbußen kommen, andererseits bietet die Ausführung von interpretiertem Code ein hohes Maß an Verfügbarkeit und Datensicherheit.
Ein Treiber kann nach unten und oben hin tupel- und/oder seitenorientierte Schnittstellen zur Verfügung stellen, so dass verschiedene Treiber kombinierbar bzw. derselbe Treiber kaskadiert verwendet werden kann. Dabei liefert der „oberste“ Treiber Tupel an die Query Prozessor Schicht. Die unterste Zugriffschicht kann seitenorientiert auf der datenbankeigenen Speicherschicht aufsetzen oder eine eigene Möglichkeit der persistenten Speicherung bieten (vgl. Abb. 1).
Eine denkbare Erweiterung ist zudem, einen mengenorientierten Austausch von Tupeln oder Seiten an den Schnittstellen auf Zugriffsebene zu ermöglichen, um so die Lokalität bei der Abarbeitung eines Zugriffsplans innerhalb der einzelnen Schichten zu erhöhen und gleichzeitig Ansatzpunkte zur Entkopplung und zum parallelen Abarbeiten eines Query-Plans auf modernen Rechensystemen zu ermöglichen.
Die datenbankeigenen Zugriffsmodule (verschiedene B-Baum Varianten, unsortierte Tupelcontainer) sollen die vereinbarte Schnittstelle ebenfalls unterstützen und werden ggf. weiter modularisiert. Sie stehen somit als Basis für schlanke Indexerweiterungen, wie etwa funktionale Indexe, zur Verfügung.
Jeder Treiber muss zur Integration in die darüber liegenden Datenbankschichten eine Reihe von weiteren Schnittstellen zur Verfügung stellen. Die Schnittstellen sollen im Folgenden in der Reihenfolge von oben nach unten kurz umrissen werden (vgl. Abb. 1):

• **Compilerebene:** Erweiterte Zugriffsmethoden werden möglichst keine Änderungen auf SQL Ebene mit sich bringen, dies gilt insbesondere für die DML. Allerdings muss zumindest beim Kreieren eines Zugriffspfads die Möglichkeit geboten werden einen Indextyp zu spezifizieren.

```
CREATE [<indextype>] TABLE <relname> (<fieldspeclist>);
CREATE [<indextype>] INDEX <idxname> ON <relname> (<fieldnamelist>);

CREATE filetype TABLE filetab (a CHAR(*), b INTEGER);
```

Außerdem ist die Vereinbarung einer Spracherweiterung zum Erstellen neuer Indextypen notwendig.

```
CREATE [PRIMARY|SECONDARY] INDEXTYPE <indextype> USING
      <dynamic library>;
DROP INDEXTYPE <indextype>;
```

Zusätzlich könnten aber sowohl in DDL als auch in DML Erweiterungen notwendig werden, um Index Spezifikationen möglich zu machen die mittels SQL nicht beschrieben werden können. Hierzu stellt ein Zugriffsmodul eine Schnittstelle **`parse()`** zum Parsen eines Teils eines SQL Statements zur Verfügung**,** formuliert in einer Escape-Syntax des SQL-Compilers, die falls notwendig den gewünschten Treiber auswählt. Diese SQL Erweiterungen können in einem dafür vorgesehenen Feld im Systemkatalog hinterlegt werden und stehen somit auch für die Indexwartung zur Verfügung.

```
CREATE filetype INDEX filetab_ix {store external $HOME/f.ix}
      ON filetab (b);
SELECT a FROM filetab WHERE P(b) {filetype: use filetab_ix};
```

Eine solche Sequenz kann frei im SQL Statement platziert werden und außerdem weitere, z.B. auch geschachtelte Escape-Sequenzen enthalten, die vom SQL-Compiler an die entsprechenden darunter liegenden Zugriffsmethoden weitergereicht werden.

• **Optimiererebene:** Zur Optimierungszeit muss bekannt sein, welche Zugriffpfade für eine Relation zur Verfügung stehen. Diese Informationen stellt der Systemkatalog für die erweiterten Zugriffpfade bereit. Die Einträge werden beim Erstellen eines Indexes durch entsprechende DDL Statements vorgenommen. Zusätzlich muss ersichtlich sein, welche Prädikate ein solcher Index auf den verfügbaren Feldern auswerten kann. Dies geschieht mittels benutzerdefinierter Funktionen, die einem Zugriffspfad bei dessen Registrierung von dem Treibermanager zugeordnet werden. Hier kann wahlweise die Präfixschreibweise mit dem Funktionsnamen oder die in SQL übliche Infixschreibweise mit dem entsprechenden Infixoperator verwendet werden, z.B. equals (=), lt (<), LIKE usw.

Schließlich können die Zugriffspfadmodule eine Kostenfunktion **scan_cost()** zur Verfügung stellen, die für ein gegebenes Prädikat eine Abschätzung der Zugriffsdauer berechnen.
Somit ergibt sich für den Optimierer für den Fall eines Zugriffs auf eine Relation $R(a_0,...,a_n)$ mit dem Prädikat $P(a_{P0},...,a_{Pm})$, $a_{Pi} \in \{a_0,...,a_n\}$ und der Projektionsliste $\pi(a_0,...,a_n)$ folgender Algorithmus:

$\forall$ Indexe $I(a_{I0},...,a_{Ik})$, $a_{Ij} \in \{a_0,...,a_n\}$ von R:
    Wenn $P(a_{P0},...,a_{Pm})$ kann ausgewertet werden:
        Berechne scan_cost(R,P,π)
Wähle Index mit geringsten Kosten.
Materialisiere wenn $\pi(a_0,...,a_n) \not\subset \{a_{I0},...,a_{Ik}\}$.

• **Query Prozessor:** Hier werden keine neuen Routinen benötigt. Der Query Prozessor bedient bei der Ausführung des Queryplans nach Bedarf die Schnittstellen der Zugriffsebene.
• **Zugriffsebene:** Die weitaus umfangreichste Schnittstelle wird zur Ausführungszeit benötigt. Es müssen Routinen zum Erstellen (**index_create()**) und Löschen (**index_drop()**) eines Indexes verfügbar sein. Für die Indexwartung werden zumindest die Schnittstellen **index_insert()** und **index_delete()** benötigt. Falls ein Index eine eigene effiziente Aktualisierung unterstützt, kann er zusätzlich **index_update()** zur Verfügung stellen. Die Initialisierung und Aufräumroutinen **index_open()** und **index_close()** werden vor dem ersten Zugriff bzw. beim Schließen eines Zugriffspfades gerufen. Zusätzlich kann ein Index über den aktuellen Transaktionszustand informiert werden, wenn er **index_begin_ta()**, **index_commit_ta()**, **index_prepare_ta()** und **index_abort_ta()** implementiert. Dies kann nützlich sein, falls ein Index eigene Undo Informationen vorhalten und ggf. anwenden muss oder sich nicht auf die datenbankeigene Speicherungsschicht abstützt. Schließlich werden für den eigentlichen Zugriff **index_scan_open()**, **index_scan_fetch()**, **index_scan_reset()** und **index_scan_close()** benötigt.
• **Segmentebene:** Soll ein Zugriffspfad nach oben hin seitenorientiert arbeiten, so muss er statt der Schnittstelle der Zugriffsschicht die entsprechende Schnittstelle der Speicherungsebene implementieren. Dies sind im Einzelnen die Routinen zur Wartung eines Segments **index_seg_create()** und **index_seg_drop()**. Die Bereitstellung von Routinen zur Transaktionssteuerung ist verpflichtend: **index_seg_begin_ta()**, **index_seg_commit_ta()**, **index_seg_prepare_ta()** und **index_seg_abort_ta()**. Da die Implementierung einer neuen Speicherungsschicht externe Datenhaltung und damit verteilte Transaktionen impliziert, z.B. kann ein solcher Treiber ein Gateway zu externen Flatfiles, anderen relationalen Datenbanksystemen oder Webservices darstellen, sollte ein solcher Treiber ein 2-Phasen Commit für schreibende Transaktionen unterstützen. Kann dies ein Treiber nicht gewährleisten, so ist in einer Transaktion bei der erstmalig in ein solches Segment geschrieben wird nur noch das Schreiben in diesem Segment erlaubt. Somit wird für diese Transaktion die Notwendigkeit eines verteilten Commits verhindert. Soll in einer bereits verteilten Transaktion auf ein solches Segment schreibend zugegriffen werden, so führt dies zu einem Fehler.
Die Handhabung von Seiten wird mittels **index_page_alloc()** und **index_page_delete()** gewährleistet. **index_page_fix()** sorgt dafür, dass eine benötigte Seite ggf. geladen und im Datenbankcache verfügbar gemacht wird. Zudem werden hier Schreib-/Lesezugriffe synchronisiert. Schließlich gibt **index_page_unfix()** eine geladene Seite wieder frei, wenn sie von der Zugriffsschicht nicht mehr prozessiert werden und wieder aus dem Datenbankcache verdrängt werden kann.

Vergleichbare Projekte

Informix DataBlades: Die mächtigste Schnittstelle zum Datenbanksystem bietet zur Zeit IBM mit seinem Informix Datenbanksystem an. Das Konzept der DataBlades erlaubt einem Entwickler auf fast allen Schichten des Systems Erweiterungen einzubauen. Zudem runden User-Defined Data Types und User-Defines Routines das Gesamtbild ab. Allerdings war die Zielsetzung bei der Konzeption der Schnittstellen Virtual Table Interface (VTI) und Virtual Index Interface (VII) die Möglichkeit zu bieten, externe Daten in einer ‚virtuellen' Tabelle im Datenbanksystem verfügbar zu machen. Speicherung innerhalb der Datenbank ist nur sehr eingeschränkt möglich. Zudem werden essenzielle Datenbankkonzepte wie Transaktionschutz, Locking, Logging & Recovery nicht oder unzureichend unterstützt und sind dann nur mit erheblichem implementatorischen Aufwand realisierbar.

Oracle DataCartridges: Mit Version 8i bietet Oracle zum ersten mal DataCartridges zur funktionalen Erweiterung des Datenbanksystems an. Umsetzung und Funktionsumfang ist den DataBlades weitgehend ähnlich. Dabei werden auch die in Informix fehlenden Datenbankkonzepte implementiert. Allerdings ist das Oracle DataCartridge Interface (ODCI) primär für die Implementierung von funktionalen Sekundärindexe eines einzelnen Feldes über den Lookup von RowIDs konzipiert und hat damit gravierend Mängel beim Einsatz als Schnittstelle für universelle Indexstrukturen, weil stets eine Materialisierung der Ergebnismenge notwendig ist.
Open Source Datenbanken: Diese Datenbanksysteme lassen jede Möglichkeit Eingriffe am Code vorzunehmen. Allerdings bieten sie keine klar definierte Schnittstelle an, die es erleichtert verschiedene Zugriffsmethoden mit möglichst wenig Aufwand vergleichbar und austauschbar zu implementieren.
Generalized Search Trees: GiST ist ein Framework das ein minimale Schnittstelle definiert um erweiterte Indexierung zu implementieren und findet Verwendung bei der Prototypimplementierung neuer Zugriffmethoden. Die Funktionalität eines Datenbanksystems fehlt dabei vollständig. Alle GiST Index Implementierungen lassen sich mittels eines einzigen Wrappers über die hier beschriebene Schnittstelle bedienen und stellen somit einen reichhaltigen Indexpool für Vergleichsmessungen zur Verfügung.

Zusammenfassung und Ausblick

Obwohl es bereits vergleichbare Produkte gibt, scheint keines dem Anspruch gerecht zu werden, eine vollständige Schnittstelle zur Erweiterung, Kombination und Vergleich von Indexmethoden in einem relationalen Datenbanksystem zu genügen.
Die Datenhaltung der Indexstruktur außerhalb der Datenbank ist stets mit Einschränkungen verbunden. Etwa werden Transaktionen und Mehrbenutzerbetrieb auf verschiedenen Isolationsebenen, sowie Logging und Recovery zum Teil nicht unterstützt. Selbst bei Speicherung innerhalb des Datenbanksystems leiden die Ansätze unter Einschränkungen der Funktionalität der angebotenen Schnittstellen, da diese unter anderen Zielsetzungen entworfen wurden als die, die in diesem Paper erörtert werden.
Trotzdem beweisen VTI/VII und ODCI dass es durchaus möglich ist, eine nach oben hin vollständige Schnittstelle für Indexwartung und -zugriff zu realisieren. Beiden Ansätzen fehlen jedoch Schnittstellen nach unten hin, zu Segment- und Lock-Manager des DBMS, die die verbleibenden Einschränkungen aufheben und vollwertig erweiterbare Indexstrukturen ermöglichen würden. Dabei sind keine Änderungen in der Anfragesprache und nur geringe Erweiterungen bei der Definitionssprache erforderlich.
Die Vergleichbarkeit existierender und neuer Indexstrukturen erlaubt mit diesem System weiterführende Untersuchungen unter neuen Gesichtspunkten auf dem Gebiet der Zugriffsmethoden.

Referenzen

IBM Informix Virtual-Table Interface, Version 9.4, IBM Corp., Part No. CT1TDNA, March 2003, http://www-306.ibm.com/software/data/informix/pubs/library/interim/ct1tdna-pdf.html
IBM Informix Virtual Index Interface, Version 9.4, IBM Corp., Part No. CT1TCNA, March 2003, http://www-306.ibm.com/software/data/informix/pubs/library/interim/ct1tna-pdf.html
Oracle9i Data Cartridge Developers Guide, Release 2 (9.2), Oracle Corp., Part No. A96595-01, March 2002
The GiST Indexing Project, http://gist.cs.berkeley.edu:8000/gist/