Technischer Bericht des Instituts für Informatik der Universität Rostock: CS-02-15, ISSN: 0944-5900

erhältlich unter http://www.ls-dbis.de/digbib/dbis-tr-cs-02-15

Optimierung objektorientierter Anfragen: Das Projekt CROQUE

Joachim Kröger, Andreas Heuer, mit Denny Priebe, Universität Rostock, Institut für Informatik
Kontaktadresse: heuer@informatik.uni-rostock.de

Zusammenfassung

Dieser Artikel beschreibt einige Aspekte der Optimierer-Komponente des CROQUE–Systems. Das im CROQUE–Projekt entwickelte objektorientierte Anfrageverarbeitungs-Framework bietet eine deskriptive Anfrageschnittstelle auf Basis von ODMG OQL, physische Datenunabhängigkeit, wie sie in heute kommerziell verfügbaren Produkten noch nicht umgesetzt wurde und den hier beschriebenen Anfrage–Optimierer, dessen Integration in das Gesamtprojekt vorgesehen ist. Insgesamt bietet der Anfrage–Optimierer einige interessante neuartige Features, die die Effizienz des Anfrage-Rewriting und der Anfrage–Auswertung positiv beeinflussen sollen.

Schlüsselwörter: Anfrage–Optimierung, Algebra, Rewrite-Regeln, Optimierungsablauf, Query Rewriting, Suche, Kostenmodell, Heuristiken

Abstract

This paper describes several aspects of the optimizer component of the CROQUE query rewrite and optimization system. The developed query optimizer may be integrated into CROQUE's object–oriented query processing framework further offering two main characteristics: a descriptive query language based on ODMG's OQL and physical data independence not yet sufficiently supported by nowadays commercial database systems. Summarizing the key features of the query optimizer, CROQUE provides database technology with some exciting new concepts increasing the efficiency of query rewriting and evaluation.

Key words: Query optimization, algebra, rewrite rules, optimization process, query rewriting, search, cost model, heuristics

CR Subject Classification: H.2.3, H.2.4

1 Zielsetzung des Projekts (Ausgangsfragen)

Im Rahmen des Projektes CROQUE[1] wurden Optimierungsverfahren für objektorientierte Datenbankmanagementsysteme (OODBMS) entwickelt. Im Mittelpunkt der Arbeiten standen dabei Untersuchungen zu Verfahren des "Query–Rewriting" für algebraische und kalkülartige Anfragerepräsentationen, Auswertungsverfahren für objektorientierte Anfragen sowie die Optimierung des physischen Entwurfs von OODBMS bezüglich eines gegebenen Anwendungsprofils.

Ziel dieses Projektes war die Entwicklung eines statischen, d.h. zur Übersetzungszeit einer Anfrage arbeitenden, Anfrageoptimierers. Die geschaffenen prototypischen Teil–Lösungen, deren Integration und abschließende Evaluierung im Rahmen der restlichen Projektlaufzeit vorgesehen ist, realisieren insgesamt ein objektorientiertes Anfragebearbeitungssystem mit den folgenden drei Hauptmerkmalen:

- einer deskriptiven, objektorientierten Anfrageschnittstelle auf Basis von ODMG–OQL [3],
- physischer Datenunabhängigkeit (d.h. Trennung der logischen Strukturen von den physischen Speicherstrukturen und Zugriffspfaden), die in den bisherigen objektorientierten Datenbanksystemen nicht und in relationalen Systemen nur unzureichend gegeben ist, und
- einer regel– und kostenbasierten Optimierungskomponente zur Abbildung der deskriptiven Anfrage– und Änderungsoperationen der konzeptuellen Ebene auf einen Ausführungsplan in Abhängigkeit von der gewählten Speicherstruktur.

Die Entwicklung adaptiver Verfahrensweisen für den CROQUE–Optimierer ist noch nicht abgeschlossen; ihre

[1] CROQUE (*C*ost– and *R*ule–based *O*ptimization of object–oriented *QUE*ries) ist ein Kooperationsprojekt der Universitäten Rostock und Konstanz. CROQUE wurde/wird von der DFG unter den Aktenzeichen He 1768/5–2 bzw. Scho 554/1-2 gefördert.

Integration wird aber bereits durch laufende Untersuchungen zur Modularisierung des Anfrageprozesses mittels eines Optimiererbaukastens vorbereitet. Der verfolgte regelbasierte Optimierungs–Ansatz eignet sich auch zur Unterstützung der Durchführung der geplanten Adaption.

Der hier beschriebene Ausschnitt aus dem Gesamt–Projekt konzentriert sich in erster Linie auf den Entwurf und die Realisierung des Anfrageoptimierers. Dieser Artikel ist folgendermaßen gegliedert: Abschnitt 2 gibt einen kurzen Blick auf Related Work. Anhand der in Abschnitt 3 vorgestellten CROQUE–Architektur wird die Einordnung der durchgeführten Arbeiten in den Projekt-Rahmen ermöglicht. Es wird dann zunächst ein kurzer Überblick über die Entwicklung der durchgeführten Arbeiten gegeben; auf einige der schwerpunktmäßig zum Anfrageoptimierer gehörenden Aspekte wird in Abschnitt 4 etwas detaillierter eingegangen. Anschließend werden in Abschnitt 5 mögliche Anwendungsperspektiven und in Abschnitt 6 denkbare Folgeuntersuchungen aufgeführt, bevor in Abschnitt 7 eine abschließende Zusammenfassung zum nun beendeten Teilprojekt "Anfrageoptimierer" gegeben wird.[2]

2 Related Work

In diesem Abschnitt betrachten wir ausgewählte Arbeiten, die die Ansätze von CROQUE betreffen, zunächst von Seiten der Plattformen wie objektorientierten Datenmodellen und deskriptiven Anfragesprachen, dann von Seiten der Optimierungsansätze in OODBMSen, und schließlich aus der Sicht spezieller Optimierungstechniken und unterstützender Maßnahmen wie Monoid–Kalkülen, Suchstrategien und Kostenmodellen.

Objektorientiertes Datenmodell. Das ODMG–Modell, bzw. die ODMG–ODL (Object Definition Language), standardisiert nur eine Richtung objektorientierter Datenbankmodelle: das Modell ist sehr stark auf objektorientierte Datenbankprogrammiersprachen ausgerichtet und läßt einige der für den Datenbankbereich wichtigen Flexibilität vermissen [16].

Das ODMG–Modell wurde deshalb zwar für das CROQUE-Modell als Grundlage akzeptiert, jedoch in einzelnen Aspekten verfeinert, um diese Datenbankerfordernisse, die sich insbesondere bei Anfragen auswirken, zu erfüllen.

Deskriptive Anfragesprachen. Obwohl ODMG–OQL als Standard für eine Anfragesprache eingeführt wurde [3], hat sich an der Rolle der Anfragesprachen in kommerziellen OODBMSen nichts geändert. Anfragesprachen werden derzeit nur von einem System vollständig (O_2) und von anderen in Ansätzen (etwa POET, Versant) unterstützt. Viele Systeme bleiben bei speziellen Klassen und Methoden zur Realisierung weniger Anfrageoperationen, wie etwa auch der Marktführer ObjectStore.

Optimierungsansätze in OODBMSen. Die in letzter Zeit verstärkt vorangetriebene Entwicklung deklarativer Anfragesprachen für prototypische OODBMS, vor allem in Form SQL–artiger Sprachen, hat die Konzeption und Realisierung von Optimierern für diese Sprachen motiviert.

Das Projekt EREQ entwickelte zeitlich parallel zu CROQUE die Algebra AQUA [24], die das Konzept der Funktionsanwendung in den Vordergrund stellt. Es existieren AQUA–Optimierer, die jeweils auf Erfahrungen aus EXODUS, Encore und Revelation basieren.

Monoid–Kalküle. OODBMS–Anfrageoperationen werden von einigen Forschergruppen mittels Kalkülen formalisiert. Hervorzuheben sind hier die Arbeiten [2, 6, 33]. Monoide sind als sehr allgemeine algebraische Strukturen in der Lage, Operationen sowohl auf Collection–Typen, als auch auf Skalaren uniform zu repräsentieren. Die Definition von Anfragealgebren, die jeweils spezialisierte Operatoren für die diversen auftretenden Typen und Operationen beinhalten müssen, wenn sie die konzeptuelle Anfragesprache vollständig repräsentieren sollen, läßt sich so vermeiden (z.B. im EREQ–Projekt [24] hatte dies zu Sprachen mit mehr als 100 Operatoren geführt).

Optimierungsverfahren für Kalkülansätze stehen noch am Beginn ihrer Entwicklung. Die in [6] vorgestellte Normalform hilft nur bedingt (siehe dazu auch [12]). Auch die in [34] vorgestellten Optimierungsideen gehen nicht über Umordnung und pushdown von Selektionen und Projektionen hinaus.

Suchstrategien. Zur Verwendung in OODBMS müssen die aus relationalen DBMS bekannten Such–Strategien angepaßt werden, wobei vor allem die Unterstützung der Suche unter mächtigeren Plänen (auf Basis aller vorhandenen Operatoren) deutliche Unterschiede gegenüber den in relationalen Systemen verwendeten Strategien erfordern.

In [9] wird der Suchraum heuristisch aufgespannt, d.h. während der Generierung von Ausführungsplänen wird auf Basis des Kostenmodells und einer Heuristik entschieden, ob ein Plan erzeugt wird. Auf diese Weise wird verhindert, daß nicht zu beherrschende Suchräume tatsächlich aufgespannt werden. Dies ist auch die Strategie von CROQUE, wobei hier die heuristische Suchraumaufspannung noch zusätzlich mit unterschiedlichen Suchstrategien kombiniert wird.

Die Verwendung von Optimierer–Generatoren wie Volcano [10] beschleunigt zwar einerseits die Implementierung, führt aber andererseits auch dazu, daß eine vorge-

[2] Dieser Artikel entspricht in wesentlichen Teilen dem Abschlußbericht des Rostocker Teilprojekts, der im Januar 1999 für die DFG erstellt wurde.

fertigte Suchmaschine verwendet wird, deren Strategie im wesentlichen nicht beeinflußt werden kann [1, 4, 5].

OODBMS–Kostenmodelle. Die Kostenmodelle für relationale Datenbanksysteme sind sehr weit entwickelt. Meist werden die Kosten an Operatoren der verwendeten (physischen) Algebra gebunden (etwa in [5, 9]) und den Anfrageplänen statisch zugeordnet.

Die Notwendigkeit von OODBMS–Kostenmodellen im Rahmen einer effizienten Anfrageoptimierung steht außer Frage. Zumeist werden in OODBMS–Prototypen vor allem Heuristiken als Entscheidungsgrundlage für die Auswahl von Ausführungsplänen verwendet. Ein Gegen–Beispiel ist das Kostenmodell aus [1], bei dem Ausführungspläne — wie in CROQUE — nicht nur auf Basis von Heuristiken ausgewählt, sondern auch auf Basis von Kosten miteinander verglichen werden.

3 Entwicklung der durchgeführten Arbeiten

Zunächst wird in Unterabschnitt 3.1 die CROQUE-Architektur kurz dargestellt, bevor die im Projekt–Rahmen durchgeführten Arbeiten im folgenden Unterabschnitt 3.2 beschrieben werden. Es wird hierbei nicht die tatsächliche zeitliche Abfolge der einzelnen Arbeiten wiedergegeben, sondern aufgrund der besseren Lesbarkeit sind zusammengehörige Aspekte gruppiert und in einer pragmatischen Reihenfolge angegeben. In Unterabschnitt 3.3 wird eine Abweichung vom ursprünglichen Konzept und in Unterabschnitt 3.4 wissenschaftliche Fehlschläge kurz erläutert.

3.1 CROQUE–Architektur

Die CROQUE–Architektur läßt sich in zwei ineinander geschachtelte Bereiche aufgliedern.

Die (mehr physische Sicht der) Systemarchitektur (Abb. 1) umfaßt die (mehr logische Sicht der) Optimiererarchitektur (Abb. 2). Der *Query Optimizer* in Abb. 1 und seine in Abb. 2 spezifizierten Komponenten sind durch eine dunklere Schraffur hervorgehoben. Pfeile in den Architekturskizzen kennzeichnen die Aufrufhierarchie.

Im CROQUE–Projekt existieren derzeit zwei Prototypen: der eine besteht aus der Implementierung der flexiblen internen Ebene (Storage–Manager, festes Metaschema), der andere aus den in Abb. 2 dargestellten Komponenten. Dieser zweite Prototyp implementiert einen hybriden Auswertungs–Ansatz aus Kalkül und Algebra.

User–Interface. Über das User–Interface werden alle Benutzerinformationen an die entsprechenden Subsysteme weitergeleitet und Ergebnisse dargestellt.

Tree–Viewer. Der Tree–Viewer ist ein Teil des User-Interfaces. Er dient der graphischen Visualisierung von Schemadaten wie Klassenhierarchien, sowie von Anfragepläne. Desweiteren dient der Tree–Viewer als Debugging–Tool für die Darstellung optimierter Anfrage- und Ausführungspläne.

MVDL–Parser. Mit dem Materialized View Definition Language–Parser können physische Schemainformationen in das Datenbank-Metaschema eingebracht und geändert werden (physische Schemaevolution). Dieses System umfaßt die Möglichkeit, physische Strukturen anzulegen, zu löschen, sowie Rekonstruktionsvorschriften zu (re–)definieren. Der MVDL–Parser prüft dabei die Typkorrektheit und Verlustfreiheit der Abbildung (sofern entscheidbar) und instantiiert die definierten physischen Variablen.

ODL–Parser. Mit dem ODL–Parser kann logische Schemainformation in das Metaschema des Datenbanksystems eingebracht werden. ODL–Definitionen umfassen hierbei sowohl Strukturinformationen, als auch Methoden in OQL oder C++.

OML–Parser, Update–Transformer. Der OML–Parser analysiert logische Wertänderungsoperationen und transformiert sie mit Hilfe des Query Optimizers in physische Änderungs–Ausführungspläne, die vom Query– und Updateprozessor interpretiert werden können.

OQL–Parser, Typ–Checker. Der OQL–Parser überprüft gestellte Benutzeranfragen gegen das logische Schema auf Typkorrektheit (Typinferenz für freie Variablen und Subqueries aufgrund logischer Schemainformation und der Semantik von CROQUE–OQL [30]) und leitet sie zur Transformation an den Optimierer weiter.

Query Optimizer. Der Query Optimizer ist für den gesamten Opimierungsprozeß verantwortlich. Er wird in Abb. 2 im Detail zergliedert. Der Anfrage-Optimierer erzeugt physische Ausführungspläne, die vom Query–Prozessor interpretiert werden können. Da die Komponenten des Anfrage–Optimierers später im Detail erläutert werden, wird an dieser Stelle auf die entsprechenden Abschnitte verwiesen: geparste und auf Typkorrektheit geprüfte OQL–Anfragen werden zunächst in Algebra–Ausdrücke (rein algebraischer Ansatz) bzw. eine Mischung aus Kalkül– und Algebra–Ausdrücke (hybrider Ansatz) übersetzt (siehe Unterabschnitt 4.1). Diese Ausdrücke werden mittels Rewriting umgeformt (4.2, 4.3). Dadurch wird der Suchraum aufgespannt (4.6), in dem später mittels Suchverfahren der auszuführende Plan bestimmt wird (der Aspekt Suche wird in Unterabschnitt 4.4 behandelt). Dazu wird eine Kostenbewertung mittels eines Kostenmodells durchgeführt (4.5).

Schema–Manager. Der Schema–Manager stellt die Schnittstelle zum Metaschema zur Verfügung. Hier können physische und logische Schemainformationen abgerufen und eingebracht werden.

festes Metaschema. Das feste Metaschema ist eine fest verdrahtete Version des Metaschemas, welches später durch eine ODL–Definition ersetzt wird. Über diese

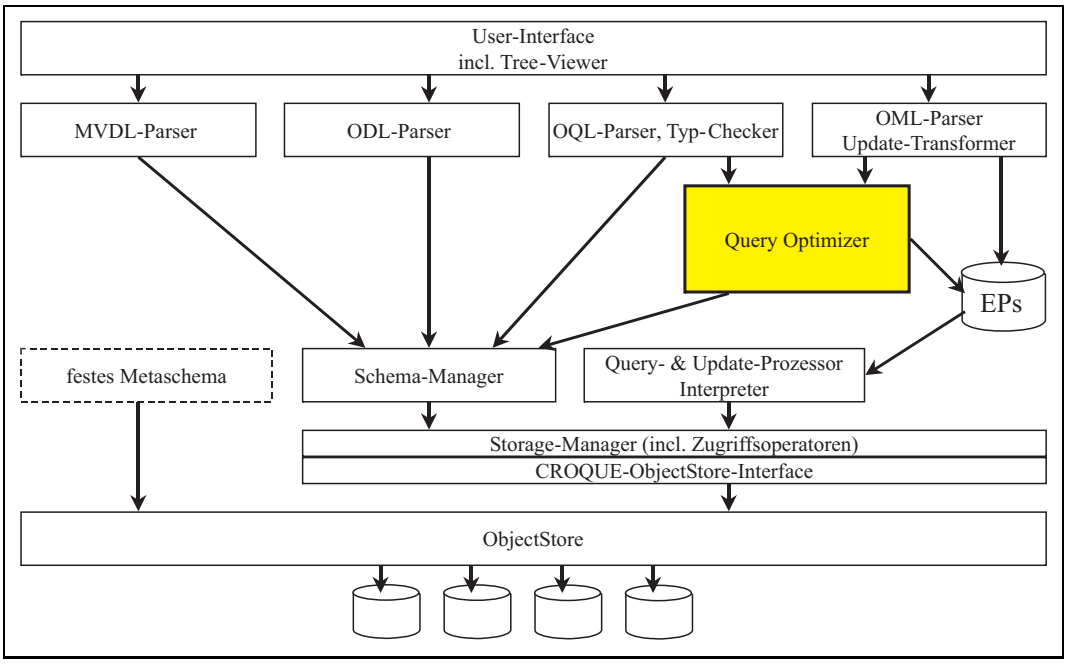

Abbildung 1 Systemarchitektur

temporäre feste Implementierung wird der Metaschema-Bootstrap sichergestellt.

Query– und Update–Prozessor. Diese Komponente führt die erzeugten Ausführungspläne auf dem Storage-Manager aus.

Storage–Manager (incl. Zugriffsoperatoren). Der Storage–Manager stellt elementare Zugriffe und Navigations– und Update–Operationen für die erweiterten Datenstrukturen des CROQUE–ObjectStore–Interfaces zur Verfügung.

CROQUE–ObjectStore–Interface. Dieses Interface erweitert die von ObjectStore bereitgestellten Datenstrukturen unter anderem um verschiedene Clusterungsmöglichkeiten.

ObjectStore. ObjectStore ist ein kommerzielles Objektspeichersystem der Firma Object Design.

Arbeiten zu den wesentlichen Komponenten des Anfrage-Optimierers werden in den nachfolgenden Abschnitten aufgeführt.

3.2 Behandelte Themenkomplexe

Themenkomplexe, die federführend in Konstanz bearbeitet wurden, werden aufgrund von Platz–Restriktionen hier jeweils nur in einem kurzen Absatz charakterisiert und in den Rahmen des Projektes eingeordnet. Dabei werden auch die wesentlichen Veröffentlichungen direkt angegeben.

Diejenigen Aspekte, die hauptsächlich in Rostock bearbeitet wurden, werden im nachfolgenden Abschnitt 4 noch einmal ausführlicher in eigenen Unterabschnitten dargestellt. Auf die Angabe von Veröffentlichungen zu diesen Aspekten wird daher in diesem Abschnitt zunächst noch verzichtet.

- *Formalisierung von Datenmodell und Anfragesprache.*
  Die in Konstanz durchgeführten Grundlagen-Arbeiten zu dieser Thematik wurden in [30,31] publiziert. Die notwendige Formalisierung des in [3] nur unzureichend beschriebenen Ansatzes bildet eine stabile Basis für die Definition der verwendeten internen Anfragerepräsentationen.
- *Definition von Kalkül und variablenfreier Algebra (als interne Anfragerepräsentationen).*
  Neben der in Konstanz durchgeführten Definition einer neuartigen, kalkülartigen Anfragerepräsentation [12] wurde in Rostock eine variablenfreie Algebra definiert, deren Vollständigkeit bezüglich der formalisierten Anfragesprache durch Angabe einer Abbildungsfunktion von der Anfragesprache in die Algebra nachgewiesen wurde.
- *Analyse der Rewrite–Regeln.*
  Die für die zunächst verwendete variablenbehaftete Algebra vorhandenen Rewrite–Regeln wurden in Rostock — soweit möglich — an die neue variablenfreie Algebra angepaßt. Die Verwendung der variablenfreien Algebra gestattet die Spezifikation deklarativer Regeln. Dies war für die variablenbehaftete Algebra nicht möglich. Die Regeln wurden dabei zudem vom eigentlichen Optimierer entkoppelt und können nun auch mittels einer geeigneten Regelverwaltung in den Optimierer eingebracht werden. Hierdurch wurde somit die technische Grundlage für eine Adaption des Optimierers geschaffen. Die Entkopplung ermöglicht

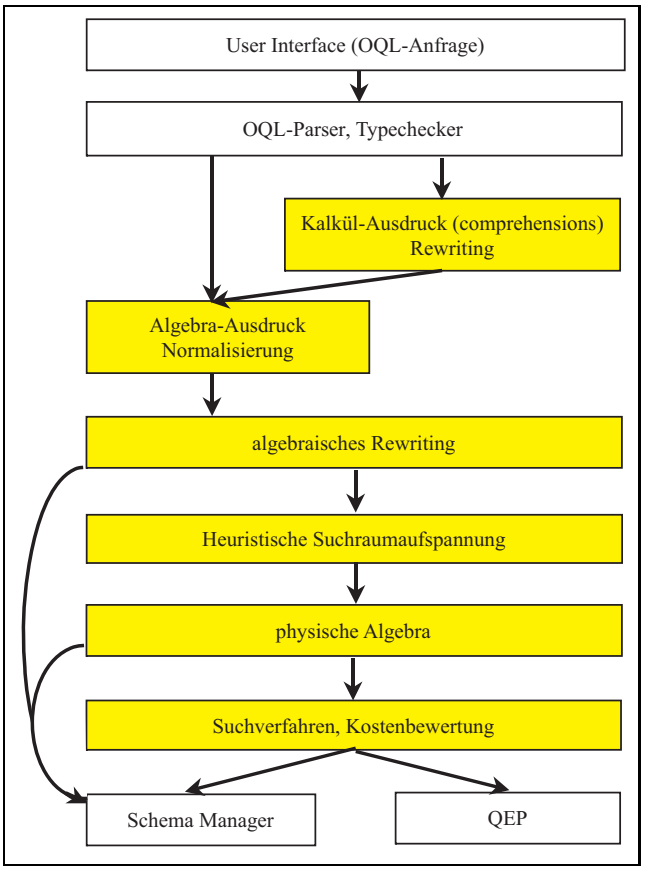

Abbildung 2 Optimiererarchitektur (*Query Optimizer*)

aber insbesondere auch die Einbeziehung von Regeln, die die interne Ebene bzw. deren tatsächliche physische Strukturen beschreiben und damit eine Berücksichtigung der physischen Freiheitsgrade im Rahmen der Optimierung.

– *Konzeption eines hybriden Auswertungsansatzes basierend auf Kalkül und Algebra.*
Die in Konstanz durchgeführten Untersuchungen zu einem neuartigen, hybriden Auswertungsansatz, der bei der Optimierung die Vorteile von Kalkül und Algebra kombiniert und dadurch zu einer effizienten Anfrage–Auswertung führt, wurden in [11, 13] publiziert. Die homogene Fortführung dieses Ansatzes mittels des formalen Modells der Query Deforestation [14] resultiert in effizienten Ausführungsplänen, die ohne die sonst üblicherweise notwendige temporäre Speicherung von Zwischenergebnissen auskommen. Das Verfahren fügt sich dabei trotzdem nahtlos in den Optimierungsablauf ein.
– *Entwicklung eines verzahnten Optimierungsablaufs.*
Um eine bessere Steuerbarkeit des Anfrage-Rewritings zu erreichen, wurde in Rostock ein alternativer Optimierungsablauf konzipiert, der die streng zweiphasige Vorgehensweise von aufeinanderfolgendem logischen und physischen Rewriting aufgibt. Durch das verzahnte Aufspannen von logischer und physischer Dimension wird eine Form des Feedback zwischen diesen beiden Phasen realisiert. Der verzahnte Optimierungsablauf ist mit dem hybriden Auswertungsansatz und der Query Deforestation kombinierbar.
– *Untersuchungen zu Query Rewriting und Suche.*
In Rostock durchgeführte experimentelle Untersuchungen belegen, daß bereits einfache Heuristiken gute Ergebnisse bei der Steuerung der implementierten Suchstrategien erzielen können. Es wurde ein Algorithmus entwickelt, der die Einhaltung gewisser Grenzen beim Optimierungsaufwand zusichert. Die Grenzwerte gehen dabei als Parameter ein.
– *Evaluierung des Kostenmodells.*
Das Kostenmodell ist integraler Bestandteil der Suche nach dem günstigsten Ausführungsplan, da es die Bewertung und damit auch die Auswahl eines Plans erst ermöglicht. Das zunächst in Rostock aufgestellte Kostenmodell wurde evaluiert und soweit nötig auch an die zwischenzeitlich veränderte, flexible interne Ebene angepaßt. Es konnte gezeigt werden, daß auch das gegenüber dem ursprünglichen Kostenmodell wesentlich schlankere, einfachere und damit auch günstigere Kostenmodell bereits gute Kosten–Abschätzungen ermöglicht. Darüber hinaus wurde festgestellt, daß ein Kostenmodell, das lediglich für Optimierungsentscheidungen verwendet wird, keine exakten Kostenwerte liefern muß. Es reicht, wenn die Kostenrelation für je zwei betrachtete Pläne korrekt wiedergegeben wird.
– *Untersuchungen zur Verwendung von Heuristiken bei der Suchraumaufspannung.*
Die vorzeitige (heuristische) Einschränkung des logischen Suchraums auf Basis einer Ordnung der Rewrite-Regeln gemäß ihrem Optimierungspotential wurde formalisiert. Der in Rostock im Rahmen von CROQUE verfolgte Ansatz ist flexibler als die Ansätze herkömmlicher Systeme und erlaubt es, die Suche des günstigsten unter den äquivalenten Ausführungsplänen zu beschleunigen, was potentiell zu besseren Optimierungsergebnissen bzw. einem geringeren Optimierungsaufwand führt.
– *Analyse der Bewertungsmöglichkeiten von Anfrageoptimierungen.*
Zunächst wurde untersucht, inwieweit Benchmarks, die als klassisches Mittel zur Bewertung von Systemen verwendet werden, für die Bewertung von Anfrageoptimierungen sinnvoll sind. Benchmarks reduzieren die Leistung eines Systems auf eine (Menge von) Zahl(en) in einer gewissen Metrik und ermöglichen somit einen Vergleich verschiedener Systeme. Für den direkten Vergleich zumal von heuristisch gesteuerten Optimierungen haben sich Benchmarks jedoch als im wesentlichen ungeeignet erwiesen. Wesentlich erfolgversprechender sind hier die Verwendung von Statistiken, die sehr genau aber relativ teuer sind, und die Berechnung fiktiver Extrempunkte, die sehr einfach und daher auch günstig umgesetzt werden kann,

aber relativ ungenau arbeitet. Eine Kombination dieser beiden Ansätze ist als Mittel der Bewertung von Anfrageoptimierungen wünschenswert.

– *Entwicklung der internen Ebene.*
Die interne Ebene von CROQUE, die wesentlich flexiblere Speicherstrukturen anbietet, als dies bei kommerziellen Systemen der Fall ist, basiert auf der automatischen Materialisierung von Views mit dem Ziel, die Antwortzeit für wiederkehrende Anfragen (bzw. Anfragen, die einem gemeinsamen Muster folgen) zu verkürzen. Für das Problem der View Maintenance wurde in [8] ein Verfahren zum effizienten inkrementellen Update der konzeptuellen Daten publiziert, die im System nur als materialisierte Views vorliegen. Als View–Definitions–Sprache wird dabei die Anfragesprache von CROQUE verwendet, so daß sich die Berücksichtigung der auf der internen Ebene vorhandenen/verwendeten Speicherstrukturen nahtlos in den Optimierungsprozeß einfügt. Physische Aspekte der Anfrageoptimierung wurden exklusiv in Konstanz bearbeitet.

3.3 Abweichung vom ursprünglichen Konzept

Die zunächst geplante Selbstoptimierung des Systems wurde vorerst von der Thematik des CROQUE–Projektes abgetrennt. Fragen der Adaption von Optimierern werden eigenständig und projektübergreifend auf einem allgemeineren Level in einer laufenden, am Rostocker Lehrstuhl betreuten, externen Arbeit behandelt. Diese Arbeit wird wesentliche Grundlagen für die Entwicklung adaptiver Verfahren für CROQUE liefern.

3.4 Wissenschaftliche Fehlschläge

Die durchgeführten Implementierungen verwendeten zunächst ein auf der SML [15,25]–Erweiterung CRML [32] basierendes Optimierer–Framework, welches keinen konkreten Optimierungsablauf vorgibt und somit dem Implementierer die Bereitstellung einer geeigneten Realisierung erlaubt [7]. Nachdem praktische Erfahrungen gezeigt haben, daß eine unkontrollierte Anwendung logischer Äquivalenzen im Rahmen des Rewritings zu inakzeptablen Optimierungszeiten führte, wurde eine Suche nach Alternativen notwendig. Da sich eine Steuerung der Suchstrategie innerhalb des CRML–Optimierer-Frameworks jedoch eher schwierig gestaltet, wurde die Nutzung dieses Frameworks aus Effizienzgründen aufgegeben und stattdessen eine reine SML-Realisierung vorgezogen.

Wie bereits kurz skizziert wurde, haben sich Benchmarks zur Bewertung von Anfrageoptimierungen als im wesentlichen ungeeignet erwiesen [18]. Aus der mit Hilfe von Benchmarks ermittelbaren Performance eines Datenbanksystems lassen sich zunächst nicht ohne weiteres Aussagen über die Qualität einzelner Optimierungen, noch über die Qualität des Anfrageoptimierers treffen, da Benchmarks ein System als Ganzes testen und nicht einzelne Komponenten, wie z.B. den Anfrageoptimierer. Obwohl der Optimierer selbstverständlich das Ergebnis eines Benchmarktests beeinflußt, können keine Rückschlüsse auf dessen Qualität gezogen werden. Z.B. könnten schlecht optimierte Anfragen lediglich aufgrund einer sehr guten Pufferverwaltung fälschlicherweise sehr gute Testergebnissse liefern. Notwendig für eine korrekte Bewertung des Anfrageoptimierers oder einzelner Anfrageoptimierungen wäre vielmehr der Vergleich der Performance der zur Ausführung ausgewählten optimierten Anfrage mit der Performance des tatsächlich optimalen Anfrageplans. Aufgrund dieser Feststellungen wird daher im Rahmen von CROQUE eher auf eine Kombination anderer Verfahren gesetzt, die in Abschnitt 4.7 kurz dargestellt werden.

4 Darstellung der erreichten Ergebnisse und Diskussion im Hinblick auf den relevanten Forschungsstand

In diesem Abschnitt werden einige der behandelten Aspekte jeweils in eigenen Unterabschnitten dargestellt, wobei Verweise auf dazu entstandene Veröffentlichungen angegeben werden. Im einzelnen wird auf die Definition einer variablenfreien Algebra als interner Anfragerepräsentation (in Unterabschnitt 4.1), die Analyse der Rewrite–Regeln (4.2), die Entwicklung eines verzahnten Optimierungsablaufs (4.3), durchgeführte Untersuchungen zu Query Rewriting und Suche (4.4), die abgeschlossene Evaluierung des Kostenmodells (4.5), Untersuchungen zur Verwendung von Heuristiken bei der Suchraumaufspannung (4.6) und die Analyse der Bewertungsmöglichkeiten von Anfrageoptimierungen (4.7) eingegangen.

4.1 Definition einer variablenfreien Algebra als interner Anfragerepräsentation

Rewrite–Regeln variablenbehafteter Algebren bestehen aufgrund der Natur dieser Algebren im Allgemeinen nicht nur aus einer linken und einer rechten Regelseite, sondern zusätzlich auch aus Code–Fragmenten, die die Anwendbarkeit der Regeln testen und die Anwendung der Regeln vornehmen. Die zunächst in CROQUE verwendete variablenbehaftete logische Algebra wurde in [27,29] durch eine variablenfreie Algebra abgelöst, wodurch die Spezifikation deklarativer Regeln für den Optimierungsprozeß ermöglicht wurde.

Die Vorteile variablenbehafteter Algebren, also deren gute Lesbarkeit und die im Allgemeinen einfache Übersetzbarkeit aus der Anfragesprache, gingen mit der Einführung der variablenfreien Algebra verloren. Variablen werden bereits im Vorfeld der Übersetzung eines Ausdruckes der Anfragesprache in die variablenfreie Algebra eliminiert, ein Zugriff auf Werte erfolgt beispielswei-

se durch Funktionskomposition und explizites Mitführen einer Umgebung für jede Operation. Die typisierten Operationen der variablenfreien Algebra sind gegenüber den Operationen der variablenbehafteten Algebra strukturreicher, so daß ein Anfrageausdruck in eine größere Anzahl von Algebraoperationen übersetzt wird. Damit sind zwar die Vorteile variablenbehafteter Algebren verlorengegangen, die Notwendigkeit der expliziten Spezifikation von Code für Rewrite–Regeln wurde aber im Gegenzug vollständig beseitigt.

Da die variablenfreie Algebra lediglich der internen Anfragerepräsentation dient, überwiegen die gewonnenen Vorteile der einfacheren Handhabbarkeit bzw. Modifizierbarkeit von Regeln sowie der Wegfall der Notwendigkeit, Gültigkeitsbereiche von Variablen zu untersuchen, die Nachteile der schlechteren Lesbarkeit und der komplexeren Übersetzung, die sich aus den strukturellen Differenzen zwischen Anfragesprache und Algebra ergeben.

4.2 Analyse der Rewrite–Regeln

Im ersten CROQUE–Prototypen waren Regeln fest in die Optimiererkomponente integriert, so daß das Einbringen neuer Regeln eine Änderung des Optimierer–Codes zur Folge hatte. Durch die im Rahmen der Einführung der variablenfreien Algebra erfolgte Entkopplung der Regeln vom Optimierer und die durch den Paradigmen-Wechsel erzwungene Definition zu dieser Algebra passender Regeln (beides wurde in [27] beschrieben) hat sich die Situation grundlegend geändert.

Zunächst eröffnet die Entkopplung vielfältige Möglichkeiten zur Steuerung des Optimierungsablaufes sowie der Regelmodifikation und –einbeziehung, so daß eine Adaption und die Berücksichtigung der physischen Freiheitsgrade im Rahmen der Optimierung ermöglicht wurden. Dem Optimierer können — etwa für eine zu Beginn der Transformationen notwendige Normalisierung — Teil–Regelmengen übergeben werden, mit denen ein spezielles Ziel (z.B. die Überführung in eine ausgewählte Normalform) erreicht werden kann. Auf diese Weise wurde das System in Hinsicht auf die verwendeten Regel(menge)n modularisiert und erweiterbar gemacht.

Darüber hinaus können nun deklarative Regeln definiert werden. Die Vorteile der Verwendung deklarativer Regeln wurden bereits in Unterabschnitt 4.1 erläutert. Die Transformation von zunächst in CROQUE verwendeten Regeln in äquivalente Regeln für die neue variablenfreie Algebra hat gezeigt, daß sich die Anzahl der zum Ausdruck einer bestimmten Umformung benötigten Regeln in der variablenfreien Darstellungsform erhöht.

Neben den gewonnenen Möglichkeiten zur Steuerung der Regel–Einbeziehung kann auch durch eine gezielte Beeinflussung der Suche nach äquivalenten Plänen (s. Unterabschnitt 4.4) und die Weiterentwicklung von Heuristiken (4.6) der Nachteil der gewachsenen Regel–Anzahl aufgewogen werden.

4.3 Entwicklung eines verzahnten Optimierungsablaufs

Als Alternative zum traditionellen, streng sequentiellen, zweiphasigen Optimierungsablauf bestehend aus logischem gefolgt von physischem Rewriting wurde in [28] ein Verfahren mit Feedback entwickelt, bei dem das Aufspannen der logischen und physischen Dimension des Suchraums verzahnt vorgenommen wird. Auf diese Weise kann bereits während der logischen Optimierung die Qualität der vorgenommenen Umformungen durch Betrachten von physischen Plänen bewertet werden, um zukünftige Transformationen logischer Pläne geeignet zu steuern.

Eine Interaktion zwischen logischer und physischer Suchraumaufspannung wurde bereits in anderen Systemen realisiert. Die in CROQUE vorgenommene parametrisierte Implementierung des Verfahrens basiert in ihren Grundzügen auf der für das Projekt EXODUS entwickelten Vorgehensweise [9]. Die Probleme der in EXODUS realisierten Suchraumaufspannung, die durch die ausschließliche Anwendung der am besten bewerteten Rewrite–Regel auf den Ausdruck mit der zu erwartenden absoluten Kostenverbesserung auftraten, wurden aber im Ansatz von CROQUE dadurch beseitigt, daß immer die n besten Regeln (quasi parallel) auf Pläne angewendet werden, die vor der Transformation die geringsten zu erwartenden Kosten aufweisen.

Dadurch wurde sowohl die strikte Trennung von logischer und physischer Optimierung aufgeweicht, als auch eine Bewertung der Qualität logischer Transformationen im Kontext der aktuellen Datenbankeigenschaften erreicht.

4.4 Untersuchungen zu Query Rewriting und Suche

Es ist das Ziel der Optimierung, eine günstige Repräsentation zu finden, ohne dabei alle Möglichkeiten betrachten zu müssen, d.h. ohne alle Pläne zu betrachten und zu bewerten. Dennoch sollen keine erfolgversprechenden Pläne vorzeitig entfernt werden. Auf der Ebene der Ausführungspläne wird dieser Vorgang durch Suchstrategien gesteuert. Das Erzeugen physischer Pläne und die Kostenbewertung werden dabei in CROQUE erst bei Bedarf (“on demand”) vorgenommen.

Die in [19] beschriebene Verfahrensweise basiert auf einem Suchraum, bei dem jedem logischen Operator genau eine Implementierung zugeordnet wurde, so daß die physische Dimension tatsächlich nur eine Ausdehnung von einer Repräsentation besaß. Im dadurch aufgespannten eindimensionalen Suchraum wurden die erzeugten Pläne bezüglich der Anzahl der zu ihrer Erzeugung notwendigen Transformationsschritte priorisiert ausgewählt. Das Verfahren resultierte in teilweise sehr guten Ergebnissen.

Allerdings gilt für die zugrundeliegende Heuristik, daß eine größere Anzahl an Regelanwendungen in einem besseren Plan resultiert, wie für jede Heuristik, daß

es sich hierbei um einen Erfahrungswert bzw. eine begründete Annahme handelt: in vielen Fällen ist eine Bestätigung der Aussage der Heuristik zu erwarten. In einigen Fällen kann auch diese Heuristik jedoch fehlschlagen.

Aufgrund der positiven Ergebnisse dieser doch sehr einfachen Heuristik, wurde der in Unterabschnitt 4.6 beschriebene heuristische Ansatz der Suchraumaufspannung beschritten, der eine Verfeinerung der zunächst verwendeten Konzepte darstellt.

4.5 Evaluierung des Kostenmodells

Die Abschätzung der in CROQUE bei Ausführung eines Planes anfallenden Kosten wird durch die in [17,20] angegebene Definition eines Kostenmodells — auch für die Verwendung physischer Clusterung — ermöglicht.

Das dort beschriebene Modell ist wesentlich schlanker als das zu Beginn der Arbeiten im CROQUE–Projekt zunächst aufgestellte Ad–hoc–Kostenmodell, da zu dieser Frage festgestellt wurde [17], daß

- die im Rahmen des geplanten Projekt–Umfangs zu berechnenden Kostenwerte nicht als exakte Zeitangaben benötigt werden (es reicht vielmehr für die notwendige Entscheidung zwischen je zwei Plänen aus, wenn die Kostenrelation für diese Pläne korrekt wiedergegeben wird).
- das wesentlich einfachere und damit auch günstigere Kostenmodell bereits ausreichend gute Kosten–Abschätzungen ermöglicht.

Der damit aufgegebene Versuch, die Kosten möglichst exakt zu bestimmen um daraus die realen Ausführungszeiten ableiten zu können, scheint im nachhinein keinesfalls zu rechtfertigen, da die Komplexität des Kostenmodells direkten Einfluß auch auf den Optimimerungsaufwand insgesamt hat.

4.6 Untersuchungen zur Verwendung von Heuristiken bei der Suchraumaufspannung

Während der physischen Optimierung konkrete Informationen über Speicherstrukturen und Zugriffspfade vorliegen, muß die logische Optimierung allein mit Statistiken und Erfahrungswerten (also Heuristiken) auskommen, wenn sie für sich selbst betrachtet wird (also bei Nicht–Berücksichtigung des in Unterabschnitt 4.3 beschriebenen verzahnten Verfahrens mit Feedback).

Der Aufwand bei der logischen Optimierung besteht zum überwiegenden Teil im Erzeugen äquivalenter Ausdrücke mittels Rewriting. Vor allem aufgrund der erweiterten Fähigkeiten objekt–relationaler und objekt–orientierter Datenbanksysteme bezüglich der verwendeten Datenmodelle und Speicherstrukturen (hier liegt einer der Themenschwerpunkte in CROQUE; wie in Unterabschnitt 3.2 beschrieben, ist CROQUE ein Vorreiter bei der Flexibilisierung interner Strukturen) scheint es sinnvoll, die verwendeten Rewrite–Regeln nicht — wie bisher — als unabhängig von einer konkreten Situation zu sehen, sondern ihren Nutzen in Abhängigkeit von den vorliegenden Datenbankeigenschaften zu bewerten. Damit verbunden ist einerseits die in Unterabschnitt 4.3 beschriebene Interaktion zwischen logischer und physischer Optimierung und andererseits die gezielte Verwendung von veränderlichen (adaptiven) Heuristiken.

Die entwickelte Ordnung der Rewrite–Regeln nach ihrem Optimierungspotential (beschrieben in [21–23,26]) ermöglicht eine vorzeitige Einschränkung des Suchraums. Durch die Angabe eines Parameters kann festgelegt werden, wie stark der Suchraum während seiner Erzeugung beschnitten werden soll (im Extremfall wird ein vollständiges Aufspannen der algebraischen Dimension zugelassen). Das Verfahren wurde so konzipiert, daß es (vorerst lediglich manuell) flexibel auf Änderungen der beobachteten Anfragesituation reagieren kann. Weitergehende Untersuchungen zum Adaptions–Aspekt werden derzeit in einer gesonderten Arbeit durchgeführt (vgl. Unterabschnitt 3.3).

4.7 Analyse der Bewertungsmöglichkeiten von Anfrageoptimierungen

Für die Bewertung der Optimierung einer konkreten Anfrage gibt es gemäß [18] folgende zwei Ansätze: Man kann versuchen, die Bewertung sofort nach der Abarbeitung der Anfrage vorzunehmen (online) oder dies zu einem späteren Zeitpunkt nachholen (offline). Falls die Bewertung online vorgenommen wird, ist es aufgrund der im allgemeinen vorliegenden Ressourcenknappheit nicht sinnvoll, exhaustiv nach besseren Anfrageplänen zu suchen, da dies dann auch gleich während der Optimierung hätte getan werden können. Stattdessen müssen Abschätzungen für die Qualität der Optimimerung getroffen werden.

Das Verfahren der fiktiven Extrempunkte kann eine — jedoch lediglich sehr grobe — Einordnung der Qualität einer Optimierung sofort nach Ausführung einer Anfrage liefern. Die fiktiven Extrempunkte sind dabei zum einen ein Minimum, das von keinem realen System unterboten werden kann und andererseits ein Maximum, das die größtmögliche Bearbeitungszeit festzulegen versucht. Beide Werte verwenden die jeweiligen Extremwerte der Parameter Systemlast, Kommunikation, Datenvolumen und Selektivität und berechnen auf dieser Grundlage mittels des Kostenmodells die Grenzen an denen die tatsächliche Ausführung gemessen wird. Im Rahmen von CROQUE ist dieses Verfahren aufgrund des gewählten einfachen Kostenmodells nicht ohne weiteres umsetzbar. Die Grundidee, einfach mittels des Kostenmodells die Qualität zu messen, wird aber auch in den Ansatz von CROQUE Eingang finden.

Wesentlich genauer als das Verfahren der fiktiven Extremwerte ist die Verwendung von Statistiken über

die Kostenschätzungen/Ausführungszeiten früher bearbeiteter Anfragen. Nach einer genügend langen Laufzeit sollte es möglich sein, Tendenzen zur Bewertung von Optimierungen aus den vorhandenen Statistiken abzuleiten. Auch hier bereitet das Kostenmodell von CROQUE noch Probleme. Es besteht aber die begründete Hoffnung, daß sich brauchbare Zeitschätzungen aus den abstrakten Kostenwerten ableiten lassen werden, da das Kostenmodell zwar gewisse Kostenparameter vernachlässigt, CROQUE sich aber als Einzelnutzersystem relativ stabil hinsichtlich der Ungenauigkeiten des Kostenmodells verhalten wird.

Bezüglich der Bewertung von Optimierungen sind noch weitergehende Untersuchungen notwendig.

5 Mögliche Anwendungsperspektiven

Insgesamt wurde in CROQUE der Nachweis erbracht, daß ein objektorientiertes Anfragebearbeitungssystem mit den drei — aus unserer Sicht absolut wünschenswerten/notwendigen, in Abschnitt 1 beschriebenen und in derzeit kommerziell verfügbaren Systemen nicht umgesetzten — Hauptmerkmalen tatsächlich realisiert werden kann.

Die im Rahmen des Projektes konzipierten und auch prototypisch implementierten Verfahren und Optimiererbestandteile lassen sich durchaus als Basis für weitergehende Datenbank–Forschungsthemen und –projekte verwenden (siehe dazu auch Unterabschnitt 6). Eine direkte wirtschaftliche Verwertbarkeit scheint jedoch nicht gegeben zu sein.

Vielmehr sind die Ideen, die hinter den gewonnenen Erkenntnissen und den erzielten Ergebnissen stehen, sowohl für die Verwendung in weiteren Prototypen hilfreich und könnten teilweise auch in kommerzielle Systeme, wie das in CROQUE als zugrundeliegender, persistenter Objektspeicher verwendete OODBMS ObjectStore oder das Produkt POET, eingehen. Die genannten Systeme stehen stellvertretend für die Datenbank–Entwicklungslinie der persistenten, objektorientierten Programmiersprachen und bieten beispielsweise keine deklarative Anfragesprache, wie das als Standard konzipierte und in CROQUE umgesetzte ODMG–OQL. Somit sind die in CROQUE erzielten Ergebnisse, die durch die Erarbeitung von Optimierungskonzepten weit über die bloße Verwendung von OQL hinausgehen, auch für die Umsetzung in diesen Produkten interessant.

6 Denkbare Folgeuntersuchungen

Der Optimierer von CROQUE wurde zunächst als statischer Optimierer konzipiert, d.h. alle Entscheidungen werden zur Übersetzungszeit der Anfrage getroffen. Erweiterungen hin zu dynamischen Ausführungsplänen, die einige Entscheidungen auf den Ausführungszeitpunkt verschieben, waren nicht Kernbestandteil dieses Vorhabens.

Zur weiteren Verbesserung der Optimierung vor allem von vorübersetzten (nicht–Ad–hoc–) Anfragen bieten sich Folgeuntersuchungen zu dynamischen Ausführungsplänen an.

Zum Nachweis der Qualität der entwickelten Optimierungskonzepte sind Folgeuntersuchungen zur Umsetzung der Implementierungen auf weitere Plattformen, wie etwa modernere SQL-Datenbanksysteme, wünschenswert. Bei diesen Untersuchungen sollten auch die Auswirkungen des zu erwartenden SQL3-Standards berücksichtigt werden.

Wie in Unterabschnitt 4.7 aufgeführt, sind darüber hinaus weitergehende Untersuchungen bezüglich der Bewertung von Optimierungen notwendig.

7 Zusammenfassung

Der an der Universität Rostock verfolgte Ansatz im nun abgeschlossenen Teil–Projekt konzentrierte sich in erster Linie auf den Entwurf und die Realisierung des Anfrage–Optimierers. Aufbauend auf einer Formalisierung des SQL–Nachfolger–Vorschlages OQL wurde zunächst ein Optimierungs-Ablauf konzipiert, der die traditionelle Trennung in logische und physische Optimierung aufgibt. Dadurch wird für den Optimierer die Möglichkeit des Austauschs von "Feedback" zwischen diesen beiden wesentlichen Optimierungsphasen geschaffen.

Für die Anfragesprache OQL wurde ein neuartiger hybrider Auswertungsansatz entwickelt, der durch die Definition eines kalkülartigen Formalismus und einer variablenfreien Algebra auch erfolgreich umgesetzt werden konnte. Dieser Ansatz erlaubt zum einen die Verwendung von rein deklarativen Äquivalenzregeln und vereint zum anderen die Stärken der beiden verwendeten Formalismen bei der Darstellung und Manipulation von Anfrage–Ausdrücken.

Zur Umsetzung der geplanten regel– und kostenbasierten Optimierungskomponente wurde zunächst ein auf die in CROQUE verwendeten Operatoren und Algorithmen zugeschnittenes Kostenmodell aufgestellt und verschiedene Suchstrategien implementiert. Darauf aufbauend wurden unterschiedliche Heuristiken entwickelt, deren Verwendung speziell den Optimierungsaufwand verringern und gleichzeitig ein besseres Optimierungsergebnis erreichen.

Die erarbeiteten Konzepte wurden zur Evaluierung in einem Prototypen umgesetzt. Als Mittel zur Bewertung von Optimierungen wurden Optimierer–Benchmarks untersucht, die aber als insgesamt ungeeignet verworfen werden mußten. Die Konzeption eines Optimierer-Baukastens dauert noch an.

Fragen des physischen Datenbank–Designs und die Entwicklung der über die Fähigkeiten kommerzieller Systeme weit hinausgehenden internen Ebene, des flexiblen Storage–Managers und der dazugehörigen Ausführungsstrategien, die die Basis des Projektes bilden, wurden

exklusiv beim Projektpartner, der Universität Konstanz, untersucht. Diese Untersuchungen und abschließende Implementierungen zur Integration der geschaffenen Prototypen werden noch bis zum dortigen Projektende fortgeführt.

Acknowledgements: Wir danken unseren Konstanzer Projektpartnern Marc H. Scholl, Torsten Grust und Dieter Gluche für die gemeinsame Arbeit an CROQUE und unseren Studenten Ralf Asmus, Regina Illner, Holger Janz, Michael Jonas, Stefan Paul, Beate Porst, Denny Priebe und Steffen Rost für die Ausgestaltung und Implementierung der verschiedenen Konzepte in den Prototypen der jeweiligen Projektphasen.

Literatur

1. J.A. Blakeley, W.J. McKenna, and G. Graefe. Experiences Building the Open OODB Query Optimizer. In *Proc. of the ACM SIGMOD Conference*, pages 287–296, Washington DC, USA, May 1993.
2. Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong. Comprehension Syntax. *SIGMOD Record*, 23:87–96, March 1994.
3. R.G.G. Cattell, editor. *The Object Database Standard: ODMG-93, Release 1.2*. Morgan-Kaufmann, San Mateo, CA, 1996.
4. R. L. Cole and G. Graefe. Optimization of Dynamic Query Evaluation Plans. In *Proc. of the ACM SIGMOD 23,2*, pages 150–160, 1994.
5. D. Das and D. Batory. Prairie: A Rule Specification Framework for Query Optimizers. In *11th IEEE Conference on Data Engineering*, pages 201–210, Taipei, Taiwan, March 1995.
6. L. Fegaras and D. Maier. Towards an Effective Calculus for Object Query Languages. In *Proc. of the ACM SIGMOD Conference*, pages 47–58, San Jose, CA, 1995.
7. L. Fegaras, D. Maier, and T. Sheard. Specifying Rule–based Query Optimizers in a Reflective Framework. In *Proc. of the 3rd Int. Conference on Deductive and Object-Oriented Databases*, pages 146–168, New York, December 1993. Springer.
8. D. Gluche, T. Grust, C. Mainberger, and M.H. Scholl. Incremental Updates for Materialized Views with User-Defined Functions. In *Proc. of the Fifth Int. Conference on Deductive and Object-Oriented Databases (DOOD'97), Montreux, Switzerland, LNCS 1341, Springer*, pages 52–66, December 1997.
9. G. Graefe and D.J. DeWitt. The EXODUS Optimizer Generator. In *Proc. of the ACM SIGMOD Int. Conference on Management of Data*, pages 160–172, San Francisco, CA, USA, May 1987.
10. G. Graefe and W.J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *IEEE Conference on Data Engineering 4/1993*, pages 209–218, Vienna, Austria, April 1993.
11. T. Grust, J. Kröger, D. Gluche, A. Heuer, and M.H. Scholl. Query Evaluation in CROQUE — Calculus and Algebra Coincide. In *Proc. of the 15th British National Conference on Databases (BNCOD 15), London, UK, LNCS 1271, Springer*, pages 84–100, July 1997.
12. T. Grust and M.H. Scholl. Translating OQL into Monoid Comprehensions — Stuck with Nested Loops? Technical Report 3a/1996, Department of Mathematics and Computer Science, Database Research Group, University of Konstanz, September 1996. Revised Version.
13. T. Grust and M.H. Scholl. Hybrid Strategies for Query Translation and Optimisation. Technical Report 72/1998, Department of Mathematics and Computer Science, Database Research Group, University of Konstanz, October 1998. Also appeared as ESPRIT Pastel research report RT2R1. This is collaborative work with Pastel RT2 members.
14. T. Grust and M.H. Scholl. Query Deforestation. Technical Report 68/1998, Department of Mathematics and Computer Science, Database Research Group, University of Konstanz, June 1998.
15. R. Harper. *Introduction to Standard ML*. School of Computer Science, Carnegie Mellon University, Pittsburgh, 1993.
16. A. Heuer. *Objektorientierte Datenbanken - Konzepte Modelle, Standards und Systeme*. Addison-Wesley, Bonn, 1997. 2. aktualisierte und erweiterte Auflage.
17. H. Janz. Anpassung des CROQUE-Kostenmodells an die flexible interne Ebene. Diplomarbeit, Universität Rostock, Fachbereich Informatik, July 1997.
18. M. Jonas. Entwicklung und Test von Verfahren zur Bewertung von Anfrageoptimierungen. Studienarbeit, Universität Rostock, Fachbereich Informatik, December 1998.
19. J. Kröger, R. Illner, S. Rost, and A. Heuer. Query Rewriting and Search in CROQUE. Preprint CS–15–98, Universität Rostock, Fachbereich Informatik, December 1998. Eingereicht zur Veröffentlichung.
20. J. Kröger, H. Janz, and A. Heuer. About Reducing the Costs for Cost Calculation in CROQUE. Preprint, CS Dept., University of Rostock, 1999. In preparation.
21. J. Kröger, S. Paul, and A. Heuer. On the Ordering of Rewrite Rules (Extended Abstract). In *Proc. of the Second East-European Symposium on Advances in Databases and Information Systems (ADBIS'98), Poznan, Poland, LNCS 1475, Springer*, pages 157–159, September 1998.
22. J. Kröger, S. Paul, and A. Heuer. Query Optimization: On the Ordering of Rules. Preprint CS–09–98, Universität Rostock, Fachbereich Informatik, June 1998.
23. J. Kröger, S. Paul, and A. Heuer. Query Optimization: Ordering Rules? Preprint CS–12–98, Universität Rostock, Fachbereich Informatik, June 1998.
24. T. W. Leung, G. Mitchell, B. Subramanian, B. Vance, S. L. Vandenberg, and S. B. Zdonik. The AQUA Data Model and Algebra. In *Proc. of the 4th Int. Workshop on Database Programming Languages (DBPL)*, pages 157–175, New York, August 1993. Springer.
25. R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.
26. S. Paul. Analyse des Optimierungspotentials der algebraischen Äquivalenzregeln in CROQUE. Diplomarbeit, Universität Rostock, Fachbereich Informatik, December 1997.
27. D. Priebe. Konzeption einer variablenfreien CROQUE–Algebra und Implementierung einer geeigneten Regelverwaltung. Diplomarbeit, Universität Rostock, Fachbereich Informatik, November 1998.

28. D. Priebe. Konzeption und Implementierung des Optimierungsablaufs in CROQUE. Studienarbeit, Universität Rostock, Fachbereich Informatik, June 1998.
29. D. Priebe, J. Kröger, and A. Heuer. Variablen in Algebren? — Die CROQUE–Algebra. Technischer Bericht in Vorbereitung, Fachbereich Informatik, Universität Rostock, 1999. Fertigstellung voraussichtlich im Februar.
30. H. Riedel and M.H. Scholl. The CROQUE–Model: Formalization of the Data Model and Query Language. Technical Report 23/1996, Department of Mathematics and Computer Science, Database Research Group, University of Konstanz, December 1996.
31. H. Riedel and M.H. Scholl. A Formalization of ODMG Queries. In *Proc. of the 7th Int. Conference on Database Semantics (DS-7), Leysin, Switzerland*, October 1997.
32. T. Sheard. Guide to using CRML – Compile-Time Reflective ML. Technical report, Pacific Software Research Center, Oregon Graduate Institute of Science & Technology, Beaverton, Oregon, October 1993.
33. P. Trinder. Comprehensions, a Query Notation for DBPLs. In *Proc. of the Third Int. Workshop on Database Programming Languages, Nafplion, Greece*, pages 55–68, 1991.
34. P. Trinder and P. Wadler. Improving List Comprehension Database Queries. In *Proc. of TENCON'89, Bombay, India*, pages 186–192, November 1989.