Anfrageoptimierung in objektrelationalen Datenbanken durch kostenbedingte Termersetzungen

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades
Doktor der Naturwissenschaften
Dr. rer. nat.
genehmigte Dissertation

von
Dipl.-Math. Mazeyar E. Makoui
geboren am 18.07.1977 in Teheran (Iran)

2006

Referent: Prof. Dr. Udo W. Lipeck

Korreferent: Prof. Dr. Rainer Parchmann

Tag der Promotion: 20.09.2006

Zusammenfassung

In dieser Arbeit wird ein Optimierungssystem für relationale und objektrelationale Anfragen entwickelt. Dabei wird ein Optimierungskonzept vorgestellt, welches den bisherigen dreiphasigen Optimierungsablauf einer deklarativen Anfrage (algebraische, physische und kostenbasierte Optimierung) hin zu einer umfassenden Einphasenoptimierung vereinfacht. Durch die Unterteilung in Phasen war es nicht möglich, Optimierungsregeln beliebig anzuordnen, obwohl häufig kostenbezogene Kriterien Fehlentscheidungen bei algebraischen Umformungen verhindern würden (z. B. bei der Anordnung von Verbunden). Aus diesem Grund wird ein Termersetzungsverfahren vorgestellt, welches als Basis die Ersetzungsregeln der relationalen Algebra besitzt. Zur Realisierung werden demzufolge algebraische Regeln um physische Implementierungsarten der bekannten relationalen Operatoren erweitert. Diese dienen dann als Grundlage für ein neues Termersetzungssystem, welches flexibel bedingte Ersetzungen durchführen kann. Hiermit erreichen wir eine maximale Erweiterbarkeit, da neue Implementierungen von Operatoren in Beziehung mit Vorhandenen gesetzt werden können. Aus diesem Grund wird eine erweiterte Regelsprache, eine neue Regelanordnungssprache und eine neue Datenstruktur für Ausführungspläne vorgestellt, um folgende drei Steuerungsstufen der Regelanwendung spezifizieren zu können:

(a) *lokale Restriktionen*: Es werden individuelle Bedingungen, unter denen eine Regel angewandt wird, definiert. Da das Hauptziel eine Minimierung der Anzahl der durchgeführten Ersetzungen ist, wird die Bedingungsmenge nicht nur auf semantische und syntaktische Bedingungen begrenzt, sondern mit Kostenbedingungen angereichert. Dafür wird ein Kostenmodell vorgestellt, mit dessen Hilfe Entscheidungen für Ausführungspläne getroffen werden können.

(b) *spezifizierte Regelabfolgen*: Anwendungen von Regeln werden durch reguläre Ausdrücke iteriert und zur Generierung von alternativen Ausführungsplänen definiert. Damit können bisherige statische Regelabfolgen z. B. dynamisch an neue Datenbestände angepasst werden.

(c) *globale Strategien*: Beliebige Strategien zur Suche von Optima können flexibel kombiniert werden. Um das exponentielle Wachstum der Anzahl von verschiedenen Ausführungsplänen zur Laufzeit kontrollierbar zu machen, werden Suchstrategien in Form von Kosten- bzw. Variantengrenzen eingeführt.

Zur Verdeutlichung der Flexibilität des realisierten Anfrageoptimierers wird für den objektrelationalen Fall das Optimierungskonzept um die Berücksichtigung eines räumlichen Datenbanksystems erweitert. Dazu werden neue, bislang nicht im Termersetzungsoptimierer angedachte räumliche Selektionsregeln definiert. Daraus resultiert ebenfalls die Speicherung von zusätzlichen Statistiken, wie Geometrietypen und deren Vorkommen, die zur Bedingungsprüfung bei der Verwendung von Regeln benötigt werden.

Es wird eine Experimentierumgebung geschaffen, mit deren Hilfe man die Anordnung von Optimierungsregeln hinsichtlich eines Datenbankinhaltes und den dazu gehörenden Anfragen, abgeschätzt durch das vorgestellte Kostenmodell, validieren kann. Insbesondere werden Experimente durchgeführt, um zu zeigen inwieweit sich bisherige Optimierungsstrategien im relationalen Fall auf den objektrelationalen Fall übertragen lassen.

Abschließende eigene Tests zeigen, dass im relationalen Fall bisherige kommerzielle Optimierer vergleichbare Resultate, wie das in dieser Arbeit vorgeschlagene Optimierungskonzept, liefern. Im objektrelationalen Fall aber können mit bedingten Termersetzungen deutlich schnellere Ausführungspläne generiert werden.

Stichworte:

Anfrageoptimierung
Termersetzungssystem
Räumliche Anfragen

Abstract

In this thesis an optimization system for relational and object-relational queries is designed. An optimization process is developed, which simplifies the current three-phase optimization sequence of declarative queries (algebraic, physical and cost-based optimization) to one comprehensive one-phase optimization. By separating the phases it was not possible to arrange optimization rules arbitrarily, although cost-based criteria would often avoid wrong decisions of an algebraic transformation, e. g. during the join ordering. For that reason a term rewriting system is presented, which uses basic rewrite rules of the relational algebra. The algebraic rewrite rules are extended with physical implementations of alternatives of known relational operators. They serve as a basis of a new term rewriting system that can perform conditional term rewriting. In this way we achieve a maximal extensibility, since new implementations of operators can be compared with existing ones. Therefore, a new extended rule language, a new rule sequencing language and a new data structure for execution trees are presented, in order to specify the following three controlling stages of rule application:

(a) *local restrictions*: Individual conditions, on which a rule is applied, are defined. Since the principal purpose is the minimization of the number of rewrites, the condition quantity is not only limited to semantical and syntactical restrictions, but also enriched with cost conditions. Therefore, a cost model is presented, with whose assistance decisions for execution trees can be made.

(b) *specifying rule patterns*: Rule applications are defined with regular expressions for iteration and for generating alternative execution trees. In this way static rule patterns for example can be dynamically adapted to new datasets.

(c) *global strategies*: Arbitrary search strategies can be flexibly combined. To control the exponential growth of the number of alternative execution trees during run-time, common search strategies are introduced as cost- and variants-limitations.

To clarify the flexibility of the realized query optimizer, an object-relational case of the query optimization concept for spatial databases is presented. For that purpose, new selection rules are defined which have never before been used in a term rewriter optimizer. This yields the autonomous storing of additional statistics, to geometry types and their occurrences, which are needed for condition checks during the rule application.

A framework for experiments is created, to validate the ordering of optimization rules regarding the database content and the corresponding workload, estimated by the presented cost model. In particular, experiments are performed in order to establish to what extent past optimization strategies can be transferred from the relational case to the object-relational one.

Finally, our own tests demonstrate that in the relational case commercial optimizers supply comparable results like the optimization concept suggested in this work. In the object-relational case, however, faster execution trees can be generated with conditioned term rewriting.

Keywords:

Query Optimization
Term Rewriting
Spatial Queries

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Ablauf der Anfragebearbeitung . . . 1
- 1.2 Die klassischen Optimierungsphasen . . . 2
- 1.3 Ein Ansatz zum Aufbrechen der Optimierungsphasen – Das System von bedingten Termersetzungen . . . 3
- 1.4 Bedingte Termersetzung bei objektrelationalen Operatoren . . . 5
- 1.5 Ziel dieser Dissertation . . . 5
- 1.6 Verwandte Arbeiten . . . 6
- 1.7 Klassifikation des entwickelten Optimierungskonzeptes . . . 8
- 1.8 Aufbau der Dissertation . . . 10

2 Kostenmodell **11**
- 2.1 Allgemeine Konventionen dieser Arbeit . . . 12
- 2.2 Definition eines Kostenterms . . . 13
- 2.3 Definition einer Kostenfunktion . . . 14
- 2.4 Darstellung von Mengenoperatoren . . . 17
- 2.5 Selektivitätsabschätzung von Prädikaten . . . 17
  - 2.5.1 Abschätzung von Kardinalitäten und Seitenanzahlen . . . 18
  - 2.5.2 Schätzungen für Attributwertanzahlen . . . 19
- 2.6 Abschätzung von Tupellängen . . . 21
- 2.7 Kostenfunktionen . . . 22
  - 2.7.1 Selektion . . . 22
  - 2.7.2 Projektion . . . 22
  - 2.7.3 Join . . . 23
  - 2.7.4 Antisemi-/Semijoin . . . 24
  - 2.7.5 Aggregation mit und ohne Gruppierung . . . 25
  - 2.7.6 Erzeugung von Indexen und Sortierungen . . . 26
- 2.8 Pipelining . . . 26
- 2.9 Vererbung von Indexen und Sortierungen . . . 29
- 2.10 Definition eines generalisierten Baumes . . . 31
- 2.11 Beispiel für die Kostenberechnung eines Ausführungsplanes . . . 33
- 2.12 Fazit des Kostenmodells . . . 35

3 Optimierung von Ausführungsplänen mit Hilfe eines Termersetzungssystems **37**
- 3.1 Ein klassisches System von bedingten Termersetzungen . . . 37
- 3.2 Erweiterung der Regeln der Relationalen Algebra um Bedingungen . . . 37
  - 3.2.1 Implementierungsunabhängige relationale Ersetzungsregeln . . . 38
  - 3.2.2 Erzeugungsregeln für Sortierungen und Indexe . . . 43
  - 3.2.3 Implementierungsabhängige relationale Ersetzungsregeln . . . 43
- 3.3 Generalisierte Regeln . . . 44
- 3.4 Erweiterung der Algebraischen Optimierung um kostenbedingte Termersetzungen 46
  - 3.4.1 Einführendes Beispiel zur kostenbedingten Termersetzung . . . 46

3.4.2 Herleitung einer kostenbedingten Termersetzungsregel 48
3.4.3 Entwicklung einer bedingten Ersetzungsregel für die Projektion 49
3.5 Erweiterung des Join-Ordering-Algorithmus um Selektionen 52
3.5.1 Abstraktes Beispiel für den bedingten Join-Ordering-Algorithmus 56
3.5.2 Laufzeitbetrachtung des erweiterten Join-Ordering-Algorithmus 57
3.6 Fazit der bedingten Termersetzungsregeln 57

4 Erweiterung auf objektrelationale Operatoren 59
4.1 Räumliche Datentypen 60
4.2 Räumliche Operatoren 60
4.3 Beispiel für eine objektrelationale Anfrage 61
4.4 Motivation zur differenzierten Betrachtung von Geometrietypen und Prädikatsfunktionen 63
4.5 Geometrietypen und ihre Verteilung 63
4.6 Topologische Beziehungsoperatoren 65
4.7 Vorselektierung bei topologischen Beziehungsanfragen 67
4.8 Funktionsweise von räumlichen Operatoren 68
4.9 Erweiterung des relationalen Kostenmodells auf objektrelationale Operatoren . . 69
4.9.1 Selektivitätsabschätzungen von räumlichen Prädikaten 70
4.9.2 Abschätzung von Kardinalitäten und Seitenanzahlen 70
4.9.3 Schätzungen für Attributwertanzahlen 70
4.9.4 Abschätzungen von Tupellängen 71
4.10 Räumliche Kostenfunktionen 71
4.10.1 Gewichtungsfaktor für objektrelationale Operatoren 71
4.10.2 Räumliche Selektion 73
4.10.3 Räumlicher Join 73
4.10.4 Pipelining 74
4.10.5 Vererbung von Sortierungen und Indexen 74
4.11 Beispiel für eine objektrelationale Kostenberechnung 74
4.12 Fazit der Erweiterung des Kostenmodells auf räumliche Operatoren 76

5 Der bedingte Termersetzungsoptimierer 77
5.1 Das Matching von generalisierten Regeln 77
5.1.1 Auswerten von Bedingungen: pre- und postconditions 78
5.1.2 Definition von Varianten eines Ausführungsplanes 78
5.1.3 Generierung von neuen Varianten 79
5.2 Definition einer Regelsteuerungssprache 79
5.3 Beschränkung der Möglichkeiten zur Variantenerzeugung 81
5.3.1 Dynamische Kostengrenze 81
5.3.2 Statische Variantengrenze 82
5.3.3 Dynamische Variantengrenze 83
5.4 Aufbau des bedingten Termersetzungsoptimierungssimulators 83
5.4.1 Modul 1: Der relationale Parser 84
5.4.2 Modul 2: Die Steuereinheit 84
5.4.3 Modul 3: Das Termersetzungssystem 85
5.5 Ein Beispiel für den Suchraum des Termersetzers 85
5.6 Fazit der Termersetzung 85

6 Entwicklung von Optimierungsstrategien 87
6.1 Einführung in die Strategiebetrachtung 87

6.2 Optimierungsstrategien . . . 88
6.2.1 Kostenbedingte Strategien . . . 88
6.2.2 Strategien mit Hilfe von Variantengrenzen . . . 90
6.3 Ergebnisse der Experimente . . . 90
6.3.1 Resultate der relationalen Optimierung mit RELOpt . . . 91
6.3.2 Fazit der relationalen Optimierung . . . 93
6.3.3 Objektrelationale Tests mit ATKIS-Daten . . . 94
6.3.4 Resultate der objektrelationalen Optimierung mit RELOpt . . . 96
6.3.5 Fazit der objektrelationalen Optimierung . . . 97
6.4 Erfahrungen bei der Verwendung der Regeln . . . 98
6.4.1 Verschmelzung von Regeln am Beispiel der Anordnung von Joins . . . 98
6.4.2 Strategien durch Präferierung von Ersetzungsregeln . . . 99
6.5 Zusammenfassung der Ergebnisse . . . 100

7 Implementierung 103
7.1 Das System des Anfragensimulators RELOpt . . . 103
7.1.1 Package *dbs.unihannover.sopt.struc.general* . . . 103
7.1.2 Package *dbs.unihannover.sopt.struc.general.optimizer* . . . 104
7.1.3 Package *dbs.unihannover.sopt.struc.general.tree* . . . 108
7.1.4 Package *dbs.unihannover.sopt.struc.general.tree.condition* . . . 110
7.2 Sequenzablaufdiagramm der Einphasenoptimierung . . . 113
7.3 Komponentendiagramm für die Einphasenoptimierung . . . 114
7.4 Fazit der Implementierung . . . 114

8 Ausblick 115

A Leistungsstatistiken 117

B Konkretes Beispiel für den bedingten Join-Ordering-Algorithmus 119

C Gewichtungsfaktoren für topologische Funktionen 123

D Tabellarische Übersichten der Optimierungsresultate 127
D.1 Relationale Tests mit der TPC-H Version 2.3.0 Datenbank . . . 127
D.2 Objektrelationale Tests mit ATKIS-Daten . . . 137
D.2.1 oQuery 1 . . . 139
D.2.2 oQuery 2 . . . 140
D.2.3 oQuery 3 . . . 143
D.2.4 oQuery 4 . . . 146
D.2.5 oQuery 5 . . . 149
D.2.6 oQuery 6 . . . 153
D.2.7 oQuery 7 . . . 155
D.2.8 oQuery 8 . . . 158
D.2.9 Ergebnisse der räumlichen Anfragen . . . 161

Abbildungsverzeichnis 165

Tabellenverzeichnis 167

Literaturverzeichnis 169

Ich verzeihe keine Fehler,
noch weniger meine eigenen.

Sir Peter Ustinov 1921-2004



1 Einleitung

Ermöglicht man es dem Benutzer, eine Anfrage an eine Datenbank deklarativ zu stellen, folgt zwingend daraus, dass die Datenbank selbst für die Ausführung verantwortlich ist. Durch die Vielzahl von äquivalenten Ausführungsmöglichkeiten muss das System zum einen selbst entscheiden, welche Ausführungspläne exakt der Anfrage entsprechen, und zum anderen – der weitaus wichtigere Aspekt – welcher von den vielen äquivalenten Ausführungsplänen das Optimierungsziel am besten annähert. Mögliche Optimierungsziele sind u. a. die Antwortzeiten, die Ressourcenbelastung und somit die Laufzeit und/oder der Speicherplatzverbrauch. Diese Ziele müssen gegeneinander abgewägt und verfolgt werden.

1.1 Ablauf der Anfragebearbeitung

Abbildung 1.1 zeigt den üblichen Ablauf einer Anfragebearbeitung. Dabei gibt der Benutzer per SQL-Statement eine deklarative Anfrage ein, die der Anfrageoptimierer der Datenbank in die bevorzugte algebraische Form überführt. Dabei werden Zugriffe auf Sichten durch Substitutionen der Sichtendefinitionen aufgelöst (Sichtauflösung / Sichtexpansion). Es folgt die Entschachtelung von Unteranfragen (vgl. [41]) mit nachfolgender Standardisierung in eine (konjunktive) Form (vgl. [35]), welche dem Optimierer der Datenbank als Grundlage für die anschließende Optimierung dient. Danach wird die in Abschnitt 1.2 beschriebene Optimierung der Ausführungspläne durchgeführt und, nachdem einer gewählt wurde, wird dessen Ausführungscode erzeugt und abgearbeitet. Schließlich wird das Ergebnis ausgegeben.

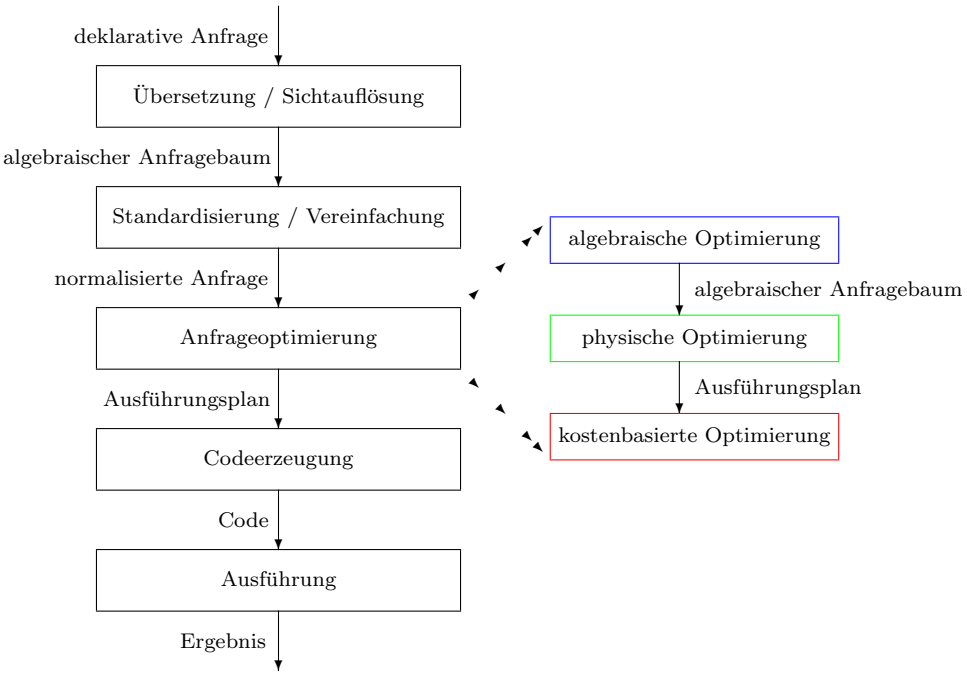


Abbildung 1.1: Ablauf der Anfragebearbeitung

1.2 Die klassischen Optimierungsphasen

Die bisherigen Strategien aus Lehrbüchern, die zur Standardliteratur über Datenbanksysteme gehören, wie etwa [8], [15], [19], [23], [33], [38], [69], [74], [81] und [87], unterteilen die Optimierung eines Ausführungsplanes einer relationalen Datenbankanfrage weitgehend in drei zeitlich aufeinander folgende Phasen. Dabei ist ihre Abfolge fest vorgegeben (siehe Abbildung 1.2):

(a) **Algebraische Optimierung (Logische Optimierung)**

In dieser Phase wird der bisherige, durch eine brute-force-Methode übersetzte Anfragebaum mit Hilfe von algebraischen Ersetzungsregeln (Transformationsregeln bzw. Termersetzungsregeln) optimiert. Da die deklarative Anfrage ein algebraischer Term ist, kann darauf eine feste, heuristische Abfolge von algebraischen Regeln angewandt werden (siehe Kapitel 3).

Bei der Verwendung der Ersetzungsregeln werden aus dem Data-Dictionary keine statistischen Informationen, sondern nur Relationen- und Attributnamen als Metadaten einbezogen, also z. B. keine Informationen über Kardinalitäten, Selektivitäten, Attributwertanzahlen oder Tupellängen. Als Ergebnis wird ein einzelner Operatorbaum mit relationalen Operatoren an die nächste Optimierungsphase übergeben. Nur in der Phase der algebraischen Optimierung kann sich die Baumstruktur noch wesentlich ändern, z. B. von linksorientierten Verbundbäumen (engl. left-deep join trees) zu „buschigen“ Verbundanordnungen (engl. bushy join trees).

(b) **Physische Optimierung (Interne Optimierung)**

Der algebraisch optimierte Operatorbaum wird in mehrere physische Repräsentanten übersetzt, indem das Vorhandensein von Indexen und Sortierungen berücksichtigt wird. Diese Repräsentanten werden jetzt durch die veränderte Datenstruktur zu Ausführungsplänen umgeformt, welche Tupelmengen durch sortierte Tupellisten darstellen. Außerdem können sie nicht nur gewöhnliche Tupel, sondern auch TID-Listen als Zwischenergebnisse beinhalten.

Erst in dieser Phase werden zum ersten Mal verschiedene äquivalente Ausführungspläne durch die Wahl einer Implementierungsart eines algebraischen Operators erzeugt. Es findet eine Selektion von mehreren Ausführungsplänen statt, deren Optimierungen weiterverfolgt werden (Vergleichshorizont). Die Präferierung der Implementierungen von Operatoren wird rein heuristisch durch die Anordnung der allgemeinen Laufzeiten vorgenommen, z. B. wird eine Selektion als indexbasierter Operator ausgewählt, falls ein vorhandener Index genutzt werden kann. Es werden keine neuen Indexe oder Sortierungen erzeugt. Für die gesamte Phase werden Metadaten aus dem Data-Dictionary genutzt, jedoch noch keine Kostenfunktionen zum Vergleich von verschiedenen Ausführungsplänen verwandt.

Die generierten, sich strukturell nicht gleichenden, äquivalenten Ausführungspläne dienen dann anschließend als Basis für eine kostenbasierte Optimierung.

(c) **Kostenbasierte Optimierung**

Erst in dieser Phase werden alle Informationen, die man über die Ausgangsrelationen besitzt, z. B. Kardinalität oder Selektivität von Attributen, in die Optimierung eingebracht. Hierbei werden auch kostenabhängig neue redundante Daten (z. B. Sortierungen) erzeugt, falls deren Nutzung einen globalen Kostenvorteil ermöglicht (siehe [74]).

Da ein vollständiges Durchsuchen des Anfrageraumes mit immensen Kosten verbunden

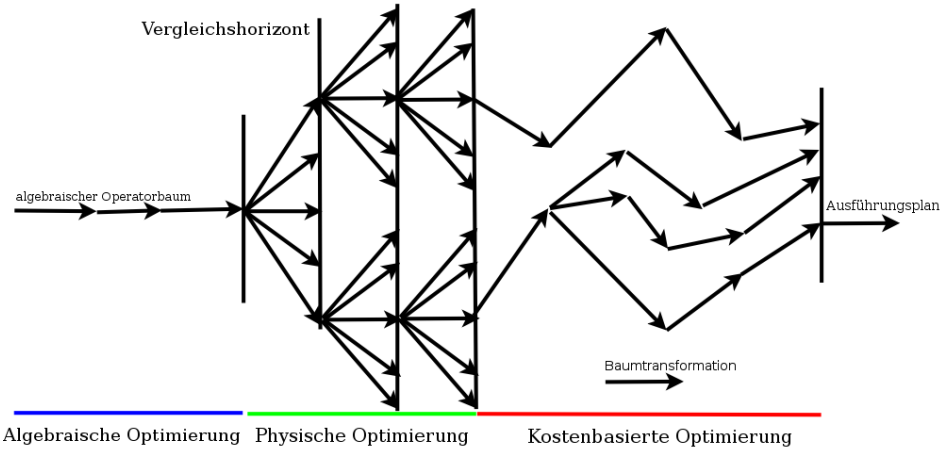


Abbildung 1.2: Dreiphasenoptimierung

wäre, muss auch in dieser Phase eine Einschränkung hin zu einer partiellen Suche vorgenommen werden. Als Entscheidungsfunktionen dienen bekannte deterministische (z. B. der Greedy-Algorithmus (engl. greedy best-first search)) und heuristische Suchstrategien (z. B. der Bergsteigeralgorithmus (engl. hill-climbing), die Simulierte Abkühlung (engl. simulated annealing)) oder Genetische Algorithmen, die alle auf Basis eines Kostenmodells (siehe Kapitel 2) Ausführungspläne in Beziehung zueinander setzen (siehe dazu [16]).

In der Literatur findet man eine Vielzahl von Algorithmen, die diese Phasenaufteilung durchbrechen und schon während der algebraischen Optimierung weitere Metadaten benutzen (siehe [38], [74]). Zu diesen Algorithmen gehören unter anderem die bekanntesten Join-Ordering Algorithmen, der Minimum-Selectivity Algorithmus [82] und das Join-Ordering mit Hilfe der Dynamischen Programmierung [86]. Beide sind kostenbasierte Suchalgorithmen, die nur auf dem algebraischen Baum agieren und zur Neubildung von äquivalenten Ausführungsplänen statistische Informationen einsetzen.

Im Falle der Dynamischen Programmierung stellte sich schon die Erweiterung auf Standard-Operatoren (Selektionen und Projektionen) als großes Hindernis heraus (siehe [76],[77]). Für dieses Problem wird ein eigener Lösungsvorschlag in Abschnitt 3.5 der vorliegenden Arbeit vorgestellt. Alle weiteren Verfahren, inklusive dem Minimum-Selectivity Algorithmus, scheitern an der Optimierung von Nicht-Standard Anfragen, da dort Kenntnisse, z. B. bezüglich der Anordnung der Selektionen, nicht mehr gelten bzw. übertragbar sind. Deshalb liefern sie in einer kombinierten Anfrage ein unbefriedigendes Ergebnis.

Im Folgenden betrachten wir abweichend von den herkömmlichen Optimierungsansätzen einen neuartigen Ansatz, der alle Informationen in einer einzigen Optimierungsphase (Einphasenoptimierung) benutzt, aber die bisherigen Optimierungsansätze feiner gesteuert kombinieren und vervollständigen kann.

1.3 Ein Ansatz zum Aufbrechen der Optimierungsphasen – Das System von bedingten Termersetzungen

In dieser Arbeit wird vorgeschlagen, die algebraischen Regeln um physische Operatoren zu erweitern. Diese Idee liefert die Basis eines flexiblen Regelwerkes, das auf physische Operatorbäume (Ausführungspläne) angewendet werden kann. Wir erreichen eine besondere Erweiterbarkeit, da

neue Implementierungen von Operatoren in Beziehung mit vorhandenen gesetzt werden können, ohne dass das eigentliche Regelsystem verändert werden muss. Um dies zu erreichen, müssen diese Operatoren als physische Implementierungsart zu den Grundoperatoren hinzugefügt werden (siehe dazu [58]).

Dieses Regelwerk wird im Folgenden als ein *Termersetzungssystem* [17] gesehen (siehe Kapitel 3). Es bietet unter anderem die Möglichkeit, Suchstrategien zur Findung des besten Ausführungsplanes als Abbild einer speziellen Reihenfolge von Optimierungsregeln darzustellen (siehe Kapitel 6). Deren Anwendung wird darauf aufbauend nur noch in einer einzigen Phase, der *Einphasenoptimierung*, kontrolliert.

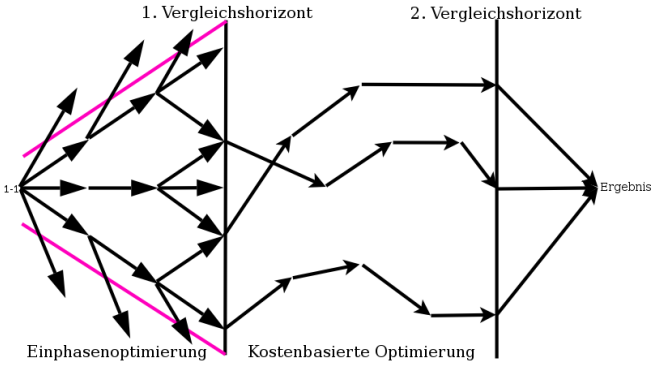


Abbildung 1.3: Einphasenoptimierung

Natürlich kann die Einphasenoptimierung auch mit einer anschließenden kostenbasierten Optimierung kombiniert werden. Dabei dienen die erzeugten Ausführungspläne der vorherigen Einphasenoptimierung als Ausgangsvarianten für eine weiterführende Suche (siehe Abbildung 1.3).

Bei jeder Regelanwendung werden neue Ausführungsvarianten erzeugt, die wieder äquivalente Ausführungspläne darstellen, deren Generierung jedoch nicht mehr rein heuristisch oder beliebig erfolgt (vgl. algebraische Optimierung). Das parallele Anwenden aller gegebenen Regeln würde nämlich die Anzahl von generierten Varianten exponentiell wachsen lassen. Würde man beispielsweise nach 10 Ersetzungen die Generierung anhalten, so hätte man bei 100 Regeln schon

$$\sum_{n=1}^{10}(100^n) \approx 1.01 \cdot 10^{20}$$

verschiedene Varianten, die verglichen und abgespeichert werden müssten (siehe Abbildung 1.4).

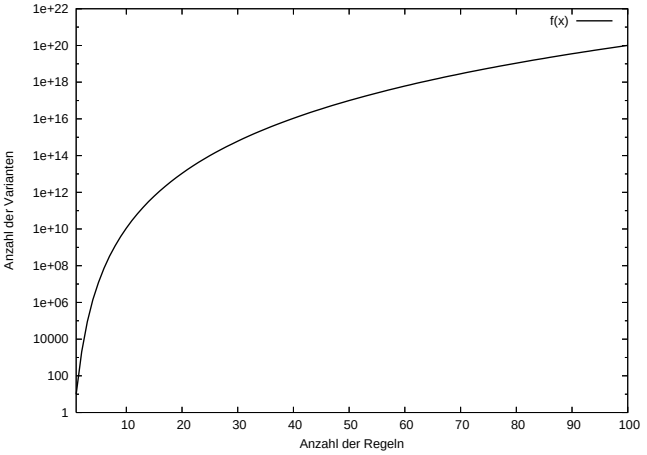


Abbildung 1.4: Wachstum der Variantenanzahl

Aus diesem Grund sollte man entweder die Erzeugung von Ausführungsplänen mit Bedingungen hinreichend beschränken oder nach einer festen Anzahl von Ersetzungen bzw. Varianten die Optimierung strikt abbrechen (siehe Kapitel 5). Diese Bedingungen bilden im Suchraum einen Korridor bzw. das Suchfenster (violette Linien in Abbildung 1.3), welches die Generierung von neuen Ausführungsplänen sinnvoll beschränkt. Um die Grenzen dieser Korridore bestmöglich angeben zu können, wird eine Regelsteuerung benötigt, deren Komponenten wie folgt zusammengefasst werden:

- *lokale Restriktionen*: Es werden individuelle Bedingungen, unter denen eine Regel angewandt wird, definiert. Diese Bedingungen können zum Beispiel aus Kostenfunktionen zusammengesetzt sein, die den vorherigen Ausführungsplan in Relation zu dem neuen setzen.

- *spezifizierte Regelabfolgen*: Die Reihenfolge der Regelanwendungen wird durch reguläre Ausdrücke für die Anwendung, Iteration und Generierung von Alternativen definiert.
- *globale Strategien*: Beliebige Suchstrategien können flexibel kombiniert werden. Als Beispiel seien heuristisch basierte Suchstrategien genannt, welche auf eine gegebene Datenmenge angepasst werden können.

Im Gegensatz zur Dynamischen Programmierung bietet der in dieser Arbeit vorgestellte Ansatz die Möglichkeit, schon während der Optimierung vollständige Ausführungspläne zu liefern. Darüber hinaus bietet er den Vorteil, Aussagen darüber treffen zu können, ob eine weitere Optimierung länger dauern würde als die komplette Ausführung des aktuellen Planes. Dies kann durch ein Kostenmodell sichergestellt werden, welches nicht nur alleine asymptotische Laufzeiten betrachtet, sondern realistische Abschätzung für die Bearbeitungsdauer der Anfrage vornimmt.

1.4 Bedingte Termersetzung bei objektrelationalen Operatoren

Objektrelationale Datenbanken (ORDB, ORDBMS) stellen das Bindeglied zwischen klassischen relationalen und objektorientierten Datenbanksystemen dar. Sie kommen überall dort zum Einsatz, wo der Anwendungsbereich eines relationalen Modells nicht mehr ausreicht (z. B. bei Nicht-Standard-Datentypen) (siehe [54]). Ein typisches Einsatzgebiet sind unter anderem Systeme zur Erfassung geographischer Daten (GIS), bei denen Koordinaten miteinander verknüpft sind oder andere Daten referenzieren. Beispielsweise referenzieren mehrere Koordinaten-Objekte eine Straße: die Koordinaten stehen in Relation zu einem Straßennamen und sind selbst Objekte, die zueinander eine Beziehung haben. Dies ermöglicht räumliche Anfragen, wie z. B. das Auffinden aller Bahnübergänge, die eine vorgegebene Strasse „schneiden“. Demzufolge sind die entscheidenden Merkmale dieser Umgebung die Komplexität der Attribute und die darauf agierenden zeitintensiven Algorithmen.

In einem DBS-Umfeld, welches die Verwendung von z. B. relationalen und räumlichen Operatoren nötig macht, scheitern die meisten Optimierer (siehe Abschnitt 1.6), da sich räumliche Operatoren nicht einfach ohne größeren Aufwand integrieren lassen. Ein flexibles Termersetzungssystem löst dieses Problem, da es für beliebige Operatoren erweitert werden kann, ohne dass strukturell neue Module entwickelt werden müssen. Nachdem ein Operator definiert worden ist und für ihn Kostenabschätzungen entwickelt worden sind, kann man den gegebenen Stamm an Regeln einfach erweitern und flexibel justieren. Als Grundlage dafür dient die Parametrisierung von relationalen Operatoren hinsichtlich ihrer Implementierungsvariante als physischer Operator (siehe Kapitel 4).

1.5 Ziel dieser Dissertation

Ziel dieser Dissertation ist die Untersuchung und Realisierung eines flexiblen Optimierungskonzeptes für objektrelationale Anfragen. Der Optimierer wird als System von bedingten Termersetzungen definiert und für beliebige objektrelationale Operatoren erweiterbar sein.

Als modular aufgebauter Optimierer (siehe Kapitel 7) kann er verschiedene Optimierungsstrategien kombinieren und bewerten (siehe Kapitel 6), so dass Aussagen über sinnvolle Operatoranordnungen gemacht werden können. Zu dem Optimierer gehören eine flexible Regelsprache (siehe Kapitel 5), eine Regelsteuerungssprache (siehe Abschnitt 5.2), eine neue Datenstruktur

für Ausführungspläne (siehe Abschnitt 2.10) und eine systematische Darstellung von selbst entwickelten Abschätzungen bzw. Kostenfunktionen (siehe Kapitel 2 und 4) für objektrelationale bzw. relationale Operatoren.

Schließlich soll eine Experimentierumgebung geschaffen werden, mit deren Hilfe man die Anordnung von Optimierungsregeln hinsichtlich eines Datenbankinhaltes und den darauf agierenden Anfragen untersuchen kann. Insbesondere werden Experimente durchgeführt, um zu zeigen inwieweit sich bisherige Optimierungsstrategien vom relationalen Fall auf den objektrelationalen Fall übertragen lassen (siehe Abbildung 1.5).

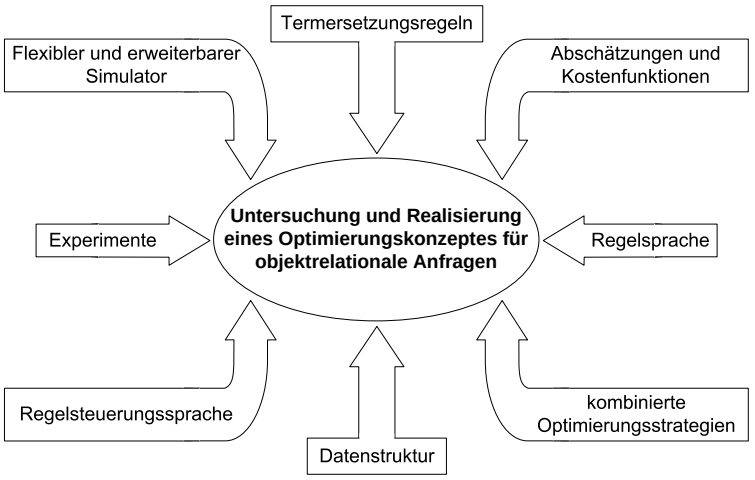


Abbildung 1.5: Mittel zur Erreichung der Dissertationsziele

1.6 Verwandte Arbeiten

Als Basis für diese Arbeit dienen insbesondere Veröffentlichungen, die aus den gemeinsamen Arbeiten der Universität Konstanz und Rostock aus den Jahren 1995 bis 1999 stammen. Die Forschungsgruppen entwickelten im Laufe des Projekts „Cost- and Rulebased Optimization of object-oriented QUEries“ (CROQUE) ein objektorientiertes Anfrageauswertungssystem, welches folgende Ausrichtungen besaß:

(a) Eine deskriptive, objektorientierte Anfrageschnittstelle.

(b) Datenunabhängigkeit, d. h. Trennung der konzeptionellen Struktur von den physischen Organisationen und Zugriffspfaden, die in den bisherigen objektorientierten Datenbanksystemen in der Regel nicht gegeben war.

(c) Eine regel- und kostenbasierte Optimierungskomponente zur Abbildung von deskriptiven Anfragen der konzeptuellen Ebene auf Ausführungspläne in Abhängigkeit von der gewählten Speicherstruktur.

Wir konzentrieren uns in dieser Arbeit auf den letzten Punkt, der zuerst 1996 [34] vorgestellt wurde. Eine heuristisch basierte Anordnung von algebraischen Anfrageregeln wurde im Jahre 1998 [48] veröffentlicht. Eine abschließende Analyse aller bis dato entwickelten Verfahren und Resultate wurde 1999 [47] bekannt gemacht. Das betrachtete Hauptziel bzgl. des Aspektes der Anfrageoptimierung kann dahingehend zusammengefasst werden, dass eine flexiblere Anordnung von algebraischen Optimierungsregeln angestrebt worden ist. Dabei wurde die Idee entwickelt, Regeln mit einem größeren „Optimierungspotential“ zu benutzen, statt diejenigen, welche keine beachtlichen Kosteneinsparungen liefern. Daraus resultierend wurde eine Gruppe von Regeln empfohlen, die priorisiert auf den Anfrageplan angewandt werden sollten. Als eine der Maßeinheiten wurde auch die Anzahl von Rewrites betrachtet und dahingehend heuristisch optimiert, dass eine Kette von mehreren Termersetzungen (bzw. die Nutzung vieler Regeln) einer kürzeren (bzw. Nutzungen weniger Regeln) zu bevorzugen sei:

„The idea of our heuristic is that using more rules for generating a plan will result in a better plan than using less rules for generation because the rules are only applied in the most favourable direction. More rule applications, i. e. improvements, thus result in better plans.“ [47]

Diese Heuristik konnte aber durch kein aussagekräftiges Resultat untermauert werden:

„[...], the pure number of rule applications is not sufficient as a means for plan ordering since the global optimal plan never was the one created using the greatest number of rule applications. [...] The obvious main drawback of our approach is the necessarily exhaustive search space generation. [...] Whereas the proposed heuristics based on the quantity of rule applications may only be used after the rewriting has completed, [...]“ [47]

Da dieser Ansatz auf algebraische Regeln beschränkt war, konnten Optimierungsstrategien bezüglich der Generierung von neuen redundanten Daten, z. B. Indexen und Sortierungen, nicht verfolgt werden. Grund dafür war die Annahme, dass eine strikte Trennung der logischen und physischen Optimierung von Vorteil wäre, da sonst die Anzahl von Regeln und die daraus resultierenden Termersetzungen unkontrollierbar würden (z. B. wurde im Zuge des COKO-KOLA Optimierers [13] (1998) bei 60 Operatoren schon eine Menge von 600 zu betrachtenden Regeln definiert).

Das Anordnen von Optimierungsregeln wurde als erstes im Rahmen von EXODUS [25], [26], [27], [28] (1986-88) und seinem Nachfolger Volcano [9], [14], [29], [61], [89], (1993/94) vorgestellt. Obwohl dem Benutzer eine Veränderung der Regelgruppen erlaubt wurde, waren die Möglichkeiten zur Anordnung von einzelnen Regeln innerhalb einer Gruppe begrenzt. Grund dafür war das Fehlen von Anpassungsmöglichkeiten an beliebige Datenbestände.

Diesen Missstand überwanden weitere Projekte, darunter COKO-KOLA [13] (1998), Starburst [32], [50], [56], [65], [68] (1988-92) und GOM [39] (1990), durch das Anbieten von flexibleren Schnittstellen. Allerdings waren diese mit einem erheblichen Programmieraufwand für jegliche Art von Regeln verbunden. Erweiterungen für objektrelationale Operatoren waren nicht möglich, ohne dass ganze Module neu entwickelt werden mussten.

Die Kostenberechnungen sowohl für alle Zwischenvarianten eines Anfrageplanes als auch für den Ausgangsbaum wurden zuerst von EXODUS und Gral [6] (1992) durchgeführt. Die dadurch verbesserten Optimierungen waren aber mit stark verlangsamten Berechnungen verbunden, so dass sich die bis dato genutzte Datenstruktur für Anfragepläne im Nachhinein als ungeeignet darstellte.

Zu den aktuelleren Arbeiten zur Anfrageoptimierung gehören die Veröffentlichungen, die im Zuge des OPT++-Projektes [36], [37] (1999/2000) entstanden. Hierbei ermöglichte man erstmals eine objektorientierte Anfrageoptimierungsumgebung, die auch „Nicht-Standard-Anfragen“ bearbeiten konnte. Obwohl diese Arbeit einen immensen Sprung hinsichtlich Flexibilität und Wartbarkeit des Systems darstellte, wurde auch hier eine strikte Trennung zwischen Optimierungsphasen vorgenommen.

In der vorliegenden Arbeit wird als Beispiel für die Flexibilität eines relationalen Datenbanksystems hinsichtlich objektrelationaler Operatoren ein räumliches Datenbanksystem gewählt, welches ein relationales Modell um geometrische Datentypen (z. B. Punkt, Linie, Polygon) und Operationen (z. B. Fensteranfragen) erweitert. Dafür muss auch das Kostenmodell erweitert und flexibler gemacht werden. Als Basis dienen die grundlegenden Ansätze für räumliche Datenbanksysteme, welche schon 1991 bzw. 1995 von Aref et al. in [4], [75] vorgestellt wurden.

Das in dieser Arbeit verwendete Kostenmodell basiert auf dem im Jahr 2000 veröffentlichten Artikel von Aboulnaga et al. [1]. Um die dort durchgeführten Kostenabschätzungen exakter vor-

nehmen zu können, wurde es – wie auch im relationalen Fall – verfeinert und auf Seitenkosten umgestellt. Die dafür benötigten Selektivitätsabschätzungen stammen aus einem Übersichtsartikel von Papadias et al. [60] (2001/2004). Das Basiswissen über Algorithmen und Laufzeiten von räumlichen Indexstrukturen wurde aus der Dissertation von Kleiner [42] (2003) und den Vorlesungen „Räumliche Datenstrukturen“ [51] (2004) und „Multidimensionale Datenbanken“ [53] (2006) bezogen. Zusätzlich wurden die Ergebnisse aus Artikeln über räumliche Indexstrukturen integriert [44], [46], [83] (1998/2002/2004), um Abschätzungen auch für derartige Zugriffsstrukturen tätigen zu können. Feinheiten zu räumlichen Datenbanken wurden aus Shekhar et al. [80] (2003) und einem der ersten Lehrbücher zu diesem Thema von Adam et al. [3] (1997) bezogen.

1.7 Klassifikation des entwickelten Optimierungskonzeptes

Um das in dieser Arbeit realisierte Optimierungskonzept namens `RELOpt` in Beziehung zu den vorher genannten Optimierern setzen zu können, benutzen wir eine Klassifikation, die erstmals in [48] vorgestellt und benutzt wurde. Dabei erweitern wir den Kontext noch um die in dieser Arbeit wichtigen Punkte 5-8 (siehe Tabelle 1.1):

1) Unterstützt das System eine freie Regelanordnung (z. B. durch einen Regelablaufbaum)?
2) Kann das System auch ohne eine anfangs festgelegte Regelanordnung optimieren?
3) Welche Art der Regelanordnung wird unterstützt (z. B. online, offline oder fest programmiert)?
4) Welche Art von Bedingungen können zur Regelanordnung berücksichtigt werden?
   a) kostenbasierte
   b) statistikbasierte
   c) heuristische
5) Können Erweiterungen neuer Regeln über eine eigenständige Regelsprache, z. B. mit Hilfe einer Maske, eingegeben werden?
6) Ermöglicht das System die flexible Kombination von verschiedenen Optimierungsheuristiken (z. B. Push-Down Selections mit n-best Strategie)?
7) Berücksichtigt das System objektrelationale Operatoren?
8) Durchbricht das System die starre Phasenoptimierung?

Schließlich ist `RELOpt` der einzige regelbasierte Optimierer, der eine flexible Datenstruktur – den „generalisierten“ Baum (siehe Abschnitt 2.10) – für die Darstellung von Ausführungsplänen besitzt. Bisherige Ansätze berücksichtigen teilweise verschiedene Punkte wie Flexibilität, Erweiterbarkeit oder Laufzeit nicht. Um den Aspekt Laufzeit auch für komplexere Operatoren exakt abschätzen zu können, wurde auf Basis einer attributierten Grammatik ein generalisierter Operatorbaum definiert, der Neuberechnungen von Statistiken bei der Transformation eines Ausführungsplanes, durch die im Baum vorherrschende inkrementelle Kostenberechnung, auf das Minimum reduziert (siehe Kapitel 3.3).

System / **Kriterium**	Volcano	EXODUS	COKO-KOLA	Starburst	GOM	Gral	CROQUE	OPT++	`RELOpt`
1) freie Regelanordnung	ja	nein	nein	ja	ja	ja	ja	ja	ja
2) freie Anfangsreihenfolge von Regeln	nein	ja	nein	nein	nein	ja	ja	ja	ja
3) Art der Regelanordnung	online	online	prog.	online/prog.	prog.	online	offline	prog.	online/offline/prog.
4a) kostenbasierte Bedingungen	ja	ja	nein	nein	nein	ja	ja	ja	ja
4b) statistikbasierte Bedingungen	ja	ja	nein	nein	nein	nein	ja	ja	ja
4c) heuristische Bedingungen	ja	nein	ja	ja	ja	ja	ja	ja	ja
5) flexible Regelsprache	nein	ja	nein	nein	nein	ja	ja	nein	ja
6) flexible Kombination von Strategien	ja	ja	nein	nein	nein	ja	ja	ja	ja
7) objektrelationale Operatoren	nein	nein	nein	nein	nein	nein	nein	ja	ja
8) Durchbruch der Phaseneinteilung	ja	ja	ja	nein	nein	ja	nein	nein	ja

Tabelle 1.1: Vergleich von `RELOpt` mit bisherigen Optimierungsansätzen.

1.8 Aufbau der Dissertation

Die Kapitel in dieser Arbeit gliedern sich wie folgt:

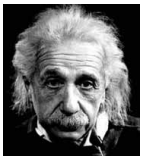
Im **Kapitel 2** werden die grundlegenden Definitionen und Funktionen für das in dieser Arbeit verwendete Kostenmodell vorgestellt. Anschließend wird im **3. Kapitel** die Optimierung von Ausführungsplänen mit Hilfe eines bedingten Termersetzungssystems entwickelt. Dafür werden Bedingungen analysiert, deren Verwendung die Anzahl von neu generierten, äquivalenten Anfragebäumen stark beschränkt, um den Berechnungsaufwand zu minimieren. Insbesondere werden Methoden zur Vereinfachung von Kostenbedingungen entwickelt, um die Berechnung von Ausführungskosten angemessen vereinfachen zu können.

Das wichtigste Kapitel ist das **4. Kapitel**. Hier wird das zuvor vorgestellte Konzept auf objektrelationale Operatoren ausgeweitet. Als Beispiel für ein objektrelationales Datenbanksystem wird ein räumliches Datenbanksystem verwendet. **Kapitel 5** stellt die Steuerung des Termersetzungsoptimierers dar und beschreibt ausführlich den modularen Aufbau des Systems.

Im **6. Kapitel** werden verschiedene Optimierungsstrategien und ihre Anwendbarkeit untersucht. Danach folgen Einzelheiten der Implementierung des Termersetzungsoptimierers `RELOpt` im **7. Kapitel**. Schließlich gibt **Kapitel 8** einen Ausblick auf weitere mögliche Forschungsarbeiten.

Man muss die Welt nicht verstehen,
man muss sich nur darin zurechtfinden.

Albert Einstein 1879-1955



2 Kostenmodell

Eine kostenbasierte Optimierung benötigt als Basis ein Kostenmodell, das dem Optimierer erlaubt, die geschätzten Kosten verschiedener Ausführungspläne in Beziehung zu setzen, um daraus eine Entscheidung für einen Ausführungsplan treffen zu können. Grund dafür ist die Tatsache, dass man nicht jeden Ausführungsplan erst anwenden sollte, um dann die Dauer der Ausführung als Messwert für die verschiedenen Pläne zu nehmen.

Wie wichtig ein gutes Kostenmodell ist, zeigt sich darin, dass kommerzielle Systeme zwar einen Kostenwert für ihre Ausführungspläne angeben, aber nicht, wie sie ihn berechnen. Somit sind Open Source Datenbanken (z. B. PostgreSQL[1], MySQL[2]) hinsichtlich einer guten Optimierung schlechter gestellt, da sie als Basis für den Optimierer meist einfache Kostenmodelle benutzen.

Die Idee bei einem Kostenmodell ist es, jegliche Meta- und Kosteninformationen im relationalen DBMS zu speichern, um mit ihrer Hilfe im Laufe der Optimierung Entscheidungen fällen zu können, welche Form der Ausführungsplan haben soll (z. B. Left-Deep-Tree vs. Bushy-Tree) und welche Operatoren bzw. Implementierungsarten an welchem Knoten genutzt werden können. Beim Hinzufügen von Operatoren (z. B. Sortierungen und Indexe) muss der Optimierer den zusätzlichen Mehraufwand für die Generierung der redundanten Daten in Beziehung mit dem originalen Ausführungsplan setzen. Solche Entscheidungen kann er jedoch erst mit einem Kostenmodell tätigen.

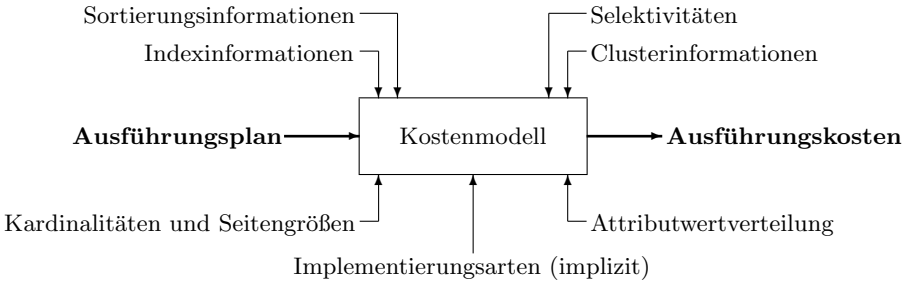


Abbildung 2.1: Funktionsweise eines Kostenmodells

Ein Kostenmodell bietet insbesondere die Funktion, die Kosten einzelner Operatoren abzuschätzen. Dazu werden diverse Parameter benötigt, wie gegebene Indexe, Clusterinformationen und Sortierungen, sowie, wenn nicht statistisch aufgeführt, neu zu berechnende Werte für z. B. Kardinalitäten, Seitengrößen oder Attributwertverteilungen. Somit muss u. a. die Anzahl der im Berechnungsprozess benötigten Tupel durch sinnvolle Funktionen abgeschätzt und verglichen werden. Das Ergebnis der Ausführungskosten berechnet sich aus mehreren Größen, die dem Zeit- oder Platzbedarf für die Anfrageausführung untergeordnet sind. Für den Zeitbedarf sind die Kosten für Zugriffe auf den externen Speicher für Tupelzugriffe bzw. Seitenzugriffe und natürlich der CPU-Berechnungsaufwand entscheidend. Für den Platzbedarf ist die Gesamtgröße aller im Speicher zu haltenden Relationen in Seiten wichtig.

[1] `http://www.postgresql.org`
[2] `http://www.mysql.com`

2.1 Allgemeine Konventionen dieser Arbeit

Um dem vielfach widersprüchlichen Sprachgebrauch in der Anfrageoptimierung entgegenzuwirken, wurden folgende Konventionen für diese Arbeit festgelegt:

$R, R_1, R_2, \ldots, R_k$	Variablen für relationale Terme
$\overline{A}, \overline{A}_1, \overline{A}_2, \ldots, \overline{A}_l$	Attributmengen
$A, B, C, D, ...$	Attribute
$dom(A)$	Wertebereich des Attributes A
$\varphi, \varphi_1, \varphi_2, \ldots, \varphi_m$	Bedingungen
a, b	Konstanten
$n(A, R)$	Anzahl unterschiedlicher Werte des Attributes A in R
$max(A, R), min(A, R)$	Maximum bzw. Minimum der Werte des Attributes A in der Relation R
$\#1, \#2, \#3, ...$	Aliase zur verkürzten Darstellung von verwendeten Funktionen
$\|R\|$	Kardinalität einer Relation R
$pg(R)$	Seitenanzahl einer Relation R
$attr(\varphi)$	Menge der in φ enthaltenen Attribute
$attr_i(\varphi)$	Menge der in φ enthaltenen Attribute, die sich auf die $i-$te Relation R_i beziehen
$\mathbf{sch}(R)$	das Schema (gleich der Menge der Attribute) einer Relation R
$\tau, \tau_1, \ldots, \tau_n$	Ausführungspläne
$\overset{1}{\underset{2}{\Rightarrow}}$	gerichtete bedingte Termersetzung mit den Bedingungen 1) und 2)
$tlg(R)$	Tupellänge einer Relation R in Bytes normiert auf die systemabhängige Seitengröße
$tlg(A, R)$	Tupellänge des Attributes A in der Relation R in Bytes normiert auf die systemabhängige Seitengröße
$cost(\tau)$	Kosten des Ausführungsplanes τ
$cost_{op}(\tau)$	Kosten des Wurzeloperatorknotens des Ausführungsplanes τ
$cost^+_{op}(\tau)$	gewichtete Kosten des räumlichen Wurzeloperatorknotens des Ausführungsplanes τ
$\omega^{\epsilon}_{(\varphi_{geo},R)}$	Gewichtungsfaktor für räumliche Operatoren
s^{ϵ}	Systemabhängiger Gewichtungsfaktor für Tupelzugriffe auf räumliche Attribute
x^{top}	Gewichtungsfaktor für die topologische Beziehungsfunktion top
y^{top}_{ij}	Gewichtungsfaktor für die topologische Beziehungsfunktion top in Abhängigkeit der Argumentgeometrietypen i und j
$Index_A(R)$	Index auf der Ergebnisrelation eines relationalen Termes R, nach dem Attribut A
$Index_{\overline{A}}(R)$	kombinierter Index auf der Ergebnisrelation eines relationalen Termes R, nach den Attributen der Attributmenge $\overline{A}$
$\|Index_A(R)\|_l$	Höhe des Indexbaumes als Maß für die Anzahl von Zugriffen zum Auslesen eines Tupels
$\|Index_A(R)\|_c$	Kardinalität der im Index referenzierten Tupel
$op(\varphi)$	liefert die booleschen Operationen in der Bedingung φ
$op(Index_A(R))$	liefert die vom $Index_A(R)$ unterstützten Zugriffsoperationen
$sel(\varphi, R)$	Selektivität der Selektionsbedingung φ bezüglich des relationalen Terms R
$func(\overline{F})$	liefert alle Aggregationsfunktionen aus der Liste der Aggregierungsterme $\overline{F}$
$gt(A)$	liefert den Geometrietyp der Argumentgeometrie A
$\overline{s_d}(A, R)$	gibt die durchschn. Größe der Geometrien A in der Rel. R bzgl. der Dimension d an
$\overline{r_d}(A, R)$	gibt die Größe des Wertebereichs für die Geometrien A aus R an bzgl. der Dimension d
$\overline{W_d}$	gibt die Größe des Anfragefensters W bzgl. der Dimension d an

$SORT_A(R)$	sortiert die Relation R nach dem Attribut A
$INDEX_A(R)$	erzeugt einen Index auf der Ergebnisrel. eines relat. Termes R nach dem Attr. A
$supports_index_access(R, A)$	liefert $true$, falls der Zugriff auf das Attribut A von R über einen passenden Index möglich ist, sonst $false$
$supports_ordered_access(R, A)$	liefert $true$, falls der gewünschte sortierte Zugriff auf das Attribut A von R möglich ist, sonst $false$

Relationale Operationen:

$R_1 \cap R_2$	Durchschnitt zweier Relationen R_1 und R_2
$R_1 \cup R_2$	Vereinigung zweier Relationen R_1 und R_2
$R_1 - R_2$	Differenz zweier Relationen R_1 und R_2
$R_1 \times R_2$	Kartesisches Produkt zweier Relationen R_1 und R_2
$\pi_{\overline{A}}(R)$	Projektion einer Relation R auf die Attributmenge $\overline{A}$
$\sigma_{\varphi}(R)$	Selektion einer Relation R gemäß der Bedingung φ
$R_1 \bowtie_{\varphi} R_2$	Verbund zweier Relationen R_1 und R_2 gemäß der Bedingung φ
$R_1 \ltimes_{\varphi} R_2$	Reduzierung der Relation R_1 auf die Tupel, die gemäß der Bedingung φ einen Joinpartner in R_2 haben (Semijoin)
$R_1 \overline{\ltimes}_{\varphi} R_2$	Reduzierung der Relation R_1 auf die Tupel, die gemäß der Bedingung φ keinen Joinpartner in R_2 haben (Antisemijoin)
$R_1 \rtimes_{\varphi} R_2$	Verbund zweier Relationen R_1 und R_2 gemäß der Bedingung φ mit anschließender Hinzufügung aller Tupel aus R_1, die keinen Joinpartner besaßen
$R_1 \ltimes_{\varphi} R_2$	Verbund zweier Relationen R_1 und R_2 gemäß der Bedingung φ mit anschließender Hinzufügung aller Tupel aus R_2, die keinen Joinpartner besaßen
$R_1 \rtimes_{\varphi} R_2$	Verbund zweier Relationen R_1 und R_2 gemäß der Bedingung φ mit anschließender Hinzufügung aller Tupel aus R_1 und R_2, die keinen Joinpartner besaßen
$\Gamma_{\overline{G}\#\overline{F}}(R)$	Gruppierung der Relation R nach den Attributen der Menge $\overline{G}$ und den gruppeninvarianten relationalen Attributtermen bzw. aggregierenden Funktionen $\overline{F}$

Im Exponenten werden Implementierungsvarianten von relationalen Operatoren angegeben:

elim	weist auf einen duplikateliminierenden Operator hin
Rel	relationaler Zugriff auf die Argumentrelation
Sort	Zugriff auf eine sortierte Argumentrelation
Index	indexbasierter Zugriff auf eine Argumentrelation
Hash	hashbasierter Zugriff auf eine Argumentrelation

Für eine Join-Implementierung folgt beispielsweise mit einfachem relationalen Zugriff auf R_1 und einem indexbasierten auf R_2: $R_1 \bowtie_{\varphi}^{Rel,Index} R_2$.

2.2 Definition eines Kostenterms

Notation 2.2.1 Ein aus der deklarativen Anfragesprache abgeleiteter Operatorbaum sei mit τ als Anfragebaum bezeichnet. Dabei werden Argumentrelationen und die darauf agierenden algebraischen Operatoren als Knoten K dieses Baumes dargestellt. Die Kanten geben die Reihenfolge der Anwendung der Operatoren an und werden „bottom up“ gelesen (siehe Abbildung 2.2).

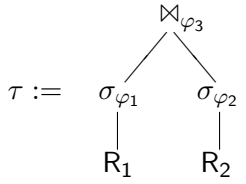


Abbildung 2.2: Beispiel für einen Anfragebaum

Notation 2.2.2 Ersetzt man die algebraischen Operatoren eines Anfragebaumes durch ihre verschiedenen physischen Implementierungsarten, resultiert daraus die Menge aller möglichen äquivalenten Ausführungspläne (siehe Abbildung 2.3). Dementsprechend sei der Suchraum der Planoptimierungen, der alle zu τ semantisch äquivalenten Ausführungspläne umfasst, mit $T(\tau)$ bezeichnet.

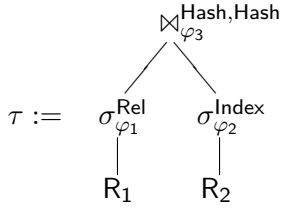


Abbildung 2.3: Beispiel für einen Ausführungsplan

Die allgemeine Kostenberechnung sei wie folgt definiert:

Definition 2.2.3 Die gesamten Kosten eines Ausführungsplanes τ setzen sich zusammen aus den Kosten der Operation der Wurzel und den rekursiv berechneten Kosten der Teilbäume $\tilde{\tau}$ an der Wurzel K:

$$\mathsf{cost}(\tau) = \mathsf{cost}_{\mathsf{op}}(\tau) + \sum_{\substack{\tilde{\tau} \text{ Teilbäume} \\ \text{am Knoten K}}} \mathsf{cost}(\tilde{\tau})$$

Dafür benötigt man die Kosten jedes einzelnen Knotens K, die wiederum abhängig sind von der dort auszuführenden Operation und den Relationen, auf denen sie angewandt werden. Dabei gehen beispielsweise die gewählten Zugriffsmethoden, die Zugriffsreihenfolgen und die Selektivitäten der Prädikate als Parameter in die Kostenberechnungen ein.

Definition 2.2.4 Zielsetzung der Optimierung ist die Generierung eines optimalen Ausführungsplanes $\tau_{\mathsf{opt}} \in \mathsf{T}(\tau)$, so dass gilt:

$$\mathsf{cost}(\tau_{\mathsf{opt}}) = \mathsf{min}\{\mathsf{cost}(\overline{\tau}) \,|\, \overline{\tau} \in \mathsf{T}(\tau)\}$$

Die verwendete Kostenfunktion ist maßgebend für die Auswahl möglicher Alternativen der Implementierung von Zugriffsmethoden. In den folgenden Abschnitten werden deshalb die benötigten Funktionen und Abschätzungen vorgestellt, die später als Basis unserer Bewertung eines Ausführungsplanes dienen werden.

2.3 Definition einer Kostenfunktion

Die in dieser Dissertation verwendeten Formeln zur Abschätzung der Ausführungspläne basieren – wie alle heute bekannten Entwicklungen – auf dem Kostenmodell von Selinger, Astrahan, Chamberlin, Lorie und Price [79], welches 1979 für das Datenbanksystem System R von IBM entwickelt wurde. Grund dafür sind die einfachen Kardinalitätsabschätzungen von Anfrageergebnissen bzw. Zwischenergebnissen. Der Hauptunterschied zu dem genannten System besteht darin, dass das in der vorliegenden Arbeit verwendete Kostenmodell nicht mehr nur Tupelanzahlen vergleicht [79], sondern auch die Kostenabschätzung auf Basis von Seitenbelegungen durchführt. Durch dieses Vorgehen lassen sich letztlich präzisere und sinnvollere Kostenaussagen treffen.

Eine Alternative zu beiden sind Modelle, die konkret Teilpläne ausführen und zur Optimierung die benötigte Bearbeitungszeit nutzen. Dazu gehört die Familie der dynamischen Optimierungsstrategien wie sie z. B. im Projekt Eddy genutzt werden (siehe dazu [5]).

Die jeweiligen Formeln zur konkreten Kostenabschätzung der unterstützten Zugriffsmethoden sind in den in diesem Kapitel aufgeführten Tabellen zusammengefasst. Dabei werden die üblichen statistischen Annahmen bezüglich der Gleichverteilung der Werte über den Wertebereich eines Attributes und bezüglich der Unabhängigkeit zwischen den Werten verschiedener Attribute getroffen.

Um zu einer zuverlässigen Abschätzung der Zugriffskosten $\mathsf{cost}(\tau)$ eines Ausführungsplanes τ zu kommen, müssen in den Kostenfunktionen diejenigen Ressourcen berücksichtigt werden, welche den größten Einfluss auf die Anfrageausführung ausüben. Hierzu gehört der externe Speicherbedarf (lokal oder dezentral abgespeicherte Relationen), die Anzahl der Disk-I/O-Zugriffe in Seiten, der im Systempuffer benötigte Speicherplatz sowie die notwendige CPU-Zeit zur Ausführung eines spezifischen Ausführungsplanes. Aus diesem Grund müssen wir für sinnvolle Kostenmodelle die Prozessorkosten (CPU-Kosten), Ein-/Ausgabe-Kosten (I/O-Kosten) und Netzwerkkosten (NET-Kosten) gleichzeitig betrachten. Dieser Sachverhalt wird nicht nur durch alle Kostenmodelle in realisierten Datenbankmanagementsystemen bestärkt, sondern ist auch in der Theorie fester Bestandteil aller Untersuchungen, siehe dazu insbesondere [79].

Vor einem Jahrzehnt fiel der größte Berechnungsaufwand im Zugriffs- und Speichersystem an, so dass dieser Wert für vereinfachte Kostenmodelle als Grundlage der Kostenberechnung diente. Natürlich wurde dabei eine mögliche falsche Abschätzung in Kauf genommen, die sich abhängig von der CPU-Belastung vergrößerte oder relativierte. Heutige Systeme besitzen immer größer werdende Caches (viele I/O-Zugriffe werden so vermieden), dadurch erlangen die CPU-Kosten einen höheren Anteil an den Gesamtkosten.[3] Als weiterer Punkt kommen die Netzwerkkommunikationskosten hinzu, da auch dezentrale Datenbanken (z. B. föderierte Datenbanken) Ausführungspläne abschätzen können müssen, bei denen die Relationen verteilt auf verschiedenen Netzwerkknoten liegen.

Bei dem für diese Dissertation entwickelten Kostenmodell werden, basierend auf den Habilitationsschriften von Mitschang [62] und Moerkotte [63], I/O-Kosten ausschließlich als Input-Kosten gesehen, da die gesamten I/O-Kosten dieselbe Größenordnung haben wie allein die Input-Kosten. Dies ist der Fall, da nach jeder Operation das Zwischenergebnis in den Speicher geschrieben wird und danach wieder für die nächste Operation ausgelesen werden muss. Beide Schritte benötigen ungefähr denselben Aufwand. Schließlich speichern alle äquivalenten Ausführungspläne dieselbe Schlussausgabe, so dass auch diese vernachlässigt werden kann. Zusätzlich muss eine spezielle Kostenkorrektur für eventuell vorkommendes Pipelining vorgenommen werden (siehe Abschnitt 2.8).

Schließlich werden die besagten Kostenmodelle zu einem neuen Modell verfeinert (u. a. durch Betrachtung von Seitenzahlen anstatt von Tupelanzahlen) und zusätzlich flexibler gestaltet, so dass ein leicht verständliches und somit einfach erweiterbares Kostenmodell entsteht, welches spätere Erweiterungen z. B. hinsichtlich objektrelationaler Operatoren (siehe Kapitel 4) unterstützt.

Definition 2.3.1 Die Abschätzung der Kosten für einen Ausführungsplan τ lautet:

$$\boxed{\mathsf{cost}(\tau) = \mathsf{cost}_{\mathsf{I/O}}(\tau) \; + \; \mathsf{w}_1 \cdot \mathsf{cost}_{\mathsf{CPU}}(\tau) \; + \; \mathsf{w}_2 \cdot \mathsf{cost}_{\mathsf{NET}}(\tau)}$$

wobei $\mathsf{w}_1, \mathsf{w}_2$ die Gewichtung von I/O-, CPU- und NET-Kosten zueinander darstellt. Dabei ist der Wert abhängig von der jeweiligen (örtlichen) Systemkonfiguration (Hard- und Software).

Um die drei stark unterschiedlichen Kostenanteile zu den gesamten Zugriffskosten zusammenfassen zu können, werden sie zueinander normiert. Dies erfolgt durch die Proportionalitätsfaktoren $\mathsf{w}_1, \mathsf{w}_2$, die das durchschnittliche Verhältnis des Aufwandes für einen Aufruf des Zugriffssystems zu einem Seitenzugriff auf den externen lokalen bzw. globalen Speicher angeben.

In unserem Modell können die drei vorkommenden Extremfälle berücksichtigt werden [79]:

(a) **CPU-Beschränkungen:** Ist die CPU der Engpass des Systems, sind natürlich I/O-Zugriffe zu bevorzugen, so dass rechenintensive Ausführungspläne bei der Optimierung benachteiligt werden. Für eine I/O-Operation werden lediglich die hierzu auszuführenden Instruktionen angesetzt. Dabei wird angenommen, dass die Zugriffszeit voll überlappend zur weiteren Verarbeitung ausgenutzt werden kann.

$$\mathsf{w}_1 = \frac{\#\text{CPU-Instruktionen pro Sekunde}}{\#\text{I/O-Instruktionen pro Sekunde}}$$

(b) **I/O-Beschränkungen:** Normalerweise sind es I/O-Zugriffe, die am meisten Zeit benötigen. Deshalb sollte man rechenintensive Ausführungspläne bevorzugen. Aus diesem Grund

[3] Cachingeffekte hinsichtlich der von der CPU benötigten Daten sind stark von der Größe des einzelnen CPU-Caches abhängig und spiegeln sich in der MIPS-Leistung des Prozessors wider (siehe A.3).

dreht sich das Verhältnis um und es wird der dominierende I/O-Kostenanteil als Zähler genommen.

$$w_1 = \frac{\#\text{CPU-Instruktionen pro Sekunde}}{\#\text{I/O-Instruktionen pro Sekunde}}$$

Eigene Erfahrungswerte aus Experimenten (siehe Kapitel 6) zeigen, dass eine Gewichtung der CPU-Kosten mit $w_1 = 0,2$ zur Generierung des besten Ausführungsplanes führen kann (siehe hierzu [24]).

(c) **NET-Beschränkungen:** Bei dezentralen Datenbanken, die einen Teil ihrer Kommunikation nicht direkt, sondern über ein Netzwerk ausführen, kommen so genannte NET-Kosten dazu:

$$w_2 = \frac{\text{Bus-Transferrate}}{\text{NET-Transferrate}}$$

Standardmäßig wird bei heutigen Datenbanksystemen der Wert w_2 auf $1,5$ gesetzt (siehe dazu [24]). Bei zentralen Datenbanken ohne NET-Kommunikation entfällt natürlich dieser Faktor, d. h. $w_2 = 0$.

Schließlich müssen Gewichtungsfaktoren proportional verändert werden, wenn man einzelne Kostenmodelle für CPU-, I/O- oder NET-Kosten austauscht. Beispielsweise müssten die Proportionalitätsfaktoren w_1, w_2 mit Hilfe eines zusätzlichen Faktors neu abgestimmt werden, wenn nur das Kostenmodell zur Berechnung der CPU-Kosten ausgetauscht wird und alle anderen Kostenfunktionen gleich bleiben.

Der rasante Wandel der CPU-MIPS-Leistungen in den letzten Jahrzehnten und die daraus resultierenden Veränderungen für die Gewichtungsfaktoren werden in der Abbildung 2.4 durch eine logarithmische Achsendarstellung verdeutlicht.[4]

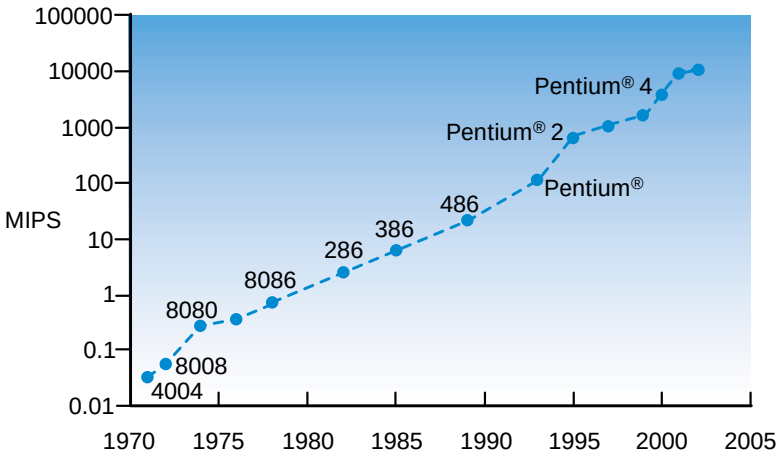


Abbildung 2.4: intel® CPU-MIPS-Leistung

Folgende Statistiken mit der anschließenden Rechnung verdeutlichen, wie einfach die Ermittlung der Proportionalitätsfaktoren w_1, w_2 ist:

$13.775\ \frac{\text{MInstr.}}{\text{s}}$	MIPS (Millionen CPU-Instruktionen pro Sekunde) (z. B. intel Pentium 4 EE 3.7 GHz (siehe Anhang A))
$297,2\ \frac{\text{MInstr.}}{\text{s}} = 4 \cdot 74,3\ \frac{\text{MInstr.}}{\text{s}}$	IOPS (Millionen I/O-Instruktionen pro Sekunde) (z. B. 4 Western Digital WF2000JB 200 GB im Raid 0 (siehe A.1))
$100 \cdot 10^9\ \frac{\text{Bits}}{\text{s}} = 125 \cdot 10^8\ \frac{\text{Bytes}}{\text{s}}$	NET-Transferrate (z. B. 100 MBit Fast Ethernet (IEEE 802.3u))
$2,133 \cdot 10^9\ \frac{\text{Bytes}}{\text{s}}$	Bus-Transferrate (z. B. PCI-X 266 (2.0) 64 Bit 266 MHz)

[4] `http://www.intel.com/network/csp/pdf/8538wp.pdf` (Für genauere Angaben siehe Anhang A.3.)

Somit ergeben sich die Gewichtungsfaktoren zu:

$$w_1 = \frac{297{,}3\ \frac{\text{MInstr.}}{\text{s}}}{13.775\ \frac{\text{MInstr.}}{\text{s}}} \approx 0,022$$

$$w_2 = \frac{2{,}133 \cdot 10^9\ \frac{\text{Bytes}}{\text{s}}}{125 \cdot 10^8\ \frac{\text{Bytes}}{\text{s}}} \approx 0.171$$

2.4 Darstellung von Mengenoperatoren

Die in jeder Datenbank implementierten Mengenoperatoren (Vereinigung, Differenz, Durchschnitt und Produkt) lassen sich leicht auf Verbundbildung zurückführen. Da alle Mengenoperatoren duplikateliminierend sind, müssen diese Kosten mit berücksichtigt werden. Somit leiten sich z. B. die Kosten für einen Vereinigungsoperator zu $\pi^{\mathsf{elim}}(R_1 \supset\!\!\bowtie\!\!\subset R_2)$ ab (siehe Tabelle 2.1).

Mengenoperation	Verbundoperation
$R_1 \cup R_2$	$R_1 \supset\!\!\bowtie\!\!\subset R_2$
$R_1 \cap R_2$	$R_1 \ltimes R_2$
$R_1 - R_2$	$R_1 \overline{\ltimes} R_2$
$R_1 \times R_2$	$R_1 \bowtie_{\mathsf{true}} R_2$

Tabelle 2.1: Darstellungen für Mengenoperatoren

2.5 Selektivitätsabschätzung von Prädikaten

Von fundamentaler Bedeutung für die Kostenberechnung ist die Frage, wie viele Tupel sich bei der Auswertung einer Bedingung qualifizieren, da erst dadurch eine Abschätzung der Größe von Zwischenergebnissen ermöglicht wird. Dieses führt zur Einführung des Begriffes Selektivität (Auswahlschärfe), also dem Anteil der sich qualifizierenden Tupel (siehe dazu [38], [57], [52] und [88]).

Definition 2.5.1 Für eine **Selektionsbedingung** φ auf einer Relation R sei die Selektivität $\mathsf{sel}(\varphi, R)$ definiert als das Verhältnis der Anzahl an Tupeln, die diese Bedingung erfüllen, zu der Gesamtanzahl an Tupeln der Relation:

$$\mathsf{sel}(\varphi, R) := \frac{|\sigma_\varphi(R)|}{|R|}$$

Für eine **Joinbedingung** φ der beiden Relationen R_1 und R_2 sei die Selektivität $\mathsf{sel}(\varphi, R_1 \times R_2)$ definiert als das Verhältnis der Tupel, die diese Joinbedingung erfüllen, zu den Tupeln des Kartesischen Produktes der beiden Relationen:

$$\mathsf{sel}(\varphi, R_1 \times R_2) := \frac{|R_1 \bowtie_\varphi R_2|}{|R_1 \times R_2|}$$

Somit liefert eine Selektion σ_φ mit einer Selektionsbedingung $\varphi \equiv A\Theta a$ und einer Selektivität $\mathsf{sel}(A\Theta a, R)$ voraussichtlich ein Ergebnis mit $\mathsf{sel}(A\Theta a, R) \cdot |R|$ Tupeln, wenn sie auf eine Relation R mit der Kardinalität $|R|$ angewandt wird.

Für viele Selektivitätsabschätzungen ist die Anzahl der verschiedenen Werte eines Attributes einer Relation eine wichtige Größe. Deshalb muss außer der Größe der Ergebnisrelation R in einer Operation auch die Anzahl der vorkommenden unterschiedlichen Attributwerte $\mathsf{n}(A, R)$ abgeschätzt werden (siehe dazu Abschnitt 2.5.2).

Unter der anfangs genannten Generalannahme, dass die in den Prädikaten $\varphi, \varphi_1, \varphi_2$ auftauchenden Attribute gleichverteilt und unabhängig sind, ergeben sich die Selektivitätsabschätzungen in Tabelle 2.2, wobei $a, b \in \mathsf{dom}(A)$ verschiedene Konstanten seien.

Um Selektivitäten von Bedingungen zwischen ganzen Termen z. B. der Form $(R_1.A \cdot 2 \geq R_1.B + 10)$ bestimmen zu können, müssten konkrete Ergebnisse der Argumentrelation vorliegen. Da man diese Informationen ohne einmaliges Ausführen nicht besitzt, gehen wir vom vereinfachten Fall aus und nehmen $\mathsf{sel}(\varphi, R) = 1$ an. Falls keine Statistiken zur Abschätzung der Selektivität vorliegen, können Standardwerte verwendet werden, die aus verschiedenen empirischen Tests ermittelt wurden.

Prädikat φ	Schätzung für $\mathsf{sel}(\varphi, R)$	Standardwert
$A = a$	$\frac{1}{\mathsf{n}(A,R)}$	$\frac{1}{10}$
$A > a, A \geq a$	$\frac{\mathsf{max}(A,R)-a}{\mathsf{max}(A,R)-\mathsf{min}(A,R)}$	$\frac{1}{3}$
$A < a, A \leq a$	$\frac{a-\mathsf{min}(A,R)}{\mathsf{max}(A,R)-\mathsf{min}(A,R)}$	$\frac{1}{3}$
$a < A \leq b$	$\frac{b-a}{\mathsf{max}(A,R)-\mathsf{min}(A,R)}$	$\frac{1}{4}$
$\leq, \geq$ analog...		
$A = B$	$\mathsf{min}\left(\frac{1}{\mathsf{n}(A,R)}, \frac{1}{\mathsf{n}(B,R)}\right)$	$\frac{1}{10}$
$A > B, A \geq B$	-	$\frac{1}{3}$
$A < B, A \leq B$	-	$\frac{1}{3}$
$\neg\varphi_1$	$1 - \mathsf{sel}(\varphi_1, R)$	-
$\varphi_1 \wedge \varphi_2$	$\mathsf{sel}(\varphi_1, R) \cdot \mathsf{sel}(\varphi_2, R)$	-
$\varphi_1 \vee \varphi_2$	$\mathsf{sel}(\varphi_1, R) + \mathsf{sel}(\varphi_2, R) - \mathsf{sel}(\varphi_1 \wedge \varphi_2, R)$	-

Tabelle 2.2: Selektivität von Prädikaten

2.5.1 Abschätzung von Kardinalitäten und Seitenanzahlen

Ein weiterer Bestandteil des Kostenmodells sind Abschätzungen von Kardinalitäten bei Zwischen- bzw. Ergebnisrelationen. Dabei muss man die Anzahl von Tupeln in der Ergebnisrelation, wie in Tabelle 2.3 angegeben, abschätzen.

Die belegten Speicherseiten können berechnet werden, indem die durchschnittliche Tupellänge $\mathsf{tlg}(R)$ (siehe Tabelle 2.6 auf Seite 21) mit der Anzahl von Tupeln $|R|$ multipliziert wird. Diesen Berechnungen liegt die Annahme zugrunde, dass die Speicherseiten stets vollständig mit Tupeln zu einer Relation gefüllt sind (was für Oracle 10g, aber z. B. nicht für IBM DB2 zutrifft).

Operation	|Operationsergebnis|
$\sigma_\varphi(R)$	$\mathsf{sel}(\varphi, R) \cdot \lvert R \rvert$
$\pi_{\overline{A}}(R)$	$\mathsf{min}\left(\lvert R \rvert, \prod_{A \in \overline{A}} \mathsf{n}(A, R)\right)$
$\Gamma_{\overline{G}\#\overline{F}}(R)$	$\mathsf{min}\left(\lvert R \rvert, \prod_{A \in \overline{G}} \mathsf{n}(A, R)\right)$
$R_1 \times R_2$	$\lvert R_1 \rvert \cdot \lvert R_2 \rvert$
$R_1 \bowtie_\varphi R_2$	$\lvert \sigma_\varphi(R_1 \times R_2) \rvert$
$R_1 \ltimes_\varphi R_2$	$\mathsf{min}\left(\lvert R_1 \rvert, \lvert \pi_{\mathsf{sch}(R_1)}(R_1 \bowtie_\varphi R_2) \rvert\right)$
$R_1 \overline{\ltimes}_\varphi R_2$	$\lvert R_1 \rvert - \lvert R_1 \ltimes_\varphi R_2 \rvert$
$R_1 \supset\!\bowtie_\varphi R_2$	$\lvert R_1 \bowtie_\varphi R_2 \rvert + \lvert R_1 \overline{\ltimes}_\varphi R_2 \rvert$
$R_1 \bowtie\!\subset_\varphi R_2$	$\lvert R_1 \bowtie_\varphi R_2 \rvert + \lvert R_2 \overline{\ltimes}_\varphi R_1 \rvert$
$R_1 \supset\!\bowtie\!\subset_\varphi R_2$	$\lvert R_1 \bowtie_\varphi R_2 \rvert + \lvert R_2 \overline{\ltimes}_\varphi R_1 \rvert + \lvert R_1 \overline{\ltimes}_\varphi R_2 \rvert$
$R_1 - R_2$	$\lvert R_1 \overline{\ltimes}_\varphi R_2 \rvert$
$R_1 \cap R_2$	$\lvert R_1 \ltimes_\varphi R_2 \rvert$
$R_1 \cup R_2$	$\lvert R_1 \rvert + \lvert R_2 \rvert - \lvert R_1 \cap R_2 \rvert$

Tabelle 2.3: Kardinalitäten

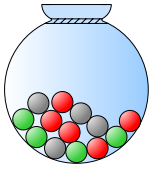
2.5.2 Schätzungen für Attributwertanzahlen

Attributwertanzahlen $\mathsf{n}(\mathsf{A},\mathsf{R})$ geben an, wie viele verschiedene Werte ein Attribut A in R annimmt. Da sie nicht in allen Fällen vorliegen, müssen sie im Laufe der Kostenberechnung abgeschätzt werden.

Bei der Selektion sind mehrere Fälle zu unterscheiden. Dabei ist es wichtig, ob das betrachtete Attribut A in der Selektionsbedingung φ vorkommt oder nicht:

(a) Sei $\mathsf{A} \notin \mathsf{attr}(\varphi), \mathsf{r} = |\sigma_\varphi(\mathsf{R})|, \mathsf{m} = \mathsf{n}(\mathsf{A},\mathsf{R})$, dann kann man nach [11] bei Gleichverteilung und Unabhängigkeit der Attribute die Abschätzung der Wertanzahlen auf folgende kombinatorische Aufgabe zurückführen:

Gegeben sei eine Urne mit n *Kugeln in* m *verschiedenen Farben. Wie viele verschiedenfarbige Kugeln erhält man, wenn man* r *Kugeln zufällig ohne Zurücklegen aus der Urne zieht?*



Die dafür entwickelte Abschätzung ist in Abbildung 2.5 angegeben.

$$\mathsf{n}(\mathsf{A},\sigma_\varphi(\mathsf{R})) = \begin{cases} \mathsf{r} & \text{falls} \quad \mathsf{r} < \frac{\mathsf{m}}{2} \\ \frac{\mathsf{r}+\mathsf{m}}{3} & \text{falls} \quad \frac{\mathsf{m}}{2} \le \mathsf{r} < 2\cdot\mathsf{m} \\ \mathsf{m} & \text{falls} \quad \mathsf{r} \ge 2\cdot\mathsf{m} \end{cases}$$

Abbildung 2.5: Kombinatorische Formel (†) zur Abschätzung von Attributwertanzahlen

(b) Wenn $\mathsf{A} \in \mathsf{attr}(\varphi)$ ist, werden Berechnungen aus der Tabelle 2.4 für die Abschätzung der Attributwertanzahlen bei der Selektion verwendet.

Wenn man Tupel aus R nach bestimmten A-Werten selektiert, ergibt sich die Anzahl von verschiedenen Attributwertanzahlen natürlich aus der Anzahl der vorgegebenen A-Werte.

Wird die Selektion durch eine Joinbedingung gegeben, dann vergrößert sich die Anzahl von A-Werten nicht:

$$\mathsf{n}\left(\mathsf{A},\sigma_{\mathsf{A}\Theta\mathsf{B}}(\mathsf{R})\right) \le \mathsf{n}(\mathsf{A},\mathsf{R})\,.$$

Bedingung φ	$\mathsf{n}(\mathsf{A},\sigma_\varphi(\mathsf{R}))$
$\mathsf{A}\Theta\mathsf{a}$	$\mathsf{sel}(\varphi,\mathsf{R})\cdot\mathsf{n}(\mathsf{A},\mathsf{R})$
$\mathsf{A}\Theta\mathsf{B}$ mit $\Theta \notin \{=\}$	$\mathsf{n}(\mathsf{A},\mathsf{R})$
$\mathsf{A}=\mathsf{B}$	$\min\left(\mathsf{n}(\mathsf{A},\mathsf{R}),\mathsf{n}(\mathsf{B},\mathsf{R})\right)$
$\varphi_1 \wedge \varphi_2$	$\mathsf{n}\left(\mathsf{A},\sigma_{\varphi_1}(\sigma_{\varphi_2}(\mathsf{R}))\right)$
$\varphi_2 \vee \varphi_2$	$\mathsf{n}(\mathsf{A},\sigma_{\varphi_1}(\mathsf{R}) \cup \sigma_{\varphi_2}(\mathsf{R}))$

Tabelle 2.4: Schätzungen für Attributwertanzahlen einer Selektion

Bei Equijoins ist die Anzahl von A-Werten nicht größer als die Anzahl von B-Werten: $\mathsf{n}\left(\mathsf{A},\sigma_{\mathsf{A}=\mathsf{B}}(\mathsf{R})\right) \le \mathsf{n}(\mathsf{B},\mathsf{R})$. Beinhaltet die Selektionsbedingung eine Konjunktion oder Disjunktion, so werden die im Kapitel 3.4 vorgestellten Regeln angewandt, um die Berechnungen zu vereinfachen. Negationen werden mit Hilfe von DeMorgans Gesetz vor der Abschätzung nach „innen gezogen“, so dass $\neg\mathsf{A} > \mathsf{a}$ z. B. zu $\mathsf{A} \le \mathsf{a}$ werden würde.

Die Abschätzungen für alle anderen Operatoren, wie z. B. Produkt oder Outer-Join, leiten sich aus den Überlegungen für die Selektion oder dem Join ab. Bei einigen Operatoren können genauere Abschätzungen angegeben werden. Diese sind in der Tabelle 2.5 aufgeführt und nachfolgend beschrieben. In der Tabelle steht A für ein vorhandenes Attribut aus R_1 und B aus R_2.

Operation	Fall	n(A,Operation)
$\pi_{\overline{A}}(R)$	$A \in \overline{A}$	$n(A, R)$
$\Gamma_{\overline{G}\#\overline{F}}(R)$	$A \in \overline{G}$	$n(A, R)$
	$A = \mathsf{count}$	1
	$A \in \{\mathsf{sum}, \mathsf{avg}, \mathsf{min}, \mathsf{max}\}$	$\lvert\Gamma_{\overline{G}\#\overline{F}}(R)\rvert$
$R_1\{\times, \rtimes, \ltimes_\varphi\}R_2$	$A \in (\mathsf{sch}(R_1) - \mathsf{sch}(R_2))$	$n(A, R_1)$
	$A \in (\mathsf{sch}(R_2) - \mathsf{sch}(R_1))$	$n(A, R_2)$
	$A \in (\mathsf{sch}(R_1) \cap \mathsf{sch}(R_2))$	$\min(n(A, R_1), n(A, R_2))$
$R_1 \rtimes\ltimes_\varphi R_2$		$n(A, \sigma_\varphi(R_1 \times R_2))$
$R_1 \ltimes_\varphi R_2$	$A \in \mathsf{attr}(\varphi)$	$n(A, \pi_{\mathsf{sch}(R_1)}(R_1 \rtimes\ltimes_\varphi R_2))$
	$A \notin \mathsf{attr}(\varphi)$	Kombinatorische Formel (†) mit $r = \lvert R_1 \ltimes_\varphi R_2\rvert, m = \max(n(A, R_1), n(B, R_2))$
$R_1 \overline{\ltimes}_\varphi R_2$	$A \in \mathsf{attr}(\varphi) \wedge \mathsf{op}(\varphi) \in \{=\}$	$n(A, R_1) - n(A, R_1 \ltimes_\varphi R_2)$
	$A \in \mathsf{attr}(\varphi) \wedge \mathsf{op}(\varphi)$ sonst	$n(A, R_1)$
	$A \notin \mathsf{attr}(\varphi)$	Kombinatorische Formel (†) mit $r = \lvert R_1 \overline{\ltimes}_\varphi R_2\rvert, m = \max(n(A, R_1), n(B, R_2))$
$R_1 \rtimes_\varphi R_2$	$A \in \mathsf{sch}(R_1)$	$n(A, R_1)$
	$A \in \mathsf{sch}(R_2)$	$n(A, \sigma_\varphi(R_1 \times R_2))$
$R_1 \ltimes_\varphi R_2$	$A \in \mathsf{sch}(R_1)$	$n(A, \sigma_\varphi(R_1 \times R_2))$
	$A \in \mathsf{sch}(R_2)$	$n(A, R_2)$
$R_1 - R_2$	$\mathsf{sch}(R_1) = \mathsf{sch}(R_2)$	$n(A, R_1 \overline{\ltimes} R_2)$
$R_1 \cap R_2$	$\mathsf{sch}(R_1) = \mathsf{sch}(R_2)$	$n(A, R_1 \ltimes R_2)$
$R_1 \cup R_2$	$\mathsf{sch}(R_1) = \mathsf{sch}(R_2)$	$n(A, R_1) + n(A, R_2) - n(A, R_1 \cap R_2)$

Tabelle 2.5: Schätzungen für Attributwertanzahlen

Bei Projektionen und dem Kartesischen Produkt wird die Anzahl unterschiedlicher Attributwerte nicht verändert. Da (Anti-)Semijoins Tupel aus R_1 auswählen, die eine Bedingung erfüllen und dementsprechend einer Selektion ähneln, benutzen wir für den Fall $A \notin \mathsf{attr}(\varphi)$ ((†)) aus der Kombinatorischen Formel (siehe Abbildung 2.5) die Abschätzung mit $r = \lvert R_1 \ltimes_\varphi R_2\rvert$ und $m = \max(n(A, R_1), n(B, R_2))$. Bei der Aggregation können Attribute durch das Anwenden der Group-by-Funktionen hinzukommen. Da wir während der Optimierungsphase diese nicht vornehmen können, wird von einer Gleichverteilung der Attributwerte ausgegangen, was dazu führt, dass gleich große Gruppen entstehen. Das bedeutet unter der Annahme, dass null-Werte mitgezählt werden für $\mathsf{count}()$, dass $n(A, R) = 1$ ist. Für alle anderen Funktionen gehen wir davon aus, dass wir für jede neu entstandene Gruppe einen unterschiedlichen Wert für das neu generierte Attribut erhalten $n(A, R) = \lvert\Gamma_{\overline{G}\#\overline{F}}(R)\rvert$.

Bei einem Full-Outer-Join bleiben die Attributwertanzahlen erhalten, da auch die Tupel, die keinen Joinpartner finden, anschließend zur Tabelle hinzugefügt werden. Ähnlich verhält es sich mit dem Left-Outer-Join und dem Right-Outer-Join. Hier ändern sich allerdings die Werteanzahlen der Relation, bei der die Tupel ohne Joinpartner nicht betrachtet werden hin zu denen eines normalen Joins.

2.6 Abschätzung von Tupellängen

Nach dem Ausführen einer Operation muss zur Berechnung der neuen Seitengröße $\mathsf{pg}(R)$ einer Ergebnisrelation R auch die Tupellänge $\mathsf{tlg}(R)$ abgeschätzt werden, da $\mathsf{pg}(R) = |R| \cdot \mathsf{tlg}(R)$ ist.

Eine Verfeinerung der vereinfachenden Annahme von gleich großen Attributlängen wird durch Berücksichtigung des Wissens während der Optimierung über das Schema der Relationen und die Größe der einzelnen Tupel erreicht. Daraus resultierend kann die Größe der Relation präziser abgeschätzt und somit können die Kostenabschätzungen noch feiner gestaltet werden. In unserem Fall werden keine Durchschnittsbildungen für Tupellängen berechnet, sondern es wird mit der Annahme von gleich langen Tupeln pro Relation gearbeitet. Zusätzlich gehen wir auch von gleich großen Seiten aus und normieren die Tupellängen auf die systembedingte Seitengröße.

Beim Selektionsoperator ($\sigma_\varphi(R)$) verändert sich laut Definition die Attributmenge der Argumentrelation nicht, somit auch nicht die Tupellänge. Dasselbe gilt auch für die Vereinigung zweier Relationen ($R_1 \cup R_2$), die in der relationalen Algebra nur für Relationen mit gleichem Schema definiert ist. Auch bei der Differenz ($R_1 - R_2$) und dem daraus abgeleiteten Schnitt zweier Relationen ($R_1 \cap R_2$) wird aus demselben Grund an der Tupellänge der Argumentrelation nichts verändert.

Beim Semi- bzw. Antisemijoin wird nur die Anzahl von Tupeln aus der ersten Argumentrelation reduziert. Demzufolge bleibt auch hier die Tupellänge gleich.

Bei einer Projektion $\left(\pi_{\overline{A}}(R)\right)$ müssen die Attributlängen der resultierenden Ergebnisrelation addiert werden. Da beim Kartesischen Produkt ($R_1 \times R_2$) von unterschiedlichen Attributmengen ausgegangen wird, müssen auch hier die beiden Tupellängen addiert werden. Beim Join ($R_1 \bowtie_\varphi R_2$) ist die Berechnung aufgrund der Ableitung als Selektion auf einem Kreuzprodukt identisch. Bei einem Equijoin wissen wir zusätzlich, dass die Verbundattribute (VA) nur einmal in der Ergebnisrelation vorkommen. Somit verbessert sich die Abschätzung hin zu: $\mathsf{tlg}(R_1) + \mathsf{tlg}(R_2) - \Sigma_{A \in VA}\, \mathsf{tlg}(A, R_1)$.

Schließlich wird die Tupellänge der Ergebnisrelation einer Aggregation ($\Gamma_{\overline{G}\#\overline{F}}(R)$) dahingehend abgeschätzt, dass man von einer Projektion der Attribute aus $\overline{F}$ ausgeht und deren einzelne Tupellängen aufsummiert.

Operation	Tupellänge
$\mathsf{tlg}\,(\sigma_\varphi(R))$	$\mathsf{tlg}(R)$
$\mathsf{tlg}\left(\pi_{\overline{A}}(R)\right)$	$\Sigma_{A \in \overline{A}}\, \mathsf{tlg}(A, R)$
$\mathsf{tlg}\,(R_1 \times R_2)$	$\mathsf{tlg}(R_1) + \mathsf{tlg}(R_2)$
$\mathsf{tlg}\,(R_1 \cup R_2)$	$\mathsf{tlg}(R_1)$
$\mathsf{tlg}\,(R_1 \cap R_2)$	$\mathsf{tlg}(R_1)$
$\mathsf{tlg}\,(R_1 - R_2)$	$\mathsf{tlg}(R_1)$
$\mathsf{tlg}\,(R_1 \bowtie_\varphi R_2)$	$\mathsf{tlg}(R_1) + \mathsf{tlg}(R_2)$
$\mathsf{tlg}\,(R_1 \bowtie_\varphi R_2)$ mit $\mathsf{op}(\varphi) \in \{=\}$	$\mathsf{tlg}(R_1) + \mathsf{tlg}(R_2)$ $-\Sigma_{A \in VA}\, \mathsf{tlg}(A, R_1)$
$\mathsf{tlg}\,(R_1 \ltimes_\varphi R_2)$	$\mathsf{tlg}(R_1)$
$\mathsf{tlg}\,(R_1 \overline{\ltimes}_\varphi R_2)$	$\mathsf{tlg}(R_1)$
$\mathsf{tlg}\left(\Gamma_{\overline{G}\#\overline{F}}(R)\right)$	$\Sigma_{A \in \overline{F}}\, \mathsf{tlg}(A, R)$

Tabelle 2.6: Abschätzungen von Tupellängen

2.7 Kostenfunktionen

In dieser Arbeit wird zwischen I/O-Kosten, NET-Kosten und CPU-Kosten unterschieden. Dabei sind die Formeln für NET-Kosten nicht explizit angegeben. Für diese Kosten müssen neue Transportoperatoren abgeschätzt werden, für die man nur die Kardinalitäten der zu transportierenden Ergebnisrelationen berücksichtigen muss. Daraus folgt, dass bei dezentralen Systemen $\mathsf{cost}_{\mathsf{CPU,I/O}} = 0$ gesetzt wird, da dort nur die überproportional hohen Transportkosten betrachtet werden.

Die nachfolgenden I/O-Kostenfunktionen beziehen sich auf Seitenzugriffe und gehen davon aus, dass der Puffer eine Seite pro Relation aufnehmen kann. Somit können die gesamten Argumentrelationen nicht komplett in den Hauptspeicher geladen werden. Die CPU-Kosten belaufen sich aus den daraus proportional resultierenden Tupelzugriffen. Dagegen werden die dazugehörenden Indexe nur einmal eingelesen ($\mathsf{pg}(\mathsf{Index_A(R)})$). Es wird davon ausgegangen, dass nach einem einmaligen Einstieg in den Baum, alle benötigten Tupel sequenziell gelesen werden können. Aus diesem Grund werden die CPU-Kosten für den Zugriff über einen Index mit $|\mathsf{Index_A(R)}|_\mathsf{I}$ abgeschätzt.

2.7.1 Selektion

In der nachfolgenden Tabelle 2.7 werden vier Zugriffsvarianten für den Selektionsoperator abgeschätzt. Pro relationalem Scan $\sigma^{\mathsf{Rel}}(\mathsf{R})$ wird die komplette Relation R einmal tupelweise abgearbeitet. Liegt das zu selektierende Attribut in sortierter Reihenfolge vor, kann der Zugriff frühzeitig abgebrochen werden. Mit Hilfe eines Indexes in geclusterter oder ungeclusterter Form lassen sich je nach Existenz ebenfalls Kosten gegenüber dem kompletten Zugriff einsparen.

$\sigma_\varphi(\mathsf{R})$	I/O-Kosten	CPU-Kosten
$\sigma_\varphi^{\mathsf{Rel}}(\mathsf{R})$	$\mathsf{pg(R)}$	$\lvert\mathsf{R}\rvert$
$\sigma_\varphi^{\mathsf{Sort}}(\mathsf{R})$	$\mathsf{sel}(\varphi,\mathsf{R}) \cdot \mathsf{pg(R)}$	$\mathsf{sel}(\varphi,\mathsf{R}) \cdot \lvert\mathsf{R}\rvert$
$\sigma_\varphi^{\mathsf{Index}}(\mathsf{R})$ Index geclustert	$\mathsf{pg}(\mathsf{Index_A(R)})+$ $\mathsf{sel}(\varphi,\mathsf{R}) \cdot \mathsf{pg(R)}$	$\lvert\mathsf{Index_A(R)}\rvert_\mathsf{I}+$ $\mathsf{sel}(\varphi,\mathsf{R}) \cdot \lvert\mathsf{R}\rvert$
$\sigma_\varphi^{\mathsf{Index}}(\mathsf{R})$ Index nicht geclustert	$\mathsf{pg}(\mathsf{Index_A(R)})+$ $\mathsf{sel}(\varphi,\mathsf{R}) \cdot \lvert\mathsf{R}\rvert$	$\lvert\mathsf{Index_A(R)}\rvert_\mathsf{I}+$ $\mathsf{sel}(\varphi,\mathsf{R}) \cdot \lvert\mathsf{R}\rvert$

Tabelle 2.7: Selektionskosten

2.7.2 Projektion

Projektionskosten werden in dieser Arbeit in vier Varianten unterschieden. Bei einem relationalen Scan werden die nicht benötigten Attribute einer Relation R einfach abgeschnitten. Falls dabei eine Duplikateliminierung vorgenommen werden soll, entsteht noch der zusätzliche Aufwand einer zur Laufzeit vorgenommenen Sortierung. Falls die Relation nach dem zu projizierenden Attribut sortiert vorliegt, kann man diesen Mehraufwand aus der Kostenfunktion entfernen.

$\pi_{\overline{\mathsf{A}}}(\mathsf{R})$	I/O-Kosten	CPU-Kosten
$\pi_{\overline{\mathsf{A}}}^{\mathsf{Rel}}(\mathsf{R})$	$\mathsf{pg(R)}$	$\lvert\mathsf{R}\rvert$
$\pi_{\overline{\mathsf{A}}}^{\mathsf{Rel/elim}}(\mathsf{R})$	$\mathsf{pg(R)} + \lvert\mathsf{R}\rvert \cdot \frac{\mathsf{pg(R)}}{2}$	$\lvert\mathsf{R}\rvert + \lvert\mathsf{R}\rvert \cdot \frac{\lvert\mathsf{R}\rvert}{2}$
$\pi_{\overline{\mathsf{A}}}^{\mathsf{Sort[/elim]}}(\mathsf{R})$	$\mathsf{pg(R)}$	$\lvert\mathsf{R}\rvert$
$\pi_{\overline{\mathsf{A}}}^{\mathsf{Index}}(\mathsf{R})$	$\mathsf{pg}\left(\mathsf{Index}_{\overline{\mathsf{A}}}(\mathsf{R})\right)$	$\lvert\mathsf{Index}_{\overline{\mathsf{A}}}(\mathsf{R})\rvert_\mathsf{I}+$ $\lvert\pi_{\overline{\mathsf{A}}}(\mathsf{R})\rvert$

Tabelle 2.8: Projektionskosten

Eine indexbasierte Projektion vereinfacht das Entfernen zusätzlicher Attribute, indem das Ergebnis direkt aus dem Index gelesen werden kann. Dabei entstehen die I/O-Kosten durch das Einlesen des Indexes und die CPU-Kosten durch das Ausnutzen des Indexes, zuzüglich der Berechnungskosten für das Bestimmen der auftretenden Attributwerte, die schätzungsweise $|\pi_{\overline{A}}(R)|$ Tupelzugriffe erfordern (siehe Tabelle 2.8).

2.7.3 Join

Tabelle 2.9 gibt Auskunft über die verwendeten Abschätzungen für Kosten bei sieben verschiedenen Joinmethoden. Dabei werden beim Merge-Join ($\rtimes\!\!\ltimes_{\varphi}^{\mathsf{Sort,\,Sort}}$) und dem Hash-Join ($\rtimes\!\!\ltimes_{\varphi}^{\mathsf{Hash,\,Hash}}$) algorithmisch bedingt nur Gleichheitsverbundbedingungen zugelassen und abgeschätzt. Eine Kostenfunktion für den Index-Rel-Join ($\rtimes\!\!\ltimes_{\varphi}^{\mathsf{Index,\,Rel}}$) gleicht dem Rel-Index-Join ($\rtimes\!\!\ltimes_{\varphi}^{\mathsf{Rel,\,Index}}$) mit vertauschten Argumenten und wird daher nicht explizit angegeben.

Im Folgenden enthält die Menge $\overline{\mathsf{A}}$ diejenigen Joinattribute, die sich nur auf R_1, und die Menge $\overline{\mathsf{B}}$, die sich auf R_2 beziehen. Bei Equijoins verringert sich diese Menge auf das linksseitige Joinattribut A und das rechtsseitige B.

$\mathsf{R}_1 \rtimes\!\!\ltimes_{\varphi} \mathsf{R}_2$	I/O-Kosten	CPU-Kosten
$\mathsf{R}_1 \rtimes\!\!\ltimes_{\varphi}^{\mathsf{Rel,\,Rel}} \mathsf{R}_2$	$\mathsf{pg}(\mathsf{R}_1) + \lvert\mathsf{R}_1\rvert \cdot \mathsf{pg}(\mathsf{R}_2)$	$\lvert\mathsf{R}_1\rvert + \lvert\mathsf{R}_1\rvert \cdot \lvert\mathsf{R}_2\rvert$
$\mathsf{R}_1 \rtimes\!\!\ltimes_{\mathsf{R}_1.\mathsf{A}=\mathsf{R}_2.\mathsf{B}}^{\mathsf{Sort,\,Sort}} \mathsf{R}_2$	$\mathsf{pg}(\mathsf{R}_1) + \mathsf{pg}(\mathsf{R}_2)$	$\lvert\mathsf{R}_1\rvert + \lvert\mathsf{R}_2\rvert$
$\mathsf{R}_1 \rtimes\!\!\ltimes_{\varphi}^{\mathsf{Rel,\,Index}} \mathsf{R}_2$ Index nicht geclustert	$\mathsf{pg}(\mathsf{R}_1) + \mathsf{pg}(\mathsf{Index}_{\mathsf{B}}(\mathsf{R}_2))$ $+\lvert\mathsf{R}_1\rvert \cdot \lvert\sigma_{\wedge_{\mathsf{X}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{X}\Theta\mathsf{a}}(\mathsf{R}_2)\rvert$	$\lvert\mathsf{R}_1\rvert + \lvert\mathsf{R}_1\rvert \cdot (\lvert\mathsf{Index}_{\mathsf{B}}(\mathsf{R}_2)\rvert_{\mathsf{I}} +$ $\lvert\sigma_{\wedge_{\mathsf{X}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{X}\Theta\mathsf{a}}(\mathsf{R}_2)\rvert)$
$\mathsf{R}_1 \rtimes\!\!\ltimes_{\varphi}^{\mathsf{Rel,\,Index}} \mathsf{R}_2$ Index geclustert	$\mathsf{pg}(\mathsf{R}_1) + \mathsf{pg}(\mathsf{Index}_{\mathsf{B}}(\mathsf{R}_2)) +$ $+\lvert\mathsf{R}_1\rvert \cdot \mathsf{pg}\left(\sigma_{\wedge_{\mathsf{X}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{X}\Theta\mathsf{a}}(\mathsf{R}_2)\right)$	$\lvert\mathsf{R}_1\rvert + \lvert\mathsf{R}_1\rvert \cdot (\lvert\mathsf{Index}_{\mathsf{B}}(\mathsf{R}_2)\rvert_{\mathsf{I}} +$ $\lvert\sigma_{\wedge_{\mathsf{X}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{X}\Theta\mathsf{a}}(\mathsf{R}_2)\rvert)$
$\mathsf{R}_1 \rtimes\!\!\ltimes_{\varphi}^{\mathsf{Index,\,Index}} \mathsf{R}_2$ Index [nicht] geclustert	$\mathsf{pg}\left(\mathsf{Index}_{\mathsf{A}}(\mathsf{R}_1)\right)$ $+\mathsf{pg}\left(\mathsf{Index}_{\mathsf{B}}(\mathsf{R}_2)\right)$ $+\mathsf{min}\left(\lvert\mathsf{R}_1\rvert, \prod_{\mathsf{X}\in\overline{\mathsf{A}}} \mathsf{n}(\mathsf{X}, \mathsf{R}_1)\right)$ $\cdot(\lvert\sigma_{\wedge_{\mathsf{X}\in\overline{\mathsf{A}}}\mathsf{R}_1.\mathsf{X}=\mathsf{a}}(\mathsf{R}_1)\rvert$ $+\lvert\sigma_{\wedge_{\mathsf{Y}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{Y}\Theta\mathsf{a}}(\mathsf{R}_2)\rvert)$	$\mathsf{min}\left(\lvert\mathsf{R}_1\rvert, \prod_{\mathsf{X}\in\overline{\mathsf{A}}} \mathsf{n}(\mathsf{X}, \mathsf{R}_1)\right) \cdot$ $(\lvert\mathsf{Index}_{\mathsf{A}}(\mathsf{R}_1)\rvert_{\mathsf{I}} + \lvert\sigma_{\wedge_{\mathsf{Y}\in\overline{\mathsf{A}}}\mathsf{R}_1.\mathsf{Y}=\mathsf{a}}(\mathsf{R}_1)\rvert$ $+\lvert\mathsf{Index}_{\mathsf{B}}(\mathsf{R}_2)\rvert_{\mathsf{I}} + \lvert\sigma_{\wedge_{\mathsf{X}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{X}\Theta\mathsf{a}}(\mathsf{R}_2)\rvert)$
$\mathsf{R}_1 \rtimes\!\!\ltimes_{\mathsf{R}_1.\mathsf{A}=\mathsf{R}_2.\mathsf{B}}^{\mathsf{Hash,\,Hash}} \mathsf{R}_2$	$\mathsf{pg}(\mathsf{R}_1) + \mathsf{pg}(\mathsf{R}_2) +$ $\lvert\mathsf{R}_1\rvert \cdot \frac{\mathsf{pg}(\mathsf{R}_2)}{\mathsf{n}(\mathsf{B},\mathsf{R}_2)}$	$\lvert\mathsf{R}_1\rvert + \lvert\mathsf{R}_2\rvert + \mathsf{n}(\mathsf{A}, \mathsf{R}_1) + \mathsf{n}(\mathsf{B}, \mathsf{R}_2) +$ $\lvert\mathsf{R}_1\rvert \cdot \frac{\lvert\mathsf{R}_2\rvert}{\mathsf{n}(\mathsf{B},\mathsf{R}_2)}$

Tabelle 2.9: Kosten von Joins

Der Nested-Loop-Join ($\rtimes\!\!\ltimes_{\varphi}^{\mathsf{Rel,\,Rel}}$) wird algorithmisch durch zwei ineinander geschachtelte Schleifen berechnet, so dass anfangs die erste Relation R_1 eingelesen werden muss, um danach die zweite Relation R_2 so oft zu durchlaufen, wie R_1 Tupel besitzt. Hier ergeben sich also Laufzeitunterschiede, je nachdem, ob $\mathsf{R}_1 \bowtie_{\varphi} \mathsf{R}_2$ oder $\mathsf{R}_2 \bowtie_{\varphi} \mathsf{R}_1$ berechnet wird. Beim Merge-Join

($\bowtie_{\varphi}^{\mathsf{Sort},\,\mathsf{Sort}}$) ist es dagegen egal, welche Relation die kleinere Eingaberelation ist, da beide nur einmal durchlaufen werden.

Für einen Rel-Index-Join ($\bowtie_{\varphi}^{\mathsf{Rel},\,\mathsf{Index}}$) müssen die erste Relation und der Index einmal komplett gelesen und für jedes Tupel aus R_1 ein Indexscan durchgeführt werden. Um abschätzen zu können, wie viele Tupel zur gefundenen TID-Folge gelesen werden müssen, ist es nötig, die Joinbedingung zu zerlegen. Die Joinattribute, die sich auf R_2 beziehen, werden in einer Hilfsbedingung mit einem Attributwert anstatt mit einem Attribut aus R_1 verbunden, z. B. wird $\mathsf{R}_1.\mathsf{A} < \mathsf{R}_2.\mathsf{B}$ zu $\mathsf{R}_2.\mathsf{B} >$ 'a'. Die Anzahl der selektierten Tupel wird mit $|\sigma_{\wedge_{\mathsf{X}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{X}\Theta\mathsf{a}}(\mathsf{R}_2)|$ abgeschätzt.

Bei einem Index-Index-Join ($\bowtie_{\varphi}^{\mathsf{Index},\,\mathsf{Index}}$) werden beide Indexe jeweils nur einmal eingelesen. Daraufhin wird für jeden Joinattributwert bzw. für jede -wertkombination aus R_1 ein Indexscan auf beiden Relationen durchgeführt. In der ersten Relation wird nach dem konkreten Wert gesucht, in der zweiten nach Tupeln, die die Bedingung mit dem Attributwert erfüllen. Dabei werden zur Abschätzung der zu lesenden Tupel auch in diesem Fall Hilfsbedingungen erzeugt.

Abschließend ergeben sich die Kosten eines Hash-Joins ($\bowtie_{\varphi}^{\mathsf{Hash},\,\mathsf{Hash}}$) durch das einmalige Einlesen der ersten und zweiten Argumentrelation $\mathsf{pg}(\mathsf{R}_1) + \mathsf{pg}(\mathsf{R}_2)$ (partitioning phase, building phase)[5], addiert mit den Kosten für das Joinen der einzelnen Buckets (join phase, probing phase):

$$\frac{|\mathsf{R}_1|}{\mathsf{n}(\mathsf{A},\mathsf{R}_1)} \cdot \frac{\mathsf{pg}(\mathsf{R}_2)}{\mathsf{n}(\mathsf{B},\mathsf{R}_2)} \cdot \mathsf{n}(\mathsf{A},\mathsf{R}_1) = |\mathsf{R}_1| \cdot \frac{\mathsf{pg}(\mathsf{R}_2)}{\mathsf{n}(\mathsf{B},\mathsf{R}_2)}$$

Die CPU-Kosten addieren sich durch das jeweilige Einlesen der ersten und zweiten Argumentrelation $|\mathsf{R}_1| + |\mathsf{R}_2|$ (partitioning phase, building phase), plus dem Verbinden der Buckets mit $\mathsf{n}(\mathsf{A},\mathsf{R}_1) + \mathsf{n}(\mathsf{B},\mathsf{R}_2)$ (probing phase, matching phase) und der abschließenden Summation der Kosten für das Joinen der Tupel in den Buckets $|\mathsf{R}_1| \cdot \frac{|\mathsf{R}_2|}{\mathsf{n}(\mathsf{B},\mathsf{R}_2)}$ (join phase).

Alle vorgestellten Kostenfunktionen gelten ebenfalls für entsprechende Outer-Joinoperatoren.

2.7.4 Antisemi-/Semijoin

Gegenüber einem Nested-Loop-Join ($\bowtie_{\varphi}^{\mathsf{Rel},\,\mathsf{Rel}}$) ist der Nested-Loop-Semijoin ($\ltimes_{\varphi}^{\mathsf{Rel},\,\mathsf{Rel}}$) im Vorteil, da der innere Schleifendurchlauf abgebrochen werden kann, wenn ein die Verbundbedingung erfüllendes Tupel gefunden worden ist. Somit wird für die Kostenabschätzung wichtig, wie viele Tupel durchschnittlich bei jedem inneren Schleifendurchlauf gelesen werden.

Es ergibt sich der Anteil der R_2-Joinpartner wie in Tabelle 2.10 als $\mathsf{sel}(\wedge_{\mathsf{X}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{X}\Theta\mathsf{a},\mathsf{R}_2)$. Unter der Annahme, dass die Attributwerte gleichverteilt und nicht in sortierter Reihenfolge vorliegen, werden für qualifizierende Tupel bei einem inneren Schleifendurchlauf durchschnittlich $\frac{1}{\mathsf{sel}(\wedge_{\mathsf{X}\in\overline{\mathsf{B}}}\mathsf{R}_2.\mathsf{X}\Theta\mathsf{a},\mathsf{R}_2)}$ viele Tupel gelesen. Falls kein Joinpartner existiert, muss die gesamte zweite Argumentrelation durchlaufen werden, so dass wir $|\mathsf{R}_2|$ Tupelzugriffe bekommen, welche mit $\frac{\mathsf{pg}(\mathsf{R}_2)}{|\mathsf{R}_2|}$ multipliziert Kosten von $\mathsf{pg}(\mathsf{R}_2)$ verursacht.

Ein Index-Rel-Semijoin ($\bowtie_{\varphi}^{\mathsf{Index},\,\mathsf{Rel}}$) lohnt sich nur bei besonders selektiven Bedingungen, da dort nach dem Einlesen des Indexes, nur noch der Rest des sich qualifizierenden Tupels gele-

[5] Passt die Hash-Tabelle für R_1, wie in unserem Fall, nicht in den Hauptspeicher, dann werden für die I/O-Kosten 2 und für die CPU-Kosten 3 Phasen unterschieden. Bei der oben durchgeführten Abschätzung gehen wir von der Annahme aus, dass die Anzahl der Hash-Buckets gleich der Anzahl von verschiedenen Attributwerten ist. Als Hash-Implementierung wurde die in [78] algorithmisch vorgestellte Grace Hash Join-Variante gewählt. Darüber hinaus finden sich dort alle Implementierungsalgorithmen für die in dieser Arbeit verwendeten Joinoperatoren.

sen werden muss; im Gegensatz zur Nested-Loop-Variante, bei der immer die komplette erste Argumentrelation verarbeitet werden muss.

Bei einem Hash-Semijoin ($R_1 \ltimes_{\varphi}^{\mathsf{Hash,\,Hash}} R_2$) wird nach dem Einlesen der Argumentrelationen nur noch die Existenz der gehashten Buckets überprüft (die „join phase“ entfällt), da beim Semijoin nur Tupel von R_1 in der Ergebnisrelation vorkommen. Somit vereinfachen sich die Kosten hin zu denen eines Merge-Joins der Buckets.

Die Kostenfunktionen ergeben sich analog zu den vorherigen Join-Formeln für den Semi- bzw. Antisemijoin (siehe Tabelle 2.10).

$R_1 \ltimes_{\varphi} R_2$	I/O-Kosten	CPU-Kosten
$R_1 \ltimes_{\varphi}^{\mathsf{Rel,\,Rel}} R_2$	$\mathsf{pg}(R_1) + \lvert R_1 \overline{\ltimes}_{\varphi} R_2\rvert \cdot \mathsf{pg}(R_2) +$ $\lvert R_1 \ltimes_{\varphi} R_2\rvert \cdot \frac{1}{\mathsf{sel}(\wedge_{X \in \overline{B}} R_2.X \Theta a, R_2)} \cdot \frac{\mathsf{pg}(R_2)}{\lvert R_2\rvert}$	$\lvert R_1\rvert + \lvert R_1 \overline{\ltimes}_{\varphi} R_2\rvert \cdot \lvert R_2\rvert +$ $\lvert R_1 \ltimes_{\varphi} R_2\rvert \cdot \frac{1}{\mathsf{sel}(\wedge_{X \in \overline{B}} R_2.X \Theta a, R_2)}$
$R_1 \ltimes_{\varphi}^{\mathsf{Sort,\,Sort}} R_2$	$\mathsf{pg}(R_1) + \mathsf{pg}(R_2)$	$\lvert R_1\rvert + \lvert R_2\rvert$
$R_1 \ltimes_{\varphi}^{\mathsf{Index,\,Rel}} R_2$ Index [nicht] geclustert	$\mathsf{pg}\left(\mathsf{Index}_A(R_1)\right) + \mathsf{pg}(R_1 \ltimes_{\varphi} R_2)$ $\lvert R_1 \overline{\ltimes}_{\varphi} R_2\rvert \cdot \mathsf{pg}(R_2) +$ $\lvert R_1 \ltimes_{\varphi} R_2\rvert \cdot \frac{1}{\mathsf{sel}(\wedge_{X \in \overline{B}} R_2.X \Theta a, R_2)} \cdot \frac{\mathsf{pg}(R_2)}{\lvert R_2\rvert}$	$\lvert R_1\rvert \cdot \lvert \mathsf{Index}_A(R_1)\rvert_I + \lvert R_1 \ltimes_{\varphi} R_2\rvert +$ $\lvert R_1 \overline{\ltimes}_{\varphi} R_2\rvert \cdot \lvert R_2\rvert +$ $\lvert R_1 \ltimes_{\varphi} R_2\rvert \cdot \frac{1}{\mathsf{sel}(\wedge_{X \in \overline{B}} R_2.X \Theta a, R_2)}$
$R_1 \ltimes_{\varphi}^{\mathsf{Rel,\,Index}} R_2$ Index [nicht] geclustert	$\mathsf{pg}(R_1) + \mathsf{pg}\left(\mathsf{Index}_B(R_2)\right)$	$\lvert R_1\rvert + \lvert R_1\rvert \cdot \lvert \mathsf{Index}_B(R_2)\rvert_I$
$R_1 \ltimes_{\varphi}^{\mathsf{Index,\,Index}} R_2$ Index [nicht] geclustert	$\mathsf{pg}\left(\mathsf{Index}_A(R_1)\right) + \mathsf{pg}\left(\mathsf{Index}_B(R_2)\right)$ $+\min\left(\lvert R_1\rvert, \prod_{X \in \overline{A}} \mathsf{n}(X, R_1)\right)$ $\cdot \lvert \sigma_{\wedge_{X \in \overline{A}} R_1.X = a}(R_1)\rvert$	$\min\left(\lvert R_1\rvert, \prod_{X \in \overline{A}} \mathsf{n}(X, R_1)\right) \cdot$ $(\lvert \mathsf{Index}_A(R_1)\rvert_I$ $+\lvert \mathsf{Index}_B(R_2)\rvert_I$ $+\lvert \sigma_{\wedge_{X \in \overline{B}} R_2.X \Theta a}(R_2)\rvert)$
$R_1 \ltimes_{R_1.A = R_2.B}^{\mathsf{Hash,\,Hash}} R_2$	$\mathsf{pg}(R_1) + \mathsf{pg}(R_2) +$ $\mathsf{n}(A, R_1) + \mathsf{n}(B, R_2)$	$\lvert R_1\rvert + \lvert R_2\rvert +$ $\mathsf{n}(A, R_1) + \mathsf{n}(B, R_2)$

Tabelle 2.10: Kosten eines Antisemi- bzw. Semijoins

2.7.5 Aggregation mit und ohne Gruppierung

Aggregationen ($\Gamma_{\overline{G} \# \overline{F}}$) lassen sich in zwei Typen einteilen: Ist der Teil der Bedingung vor dem Doppelkreuz leer, handelt es sich um eine Aggregation ohne Gruppierung. Bei diesem Typ wird ein einzelnes Tupel für die angewendeten Group-by-Funktionen zurückgeliefert.

$\Gamma_{\#\overline{F}}(R)$	I/O-Kosten	CPU-Kosten
$\Gamma_{\#\overline{F}}^{\mathsf{Rel}}$	$\mathsf{pg}(R)$	$\lvert R\rvert$
$\Gamma_{\#\overline{F}}^{\mathsf{Index}}$	$\mathsf{pg}(\mathsf{Index}_{\overline{F}}(R))$	$\lvert \mathsf{Index}_{\overline{F}}(R)\rvert_I + \lvert \mathsf{Index}_{\overline{F}}(R)\rvert_c$

Tabelle 2.11: Kosten einer Aggregation ohne Gruppierung

Eine Aggregation mit Gruppierung ähnelt einer Projektion mit Duplikateliminierung. Wird ein Relationenscan verwendet, muss für jeden neuen Wert geprüft werden, ob entsprechende Attri-

butwerte bereits gelesen wurden. Dabei können die Aggregierungsfunktionen berechnet werden, ohne weitere Kosten zu erzeugen. Im Mittel beträgt der bereits betrachtete Teil der Relation etwa die Hälfte der Gesamtrelation. Die Kosten für beide Typen von Aggregationen sind in den Tabellen 2.11 und 2.12 angegeben.

Da die Aggregierungsfunktionen parallel berechnet werden können, wird grundsätzlich nur ein Relationenscan verwendet, außer es existiert ein nutzbarer Index auf den Attributen innerhalb der Funktionen. In solch einem Fall können die benötigten Daten direkt aus ihm gelesen werden, ohne dass Zugriffe auf die Relation nötig sind.

Falls in der Relation die einzelnen Attribute aus $\overline{G}$ sortiert sind, oder eine Sortierung über alle Attribute aus $\overline{G}$ existiert, kann ein sequentiell arbeitender Merge-Algorithmus zur Gruppierung genutzt werden. Es ist verhältnismäßig selten, dass in einer Basisrelation mehr als ein Attribut sortiert ist. Nach der Anwendung von Verbundoperatoren kann dies aber durchaus der Fall sein.

$\Gamma_{\overline{G}\#\overline{F}}(R)$	I/O-Kosten	CPU-Kosten
$\Gamma^{Rel}_{\overline{G}\#\overline{F}}$	$pg(R) + \lvert R\rvert \cdot \frac{pg(R)}{2}$	$\lvert R\rvert + \lvert R\rvert \cdot \frac{\lvert R\rvert}{2}$
$\Gamma^{Sort}_{\overline{G}\#\overline{F}}$	$pg(R)$	$\lvert R\rvert$
$\Gamma^{Index}_{\overline{G}\#\overline{F}}$	$pg(Index_{\overline{G}\cup\overline{F}}(R))$	$\lvert Index_{\overline{G}\cup\overline{F}}(R)\rvert_I +$ $\lvert Index_{\overline{G}\cup\overline{F}}(R)\rvert_c + \lvert\Gamma_{\overline{G}\#\overline{F}}(R)\rvert$

Tabelle 2.12: Kosten einer Aggregation mit Gruppierung

Ist ein kombinierter Index auf den Attributen, nach denen gruppiert wird, und auf den Attributen innerhalb der Gruppierungsfunktionen vorhanden, ergeben sich die angefragten Daten unmittelbar aus diesem Index.

2.7.6 Erzeugung von Indexen und Sortierungen

Um festzustellen, ob sich die zwischenzeitliche Sortierung oder Indexbildung auf einer Relation nach dem Attribut A lohnt, benötigt man Kostenformeln, um das einmalige Erzeugen ins Verhältnis mit der mehrmaligen Nutzung einer Sortierung bzw. eines Indexes zu setzen.

Diese Kosten sind abhängig von dem zugrundeliegenden Sortierungsalgorithmus. In unserem Fall gehen wir von einer Laufzeit von $O(n \cdot \log(n))$ aus, mit der zusätzlichen Annahme, dass alle zu sortierenden Tupel der Laufzeit entsprechend mehrmals eingelesen werden müssen.[6]

	I/O-Kosten	CPU-Kosten
$SORT_A(R)$	$pg(R) \cdot \log_2(pg(R))$	$\lvert R\rvert \cdot \log_2(\lvert R\rvert)$
$INDEX_A(R)$	$pg(R) \cdot \log_2(pg(R))$	$\lvert R\rvert \cdot \log_2(\lvert R\rvert)$

Tabelle 2.13: Kosten der Erzeugung von Indexen und Sortierungen

2.8 Pipelining

Eine Möglichkeit, das Speichern von Zwischenrelationen zu vermeiden, stellt das so genannte Pipelining dar. Die Idee dabei ist, das Ergebnis einer Operation, wenn möglich ohne Zwischenspeicherung, direkt an die nächste Operation zu übergeben. Dadurch entstehen keine lesenden

[6] In diesem Fall wird von einer asymptotischen Aufwandsanalyse in Höhe von $O(n \cdot \log(n))$ ausgegangen. Gute Sortierverfahren, z. B. das Sortieren durch Mischen (MergeSort), erreichen diese Werte. Dabei müssen jedoch – wegen der notwendigen Zwischenspeicherung von partiell sortierten Zwischenergebnissen – die Kosten für schreibende Zugriffe gleich den Kosten für lesende Zugriffe angesetzt werden (siehe [52]).

Kosten bei der „darüberliegenden“ Operation. Somit werden die $\mathsf{cost}_{\mathsf{I/O}}$ des folgenden Operators auf 0 gesetzt.

Pipelining ist aber nicht immer durchführbar. Einige Implementierungsarten von Operatoren, z. B. der Nested-Loop-Join, benötigen das mehrfache Einlesen der Argumentrelation. Diese Operatoren „blockieren“ Pipelining, da in diesem Fall die Argumentrelation vollständig materialisiert werden muss. Werden Indexe genutzt (geclusterte sowie nicht-geclusterte), so ist prinzipiell kein Pipelining möglich, da sie erst die gesamte Eingabemenge materialisieren, bevor sie einzelne Tupel an den nächsten Operator weitergeben. Des Weiteren hat Pipelining keinen Einfluss auf die CPU-Kosten eines Operators, da die weitergereichten Tupel genau so verarbeitet werden müssen, wie neu eingelesene.

In [38] werden Pipeliningmuster definiert, bei denen kombinierte, aufeinanderfolgende Operatoren ein Pipelining ermöglichen. Die Implementierungsart der zuerst ausgeführten Operation ist nicht relevant, da sie nur eine Tupelfolge liefern muss, die als Eingabe für die nachfolgende Operation dient.

Daher vereinfachen wir die Angabe der Pipeliningmuster, indem wir nur noch den ersten Operator in der Bearbeitungsreihenfolge angeben. Beispielsweise steht für $\mathsf{R}_1 \bowtie_\varphi \mathsf{R}_2$ für die Relation R_1 in der Tabelle 2.15 „ja“, wenn das Pipelining der ersten Argumentrelation erlaubt ist. Grafisch wird ein Pipelining im Anfragebaum bzw. Ausführungsplan durch Verdoppelung der Kanten zwischen den Operatoren dargestellt (siehe Abbildung 2.6).

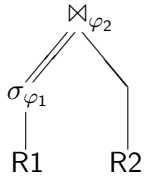


Abbildung 2.6: Beispiel für einen Anfragebaum mit Pipelining

Die nachfolgenden Tabellen liefern eine Antwort darauf, welche Argumentrelationen sich durch Pipelining übergeben lassen.

- Selektion und Projektion:

  Bei $\sigma_\varphi^{\mathsf{Rel}}$ und $\sigma_\varphi^{\mathsf{Sort}}(\mathsf{R})$ kann jedes einzeln übergebene Tupel der Eingabe direkt auf die Bedingung φ hin überprüft ausgegeben werden. Bei einem Indexzugriff ist dies hingegen nicht möglich, denn dafür müsste die gesamte Eingaberelation bereitstehen.

$\sigma_\varphi(\mathsf{R})$	Pipelining von R	$\pi_{\overline{\mathsf{A}}}(\mathsf{R})$	Pipelining von R
$\sigma_\varphi^{\mathsf{Rel}}(\mathsf{R})$	ja	$\pi_{\overline{\mathsf{A}}}^{\mathsf{Rel/elim}}(\mathsf{R})$	nein
$\sigma_\varphi^{\mathsf{Sort}}(\mathsf{R})$	ja	$\pi_{\overline{\mathsf{A}}}^{\mathsf{Sort}}(\mathsf{R})$	ja
$\sigma_\varphi^{\mathsf{Index}}(\mathsf{R})$	nein	$\pi_{\overline{\mathsf{A}}}^{\mathsf{Index}}(\mathsf{R})$	nein

Tabelle 2.14: Pipelining bei Projektionen und Selektionen

  Für eine Projektion $\pi_{\overline{\mathsf{A}}}^{\mathsf{Rel/elim}}(\mathsf{R})$ muss in jedem Fall die Argumentrelation mehrfach durchlaufen werden, um Duplikate entfernen zu können. Wenn die Relation nach dem zu projizierenden Attribut sortiert vorliegt ($\pi_{\overline{\mathsf{A}}}^{\mathsf{Sort}}(\mathsf{R})$), kann Pipelining durchgeführt werden. Wieder ist diese Möglichkeit beim Zugriff über einen Index nicht gegeben (siehe Tabelle 2.14).

- Join:

  Der Nested-Loop-Join ($\mathsf{R}_1 \bowtie_\varphi^{\mathsf{Rel,\,Rel}} \mathsf{R}_2$) greift auf die erste Argumentrelation zu und durchläuft für jedes Tupel aus R_1 einmal die zweite Relation R_2. Dabei können die Tupel von R_1 der Reihe nach verwendet werden. Die zweite Relation muss aber mehrmals gelesen werden, welches Pipelining verhindert. Die gleiche Argumentation gilt für den Rel-Index-Join ($\mathsf{R}_1 \bowtie_\varphi^{\mathsf{Rel,\,Index}} \mathsf{R}_2$).

Da beim Hash-Join ($R_1 \bowtie_\varphi^{Hash,\,Hash} R_2$) generell beide Eingaben gehasht werden, um danach einen Nested-Loop-Join auf den relevanten Buckets durchzuführen, wird auch hier dieselbe Argumentation angewandt. Es sei hier jedoch erwähnt, dass es auch Implementierungsarten gibt, wie z. B. den Pipeline-Hash-Join, die das Weiterreichen von Argumentrelationen explizit ermöglichen. Da wir uns nur auf die Standard-Hash-Join-Operatoren beziehen, wird diese spezielle Implementierung nicht mit betrachtet.

Beim Merge-Join Operator ($R_1 \bowtie_\varphi^{Sort,\,Sort} R_2$) werden beide Relationen nur einmal durchlaufen. Somit können auch beide Eingaberelationen R_1 und R_2 gepipelined werden. Auch in diesem Fall wird der einfache Operator betrachtet, der keine der beiden Argumentrelationen vorher sortiert. Dies ist auch an den vorherigen Kostenformeln in der Tabelle 2.9 erkennbar. Grundsätzlich werden beim Merge-Join nur Gleichheitsprädikate unterstützt ($\Theta_x \in \{=\}$).

$R_1\{\{\sqsupset\|\bowtie\|\sqsubset\}\,\|\,\ltimes\}_\varphi\,R_2$	Pipelining von R_1	Pipelining von R_2
$R_1\{\{\sqsupset\|\bowtie\|\sqsubset\}\,\|\,\ltimes\}_\varphi^{Rel,Rel}\,R_2$	ja	nein
$R_1\{\{\sqsupset\|\bowtie\|\sqsubset\}\,\|\,\ltimes\}_\varphi^{Sort,\,Sort}\,R_2$	ja	ja
$R_1\{\{\sqsupset\|\bowtie\|\sqsubset\}\,\|\,\ltimes\}_\varphi^{Rel,\,Index}\,R_2$	ja	nein
$R_1\{\{\sqsupset\|\bowtie\|\sqsubset\}\,\|\,\ltimes\}_\varphi^{Index,\,Rel}\,R_2$	nein	nein
$R_1\{\{\sqsupset\|\bowtie\|\sqsubset\}\,\|\,\ltimes\}_\varphi^{Index,\,Index}\,R_2$	nein	nein
$R_1\{\{\sqsupset\|\bowtie\|\sqsubset\}\,\|\,\ltimes\}_\varphi^{Hash,\,Hash}\,R_2$	nein	nein

Tabelle 2.15: Pipelining bei verschiedenen Joinoperationen

- Semi- und Antisemijoin:

  Alle Semi- bzw. Antisemijoinalgorithmen besitzen dieselben Voraussetzungen wie normale Joinalgorithmen. Daraus folgt, dass auch die gleichen Möglichkeiten für Pipelining bestehen. Allerdings gilt die Eigenschaft der Kommutativität bei diesen besonderen Joins nicht, so dass beim Index-Rel-Semijoin ($R_1 \ltimes_\varphi^{Index,\,Rel} R_2$) nicht dasselbe wie für den Rel-Index-Semijoin ($R_1 \ltimes_\varphi^{Rel,\,Index} R_2$) angenommen werden kann. Bei einem Index-Rel-Semijoin wird nämlich auf die erste Relation R_1 mit einem Index zugegriffen, der Pipelining verhindert. Da die zweite Relation R_2 mehrfach durchlaufen wird, ist auch hier kein Pipelining möglich.

- Left-[Right]-Outer-Join:

  Die Tabelle 2.15 gilt für Left-Outer-Joins ($R_1 \sqsupset\!\bowtie_\varphi R_2$) sowie für Right-Outer-Joins ($R_1 \bowtie\!\sqsubset_\varphi R_2$) mit vertauschten Relationen. Sie stimmt außerdem mit der Tabelle für Full-Outer-Joins ($R_1 \sqsupset\!\bowtie\!\sqsubset_\varphi R_2$) überein, nur dass dort kein Indexzugriff auf die zweite Relation verwendet werden kann. Alle Begründungen entsprechen denen der originalen Join-Implementierungen.

- Mengenoperatoren:

  Mengenoperationen können auf Joins bzw. Anti-/Semijoins zurückgeführt werden.

- Aggregation:

  Beim Gruppieren von Ergebnistupeln ($\Gamma_{\overline{G}\#\overline{F}}(R)$) muss für jedes Tupel der bereits gelesene Teil erneut durchlaufen werden. Dies kann innerhalb eines Durchlaufes geschehen, wenn die Argumentrelation sortiert vorliegt. Wenn ein Index genutzt wird, ist grundsätzlich kein Pipelining möglich (siehe Tabelle 2.16).

$\Gamma_{\overline{G}\#\overline{F}}(R)$	Pipelining von R
$\Gamma_{\overline{G}\#\overline{F}}^{Rel}(R)$	nein
$\Gamma_{\overline{G}\#\overline{F}}^{Sort}(R)$	ja
$\Gamma_{\overline{G}\#\overline{F}}^{Index}(R)$	nein

Tabelle 2.16: Pipelining bei einer Aggregation

- Indexe und Sortierungen:

  Da für das Indizieren und Sortieren die gesamte Argumentrelation vorliegen muss, blockiert ihre Erzeugung das Pipelining in beiden Fällen (siehe Tabelle 2.17).

	Pipelining von R
$\mathsf{SORT_A(R)}$	nein
$\mathsf{INDEX_A(R)}$	nein

Tabelle 2.17: Pipelining bei Indexen und Sortierungen

2.9 Vererbung von Indexen und Sortierungen

Neben Pipelining ist die Vererbung von Indexen und Sortierungen der zweite große Entscheidungsgrund für bestimmte Anordnungen von Operatoren. In dieser Arbeit wird davon ausgegangen, dass nach dem Benutzen eines Indexes dieser nicht weiter benutzt werden kann, da er sich auf eine obsolete Relation bezieht. Demzufolge wird das Hauptaugenmerk der Betrachtungen auf die Weitergabe von Sortierungen gelegt. Dafür wird die eingangs definierte Funktion $\mathsf{supports_ordered_access}()$ verwendet, die für Basisrelationen im Data Dictionary spezifiziert ist. In diesem Abschnitt sind die dafür benötigten Formeln aufgeführt.

- Selektion und Projektion

  Der Selektionsoperator wählt zwar bestimmte Tupel aus einer Relation aus, aber verändert dabei ihre Reihenfolge nicht. Deshalb können vorliegende Sortierungen weitergereicht werden. Beim Indexzugriff gehen wir davon aus, dass die Selektionsbedingung nur ein Attribut A besitzt, auf dem auch der verwendete Index existiert. Beim Zugriff über den Index kann die Ausgabe auch sortiert weitergereicht werden.

  Beim Projektionsoperator ohne Duplikateliminierung wird ebenfalls die Reihenfolge der Argumentrelationstupel nicht verändert. Existiert eine Sortierung auf dem Attribut $\mathsf{A} \in \overline{\mathsf{A}}$, so wird diese ebenfalls vererbt.

τ	$\mathsf{supports_ordered_access}\,(\tau, \mathsf{A}) =$
$\sigma_{\varphi}^{\mathsf{Rel}}(\mathsf{R})$	$\mathsf{supports_ordered_access}\,(\mathsf{R}, \mathsf{A})$
$\sigma_{\varphi}^{\mathsf{Sort}}(\mathsf{R})$	$\mathsf{supports_ordered_access}\,(\mathsf{R}, \mathsf{A})$
$\sigma_{\varphi}^{\mathsf{Index}}(\mathsf{R})$	$\mathsf{supports_index_access}\,(\mathsf{R}, \mathsf{A})$
$\pi_{\overline{\mathsf{A}}}^{\mathsf{Rel}}(\mathsf{R})$	$\mathsf{supports_ordered_access}\,(\mathsf{R}, \mathsf{A})$
$\pi_{\overline{\mathsf{A}}}^{\mathsf{Sort}}(\mathsf{R})$	$\mathsf{supports_ordered_access}\,(\mathsf{R}, \mathsf{A})$
$\pi_{\overline{\mathsf{A}}}^{\mathsf{Index}}(\mathsf{R})$	$\mathsf{supports_index_access}\,(\mathsf{R}, \mathsf{A})$

Tabelle 2.18: Vererbung von Sortierungen bei Selektion und Projektion

- Join

  Im Folgenden sei $\overline{\mathsf{B}}$ die Menge der Joinattribute, die sich auf $\mathsf{R_1}$, und $\overline{\mathsf{C}}$ die Menge der Joinattribute, die sich auf $\mathsf{R_2}$ beziehen. $\overline{\mathsf{B}}_{\varphi}$ und $\overline{\mathsf{C}}_{\varphi}$ sind die zusammengehörigen Attribute eines Prädikates $\varphi, \Theta_{\varphi} \in \{\geq, >, \neq, =, <, \leq\}$ der Vergleichsoperator, $\overline{\mathsf{A}} \subseteq (\mathsf{sch}(\mathsf{R_1}) - \overline{\mathsf{B}})$, $\overline{\mathsf{D}} \subseteq (\mathsf{sch}(\mathsf{R_2}) - \overline{\mathsf{C}})$ und $\varphi \equiv \overline{\mathsf{B}}_{\varphi}\, \Theta_{\varphi}\, \overline{\mathsf{C}}_{\varphi}$.

  Bei einem Merge-Join werden nur Prädikate unterstützt, die $\Theta_{\varphi} \in \{=\}$ enthalten. Damit ein Merge-Join angewandt werden kann, müssen die Argumentrelationen auf den Joinattributen sortiert sein. Da er die Reihenfolge der Tupel nicht ändert, bleiben Sortierungen erhalten. Bei einem Nested-Loop-Join wird die Reihenfolge der Tupel aus $\mathsf{R_1}$ nicht verändert,

τ	supports_ordered_access $(\tau, ...) =$			
	$\overline{A}$	$\overline{B}_\varphi$	$\overline{C}_\varphi$	$\overline{D}$
$R_1 \bowtie_\varphi^{\text{Rel, Rel}} R_2$	supports_ordered_access $(R_1, \overline{A})$	supports_ordered_access $(R_1, \overline{B}_\varphi)$	supports_ordered_access $(R_1, \overline{B}_\varphi)$, falls $\Theta_\varphi \in \{=\}$	false
$R_1 \bowtie_\varphi^{\text{Sort, Sort}} R_2$	supports_ordered_access $(R_1, \overline{A})$	true	true	supports_ordered_access $(R_2, \overline{D})$
$R_1 \bowtie_\varphi^{\text{Rel, Index}} R_2$	supports_ordered_access $(R_1, \overline{A})$	supports_ordered_access $(R_1, \overline{B}_\varphi)$	supports_ordered_access $(R_1, \overline{B}_\varphi)$, falls $\Theta_\varphi \in \{=\}$	false
$R_1 \bowtie_\varphi^{\text{Index, Index}} R_2$	false	true	true, falls $\Theta_\varphi \in \{=\}$	false
$R_1 \bowtie_\varphi^{\text{Hash, Hash}} R_2$	false	false	false	false

Tabelle 2.19: Vererbung von Sortierungen beim Join

wodurch sortierte Spalten weiterhin sortiert bleiben. Ist R_1 auf einem Joinattribut sortiert, dann wird das dazugehörige Attribut aus R_2 ebenfalls im Ergebnis des Joins sortiert sein (vorausgesetzt $\Theta_\varphi \in \{=\}$) (siehe Tabelle 2.19).

Ansonsten kann man nicht annehmen, dass Sortierungen entstehen oder erhalten bleiben. Das Gleiche gilt für den Rel-Index-Join. Die Formeln für Index-Rel-Joins sind dieselben wie für Rel-Index-Joins mit vertauschten Argumentrelationen. Bei einem Index-Index-Join werden die Tupel nach den Joinattributen sortiert, da die Werte der Reihe nach von den Indexen geliefert werden. Sortierungen auf Nicht-Joinattributen gehen dabei im Allgemeinen verloren.

τ	supports_ordered_access $(\tau, ...) =$	
	$\overline{A}$	$\overline{B}_\varphi$
$R_1 \ltimes_\varphi^{\text{Rel, Rel}} R_2$	supports_ordered_access $(R_1, \overline{A})$	supports_ordered_access $(R_1, \overline{B}_\varphi)$
$R_1 \ltimes_\varphi^{\text{Sort, Sort}} R_2$	supports_ordered_access $(R_1, \overline{A})$	true
$R_1 \ltimes_\varphi^{\text{Index, Rel}} R_2$	false	true
$R_1 \ltimes_\varphi^{\text{Rel, Index}} R_2$	supports_ordered_access $(R_1, \overline{A})$	supports_ordered_access $(R_1, \overline{B}_\varphi)$
$R_1 \ltimes_\varphi^{\text{Index, Index}} R_2$	false	true
$R_1 \ltimes_\varphi^{\text{Hash, Hash}} R_2$	false	false

Tabelle 2.20: Vererbung von Sortierungen beim Semijoin

- Antisemi-/Semijoin

  Die Tabelle 2.20 für Semijoins enthält als zusätzlichen physischen Operator den Index-Rel-Join. Ferner stimmt sie mit der vorherigen überein, nur dass keine Attribute aus R_2 in der Ergebnisrelation vorkommen, da diese nur darüber entscheiden, ob ein Tupel aus R_1 erhalten bleibt oder nicht. Bei einem Index-Rel-Join werden die Tupel aus R_1 nach den Joinattributen sortiert ausgegeben, da der Index die Tupel der Reihe nach ausgibt. Vorherige Sortierungen bleiben dabei nicht erhalten.

- Outer-Join

  Beim Left-Outer-Join werden die Tupel der linken Argumentrelation zum Ergebnis hin-

zugefügt, auch wenn sie keinen Joinpartner gefunden haben. Die linke Argumentrelation muss dabei komplett eingelesen werden, weshalb ein Indexzugriff keinen Sinn hat. Bei der rechten Argumentrelation können durch das Einfügen von null-Werten keine Sortierungen entstehen oder erhalten bleiben.

Das Vererben von Sortierungen kann bei einem Right-Outer-Join wie bei einem Left-Outer-Join mit vertauschten Argumenten betrachtet werden, so dass Tabelle 2.21 nur eines zusammenfasst. Es ist sogar möglich, Sortierungen für einen Full-Outer-Join zu übernehmen. Der Unterschied ist lediglich, dass die Tupel der zweiten Relation ohne Verbundpartner im letzten Schritt hinzugefügt werden.

Da aus beiden Relationen alle Tupel gelesen werden, ist nur ein Nested-Loop- und ein Merge-Join denkbar. Wird ein Merge-Full-Outer-Join verwendet, kann man bei beiden Relationen nicht davon ausgehen, dass Sortierungen erhalten bleiben.

τ	supports_ordered_access $(\tau, ...) =$ $\overline{\mathsf{A}}$	$\overline{\mathsf{B}}_\varphi$	$\overline{\mathsf{C}}_\varphi$	$\overline{\mathsf{D}}$
$\mathsf{R}_1 \supset\!\bowtie_\varphi^{\mathsf{Rel},\,\mathsf{Rel}} \mathsf{R}_2$	supports_ordered_access $(\mathsf{R}_1, \overline{\mathsf{A}})$	supports_ordered_access $(\mathsf{R}_2, \overline{\mathsf{B}}_\varphi)$	false	false
$\mathsf{R}_1 \supset\!\bowtie_\varphi^{\mathsf{Sort},\,\mathsf{Sort}} \mathsf{R}_2$	supports_ordered_access $(\mathsf{R}_1, \overline{\mathsf{A}})$	true	false	false
$\mathsf{R}_1 \supset\!\bowtie_\varphi^{\mathsf{Rel},\,\mathsf{Index}} \mathsf{R}_2$	supports_ordered_access $(\mathsf{R}_1, \overline{\mathsf{A}})$	supports_ordered_access $(\mathsf{R}_2, \overline{\mathsf{B}}_\varphi)$	false	false
$\mathsf{R}_1 \supset\!\bowtie_\varphi^{\mathsf{Hash},\,\mathsf{Hash}} \mathsf{R}_2$	false	false	false	false

Tabelle 2.21: Vererbung von Sortierungen beim Left-Outer-Join

- Mengenoperationen

  Mengenoperationen können auf Joins bzw. Anti-/Semijoins zurückgeführt werden.

- Aggregation

  Bei Spalten, die durch Gruppierungsfunktionen, wie z. B. count(), neu entstanden sind, kann nicht davon ausgegangen werden, dass Sortierungen entstehen (siehe Tabelle 2.22).

τ	supports_ordered_access $(\tau, \mathsf{A}) =$
$\Gamma^{\mathsf{Rel}}_{\overline{\mathsf{G}}\#\overline{\mathsf{F}}}(\mathsf{R})$	supports_ordered_access (R, A)
$\Gamma^{\mathsf{Sort}}_{\overline{\mathsf{G}}\#\overline{\mathsf{F}}}(\mathsf{R})$	true, wenn $\mathsf{A} = \overline{\mathsf{F}}$
$\Gamma^{\mathsf{Index}}_{\overline{\mathsf{G}}\#\overline{\mathsf{F}}}(\mathsf{R})$	true, wenn $\mathsf{A} = \overline{\mathsf{F}}$

Tabelle 2.22: Vererbung von Sortierungen bei Aggregationen

2.10 Definition eines generalisierten Baumes

Im Zuge dieser Arbeit wurde eine Datenstruktur für Anfragebäume und den daraus resultierenden Ausführungsplänen benötigt, welche sämtliche Informationen, die während der Optimierung zur Kostenberechnung verwendet werden, schnell abspeichert und genauso schnell wieder zugreifbar macht (siehe Fabig [20] für eine detailliertere Darstellung).

Definition 2.10.1 Ein *generalisierter Baum* ist ein um Attribute angereicherter Operatorbaum, der im Sinne von Attributen einer attributierten Grammatik abgeleitet wird.

Die Attribute der einzelnen Knoten sind relationenbezogene Daten, Metainformationen und Funktionen, die beim Erzeugen des Knotens gesetzt bzw. berechnet werden (s. u.).

Ein wichtiger Grund für die Wahl einer attributierten Grammatik zur Erzeugung eines Ableitungsbaumes, der in unserem Fall einen Ausführungsplan darstellt, ist die Berücksichtigung von kontextabhängigen Spracheigenschaften. Dadurch können Bedingungen, die durch eine kontextfreie Grammatik nicht abzudecken sind, z. B. Kostenwerte in den Knoten, mitgeführt werden.

Ein weiterer Vorteil besteht darin, dass durch synthetische und inherite Attribute Eigenschaften von Teilbäumen des attributierten Ableitungsbaumes einfach in einen anderen Teilbaum verschoben werden können. Damit können Bedingungen erfüllt werden, mit denen die Sprache, die die Grammatik beschreibt, sinnvoll beschränkt werden kann. Dazu gehören inherite, absteigende und synthetische, aufsteigende Attribute. Beim generalisierten Baum werden nur synthetische Attribute verwendet, da sonst Zyklen vorkommen könnten. Daraus folgt, dass auf Ausführungspläne in dieser Arbeit per bottom-up Prinzip zugegriffen wird.

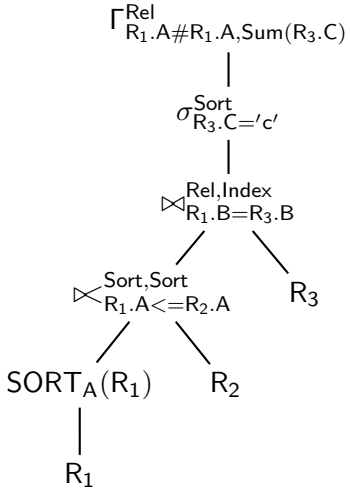


Abbildung 2.7: Beispiel für einen generalisierten Baum

Im Folgenden werden alle Attribute aufgeführt, die beim Erzeugen eines Knotens gesetzt werden müssen:

- Physische Umsetzung (z. B. ^{Rel,Rel}, falls nicht mehr gültig, wird sie automatisch mit Hilfe der Statistiken bestimmt)
- Bedingungen (φ)

Weiterhin müssen auch neue Attributwerte berechnet werden:

- Schema der Ergebnisrelation ($\mathbf{sch}(R)$)
- Kardinalität ($|R|$)
- Attributwertanzahlen ($\mathsf{n}(A, R)$)
- Pagegrößen ($\mathsf{pg}(R)$)
- Kosten des Teilbaumes ($\mathsf{cost}(\tau)$)
- Index-Unterstützungen ($\mathsf{supports_index_access}(R, A)$)
- Sortierungen ($\mathsf{supports_ordered_access}(R, A)$)

Diese werden hier immer bottom-up bestimmt und entsprechen demzufolge synthetischen Attributen im Sinne einer attributierten Grammatik. Entscheidend ist, dass Kosten ständig inkrementell, z. B. bei Austausch eines Knotens oder Teilbaumes, nachberechnet werden können.

Abbildung 2.7 zeigt beispielhaft einen generalisierten Baum, als Repräsentanten eines Ausführungsplanes ohne Attribute.

2.11 Beispiel für die Kostenberechnung eines Ausführungsplanes

Gegeben seien folgende Relationen aus der TPC-H Benchmarkdatenbank (siehe `http://www.tpc.org/tpch/spec/tpch2.3.0.pdf` für eine komplette Schemaübersicht):

orders(l_orderkey → o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment)

lineitem(l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag, l_linestatus, l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_shipmode, l_comment)

Mit der dazugehörigen TPC-H Query 4, welche an ein zentrales Datenbanksystem gestellt wird:

$$\tau := \Gamma_{\text{orders.o_orderpriority}\#\text{orders.o_orderpriority},\text{count}(*)\text{ as O.order_count}} \left(\sigma_{\text{orders.o_orderdate}>='1993-07-01' \wedge \text{orders.o_orderdate}<'1993-10-01'}(\text{orders}) \bowtie_{\text{orders.o_orderkey}=\text{lineitem.l_orderkey}} \left(\sigma_{\text{lineitem.l_commitdate}<\text{lineitem.l_receiptdate}}(\text{lineitem})\right)\right)$$

Ein Ausschnitt der verwendeten Statistiken lautet:

|orders|	=	15.001.500
pg(orders)	=	236.227
n(o_orderkey, orders)	=	15.001.500
n(o_comment)	=	7.811.655
n(o_orderdate)	=	2.395
n(o_custkey)	=	870.569
n(o_clerk)	=	9.998
n(o_orderstatus)	=	3
n(o_orderpriority)	=	5
n(o_totalprice)	=	10.429.377

n(o_shippriority)	=	1
min(o_orderdate, orders)	=	01.01.92
max(o_orderdate, orders)	=	02.08.98
|lineitem|	=	59.931.416
pg(lineitem)	=	1.024.974
n(l_orderkey, lineitem)	=	15.684.741
n(l_receiptdate, lineitem)	=	2.487
n(l_commitdate, lineitem)	=	2.420
w_1	=	0, 02
w_2	=	0

In Abbildung 2.8 wird der aus der Anfrage direkt übersetzte Ausführungsplan dargestellt.

Zur Berechnung der Kosten wird die in Abschnitt 2.2 vorgestellte Vorschrift verwendet. Demzufolge werden die Kosten jedes einzelnen Operators aufaddiert. Da $w_2 = 0$ ist, werden diese Kostenterme nicht gesondert aufgeführt:

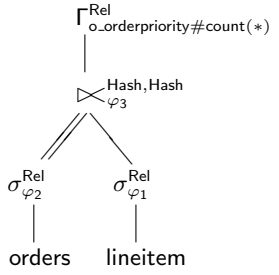


Abbildung 2.8: TPC-H Query 4 Ausführungsplan

1. Schritt $\tau_1 := \sigma^{\text{Rel}}_{\varphi_1}(\text{lineitem})$ mit $\varphi_1 := \text{lineitem.l_commitdate} < \text{lineitem.l_receiptdate}$

Kosten:

$$\begin{aligned} \text{cost}_{\text{op}}(\tau_1) &= \text{pg}(\text{lineitem}) + w_1 \cdot |\text{lineitem}| \\ &= 1.024.974 + 0,02 \cdot 59.931.416 \\ &= 2.223.602,32 \end{aligned}$$

Kardinalität:

$$\begin{aligned} |\tau_1| &= \text{sel}(\varphi_1, \text{lineitem}) \cdot |\text{lineitem}| \\ &= \frac{1}{3} \cdot 59.931.416 \\ &= 19.977.138,\overline{6} \end{aligned}$$

Attributwertanzahlen :
$$\begin{aligned} \mathsf{n}(\mathsf{l_orderkey}, \tau_1) &= \frac{|\tau_1| + \mathsf{n}(\mathsf{l_orderkey}, \mathsf{lineitem})}{3} \\ &= \frac{19.977.138,\overline{6} + 15.684.741}{3} \\ &= 11.887.293,\overline{2} \end{aligned}$$

Seitenanzahl:
$$\mathsf{pg}(\tau_1) = 341.658$$

2. Schritt $\tau_2 := \sigma^{\mathsf{Rel}}_{\varphi_2}(\mathsf{orders})$ mit $\varphi_2 := \mathsf{orders.o_orderdate} >=' 1993-07-01' \wedge \mathsf{orders.o_orderdate} <' 1993-10-01'$

Kosten:
$$\begin{aligned} \mathsf{cost}_{\mathsf{op}}(\tau_2) &= \mathsf{pg}(\mathsf{orders}) + \mathsf{w}_1 \cdot |\mathsf{orders}| \\ &= 236.227 + 0,02 \cdot 15.001.500 \\ &= 536.257 \end{aligned}$$

Kardinalität:
$$\begin{aligned} |\tau_2| &= \mathsf{sel}(\varphi_2, \mathsf{orders}) \cdot |\mathsf{orders}| \\ &= \frac{01.10.93 - 01.07.93}{\mathsf{max}(\mathsf{o_orderdate}, \mathsf{orders}) - \mathsf{min}(\mathsf{o_orderdate}, \mathsf{orders})} \cdot 15.001.500 \\ &= \frac{01.10.93 - 01.07.93}{02.08.98 - 01.01.92} \cdot 15.001.500 \\ &= 569.677,215 \end{aligned}$$

Attributwertanzahlen:
$$\begin{aligned} &\mathsf{n}(\mathsf{o_orderkey}, \tau_2) = |\tau_2| = 569.677,215 \\ &\mathsf{n}(\mathsf{o_totalprice}, \tau_2) = |\tau_2| = 569.677,215 \\ &\mathsf{n}(\mathsf{o_comment}, \tau_2) = |\tau_2| = 569.677,215 \\ &\mathsf{n}(\mathsf{o_orderpriority}, \tau_2) = \mathsf{n}(\mathsf{o_orderpriority}, \mathsf{orders}) = 5 \\ &\mathsf{n}(\mathsf{o_clerk}, \tau_2) = \mathsf{n}(\mathsf{o_clerk}, \mathsf{orders}) = 9.998 \\ &\mathsf{n}(\mathsf{o_orderstatus}, \tau_2) = \mathsf{n}(\mathsf{o_orderstatus}, \mathsf{orders}) = 3 \\ &\mathsf{n}(\mathsf{o_shippriority}, \tau_2) = \mathsf{n}(\mathsf{o_shippriority}, \mathsf{orders}) = 1 \end{aligned}$$

Mit der Verwendung der Kombinatorischen Formel (†) ergibt sich:
$$\begin{aligned} \mathsf{n}(\mathsf{o_custkey}, \tau_2) &= \frac{|\tau_2| + \mathsf{n}(\mathsf{o_custkey}, \mathsf{orders})}{3} \\ &= \frac{569.677,215 + 870.569}{3} \\ &= 480.082,072 \end{aligned}$$

$$\begin{aligned} \mathsf{n}(\mathsf{o_orderdate}, \tau_2) &= \mathsf{sel}(\varphi_2, \mathsf{orders}) \cdot \mathsf{n}(\mathsf{o_orderdate}, \mathsf{orders}) \\ &= 0,037974684 \cdot 2395 \\ &= 90,949 \end{aligned}$$

Seitenanzahl: $\mathsf{pg}(\tau_2) = \frac{\mathsf{pg}(\mathsf{orders}) \cdot |\tau_2|}{|\mathsf{orders}|} = \frac{236227 \cdot 569.677,215}{15001500} = 8.970,646$

3. Schritt $\tau_3 := \tau_2 \rtimes^{\mathsf{Hash},\mathsf{Hash}}_{\varphi_3} \tau_1$ mit $\varphi_3 := \mathsf{orders.o_orderkey} = \mathsf{lineitem.l_orderkey}$

Kosten:
$$\begin{aligned} \mathsf{cost}(\tau_3) = &\mathsf{pg}(\tau_1) + \mathsf{n}(\mathsf{o_orderkey}, \tau_2) + \mathsf{n}(\mathsf{l_orderkey}, \tau_1) \\ &+ \mathsf{w}_1 \cdot (|\tau_2| + |\tau_1| + \mathsf{n}(\mathsf{o_orderkey}, \tau_2) + \mathsf{n}(\mathsf{l_orderkey}, \tau_1)) \\ = &341.658 + 569.677,215 + 11.887.293,\overline{2} \\ &+ 0,02 \cdot (569.677,215 + 19.977.138,\overline{6} + 569.677,215 + 11.887.293,\overline{2}) \\ = &13.458.704,163 \end{aligned}$$

Kardinalität:
$$\begin{aligned} |\tau_3| &= |\pi_{\mathsf{sch}(\mathsf{orders})}(\tau_2 \bowtie^{\mathsf{Hash},\mathsf{Hash}}_{\varphi_3} \tau_1)| \\ &= \mathsf{min}(\mathsf{min}(|\tau_2|, \tau_2 \bowtie^{\mathsf{Hash},\mathsf{Hash}}_{\varphi_3} \tau_1)|, \Pi_{\mathsf{A} \in \mathsf{sch}(\mathsf{orders})} \mathsf{n}(\mathsf{A}, \tau_2 \bowtie^{\mathsf{Hash},\mathsf{Hash}}_{\varphi} \tau_1)) \\ &= \mathsf{min}(\mathsf{min}(|\tau_2|, |\sigma_{\varphi_3}(\tau_2 \times \tau_1))|, \Pi_{\mathsf{A} \in \mathsf{sch}(\mathsf{orders})} \mathsf{min}(|\tau_2|, \mathsf{n}(\mathsf{A}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)))) \\ &= \mathsf{min}(\mathsf{min}(569.677,215; 957.368,554), \\ &\quad 321.547.359.793.141.516.575.738.064.022,017) \\ &= 569.677,215 \end{aligned}$$

$$
\begin{aligned}
|\sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)| &= \mathsf{sel}(\varphi_3, \tau_2 \times \tau_1) \cdot |\tau_2 \times \tau_1| \\
&= \mathsf{min}(\frac{1}{\mathsf{n}(\mathsf{o_orderkey}, \tau_2)}, \frac{1}{\mathsf{n}(\mathsf{l_orderkey}, \tau_1)}) \cdot |\tau_2| \cdot |\tau_1| \\
&= \mathsf{min}(0,000001755; 0,000000084) \cdot 569.677,215 \cdot 19.977.138,\overline{6} \\
&= 957.368,554
\end{aligned}
$$

Attributwertanzahlen:
$\mathsf{n}(\mathsf{o_orderpriority}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) = \mathsf{n}(\mathsf{o_orderpriority}, \tau_2) = 5$
$\mathsf{n}(\mathsf{o_clerk}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) = \mathsf{n}(\mathsf{o_clerk}, \tau_2) = 9.998$
$\mathsf{n}(\mathsf{o_orderstatus}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) = \mathsf{n}(\mathsf{o_orderstatus}, \tau_2) = 3$
$\mathsf{n}(\mathsf{o_shippriority}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) = \mathsf{n}(\mathsf{o_shippriority}, \tau_2) = 1$
$\mathsf{n}(\mathsf{o_orderdate}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) = \mathsf{n}(\mathsf{o_orderdate}, \tau_2) = 90,949$

Mit der Verwendung der Kombinatorischen Formel (†) ergibt sich:

$$
\begin{aligned}
\mathsf{n}(\mathsf{o_custkey}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) &= \frac{|\sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2)| + \mathsf{n}(\mathsf{o_custkey}, \tau_2)}{3} \\
&= \frac{957.368,554 + 480082,072}{3} \\
&= 479.150,209
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{n}(\mathsf{o_totalprice}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) &= \frac{|\sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2)| + \mathsf{n}(\mathsf{o_totalprice}, \tau_2)}{3} \\
&= \frac{957.368,553 + 569677,215}{3} \\
&= 509.015,256
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{n}(\mathsf{o_comment}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) &= \frac{|\sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2)| + \mathsf{n}(\mathsf{o_comment}, \tau_2)}{3} \\
&= \frac{957.368,554 + 569.677,215}{3} \\
&= 509.015,256
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{n}(\mathsf{o_orderkey}, \sigma^{\mathsf{Rel}}_{\varphi_3}(\tau_2 \times \tau_1)) &= \mathsf{min}(\mathsf{n}(\mathsf{o_orderkey}, \tau_2), \mathsf{n}(\mathsf{l_orderkey}, \tau_1)) \\
&= \mathsf{min}(569.677,215; 19977138,\overline{6}) \\
&= 569.677,215
\end{aligned}
$$

Seitenanzahl: $\mathsf{pg}(\tau_3) = 8.970,646$

4. Schritt $\tau_4 := \Gamma^{\mathsf{Rel}}_{\mathsf{o_orderpriority}\#\mathsf{count}}(\tau_3)$

Kosten:

$$
\begin{aligned}
\mathsf{cost}_{\mathsf{op}}(\tau_4) &= \mathsf{pg}(\tau_3) + |\tau_3| \cdot \frac{\mathsf{pg}(\tau_3)}{2} + \mathsf{w}_1 \cdot (|\tau_3| + |\tau_3| \cdot \frac{\tau_3}{2}) \\
&= 8.970,646 + 569.677,215 \cdot \frac{8.970,646}{2} \\
&\quad + 0,02 \cdot \left(569.677,215 + 569.677,215 \cdot \frac{569.677,215}{2}\right) \\
&= 3.245.332.685,13
\end{aligned}
$$

Gesamtkosten :

$$
\begin{aligned}
\mathsf{cost}(\tau) &= \mathsf{cost}_{\mathsf{op}}(\tau_1) + \mathsf{cost}_{\mathsf{op}}(\tau_2) + \mathsf{cost}_{\mathsf{op}}(\tau_3) + \mathsf{cost}_{\mathsf{op}}(\tau_4) \\
&= 2.223.602,32 + 536.257 + 13.458.704,163 + 3.245.332.685,13 \\
&= 3.261.551.248,614
\end{aligned}
$$

2.12 Fazit des Kostenmodells

In diesem Kapitel wurden zu Beginn alle benötigten Konventionen getroffen, um die verschiedenen Aspekte der Anfragenoptimierung einheitlich angehen zu können. Des Weiteren lag die

Hauptaufgabe darin, ein präzises Kostenmodell vorzustellen, welches systemabhängige Abschätzungen für Kosten von Ausführungsplänen ermöglicht. Diese Pläne können als generalisierte Bäume mit einfachen Zugriffsmethoden verwendet werden.

Im Gegensatz zur allgemeinen Annahme, dass die Exaktheit eines Kostenmodells nicht hauptausschlaggebend für eine gute Optimierung ist und Abschätzungen nur die realen Kostendifferenzen der einzelnen Ausführungspläne für die gleiche Anfrage in dem Modell bereitstellen müssen, wurde im Zuge dieser Arbeit die Erfahrung gemacht, dass ein gutes Kostenmodell die Basis jeglicher guten Optimierung ist. Insbesondere bei späteren Tests zeigten sich grobe Abschätzungen mit bekannten Standardwerten als Hauptfehlerquelle für falsche Entscheidungen.

Einer der Hauptgründe für die Entwicklung eines präzisen Kostenmodells ist der Wunsch, Kostengrößen proportional zu Laufzeiten angeben zu können, d. h. wenn eine Anfrage einen großen Kostenwert hat, dann folgt daraus, dass sie längere Zeit zur Ausführung benötigt. Dieser Punkt ist wichtig, weil damit die Entscheidung zur Fortführung der Optimierung oder dem sofortigen Ausführen des bislang als besten ausgegebenen Ausführungsplanes getroffen werden kann.

In dieser Arbeit wurden alle getätigten Abschätzungen bzw. Kostenformeln für relationale Operatoren mit Hilfe des TPC-H Benchmarkes Version 2.3.0 (siehe Abschnitt 6.3.1) und am Beispiel von räumlichen Operatoren für objektrelationale (siehe Abschnitt 6.3.3) validiert.

Allgemein sollten zwei Annahmen, welche zu unpräziseren Kostenabschätzungen führen, verbessert werden:

(a) Bei der Selektivitätsabschätzung sind Histogramme zu bevorzugen, da die Standardwerte hier das Optimierungsergebnis stark verschlechtern. Spätere Tests (siehe Abschnitt 6.3.1) zeigten, dass ungenaue Selektivitätsabschätzungen und Attibutwertverteilungen das Optimierungsergebnis sogar unbrauchbar machen können.

(b) Die Annahme von gleich großen Tupellängen sollte durch exakte Längen ersetzt werden. Daraus resultieren bessere Abschätzungen bezüglich der Seitenbelegungen, die wiederum die Kostenabschätzung stark beeinflussen.

Heute erlangt das Thema „Statistiken“ eine immer wichtigere Rolle und wird unter dem Namen „statistical database profile“ geführt. Dabei wird das statistische Profil einer Datenbank durch Verwendung von Statistiken und statistischen Analysen summiert, welches dann als Grundlage für die Anfrageoptimierung, den physischen Datenbank-Entwurf und die Leistungsvorhersagen benutzt wird. Kommerzielle Datenbanken wie z. B. Oracle oder IBM statten ihre neuen Optimierer mit einer künstlichen Intelligenz aus, so dass sie autonom Statistiken sammeln und aktualisieren können, ohne dabei auf einen Benutzer angewiesen zu sein.

Nachdem das Kostenmodell zur Abschätzung von Ausführungsplänen entwickelt worden ist, wird im nachfolgenden Kapitel die Methodik erläutert, mit der man optimierte äquivalente Varianten eines Ausführungsplanes generieren kann.

Ein Blitzableiter auf einem Kirchturm ist das denkbar stärkste Mißtrauensvotum gegen den lieben Gott.

Karl Kraus 1874-1936



3 Optimierung von Ausführungsplänen mit Hilfe eines Termersetzungssystems

Zu Beginn dieses Kapitels werden algebraische Termersetzungsregeln beschrieben und der Grundstein für die Optimierung von Ausführungsplänen mit Hilfe eines Termersetzungsoptimierers gelegt. Nach der Erweiterung auf physische Implementierungsarten der eingesetzten algebraischen Operatoren werden ebenfalls Regeln zum Erzeugen von Sortierungen und Indexen angegeben. Anschließend werden diese Regeln zu generalisierten Regeln verallgemeinert. Danach werden zum ersten Mal algebraische Ersetzungsregeln mit Kostenbedingungen versehen. Darauf aufbauend wird für wichtige Regeln eine Methodik vorgestellt, mit deren Hilfe der Berechnungsaufwand für die Bedingungen verringert werden kann. Seinen Abschluss findet dieses Kapitel durch die Beschreibung der Erweiterung von einem Join-Ordering-Verfahren mit Hilfe von bedingten Termersetzungsregeln.

3.1 Ein klassisches System von bedingten Termersetzungen

Operatorbäume (Anfragebäume und Ausführungspläne) können mit Hilfe der Relationalen Algebra als Terme dargestellt werden. Diese Darstellungsmöglichkeit bietet die Flexibilität, bekannte Werkzeuge wie ein Termersetzungssystem zu benutzen, um einen originalen Anfragebaum in einen optimalen Ausführungsplan zu transformieren.

Den Kern eines Termersetzungssystems bildet eine Menge von Ersetzungsregeln.[1] Mit deren Hilfe kann durch Ersetzen von Teiltermen ein Term τ_1 zu einem Term τ_2 transformiert werden. Hierbei bestimmen die Regeln des Termersetzungssystems, welche Terme ersetzbar sind und durch welche neuen Terme ersetzt werden können.

Definition 3.1.1 Eine Regel hat die Form

$$\tau_1 \underset{2}{\overset{1}{\rightleftharpoons}} \tau_2$$

wobei τ_1 und τ_2 Muster für bestimmte Teile von Ausführungsbäumen bzw. -plänen sind. Das Überführungssymbol „ $\rightleftharpoons$ “ gibt die Ersetzungsrichtungen mit den Pfeilspitzen an und weist mit den angeordneten Zahlen auf zusätzliche Bedingungen hin. Eine Ersetzung findet nur dann statt, wenn alle Bedingungen für die jeweilige Richtung erfüllt sind.

3.2 Erweiterung der Regeln der Relationalen Algebra um Bedingungen

Im folgenden Abschnitt werden alle in dieser Arbeit verwendeten Termersetzungsregeln aufgeführt. Dabei unterscheiden wir zwischen implementierungsunabhängigen und -abhängigen relationalen Ersetzungsregeln.

[1] Eine umfassende Definition von Termersetzungssystemen kann bei Bündgen [10] nachgelesen werden.

3.2.1 Implementierungsunabhängige relationale Ersetzungsregeln

A) Rechenregeln für Mengenoperationen:

$$R \cup R \rightleftharpoons R \qquad R - R \rightleftharpoons \emptyset \qquad R \cap R \rightleftharpoons R$$

B) Rechenregeln für Mengenoperationen und Selektionen:

$$\sigma_{\varphi_1}(R) \cup \sigma_{\varphi_2}(R) \rightleftharpoons \sigma_{\varphi_1 \vee \varphi_2}(R)$$
$$\sigma_{\varphi_1}(R) \cap \sigma_{\varphi_2}(R) \rightleftharpoons \sigma_{\varphi_1 \wedge \varphi_2}(R)$$
$$\sigma_{\varphi_1}(R) - \sigma_{\varphi_2}(R) \rightleftharpoons \sigma_{\varphi_1 \wedge \neg\varphi_2}(R)$$

C) Rechenregeln für die leere Relation und die Relation mit einem leeren Tupel:

$$\emptyset - R \rightleftharpoons \emptyset \qquad \emptyset \times R \rightleftharpoons \emptyset \qquad \emptyset \cup R \rightleftharpoons R \qquad \emptyset \cap R \rightleftharpoons \emptyset$$
$$R - \emptyset \rightleftharpoons R \qquad R \times \emptyset \rightleftharpoons \emptyset \qquad R \cup \emptyset \rightleftharpoons R \qquad R \cap \emptyset \rightleftharpoons \emptyset$$

$$R \bowtie_\varphi \emptyset \rightleftharpoons \emptyset \qquad R \ltimes_\varphi \emptyset \rightleftharpoons \emptyset \qquad R \,\overline{\ltimes}_\varphi\, \emptyset \rightleftharpoons R \qquad \sigma_\varphi(\emptyset) \rightleftharpoons \emptyset$$
$$\emptyset \bowtie_\varphi R \rightleftharpoons \emptyset \qquad \emptyset \ltimes_\varphi R \rightleftharpoons \emptyset \qquad \emptyset \,\overline{\ltimes}_\varphi\, R \rightleftharpoons \emptyset \qquad \pi_{\overline{A}}(\emptyset) \rightleftharpoons \emptyset$$

$$R \bowtie_\varphi \{()\} \rightleftharpoons R \qquad R \ltimes_\varphi \{()\} \rightleftharpoons R \qquad R \,\overline{\ltimes}_\varphi\, \{()\} \rightleftharpoons \emptyset \qquad \{()\} \times R \rightleftharpoons R$$
$$\{()\} \bowtie_\varphi R \rightleftharpoons R \qquad\qquad\qquad\qquad\qquad\qquad R \times \{()\} \rightleftharpoons R$$

$$\{()\} \,\overline{\ltimes}_\varphi\, R \rightleftharpoons \begin{cases} \{()\}, & \text{wenn } R = \emptyset; \\ \emptyset, & \text{sonst} \end{cases} \qquad \{()\} \ltimes_\varphi R \rightleftharpoons \begin{cases} \emptyset, & \text{wenn } R = \emptyset; \\ \{()\}, & \text{sonst} \end{cases}$$

$$\pi_\emptyset(R) \rightleftharpoons \begin{cases} \emptyset, & \text{wenn } R = \emptyset; \\ \{()\}, & \text{sonst} \end{cases}$$

D) Eine Selektion mit einer nicht erfüllbaren Selektionsbedingung liefert eine leere Relation. Wenn diese Bedingung für alle Tupel der Relation R erfüllt ist, ist sie infolgedessen redundant:

$$\sigma_{\mathsf{true}}(R) \rightleftharpoons R \qquad \sigma_{\mathsf{false}}(R) \rightleftharpoons \emptyset$$

E) Eine Projektion auf alle Attribute einer Relation ist redundant:

$$\pi_{\mathbf{sch}(R)}(R) \rightleftharpoons R$$

F) Join, Vereinigung, Schnitt und das Kreuzprodukt sind kommutativ:

$$R_1 \bowtie_\varphi R_2 \rightleftharpoons R_2 \bowtie_\varphi R_1 \qquad R_1 \cup R_2 \rightleftharpoons R_2 \cup R_1$$
$$R_1 \cap R_2 \rightleftharpoons R_2 \cap R_1 \qquad R_1 \times R_2 \rightleftharpoons R_2 \times R_1$$

Hierbei wird davon ausgegangen, dass die Reihenfolge der Attribute einer Relation keine Rolle spielt bzw. über den Namen rekonstruierbar ist. Bekanntermaßen ist der Anti- bzw. Semijoin nicht kommutativ.

G) Vereinigung, Schnitt und das Kreuzprodukt sind assoziativ:

$$R_1 \cup (R_2 \cup R_3) \rightleftharpoons (R_1 \cup R_2) \cup R_3 \rightleftharpoons R_1 \cup R_2 \cup R_3$$
$$R_1 \cap (R_2 \cap R_3) \rightleftharpoons (R_1 \cap R_2) \cap R_3 \rightleftharpoons R_1 \cap R_2 \cap R_3$$
$$R_1 \times (R_2 \times R_3) \rightleftharpoons (R_1 \times R_2) \times R_3 \rightleftharpoons R_1 \times R_2 \times R_3$$

Auch Joins sind assoziativ, sofern sich die Joinbedingungen nur auf jeweils zwei Relationen beziehen. Gilt somit $\mathsf{attr}(\varphi_1) \cap \mathbf{sch}(R_3)=\emptyset$ und $\mathrm{attr}(\varphi_2) \cap \mathbf{sch}(R_1) = \emptyset$, so auch

$$\mathsf{R}_1 \bowtie_{\varphi_1} (\mathsf{R}_2 \bowtie_{\varphi_2} \mathsf{R}_3) \rightleftharpoons (\mathsf{R}_1 \bowtie_{\varphi_1} \mathsf{R}_2) \bowtie_{\varphi_2} \mathsf{R}_3 \rightleftharpoons \mathsf{R}_1 \bowtie_{\varphi_1} \mathsf{R}_2 \bowtie_{\varphi_2} \mathsf{R}_3$$

Bekanntermaßen ist der Anti- bzw. Semijoin auch nicht assoziativ.

H) Kommutativität von Selektionen:

$$\sigma_{\varphi_1}(\sigma_{\varphi_2}(\mathsf{R})) \rightleftharpoons \sigma_{\varphi_2}(\sigma_{\varphi_1}(\mathsf{R}))$$

I) Alle Konjunktionen in einer Selektionsbedingung können in mehrere Selektionen aufgebrochen bzw. nacheinander auszuführende Selektionen können durch Konjunktionen zusammengefügt werden:

$$\sigma_{\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n}(\mathsf{R}) \rightleftharpoons \sigma_{\varphi_1}(\sigma_{\varphi_2}(\ldots(\sigma_{\varphi_n}(\mathsf{R}))\ldots))$$

J) Mit Hilfe der folgenden ersten bedingten Termersetzungsregel können alle geschachtelten Projektionen eliminiert werden:

$$\pi_{\overline{\mathsf{A}}_1}\left(\pi_{\overline{\mathsf{A}}_2}\left(\ldots\left(\pi_{\overline{\mathsf{A}}_n}(\mathsf{R})\right)\ldots\right)\right) \underset{1}{\rightleftharpoons} \pi_{\overline{\mathsf{A}}_1}(\mathsf{R})$$

1) falls $\overline{\mathsf{A}}_1 \subseteq \overline{\mathsf{A}}_2 \subseteq \ldots \subseteq \overline{\mathsf{A}}_n \subseteq \mathbf{sch}(\mathsf{R})$

Bei der Anwendung dieser Regel von rechts nach links wird durch die Bedingung 1) garantiert, dass der erzeugte Term wohldefiniert ist.

K) Eine Selektion kann mit einer Projektion vertauscht werden, falls die Projektion keine Attribute der Selektionsbedingung entfernt. Es gilt also Folgendes:

$$\pi_{\overline{\mathsf{A}}}(\sigma_{\varphi}(\mathsf{R})) \overset{1}{\rightleftharpoons} \sigma_{\varphi}\left(\pi_{\overline{\mathsf{A}}}(\mathsf{R})\right)$$

1) falls $\mathsf{attr}(\varphi) \subseteq \overline{\mathsf{A}}$

Ist dies nicht der Fall, kann die Vertauschung vorgenommen werden, wenn die Projektion um die notwendigen Attribute erweitert wird, die nach der Selektion wieder wegprojiziert werden können:

$$\pi_{\overline{\mathsf{A}}_1}\left(\sigma_{\varphi}\left(\pi_{\overline{\mathsf{A}}_2}(\mathsf{R})\right)\right) \underset{1}{\rightleftharpoons} \pi_{\overline{\mathsf{A}}_1}(\sigma_{\varphi}(\mathsf{R}))$$

1) falls $\overline{\mathsf{A}}_1 \cup \mathsf{attr}(\varphi) = \overline{\mathsf{A}}_2 \subseteq \mathbf{sch}(\mathsf{R})$

L) Jede Selektion kann an einer Joinoperation (oder einem Kreuzprodukt) „vorbeigeschoben“ werden, falls sie nur Attribute genau eines Join-Argumentes verwendet.

$$\sigma_{\varphi_1}(\mathsf{R}_1 \bowtie_{\varphi_2} \mathsf{R}_2) \overset{1}{\rightleftharpoons} \sigma_{\varphi_1}(\mathsf{R}_1) \bowtie_{\varphi_2} \mathsf{R}_2$$

1) falls $\mathsf{attr}(\varphi_1) \subseteq \mathbf{sch}(\mathsf{R}_1)$

$$\sigma_{\varphi}(\mathsf{R}_1 \times \mathsf{R}_2) \overset{1}{\rightleftharpoons} \sigma_{\varphi}(\mathsf{R}_1) \times \mathsf{R}_2$$

1) falls $\mathsf{attr}(\varphi) \subseteq \mathbf{sch}(\mathsf{R}_1)$

Allgemeiner gilt darüber hinaus noch für den Join und das Kreuzprodukt:

$$\sigma_{\varphi}(\mathsf{R}_1 \bowtie_{\varphi_3} \mathsf{R}_2) \underset{2}{\overset{1,2}{\rightleftharpoons}} \sigma_{\varphi_0}(\sigma_{\varphi_1}(\mathsf{R}_1) \bowtie_{\varphi_3} \sigma_{\varphi_2}(\mathsf{R}_2))$$

$$\sigma_{\varphi}(\mathsf{R}_1 \times \mathsf{R}_2) \underset{2}{\overset{1,2}{\rightleftharpoons}} \sigma_{\varphi_0}(\sigma_{\varphi_1}(\mathsf{R}_1) \times \sigma_{\varphi_2}(\mathsf{R}_2))$$

1) falls $\mathsf{attr}(\varphi_i) \subseteq \mathbf{sch}(\mathsf{R}_i)$ für $i = 1, 2$
2) falls $\varphi = \varphi_0 \wedge \varphi_1 \wedge \varphi_2$

Eine Selektion kann auch an einem Semijoin oder Antisemijoin vorbeigeschoben werden.

$$\sigma_{\varphi_1}(\mathsf{R}_1 \ltimes_{\varphi_2} \mathsf{R}_2) \rightleftharpoons \sigma_{\varphi_1}(\mathsf{R}_1) \ltimes_{\varphi_2} \mathsf{R}_2$$

$$\sigma_{\varphi_1}(\mathsf{R}_1 \overline{\ltimes}_{\varphi_2} \mathsf{R}_2) \rightleftharpoons \sigma_{\varphi_1}(\mathsf{R}_1) \overline{\ltimes}_{\varphi_2} \mathsf{R}_2$$

M) Auf ähnliche Weise können auch Projektionen verschoben werden. Hier muss allerdings beachtet werden, dass die Joinattribute bis zur Durchführung des Joins erhalten bleiben müssen:

$$\pi_{\overline{\mathsf{A}}}(\mathsf{R}_1 \bowtie_{\varphi} \mathsf{R}_2) \overset{1}{\rightleftharpoons} \pi_{\overline{\mathsf{A}}}\left(\pi_{\overline{\mathsf{A}}_1}(\mathsf{R}_1) \bowtie_{\varphi} \pi_{\overline{\mathsf{A}}_2}(\mathsf{R}_2)\right)$$

1) falls $\overline{\mathsf{A}}_1 = \mathbf{sch}(\mathsf{R}_1) \cap (\overline{\mathsf{A}} \cup \mathsf{attr}(\varphi))$ und
$\overline{\mathsf{A}}_2 = \mathbf{sch}(\mathsf{R}_2) \cap (\overline{\mathsf{A}} \cup \mathsf{attr}(\varphi))$

Analog gelten die Verschiebungen für das Kartesische Produkt. Dabei müssen nur die zurückzuliefernden Attribute berücksichtigt werden:

$$\pi_{\overline{\mathsf{A}}}(\mathsf{R}_1 \times \mathsf{R}_2) \overset{1,2}{\underset{2}{\rightleftharpoons}} \pi_{\overline{\mathsf{A}}_1}(\mathsf{R}_1) \times \pi_{\overline{\mathsf{A}}_2}(\mathsf{R}_2)$$

1) falls $\overline{\mathsf{A}}_1 = \mathbf{sch}(\mathsf{R}_1) \cap \overline{\mathsf{A}}$ und
$\overline{\mathsf{A}}_2 = \mathbf{sch}(\mathsf{R}_2) \cap \overline{\mathsf{A}}$

2) falls $\overline{\mathsf{A}} = \overline{\mathsf{A}}_1 \cup \overline{\mathsf{A}}_2$

Ebenso kann man Projektionen am Semijoin und am Antisemijoin vorbeischieben, sofern keine Attribute der Joinbedingung entfernt werden:

$$\pi_{\overline{\mathsf{A}}}(\mathsf{R}_1 \ltimes_{\varphi} \mathsf{R}_2) \overset{1}{\rightleftharpoons} \pi_{\overline{\mathsf{A}}}(\mathsf{R}_1) \ltimes_{\varphi} \mathsf{R}_2$$

$$\pi_{\overline{\mathsf{A}}}(\mathsf{R}_1 \overline{\ltimes}_{\varphi} \mathsf{R}_2) \overset{1}{\rightleftharpoons} \pi_{\overline{\mathsf{A}}}(\mathsf{R}_1) \overline{\ltimes}_{\varphi} \mathsf{R}_2$$

1) falls $\mathsf{attr}(\varphi) \cap \mathbf{sch}(\mathsf{R}_1) \subseteq \overline{\mathsf{A}}$

N) Des Weiteren können Selektionen mit Mengenoperationen wie Vereinigung, Schnitt und Differenz vertauscht werden, also:

$$\sigma_{\varphi}(\mathsf{R}_1 \cup \mathsf{R}_2) \rightleftharpoons \sigma_{\varphi}(\mathsf{R}_1) \cup \sigma_{\varphi}(\mathsf{R}_2)$$
$$\sigma_{\varphi}(\mathsf{R}_1 \cap \mathsf{R}_2) \rightleftharpoons \sigma_{\varphi}(\mathsf{R}_1) \cap \sigma_{\varphi}(\mathsf{R}_2)$$
$$\sigma_{\varphi}(\mathsf{R}_1 - \mathsf{R}_2) \rightleftharpoons \sigma_{\varphi}(\mathsf{R}_1) - \sigma_{\varphi}(\mathsf{R}_2)$$

zusätzlich gilt:

$$\sigma_{\varphi}(\mathsf{R}_1 \cap \mathsf{R}_2) \rightleftharpoons \sigma_{\varphi}(\mathsf{R}_1) \cap \mathsf{R}_2$$

Analog reicht es bei der Differenz aus, die Tupel nur aus der ersten Relation herauszustreichen, so dass folgt:

$$\sigma_{\varphi}(\mathsf{R}_1 - \mathsf{R}_2) \rightleftharpoons \sigma_{\varphi}(\mathsf{R}_1) - \mathsf{R}_2$$

O) Jeder Projektionsoperator kann mit der Vereinigung vertauscht werden. Sei $\mathbf{sch}(\mathsf{R}_1) = \mathbf{sch}(\mathsf{R}_2)$, dann gilt:

$$\pi_{\overline{\mathsf{A}}}(\mathsf{R}_1 \cup \mathsf{R}_2) \rightleftharpoons \pi_{\overline{\mathsf{A}}}(\mathsf{R}_1) \cup \pi_{\overline{\mathsf{A}}}(\mathsf{R}_2)$$

Eine Vertauschung der Projektion mit Durchschnitt und Differenz ist allerdings im Allgemeinen falsch.

P) Eine Selektion und ein Kreuzprodukt können zu einem Join zusammengefasst werden, wenn die Selektionsbedingung eine Joinbedingung ist:

$$\sigma_{\varphi}(\mathsf{R}_1 \times \mathsf{R}_2) \overset{1}{\rightleftharpoons} \mathsf{R}_1 \bowtie_{\varphi} \mathsf{R}_2$$

1) falls φ Joinbedingung

Q) Ein Join mit anschließender Projektion auf die Attribute einer der Argumentrelationen kann als Semijoin dargestellt werden.

$$\pi_{\overline{A}}(R_1 \bowtie_\varphi R_2) \overset{1}{\rightleftharpoons} \pi_{\overline{A}}(R_1 \ltimes_\varphi R_2)$$

1) falls $\overline{A} \subseteq \mathbf{sch}(R_1)$

R) Der Antisemijoin ist äquivalent zur Differenz aus erster Relation und Semijoin der Relationen.

$$R_1 - (R_1 \ltimes_\varphi R_2) \rightleftharpoons R_1 \overline{\ltimes}_\varphi R_2$$

Bei Semi- oder Antisemijoin darf eine Projektion auf die zweite Argumentrelation vor der Verbundbildung durchgeführt werden, sofern man keine Attribute der Joinbedingung entfernt:

$$R_1 \ltimes_\varphi R_2 \rightleftharpoons R_1 \ltimes_\varphi \pi_{\overline{A}}(R_2)$$
$$R_1 \overline{\ltimes}_\varphi R_2 \rightleftharpoons R_1 \overline{\ltimes}_\varphi \pi_{\overline{A}}(R_2)$$

1) falls $\overline{A} \supseteq \mathsf{attr}(\varphi) \cap \mathbf{sch}(R_2)$

S) Differenz und Durchschnitt lassen sich als Semi- bzw. Antisemijoin mit Vergleich auf allen Attributen darstellen.

$$R_1 - R_2 \overset{1}{\underset{1,2}{\rightleftharpoons}} R_1 \overline{\ltimes}_\varphi R_2$$
$$R_1 \cap R_2 \overset{1}{\underset{1,2}{\rightleftharpoons}} R_1 \ltimes_\varphi R_2$$

1) falls $\varphi \equiv \bigwedge_{A \in \mathbf{sch}(R_1)} (R_1.A = R_2.A)$

2) falls $\mathbf{sch}(R_1) = \mathbf{sch}(R_2)$.

T) Eine Gruppierung kann an einem Equi-Join über Fremdschlüsselbeziehungen vorbeigeschoben werden. Dafür dürfen keine Attribute, über die in der Anfrage aggregiert wird, in der Verbundbedingung vorkommen. Dagegen muss jedes Verbundattribut auch Gruppierungsattribut sein, da sonst der Join nicht mehr möglich ist[2]. Des Weiteren müssen alle Gruppierungs- sowie Aggregationsattribute aus R_1 stammen. Aus diesem Grund wird im Gegensatz zu den Angaben in der Literatur der Join durch einen Semijoin ersetzt, da ansonsten die Schemata der Ergebnisrelationen nicht identisch wären:

$$\Gamma_{\overline{G}\#\overline{F}}(R_1 \bowtie_\varphi R_2) \overset{1}{\underset{2}{\rightleftharpoons}} \Gamma_{\overline{G}\#\overline{F}}(R_1) \ltimes_\varphi R_2$$

1) falls $\mathsf{attr}(\varphi) \cap \mathbf{sch}(R_1) \subseteq (\overline{G} \cap \overline{F})$ und
$\overline{G} \cup \mathsf{attr}(\mathsf{func}(\overline{F})) \subseteq \mathbf{sch}(R_1)$ und
$\mathsf{attr}(\varphi) \cap \mathbf{sch}(R_2) =$ Primärschlüssel von R_2 und
$\mathsf{op}(\varphi) \in \{=\}$

2) falls $\mathsf{attr}(\varphi) \subseteq \mathbf{sch}(R_1) \cup \mathbf{sch}(R_2)$

U) Falls es nicht möglich ist, die Gruppierung an einem Equi-Join vollständig vorbeizuschieben, dann kann der Versuch unternommen werden, sie in eine Haupt- ($\overline{G}$) und Vorgruppierung ($\overline{G_1}$) zu zerlegen. Verglichen mit den in Regel T) angenommenen Bedingungen wird dabei die Forderung aufgegeben, dass alle aggregierten Attribute unterhalb des Joins

[2]Die hier genannten Gruppierungsregeln sind mit Hilfe von [12],[74],[85],[91],[90],[92] entstanden. In diesen Veröffentlichungen werden aber keine expliziten Aussagen getroffen, sondern nur sprachliche Methoden beschrieben. Daher sind insbesondere die Bedingungen für die hier vorgestellten Regeln selbst entwickelt worden.

vorhanden sein müssen. In diesen Fall muss dann jedoch das Verbundattribut in die Vorgruppierung mit einbezogen werden und die Aggregationsfunktionen müssen gesondert betrachtet werden.

$$\Gamma_{\overline{G}\#\overline{F}}(R_1 \bowtie_\varphi R_2) \overset{1}{\underset{2}{\rightleftharpoons}} \Gamma_{\overline{G}\#\overline{F}}(\Gamma_{\overline{G}_1\#\overline{F}}(R_1) \bowtie_\varphi R_2) \text{ mit } \overline{G}_1 := (\overline{G} \cup \mathsf{attr}(\varphi)) \cap \mathbf{sch}(R_1)$$

1) falls $\mathsf{attr}(\mathsf{func}(\overline{F})) \subseteq \mathbf{sch}(R_1)$ und
$\mathsf{attr}(\mathsf{func}(\overline{F})) \cap (\overline{G} \cup \mathsf{attr}(\varphi) \cap \mathbf{sch}(R_1)) = \emptyset$ und
$\mathsf{attr}(\varphi) \cap \mathbf{sch}(R_1) \subseteq \overline{F}$ und
$\mathsf{op}(\varphi) \in \{=\}$
2) falls $\overline{G} \subseteq \mathbf{sch}(R_1) \cup \mathbf{sch}(R_2)$ und
$\mathsf{attr}(\varphi) \subseteq \mathbf{sch}(R_1) \cup \mathbf{sch}(R_2)$

für $\mathsf{func}(\overline{F}) \in \{\mathsf{min}(), \mathsf{max}(), \mathsf{sum}()\}$.

Für $\mathsf{func}(\overline{F}) \in \{\mathsf{avg}()\}$ gilt zusätzlich noch :

$$\Gamma_{\overline{G}\#\overline{F}}(R_1 \bowtie_\varphi R_2) \overset{1}{\underset{2}{\rightleftharpoons}} \Gamma_{\overline{G}\#\overline{F}_2}(\Gamma_{\overline{G}_1\#\overline{F}_1}(R_1) \bowtie_\varphi R_2) \text{ mit } \overline{G}_1 := (\overline{G} \cup \mathsf{attr}(\varphi)) \cap \mathbf{sch}(R_1)$$

1) falls $\mathsf{attr}(\mathsf{func}(\overline{F})) \subseteq \mathbf{sch}(R_1)$ und
$\mathsf{attr}(\mathsf{func}(\overline{F})) \cap (\overline{G} \cup \mathsf{attr}(\varphi) \cap R_1) = \emptyset$ und
$\mathsf{attr}(\varphi) \cap \mathbf{sch}(R_1) \subseteq \overline{F}$ und
$\mathsf{op}(\varphi) \in \{=\}$
2) falls $\overline{G} \subseteq \mathbf{sch}(R_1) \cup \mathbf{sch}(R_2)$ und
$\mathsf{attr}(\varphi) \subseteq \mathbf{sch}(R_1) \cup \mathbf{sch}(R_2)$

$$\mathsf{func}(\overline{F}_1) := \{\#2 := \mathsf{avg},\ \#1 := \mathsf{count}\}$$

$$\mathsf{func}(\overline{F}_2) := \left\{\tfrac{\mathsf{sum}(\#2 \cdot \#1)}{\mathsf{sum}(\#1)}\right\}$$

Für $\mathsf{func}(\overline{F}) \in \{\mathsf{count}()\}$ verändert sich dagegen die Teilaggregation zu:

$$\mathsf{func}(\overline{F}_1) := \{\#1 := \mathsf{count}\}$$

$$\mathsf{func}(\overline{F}_2) := \{\mathsf{sum}(\#1)\}$$

V) Bei der Verschiebung einer Gruppierung an einer Selektion vorbei müssen die Attribute der Selektion eine Teilmenge der Attribute der Gruppierung sein:

$$\Gamma_{\overline{G}\#\overline{F}}(\sigma_\varphi(R)) \overset{1}{\underset{2}{\rightleftharpoons}} \sigma_\varphi(\Gamma_{\overline{G}\#\overline{F}}(R))$$

1) falls $\mathsf{attr}(\varphi) \subseteq \overline{F}$
2) falls $\mathsf{attr}(\varphi) \subseteq \mathbf{sch}(R)$

W) Ein Join mit anschließender Aggregation kann als Semijoin dargestellt werden, sofern sich sowohl die Gruppierungs- als auch die Aggregationsattribute nur auf eine der Argumentrelationen beziehen.

$$\Gamma_{\overline{G}\#\overline{F}}(R_1 \bowtie_\varphi R_2) \overset{1,2}{\underset{2}{\rightleftharpoons}} \Gamma_{\overline{G}\#\overline{F}}(R_1 \ltimes_\varphi R_2)$$

1) falls $\overline{G} \cup \mathsf{attr}(\mathsf{func}(\overline{F})) \subseteq \mathbf{sch}(R_1)$
2) falls $\mathsf{attr}(\varphi) \cap \mathbf{sch}(R_2) =$ Primärschlüssel von R_2 und
$\mathsf{op}(\varphi) \in \{=\}$

3.2.2 Erzeugungsregeln für Sortierungen und Indexe

Folgende Regeln ermöglichen das Einfügen von Operatoren zur Sortierung und Indexbildung:

X) Jede Relation R kann nach einem Attribut $A \in \mathbf{sch}(R)$ sortiert werden:

$$R \underset{1}{\rightleftharpoons} \mathsf{SORT}_A(R)$$

1) falls $\neg\, \mathsf{supports_ordered_access}(R, A)$

Y) Jedes Attribut A kann in einer Relation R indexiert werden:

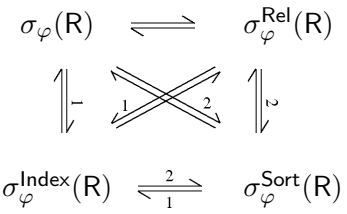
$$R \underset{1}{\rightleftharpoons} \mathsf{INDEX}_A(R)$$

1) falls $\neg\, \mathsf{supports_index_access}(R, A)$

3.2.3 Implementierungsabhängige relationale Ersetzungsregeln

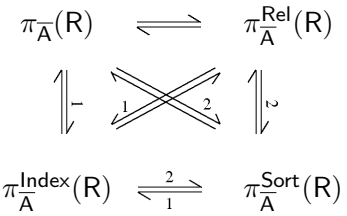
Folgende Regeln geben die verschiedenen Implementierungsarten der algebraischen Operatoren wieder. Dabei kann jeder Operator durch jede Implementierungsart zu einem physischen Operator, und diese untereinander, ersetzt werden, falls die angegebenen Bedingungen erfüllt sind.

(a) Selektionsvarianten:



1) falls $\mathsf{supports_index_access}\,(R, \varphi)$
2) falls $\mathsf{supports_ordered_access}(R, \varphi)$

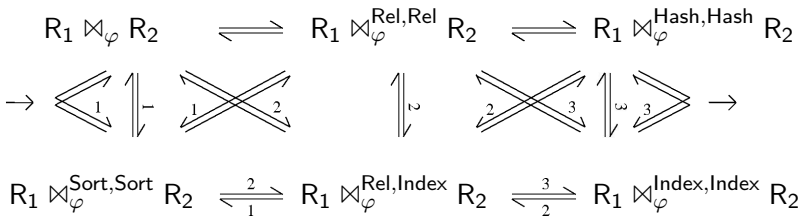
(b) Projektionsvarianten:



1) falls $\mathsf{supports_index_access}(R, \overline{A})$
2) falls $\mathsf{supports_ordered_access}(R, \overline{A})$

(c) Joinvarianten:

Um Platz zu sparen, werden die Implementierungsvarianten des Verbundes als Torus dargestellt. Dabei entspricht $\bowtie^{\mathsf{Rel,Index}}$ einem $\bowtie^{\mathsf{Index,Rel}}$ mit vertauschten Argumenten.



1) falls $\mathsf{supports_ordered_access}(\mathsf{R}_1, \varphi) \wedge$
$\mathsf{supports_ordered_access}(\mathsf{R}_2, \varphi)$
2) falls $\mathsf{supports_index_access}(\mathsf{R}_2, \varphi)$
3) falls $\mathsf{supports_index_access}(\mathsf{R}_1, \varphi) \wedge$
$\mathsf{supports_index_access}(\mathsf{R}_2, \varphi)$

Für alle weiteren relationalen Operatoren können die Regeln analog abhängig von den möglichen Implementierungsarten aufgestellt werden.

3.3 Generalisierte Regeln

Bisherige Bedingungen für Termersetzungsregeln bieten eine Vielzahl von Möglichkeiten, differenzierte Regeln zu erstellen und Kriterien aufzubauen, die eine Verbesserung des Argumentterm es erlauben. Da die Regeln zusätzlich unabhängig von Implementierungsarten der einzelnen Operatoren sind, nutzen regelbasierte Optimierer den Vorteil, dass ihre Anwendung durch eine begrenzte Grundmenge an Regeln kontrollierbar ist (siehe z. B. die 23 Regeln aus Abschnitt 3.2.1).

Dieser Vorteil birgt andererseits die Gefahr, Optimierungsmöglichkeiten zu beschränken. Beispielsweise könnte das Erzeugen einer einzelnen Sortierung dazu führen, dass mehrere Joinoperatoren diese nutzen und somit die Gesamtkosten dieser Variante des Ausführungsplanes das Optimum bilden. Um auch solche Fälle überhaupt berücksichtigen zu können, müssen zwei Erweiterungen zum bisherigen algebraischen Termersetzungsprozess hinzugefügt werden.

(a) Erweiterung der algebraischen Regeln um physische Implementierungsarten:

Definition 3.3.1 Physische Termersetzungsregeln ergeben sich aus der Kombination algebraischer Regeln (siehe Abschnitt 3.2.1) mit allen dazugehörigen möglichen Implementierungsarten für die vorkommenden relationalen Operatoren (siehe Abschnitt 3.2.3).

Beispielsweise würde man aus der Regel H, die Vertauschung von Selektionen behandelt, folgende Varianten gewinnen:

H1)

$$\sigma^{\mathsf{Rel}}_{\varphi_1}\left(\sigma^{\mathsf{Rel}}_{\varphi_2}(\mathsf{R})\right) \rightleftharpoons \sigma^{\mathsf{Rel}}_{\varphi_2}\left(\sigma^{\mathsf{Rel}}_{\varphi_1}(\mathsf{R})\right)$$

H2)

$$\sigma^{\mathsf{Rel}}_{\varphi_1}\left(\sigma^{\mathsf{Rel}}_{\varphi_2}(\mathsf{R})\right) \overset{1}{\rightleftharpoons} \sigma^{\mathsf{Rel}}_{\varphi_2}\left(\sigma^{\mathsf{Sort}}_{\varphi_1}(\mathsf{R})\right)$$

1) falls $\mathsf{supports_ordered_access}\,(\mathsf{R}, \varphi_1)$

H3)

$$\sigma^{\mathsf{Rel}}_{\varphi_1}\left(\sigma^{\mathsf{Rel}}_{\varphi_2}(\mathsf{R})\right) \overset{1}{\rightleftharpoons} \sigma^{\mathsf{Rel}}_{\varphi_2}\left(\sigma^{\mathsf{Index}}_{\varphi_1}(\mathsf{R})\right)$$

1) falls $\mathsf{supports_index_access}(\mathsf{R}, \varphi_1)$

Natürlich gewinnen wir, wie in den Regeln H2 und H3 sichtbar, auch zusätzliche Bedingungen, welche die Termersetzungen begrenzen und ein mögliches exponentielles Wachstum ihrer Anzahl verhindern (siehe Abschnitt 1.3). Aber das alleine reicht nicht aus, da wir zusätzlich noch Regeln zum Erzeugen von Indexen und Sortierungen (siehe Abschnitt 3.2.2)

verwenden müssen, um beim obigen Beispiel die Joins ersetzen zu können. Somit heben die index- bzw. sortierungserzeugenden Regeln X und Y auch das Fehlen von speziellen Zugriffsmöglichkeiten auf. Mit ($\mathsf{attr}(\varphi_1) = \mathsf{A}$) ergibt sich:

H4)
$$\sigma^{\mathsf{Rel}}_{\varphi_1}\left(\sigma^{\mathsf{Rel}}_{\varphi_2}(\mathsf{R})\right) \rightleftharpoons \sigma^{\mathsf{Rel}}_{\varphi_2}\left(\sigma^{\mathsf{Sort}}_{\varphi_1}(\mathsf{SORT}_{\mathsf{A}}(\mathsf{R}))\right)$$

H5)
$$\sigma^{\mathsf{Rel}}_{\varphi_1}\left(\sigma^{\mathsf{Rel}}_{\varphi_2}(\mathsf{R})\right) \rightleftharpoons \sigma^{\mathsf{Rel}}_{\varphi_2}\left(\sigma^{\mathsf{Index}}_{\varphi_1}(\mathsf{INDEX}_{\mathsf{A}}(\mathsf{R}))\right)$$

Wir erhalten aus der Kombination der physischen Regeln und den erzeugenden eine neue Grundmenge von Optimierungsregeln. Beispielsweise ergaben sich beim in der Einleitung erwähnten regelbasierten Optimierer COKO-KOLA [13] 600 physische Regeln, welche im Zuge der Optimierung verwendet wurden. Das bedeutet aber einen Mehraufwand von fast 30-mal mehr Regeln, welche in jedem Optimierungsschritt überprüft werden müssen.

Ein Versuch, den Aufwand zu reduzieren, wäre die Beibehaltung von Implementierungen, wenn eine Termersetzung durchgeführt wird und die dabei verwendete Regel keine Angaben zu der Implementierung der vorhandenen Operatoren macht. Jedoch ergeben sich daraus Probleme bei der Wahl des physischen Operators, wenn z. B. eine Joinumordnung vorgenommen wird (siehe Regel G). In solchen Fällen ergeben sich keine eindeutigen Joinimplementierungen, wenn alte Joinanordnungen nicht mehr existieren, da sie durch neue ersetzt worden sind.

Ein weiterer Ansatz zur Aufwandsreduzierung besteht darin, Regelabläufe heuristisch vorzunehmen. Beispielsweise kann eine aus der algebraischen Optimierung bekannte Strategie verwendet werden, die mit Hilfe der Regel L) gegebene Selektionen immer in einem Baum nach „unten drückt“ („Push-Down Selections“ Strategie). Dies übte man aber unabhängig davon aus, ob es sich im konkreten Fall wirklich lohnte oder nicht.

(b) Um auch solche Strategien im Zuge der Anfrageoptimierung bewerten zu können, sollten physische Termersetzungsregeln um Kostenbedingungen erweitert werden.

Definition 3.3.2 Physische Regeln, die um Kostenbedingungen bereichert werden, ergeben die generalisierten Regeln.

Mögliche Kostenbedingungen sind z. B. Kardinalitäten, Seitengrößen, Selektivitäten oder Attributwertanzahlen der Argumentrelationen:

H1)
$$\sigma^{\mathsf{Rel}}_{\varphi_1}\left(\sigma^{\mathsf{Rel}}_{\varphi_2}(\mathsf{R})\right) \overset{1}{\underset{2}{\rightleftharpoons}} \sigma^{\mathsf{Rel}}_{\varphi_2}\left(\sigma^{\mathsf{Rel}}_{\varphi_1}(\mathsf{R})\right),$$
$$1)\ \text{falls } \mathsf{sel}(\varphi_1, \mathsf{R}) \leq \mathsf{sel}(\varphi_2, \mathsf{R})$$
$$2)\ \text{falls } \mathsf{sel}(\varphi_2, \mathsf{R}) \leq \mathsf{sel}(\varphi_1, \mathsf{R})$$

Zu den aufwendigeren Kostenbedingungen gehört die Abschätzung der Kosten des Originalausführungsplanes (τ_1) im Verhältnis zu dem mit Hilfe der Ersetzungsregel transformierten neuen Ausführungsplan (τ_2). Daraus folgt unter der Verwendung eines Kostenmodells (siehe Kapitel 2), dass die Anwendung der Regel H3 von rechts nach links komplexer kostenbedingt eingeschränkt werden kann:

H3)
$$\tau_1 := \sigma^{\mathsf{Rel}}_{\varphi_1}\left(\sigma^{\mathsf{Rel}}_{\varphi_2}(\mathsf{R})\right) \overset{1}{\underset{2}{\rightleftharpoons}} \sigma^{\mathsf{Rel}}_{\varphi_2}\left(\sigma^{\mathsf{Index}}_{\varphi_1}(\mathsf{R})\right) =: \tau_2$$

1) falls $\mathsf{supports_index_access}(\mathsf{R}, \varphi_1)$
2) falls $\mathsf{cost}(\tau_1) \leq \mathsf{cost}(\tau_2)$

Abschließend sei erwähnt, dass alle vorgestellten Arten von Regeln – algebraische, physische und generalisierte – als generalisierte Bäume abgespeichert werden. Dabei unterscheiden sie sich hauptsächlich in den dazugehörigen mitgeführten Attributen. Rein technisch gesehen bieten sie definitionsbedingt unterschiedliche Anwendbarkeiten hinsichtlich der aufgeführten Operatorbäume: algebraische Regeln agieren auf Anfragebäumen, phyische und generalisierte auf Ausführungsplänen.

3.4 Erweiterung der Algebraischen Optimierung um kostenbedingte Termersetzungen

Nachdem die Basis für einen termersetzungsbasierten Optimierer in den letzten Abschnitten gelegt worden ist, folgen Betrachtungen zur Nutzung von Kostenbedingungen, um die Richtung einer Termersetzung zu präferieren. Hierzu werden Funktionen benötigt, die sich als Entscheidungsindikatoren nutzen lassen. Dabei ist die Reduzierung von redundanten Rechnungen während des Vergleiches zweier Varianten wünschenswert.

3.4.1 Einführendes Beispiel zur kostenbedingten Termersetzung

Gegeben seien zwei Relationen R_1 und R_2, sowie zwei darauf agierende Selektionen (σ_{φ_1} auf R_2 und σ_{φ_2} auf $\mathsf{R}_1 \times \mathsf{R}_2$). Die Joinbedingung ($\sigma_{\varphi_2}$) soll in diesem Fall so stark selektierend sein, dass es sich nicht rechnen würde, erst die Ausgangsrelationen durch die zugehörigen Selektionen einzuschränken und danach zu joinen (siehe Abbildung 3.1).[3]

Dazu seien folgende Statistiken gegeben:

Statistiken		
$\mathsf{sel}(\varphi_1, \mathsf{R}_2)$	=	$0,999$
$\mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2)$	=	$0,0001$
$\lvert\mathsf{R}_1\rvert$	=	10
$\mathsf{tlg}(\mathsf{R}_1)$	=	1
$\lvert\mathsf{R}_2\rvert$	=	100.000
$\mathsf{tlg}(\mathsf{R}_2)$	=	1
w_1	=	$0,02$
w_2	=	0

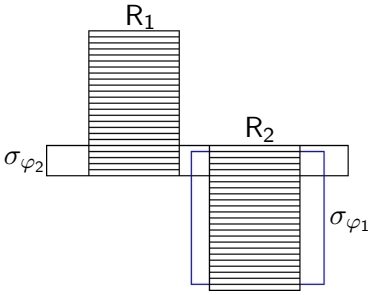


Abbildung 3.1: Beispiel für einen stark selektierenden Verbund

Gegeben sei die bekannte Selektionsverschiebungsregel für Verbunde aus Abschnitt 3.2:

L-Regel

$$\sigma_{\varphi_1}(\mathsf{R}_1 \bowtie_{\varphi_2} \mathsf{R}_2) \overset{1}{\rightleftharpoons} \mathsf{R}_1 \bowtie_{\varphi_2} \sigma_{\varphi_1}(\mathsf{R}_2)\,,$$

1) falls $\mathsf{attr}(\varphi_1) \subseteq \mathbf{sch}(\mathsf{R}_2)$

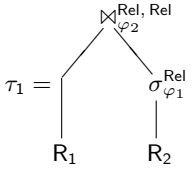
Heuristiken, die diese Regel anwenden, führen die Regel von links nach rechts aus und drücken somit die Selektion unterhalb des Verbundes. Aber gerade in unserem Beispiel ist diese Strategie nicht die richtige.

[3] Zur besseren Visualisierung liefert das Bild nicht maßstabsgetreu die Selektivitäten, die in der nachfolgenden Statistik angenommen werden.

Nachfolgend betrachten wir zwei mögliche Ausführungspläne τ_1 und τ_2 ohne gesonderte Betrachtung von möglichen Pipelinings. Dabei ist τ_1 ein Ausführungsplan mit heuristischer Optimierung durch die Anwendung der L-Regel und τ_2 ohne. Zusätzlich benutzen wir zur Kostenabschätzung das bekannte Kostenmodell aus Kapitel 2. Dabei gehen wir vom schlechtesten Fall aus und nutzen die Kostenformeln eines Nested-Loop-Joins und eines Relational-Scans als Selektionsoperatoren. Dementsprechend existieren auf den Relationen keine Indexe oder Sortierungen, die genutzt werden könnten. Darüber hinaus gehen wir von konstanten Selektivitäten aus, da sich durch die geringe Selektivität von φ_1 die Anzahl von Tupeln für den Join kaum verändert und somit auch nicht die Selektivität von φ_2. Schließlich soll das Beispiel auf einem lokalen Datenbanksystem laufen, so dass alle NET-Kosten gleich null sind und demzufolge nicht gesondert angegeben werden.

Die einzige Frage, die jetzt geklärt werden muss, lautet: Sind die Einsparungen durch die späte Basisselektion in τ_2 so hoch, dass sich der verteuerte Join insgesamt noch lohnt?

- 1. Heuristisch optimierter Ausführungsplan τ_1:

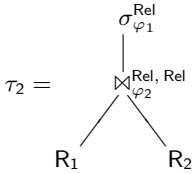
$\tau_1 =$ 

$$\begin{aligned}
\mathsf{cost}(\sigma^{\mathsf{Rel}}_{\varphi_1}(\mathsf{R}_2)) &= \mathsf{pg}(\mathsf{R}_2) + \mathsf{w}_1 \cdot |\mathsf{R}_2| = \mathsf{tlg}(\mathsf{R}_2) \cdot |\mathsf{R}_2| + \\
&\quad \mathsf{w}_1 \cdot |\mathsf{R}_2| \\
&= |\mathsf{R}_2| \cdot (\mathsf{tlg}(\mathsf{R}_2) + \mathsf{w}_1) \\
&= 100.000 \cdot (1 + 0,02) \\
&= 102.000
\end{aligned}$$

$$\begin{aligned}
\mathsf{cost}(\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \sigma_{\varphi_1}(\mathsf{R}_2)) &= \mathsf{pg}(\mathsf{R}_1) + |\mathsf{R}_1| \cdot \mathsf{pg}(\sigma_{\varphi_1}(\mathsf{R}_2)) + \mathsf{w}_1 \cdot (|\mathsf{R}_1| + |\mathsf{R}_1| \cdot |\sigma_{\varphi_1}(\mathsf{R}_2)|) \\
&= \mathsf{pg}(\mathsf{R}_1) + |\mathsf{R}_1| \cdot \mathsf{pg}(\mathsf{sel}(\varphi_1, \mathsf{R}_2) \cdot |\mathsf{R}_2|) + \mathsf{w}_1 \cdot (|\mathsf{R}_1| + |\mathsf{R}_1| \cdot \mathsf{sel}(\varphi_1, \mathsf{R}_2) \cdot |\mathsf{R}_2|) \\
&= |\mathsf{R}_1| \cdot \mathsf{tlg}(\mathsf{R}_1) + |\mathsf{R}_1| \cdot \mathsf{sel}(\varphi_1, \mathsf{R}_2) \cdot |\mathsf{R}_2| \cdot \mathsf{tlg}(\mathsf{R}_2) + \\
&\qquad \mathsf{w}_1 \cdot (|\mathsf{R}_1| + |\mathsf{R}_1| \cdot \mathsf{sel}(\varphi_1, \mathsf{R}_2) \cdot |\mathsf{R}_2|) \\
&= 10 \cdot 1 + 10 \cdot 0,999 \cdot 100.000 \cdot 1 + 0,02 \cdot (10 + 10 \cdot 0,0001 \cdot 100.000) \\
&= 999.012,2 \\
\Rightarrow \mathsf{cost}(\tau_1) &= 102.000 + 999.012,2 = 1.101.012,2
\end{aligned}$$

- 2. Nicht heuristisch optimierter Ausführungsplan τ_2:

$$\begin{aligned}
\mathsf{cost}(\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \mathsf{R}_2) &= \mathsf{pg}(\mathsf{R}_1) + |\mathsf{R}_1| \cdot \mathsf{pg}(\mathsf{R}_2) + \mathsf{w}_1 \cdot (|\mathsf{R}_1| + |\mathsf{R}_1| \cdot |\mathsf{R}_2|) \\
&= |\mathsf{R}_1| \cdot \mathsf{tlg}(\mathsf{R}_1) + |\mathsf{R}_1| \cdot |\mathsf{R}_2| \cdot \mathsf{tlg}(\mathsf{R}_2) + \\
&\quad \mathsf{w}_1 \cdot (|\mathsf{R}_1| + |\mathsf{R}_1| \cdot |\mathsf{R}_2|) \\
&= |\mathsf{R}_1| \cdot (|\mathsf{R}_2| + 1) \cdot \mathsf{w}_1 + |\mathsf{R}_1| \cdot (|\mathsf{R}_2| \cdot \mathsf{tlg}(\mathsf{R}_2) + \mathsf{tlg}(\mathsf{R}_1)) \\
&= 10 \cdot (100.000 + 1) \cdot 0,02 + 10 \cdot (100.000 \cdot 1 + 1) \\
&= 1.020.010,2
\end{aligned}$$

$\tau_2 =$ 

$$\begin{aligned}
\mathsf{cost}(\sigma^{\mathsf{Rel}}_{\varphi_1}(\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \mathsf{R}_2)) &= \mathsf{pg}(\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \mathsf{R}_2) + \mathsf{w}_1 \cdot |\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \mathsf{R}_2| \\
&= \mathsf{pg}(\mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \cdot |\mathsf{R}_1| \cdot |\mathsf{R}_2|) + \mathsf{w}_1 \cdot \mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \cdot |\mathsf{R}_1| \cdot |\mathsf{R}_2| \\
&= \mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \cdot |\mathsf{R}_1| \cdot |\mathsf{R}_2| \cdot \mathsf{tlg}(\mathsf{R}_1 \times \mathsf{R}_2) + \mathsf{w}_1 \cdot \mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \cdot |\mathsf{R}_1| \cdot |\mathsf{R}_2| \\
&= \mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \cdot |\mathsf{R}_1| \cdot |\mathsf{R}_2| \cdot (\mathsf{tlg}(\mathsf{R}_1) + \mathsf{tlg}(\mathsf{R}_2)) + \\
&\quad \mathsf{w}_1 \cdot \mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \cdot |\mathsf{R}_1| \cdot |\mathsf{R}_2| \\
&= 0,0001 \cdot 10 \cdot 100.000 \cdot (1 + 1) + 0,02 \cdot 0,0001 \cdot 10 \cdot 100.000 \\
&= 20.200 \\
\Rightarrow \mathsf{cost}(\tau_2) &= 1.020.010,2 + 202 = 1.020.212,2
\end{aligned}$$

Unter den zuvor angegebenen Einschränkungen ist τ_2 – ohne die Anwendung der Standardheuristik – ca. 7,3% billiger als τ_1. Somit können wir davon ausgehen, dass in diesem Fall $\sigma^{\mathsf{Rel}}_{\varphi_1}(\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \mathsf{R}_2)$ schneller zu berechnen ist als das von den Standardheuristiken empfohlene $\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \sigma^{\mathsf{Rel}}_{\varphi_1}(\mathsf{R}_2)$.

3.4.2 Herleitung einer kostenbedingten Termersetzungsregel

Um nicht die kompletten Kosten eines erzeugten Ausführungsplanes $\mathsf{cost}(\tau_2)$ mit denen des originalen Baumes $\mathsf{cost}(\tau_1)$ bei jeder Termersetzung ausrechnen und vergleichen zu müssen, wird im Folgenden vorgeschlagen, die Entscheidung, ob $\mathsf{cost}(\tau_1) \geq \mathsf{cost}(\tau_2)$ ist, mit Hilfe einer Indikatorfunktion zu vereinfachen. Dabei werden redundante Teile der Kostenberechnung eliminiert und nur die reine Präferierung, die benötigt wird, beibehalten. Aus diesem Grund vergleichen wir die Kostenformeln der beiden Ausführungspläne:

$$\begin{aligned}
\mathsf{cost}(\tau_1) &= |\mathsf{R}_2| \cdot (\mathsf{tlg}(\mathsf{R}_2) + \mathsf{w}_1) + \\
&\quad |\mathsf{R}_1| \cdot \mathsf{tlg}(\mathsf{R}_1) + |\mathsf{R}_1| \cdot \mathsf{sel}(\varphi_1, \mathsf{R}_2) \cdot |\mathsf{R}_2| \cdot \mathsf{tlg}(\mathsf{R}_2) + \mathsf{w}_1 \cdot (|\mathsf{R}_1| + |\mathsf{R}_1| \cdot \mathsf{sel}(\varphi_1, \mathsf{R}_2) \cdot |\mathsf{R}_2|) \\
\mathsf{cost}(\tau_2) &= |\mathsf{R}_1| \cdot (|\mathsf{R}_2| + 1) \cdot \mathsf{w}_1 \\
&\quad + |\mathsf{R}_1| \cdot (|\mathsf{R}_2| \cdot \mathsf{tlg}(\mathsf{R}_2) + \mathsf{tlg}(\mathsf{R}_1)) + \\
&\quad \mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \cdot |\mathsf{R}_1| \cdot |\mathsf{R}_2| \cdot (\mathsf{tlg}(\mathsf{R}_1) + \mathsf{tlg}(\mathsf{R}_2)) + \mathsf{w}_1 \cdot \mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \cdot |\mathsf{R}_1| \cdot |\mathsf{R}_2|
\end{aligned}$$

Da wir die Situation suchen, in der beide Ausführungspläne die gleichen Kosten besitzen, setzen wir die beiden Kostenfunktionen gleich:

$$\mathsf{cost}(\tau_1) = \mathsf{cost}(\tau_2)$$

Aufgelöst nach $\mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2)$ ergibt sich eine Grenzselektivität $\mathsf{g}^{\{\sigma^{\mathsf{Rel}},\, \bowtie^{\mathsf{Rel},\mathsf{Rel}}\}}_{\mathsf{sel}}(\mathsf{w}_1, \varphi_1, \mathsf{R}_1, \mathsf{R}_2)$:

Definition 3.4.1

$$\mathsf{g}^{\{\sigma^{\mathsf{Rel}},\, \bowtie^{\mathsf{Rel},\mathsf{Rel}}\}}_{\mathsf{sel}}(\mathsf{w}_1, \varphi_1, \mathsf{R}_1, \mathsf{R}_2) := \frac{(\mathsf{w}_1 + \mathsf{tlg}(\mathsf{R}_2)) \cdot (|\mathsf{R}_1| \cdot \mathsf{sel}(\varphi_1, \mathsf{R}_2) - |\mathsf{R}_1| + 1)}{|\mathsf{R}_1| \cdot (\mathsf{w}_1 + \mathsf{tlg}(\mathsf{R}_1) + \mathsf{tlg}(\mathsf{R}_2))}$$

Wir haben jetzt eine Ersetzungsregel, die bedingt angewandt werden kann:

Definition 3.4.2

L2-Regel

$$\sigma^{\mathsf{Rel}}_{\varphi_1}(\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \mathsf{R}_2) \underset{2}{\overset{1}{\rightleftharpoons}} \mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel}}_{\varphi_2} \sigma^{\mathsf{Rel}}_{\varphi_1}(\mathsf{R}_2)\,,$$

1) falls $\mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \geq \mathsf{g}^{\{\sigma^{\mathsf{Rel}},\, \bowtie^{\mathsf{Rel},\mathsf{Rel}}\}}_{\mathsf{sel}}(\mathsf{w}_1, \varphi_1, \mathsf{R}_1, \mathsf{R}_2)$

2) falls $\mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) \leq \mathsf{g}^{\{\sigma^{\mathsf{Rel}},\, \bowtie^{\mathsf{Rel},\mathsf{Rel}}\}}_{\mathsf{sel}}(\mathsf{w}_1, \varphi_1, \mathsf{R}_1, \mathsf{R}_2)$

In unserem Beispiel ergibt sich für den rechten Kostenterm:

$$\mathsf{g}^{\{\sigma^{\mathsf{Rel}},\, \bowtie^{\mathsf{Rel},\mathsf{Rel}}\}}_{\mathsf{sel}}(\mathsf{w}_1, \varphi_1, \mathsf{R}_1, \mathsf{R}_2) = \frac{(0,02 + 1) \cdot (10 \cdot 0,999 - 10 + 1)}{10 \cdot (0,02 + 1 + 1)} = \frac{5.049}{101.000} \approx 0,05$$

Mit $\mathsf{sel}(\varphi_2, \mathsf{R}_1 \times \mathsf{R}_2) = 0,0001$ folgt, da $0,0001 < 0,05$ ist, dass der richtige Ausführungsplan τ_2 sein sollte, da die bedingte Regel dazu rät, die Selektionen erst nach dem Join anzuwenden.

Wie stark die einzelnen Indikatorfunktionen von den zugrunde liegenden physischen Operatoren und den daraus resultierenden Kostenfunktionen abhängen, wird daran klar, dass sich beim obigen Beispiel die Aussage der Indikatorfunktion verändert, falls der Nested-Loop-Join durch einen Merge-Join ersetzt wird:

Definition 3.4.3

$$g_{\mathsf{sel}}^{\{\sigma^{\mathsf{Rel}},\bowtie^{\mathsf{Sort,Sort}}\}}(w_1,\varphi_1,R_1,R_2) := \frac{(w_1+\mathsf{tlg}(R_2))\cdot \mathsf{sel}(\varphi_1,R_2)}{|R_1|\cdot(w_1+\mathsf{tlg}(R_1)+\mathsf{tlg}(R_2))}$$

Somit folgt für unser Beispiel, dass die Selektivität von φ_2 um den Faktor 10 schwächer sein könnte, bis die Empfehlung für einen Ausführungsplan sich verändert:

$$g_{\mathsf{sel}}^{\{\sigma^{\mathsf{Rel}},\bowtie^{\mathsf{Sort,Sort}}\}}(w_1,\varphi_1,R_1,R_2) := \frac{(0,02+1)\cdot 0,999}{10\cdot(0,02+1+1)} = \frac{50.949}{101.000} \approx 0,5$$

Bei einer Hash-Join Variante werden zusätzlich die Attributwertanzahlen benötigt. Vorausgesetzt, dass auch diese konstant bleiben, ergibt sich:

Definition 3.4.4

$$g_{\mathsf{sel}}^{\{\sigma^{\mathsf{Rel}},\bowtie^{\mathsf{Hash,Hash}}\}}(w_1,\varphi_1,R_1,R_2) := \frac{w_1\cdot\mathsf{tlg}(R_2)\cdot[(\mathsf{n}(B,R_2)+|R_1|)\cdot\mathsf{sel}(\varphi_1,R_2)-|R_1|]}{\mathsf{n}(B,R_2)\cdot|R_1|\cdot(w_1+\mathsf{tlg}(R_1)+\mathsf{tlg}(R_2))}$$

Bei nicht konstanten Attributwertverteilungen lässt sich in dem Bruch kaum noch etwas kürzen, so dass der Vorteil der Redundanzminimierung der Grenzselektivität sich vermindert.

Interessant ist auch, dass im Gegensatz zu einem Input/Output-Kostenmodell wie in [59] die Fehleranfälligkeit durch das Fehlen der resultierenden Selektivität des Joins in unserem (Input-) Kostenmodell nicht mehr in Betracht gezogen werden muss, da sie in der Indikatorfunktion gar nicht vorkommt[4]. Grund dafür ist die Tatsache, dass die Kardinalität des Ergebnisses gleich ist und die daraus resultierenden Schreibkosten demzufolge auch identisch bleiben und sich deshalb beim Gleichsetzen gegenseitig aufheben.

Solche Indikatorfunktionen können für beliebige Kombinationen von Operatortypen berechnet werden und sind in allen Fällen durch die jeweilige Vereinfachung bei der Gleichsetzung schneller zu berechnen als die komplette Berechnung der Kosten jedes einzelnen Ausführungsplanes. Deshalb wird im Folgenden als weiteres Beispiel eine Indikatorfunktion für die Projektion hergeleitet.

3.4.3 Entwicklung einer bedingten Ersetzungsregel für die Projektion

Da eine Projektion ohne Duplikateliminierung keinen Einfluss auf die Anzahl von Tupeln hat, aber durch die Projektion Attribute wegfallen und somit mehr Tupel auf eine Seite gespeichert werden können, muss in diesem Fall ein Kostenmodell basierend auf Seiten verwendet werden (siehe Kapitel 2). Dementsprechend müssen auch hier für alle Implementierungsarten eigene Abschätzungen durchgeführt werden.

Erneut nutzen wir als Basis eine Termersetzungsregel aus Abschnitt 3.2:

M-Regel

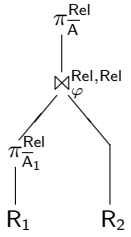
$$\pi_{\overline{A}}(R_1 \bowtie_\varphi R_2) \rightleftharpoons \pi_{\overline{A}}\left(\pi_{\overline{A}_1}(R_1) \bowtie_\varphi \pi_{\overline{A}_2}(R_2)\right)$$

$$\text{mit } \overline{A}_1 = \mathbf{sch}(R_1)\cap(\overline{A}\cup\mathsf{attr}(\varphi)) \wedge \\ \overline{A}_2 = \mathbf{sch}(R_2)\cap(\overline{A}\cup\mathsf{attr}(\varphi))$$

[4] Dies ist ein immenser Vorteil, denn obwohl eine Unabhängigkeit der Selektionen im Allgemeinen angenommen wird, kann sich die Selektivität durch das Fehlen der selektierten Tupel dennoch verändern.

Da jede Projektion einzeln betrachtet werden kann, werden wir nachfolgend nur eine Projektion $\pi_{\overline{A}_1}$ auf einer Relation R_1, ohne gegebene Sortierung, Index und Pipelining mit $\overline{R}_1 := \pi^{Rel}_{\overline{A}_1}(R_1)$ und $\overline{R}_3 := \overline{R}_1 \bowtie^{Rel,Rel}_{\varphi} R_2$ bzw. $R_3 := R_1 \bowtie^{Rel,Rel}_{\varphi} R_2$ untersuchen. Wiederum sei der Ausführungsplan τ_1 mit, τ_2 ohne heuristische Optimierung:

1. Heuristisch optimierter Ausführungsplan τ_1:[5]

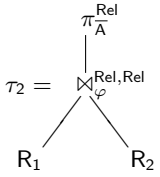
$\tau_1 =$ 

$$\begin{aligned}
\text{cost}(\pi^{Rel}_{\overline{A}_1}) &= pg(R_1) + w_1 \cdot |R_1| \\
&= tlg(R_1) \cdot |R_1| + w_1 \cdot |R_1| \\
&= (tlg(R_1) + w_1) \cdot |R_1| \\
\text{cost}(\overline{R}_1 \bowtie^{Rel,Rel}_{\varphi} R_2) &= pg(\overline{R}_1) + |\overline{R}_1| \cdot pg(R_2) + w_1 \cdot (|\overline{R}_1| + |\overline{R}_1| \cdot |R_2|) \\
&= tlg(\overline{R}_1) \cdot |\overline{R}_1| + |\overline{R}_1| \cdot tlg(R_2) \cdot |R_2| + w_1 \cdot (|\overline{R}_1| + |\overline{R}_1| \cdot |R_2|) \\
&= |R_1| \cdot (tlg(\overline{R}_1) + tlg(R_2) \cdot |R_2|) + w_1 \cdot (|R_1| + |R_1| \cdot |R_2|)
\end{aligned}$$

$$\begin{aligned}
\text{cost}(\pi^{Rel}_{\overline{A}}(\overline{R}_3)) &= pg(\overline{R}_3) + w_1 \cdot |\overline{R}_3| \\
&= tlg(\overline{R}_3) \cdot |\overline{R}_3| + w_1 \cdot |\overline{R}_3| \\
&= (tlg(\overline{R}_3) + w_1) \cdot sel(\varphi, \overline{R}_1 \times R_2) \cdot |\overline{R}_1| \cdot |R_2| \\
&= (tlg(\overline{R}_1) + tlg(R_2) + w_1) \cdot sel(\varphi, \overline{R}_1 \times R_2) \cdot |R_1| \cdot |R_2|
\end{aligned}$$

$$\begin{aligned}
\text{cost}\left(\pi^{Rel}_{\overline{A}}(\pi^{Rel}_{\overline{A}_1}(R_1) \bowtie^{Rel,Rel}_{\varphi} R_2)\right) = {} & (tlg(R_1) + w_1) \cdot |R_1| + |R_1| \cdot (tlg(\overline{R}_1) + tlg(R_2) \cdot |R_2|) + \\
& w_1 \cdot (|R_1| + |R_1| \cdot |R_2|) + (tlg(\overline{R}_1) + tlg(R_2) + w_1) \cdot sel(\varphi, \overline{R}_1 \times R_2) \cdot |R_1| \cdot |R_2|
\end{aligned}$$

2. Nicht heuristisch optimierter Ausführungsplan τ_2:

$\tau_2 =$ 

$$\begin{aligned}
\text{cost}(R_1 \bowtie^{Rel,Rel}_{\varphi} R_2) &= pg(R_1) + |R_1| \cdot pg(R_2) + w_1 \cdot (|R_1| + |R_1| \cdot |R_2|) \\
&= tlg(R_1) \cdot |R_1| + |R_1| \cdot tlg(R_2) \cdot |R_2| + w_1 \cdot (|R_1| + |R_1| \cdot |R_2|) \\
&= |R_1| \cdot (tlg(R_1) + tlg(R_2) \cdot |R_2|) + w_1 \cdot (|R_1| + |R_1| \cdot |R_2|) \\
\text{cost}(\pi^{Rel}_{\overline{A}}(R_3)) &= pg(R_3) + w_1 \cdot |R_3| \\
&= tlg(R_3) \cdot |R_3| + w_1 \cdot |R_3| \\
&= (tlg(R_1) + tlg(R_2) + w_1) \cdot sel(\varphi, R_1 \times R_2) \cdot |R_1| \cdot |R_2|
\end{aligned}$$

$$\begin{aligned}
\text{cost}\left(\pi^{Rel}_{\overline{A}}(R_1 \bowtie^{Rel,Rel}_{\varphi} R_2)\right) = {} & |R_1| \cdot (tlg(R_1) + tlg(R_2) \cdot |R_2|) + \\
& w_1 \cdot (|R_1| + |R_1| \cdot |R_2|) + (tlg(R_1) + tlg(R_2) + w_1) \cdot sel(\varphi, R_1 \times R_2) \cdot |R_1| \cdot |R_2|
\end{aligned}$$

Die interessante Frage lautet dann, wann folgende Situation eintrifft:

$$\text{cost}\left(\pi^{Rel}_{\overline{A}}(R_1 \bowtie^{Rel,Rel}_{\varphi} R_2)\right) \geq \text{cost}\left(\pi^{Rel}_{\overline{A}}(\pi^{Rel}_{\overline{A}_1}(R_1) \bowtie^{Rel,Rel}_{\varphi} R_2)\right)$$

Wiederum, aufgelöst nach $sel(\varphi, R_1 \times R_2)$, ergibt sich erneut eine Grenzselektivität $g^{\{\pi^{Rel}, \bowtie^{Rel,Rel}\}}_{sel}(w_1, \varphi, R_1, R_2)$, bei der beide Ausführungspläne die gleichen Kosten verursachen:

$$g^{\{\pi^{Rel}, \bowtie^{Rel,Rel}\}}_{sel}(w_1, \varphi, R_1, R_2) :=$$

$$\frac{w_1 \cdot (|R_2| \cdot sel(\varphi, \overline{R}_1 \times R_2) + 1) + |R_2| \cdot (tlg(R_2) + tlg(\overline{R}_1)) \cdot sel(\varphi, \overline{R}_1 \times R_2) + tlg(\overline{R}_1)}{|R_2| \cdot (w_1 + tlg(R_1) + tlg(R_2))}$$

[5] Da eine nicht duplikateliminierende Projektion betrachtet wird, kann $|R_1| = |\overline{R}_1|$ gesetzt werden.

Da in unserem Fall die Projektion keine Duplikate eliminiert, ändert sich an der Kardinalität der Argumentrelationen nichts, und somit auch nichts an der Joinselektivität. Mit $\mathsf{sel}(\varphi, \mathsf{R}_1 \times \mathsf{R}_2) = \mathsf{sel}(\varphi, \overline{\mathsf{R}}_1 \times \mathsf{R}_2)$ folgt:

Definition 3.4.5

$$\mathsf{g}_{\mathsf{sel}}^{\{\pi^{\mathsf{Rel}}, \bowtie^{\mathsf{Rel},\mathsf{Rel}}\}}(\mathsf{w}_1, \mathsf{R}_1, \mathsf{R}_2) := \frac{\mathsf{w}_1 + \mathsf{tlg}(\overline{\mathsf{R}}_1)}{|\mathsf{R}_2| \cdot (\mathsf{tlg}(\mathsf{R}_1) - \mathsf{tlg}(\overline{\mathsf{R}}_1))}$$

Bei der Gleichsetzung der Kostenformel für beliebige Joins und Anti-/Semijoins ergibt sich in jedem Fall die vorherige Indikatorfunktion[6]:

$$\mathsf{g}_{\mathsf{sel}}^{\{\pi^{\mathsf{Rel}}, \{\bowtie, \rhd\!\!<, \overline{\rhd\!\!<}\}\}}(\mathsf{w}_1, \mathsf{R}_1, \mathsf{R}_2) := \frac{\mathsf{w}_1 + \mathsf{tlg}(\overline{\mathsf{R}}_1)}{|\mathsf{R}_2| \cdot (\mathsf{tlg}(\mathsf{R}_1) - \mathsf{tlg}(\overline{\mathsf{R}}_1))}$$

Grund für die einfache Indikatorfunktion ist ein Entwurfsvorteil, der durch die Wahl der Kostenformeln im 2. Kapitel gegeben ist: Die erste Argumentrelation wird bei allen Join- bzw. Anti-/Semijoinvarianten (siehe Tabellen 2.9 und 2.10) nur einmal per Seitenzugriff gelesen, und nur dort werden Tupellängen mit berücksichtigt.

Denselben Ansatz können wir für duplikateliminierende Projektionen nutzen. Durch das Wissen, dass sich die Kardinalität der ersten Argumentrelation $|\mathsf{R}_1|$ mit $\mathsf{n}(\overline{\mathsf{A}_1}, \mathsf{R})$ abschätzen lässt und unter der Annahme, dass sich die Selektivität des Joins $\mathsf{sel}(\varphi_3, \mathsf{R}_1 \times \mathsf{R}_2)$ nicht verändert, kann man auch hier eine Indikatorfunktion angeben:

Definition 3.4.6

$$\mathsf{g}_{\mathsf{sel}}^{\{\pi^{\mathsf{Rel/elim}}, \{\bowtie, \rhd\!\!<, \overline{\rhd\!\!<}\}\}}(\mathsf{w}_1, \mathsf{R}_1, \mathsf{R}_2) :=$$

$$[-((2 \cdot \mathsf{n}(\overline{\mathsf{A}_1}, \mathsf{R}_1) \cdot (|\mathsf{R}_2| + 1) + |\mathsf{R}_1| \cdot (|\mathsf{R}_1| - 2 \cdot |\mathsf{R}_2|)) \cdot \mathsf{w}_1 +$$

$$\frac{2 \cdot \mathsf{n}(\overline{\mathsf{A}_1}, \mathsf{R}_1) \cdot (|\mathsf{R}_2| \cdot \mathsf{tlg}(\mathsf{R}_2) + \mathsf{tlg}(\overline{\mathsf{R}}_1)) + |\mathsf{R}_1| \cdot (|\mathsf{R}_1| \cdot \mathsf{tlg}(\mathsf{R}_1) - 2 \cdot |\mathsf{R}_2| \cdot \mathsf{tlg}(\mathsf{R}_2))]}{2 \cdot (\mathsf{n}(\overline{\mathsf{A}_1}, \mathsf{R}_1) \cdot (\mathsf{tlg}(\mathsf{R}_2) + \mathsf{tlg}(\overline{\mathsf{R}}_1)) - |\mathsf{R}_1| \cdot (\mathsf{tlg}(\mathsf{R}_1) + \mathsf{tlg}(\mathsf{R}_2)) \cdot |\mathsf{R}_2|}$$

In diesem Fall ergibt sich durch die Annahmen der Unabhängigkeit der Selektivitäten allerdings eine leicht pessimistischere Indikatorfunktion, die in selteneren Fällen die Projektionen verschiebt, da sich die Einsparungen durch geringere Tupelanzahlen der ersten Argumentrelation und des Joinergebnisses nicht auswirken. Grund hierfür ist das zusätzliche Wissen, dass $\mathsf{sel}(\varphi, \overline{\mathsf{R}}_1 \times \mathsf{R}_2) \leq \mathsf{sel}(\varphi, \mathsf{R}_1 \times \mathsf{R}_2)$ ist. Falls man diese Annahmen nicht tätigt, so verringert sich die Berechnungskosteneinsparung für die Präferierung von Ausführungsplänen durch die Nutzung der Indikatorfunktion, und man berechnet demzufolge die Kosten der zwei Bäume fast vollständig.

Aus den oben geführten Berechnungen ergibt sich die bedingte Termersetzungsregel:

Definition 3.4.7

M2-Regel

$$\pi_{\overline{\mathsf{A}}}(\mathsf{R}_1 \bowtie_\varphi \mathsf{R}_2) \underset{2}{\overset{1}{\rightleftharpoons}} \pi_{\overline{\mathsf{A}}}\left(\pi_{\overline{\mathsf{A}}_1}(\mathsf{R}_1) \bowtie_\varphi \mathsf{R}_2\right)$$

$$\text{mit } \overline{\mathsf{A}}_1 = \mathbf{sch}(\mathsf{R}_1) \cap (\overline{\mathsf{A}} \cup \mathsf{attr}(\varphi)) \wedge$$

$$1) \text{ falls } \mathsf{sel}(\varphi, \mathsf{R}_1 \times \mathsf{R}_2) \geq \mathsf{g}_{\mathsf{sel}}^{\{\pi^{\mathsf{Rel}\{/\mathsf{elim}\}}, \{\bowtie, \rhd\!\!<, \overline{\rhd\!\!<}\}\}}(\mathsf{w}_1, \mathsf{R}_1, \mathsf{R}_2)$$

[6] Das gilt nicht für die Index-Index-Varianten, denn in diesen Fällen verteuert die Verschiebung die Kosten immer, da sie die Indexbenutzung verhindert.

$$2) \text{ falls } \mathsf{sel}(\varphi, \mathsf{R}_1 \times \mathsf{R}_2) \leq \mathsf{g}_{\mathsf{sel}}^{\{\pi^{\mathsf{Rel}\{/\mathsf{elim}\}}, \{\bowtie, \rtimes, \overline{\rtimes}\}\}}(\mathsf{w}_1, \mathsf{R}_1, \mathsf{R}_2)$$

Unter der Annahme, dass sich die Tupellänge von R_1 durch eine Projektion halbiert, ergibt sich bei unserem anfänglichen Beispiel 3.1 im nicht duplikateliminierenden Fall eine Grenzselektivität von:

$$\mathsf{g}_{\mathsf{sel}}^{\{\pi^{\mathsf{Rel}}, \{\bowtie, \rtimes, \overline{\rtimes}\}\}}(\mathsf{w}_1, \mathsf{R}_1, \mathsf{R}_2) := \frac{0,02 + \frac{1}{2}}{10^5 \cdot (1 - \frac{1}{2})} = 1,04 \cdot 10^{-5}$$

Mit $\mathsf{sel}(\varphi, \mathsf{R}_1 \times \mathsf{R}_2) = 0,0001$ folgt, da $10^{-4} > 1,04 \cdot 10^{-5}$ ist, dass der richtige Ausführungsplan τ_1 ist, da die bedingte Regel dazu rät, auch eine Teilprojektion vor dem Join anzuwenden.

Im duplikateliminierenden Fall dreht sich die Aussage um und es wird τ_2 empfohlen, wenn wir davon ausgehen, dass $\mathsf{n}(\overline{\mathsf{A}}_1, \mathsf{R}_1) = |\mathsf{R}_1|$ ist:

$$\mathsf{g}_{\mathsf{sel}}^{\{\pi^{\mathsf{Rel}/\mathsf{elim}}, \{\bowtie, \rtimes, \overline{\rtimes}\}\}}(\mathsf{w}_1, \mathsf{R}_1, \mathsf{R}_2) :=$$

$$\frac{[-((2 \cdot 10 \cdot (10^5 + 1) + 10 \cdot (10 - 2 \cdot 10^5)) \cdot 0,02 + 2 \cdot 10 \cdot (10^5 \cdot 1 + \frac{1}{2}) + 10 \cdot (10 \cdot 1 - 2 \cdot 10^5 \cdot 1)]}{2 \cdot (10 \cdot (1 + \frac{1}{2}) - 10 \cdot (1 + 1) \cdot 10^5} = 1,124 \cdot 10^{-4} > 10^{-4}$$

Die methodische Quintessenz lautet, dass sich der Aufwand, kostenbedingte Termersetzungsregeln herzuleiten, generell lohnt. Zwar müssen die Indikatorfunktion für jede Regel und deren Implementierungsvarianten zunächst einzeln berechnet werden, jedoch spart man immer Berechnungskosten bei jeder Ersetzung.

Besonders lohnt sich die Nutzung von Kostenbedingungen innerhalb von *„Super-Regeln“*:

Definition 3.4.8 „Super-Regeln“ sind Algorithmen, welche die „normale“-Termersetzung durchbrechen und eine Gruppe von Termersetzungen strategisch zusammen ausführen. Dabei werden alle vorhandenen Statistiken bzw. Kosteninformationen verwendet, um alle benötigten Operatoren neu anzuordnen.

Zu dieser Regelart gehört z. B. das Join-Ordering mit Hilfe der Dynamischen Programmierung, das Relationen und die darauf agierenden Selektionen unterstützt durch Statistikinformationen anordnet. „Super-Regeln“ bieten durch die flexible Anordnung von Operatoren ein hohes Maß an Kosteneinsparungen. Algorithmisch bedingt ist der Berechnungsaufwand für solche Regeln überproportional hoch, d. h. sie sollten entweder ausschließlich genutzt werden oder komplett durch simplere Regelanordnungen ersetzt werden. Nachfolgend sei ein Beispiel für eine „Super-Regel“ gegeben.

3.5 Erweiterung des Join-Ordering-Algorithmus um Selektionen

Im Folgenden zeigen wir, wie der Join-Ordering-Algorithmus von Vance und Maier [86] unter Verwendung der L2-Regel verbessert werden kann, um dann als eine einzelne Join-Ordering Regel betrachtet zu werden. Dazu beschreiben wir zunächst die Arbeitsumgebung:

Zu optimieren ist ein Term bestehend aus natürlichen Joins und Selektionen über einer Menge von n verschiedenen Basisrelationen $\mathcal{R} = \{\mathsf{R}_\mathsf{i} \mid \mathsf{i} = 1, ..., \mathsf{n}\}$. Die Selektionen lassen sich darstellen als eine Menge von konjunktiv verknüpften und atomaren Selektionsbedingungen $\mathcal{P} = \{\varphi_\mathsf{j} \mid \mathsf{j} = 1, ..., \mathsf{m}\}$, von denen jede sich auf eine oder mehrere Basisrelationen bezieht. Wenn eine Bedingung φ nur Attribute einer Teilmenge $\widetilde{\mathsf{R}}$ von $\mathcal{R}$ abdeckt, schreiben wir $(\varphi, \widetilde{\mathsf{R}}) \in \mathcal{P}$.

Gegenstand der Optimierung ist einerseits die Anordnung der Joins und andererseits die Positionierung der Selektionen, die nach den Überlegungen im Abschnitt 3.4.2 eben nicht grundsätzlich, wie bei der üblichen Heuristik, so früh wie möglich ausgeführt werden sollten.

Die entstehenden *Optimierungsterme* können rekursiv dargestellt werden als $\sigma_{\Phi}(\tau_{\mathsf{left}} \bowtie \tau_{\mathsf{right}})$, wobei Φ eine Menge äußerer Selektionsbedingungen (Konjunktion) ist, welche leer sein kann (entspricht „true“), und $\tau_{\mathsf{left}}, \tau_{\mathsf{right}}$ Terme aus den oben genannten Operationen sind, die alle anderen Selektionsbedingungen enthalten. Basisterme sind von der Form $\sigma_{\varphi}(\mathsf{R_i})$, mit $(\varphi, \mathsf{R_i}) \in \mathcal{P}$, oder $\sigma_{\mathsf{true}}(\mathsf{R_i})$, wobei $\mathsf{R_i}$ eine Basisrelation ist.

Wir orientieren uns an dem in [86] vorgeschlagenen Vorgehen der Optimierung von Join-Reihenfolgen durch Dynamische Programmierung, bei dem aus den Optimierungstermen für immer größer werdende Teilmengen von $\mathcal{R}$ schließlich ein optimaler Term konstruiert wird. Als Erweiterung arbeiten wir allerdings mit durch Selektionen geschachtelten Mengen von Relationen, um auch die Positionierung der Selektionen zu optimieren.[7]

Es wird eine Tabelle `planTable` aufgebaut, deren Einträge (abhängig von der stufenweisen Berechnung der Teilmengen) durch geschachtelte Mengen von Relationen indiziert sind; ein Eintrag liefert den zugehörigen Optimierungsterm durch Angabe der äußeren Selektionsbedingungen `selections` (oben: Φ) und des Eintrags `leftSide` für den linken Teilterm (oben: τ_{left}, aber in Mengendarstellung); der Eintrag für den rechten Teilterm (oben: τ_{right}) ist daraus rekonstruierbar.

Zusätzlich werden zu jedem Eintrag – zur schrittweisen Berechnung – folgende Attribute festgehalten:

- `cardinality:` die Kardinalität des Ausführungsplanes, der durch den optimierten Term dargestellt wird,
- `selectivity:` die Selektivität des äußersten Joins des zu optimierenden Termes[8],
- `cost:` die Gesamtkosten des optimierten Teiltermes (Ausführungsplanes).

Nachfolgend wird die Klasse `optimize` vorgestellt. Sie bekommt als Eingabe alle zu verbindenden Relationen $\mathcal{R}$ und positioniert während des Aufbauens des Anfrageplanes die zusätzlich übergebene Selektionsmenge $\mathcal{P}$:

Eingabe: Relationenmenge $\mathcal{R}$, Menge aller Selektionsbedingungen $\mathcal{P}$
Ausgabe: Optimierter Anfrageplan

`optimize(Relationenmenge` $\mathcal{R}$, `Bedingungsmenge` $\mathcal{P}$`) {`

Als erster Schritt werden alle übergebenen Basisrelationen in die `planTable` eingetragen:

```
planTable := ∅;
for each Ri ∈ R do {
   planTable[Ri].cardinality := |Ri|;
   planTable[Ri].leftSide := ∅;
```

Hinter $\kappa(\tau)$ versteckt sich unser Kostenmodell, welches die Kosten einer Basisselektion aufgrund der vorhandenen Statistiktabelle zurückliefert (siehe Kapitel 2).

[7] Alle Erweiterungen gegenüber dem Original-Algorithmus werden nachfolgend blau dargestellt.
[8] Die Selektivität wird als gegeben oder zu berechnen angesehen. Falls keine Joinbedingung vorhanden ist, wird daraus resultierend ein Kartesisches Produkt mit einer Selektivität von 1 angenommen.

planTable[R_i].cost := $\kappa(\sigma_{true}(R_i))$;
planTable[R_i].selectivity := 1;
planTable[R_i].selections := $\emptyset$;
$S := \{\varphi \mid (R_i, \varphi) \in \mathcal{P}\}$
if ($S \neq \emptyset$) {

Somit enthält S alle übergebenen Selektionsbedingungen, die auf R_i agieren. Dabei soll ψ deren Konjunktion darstellen.

planTable[$\sigma_\psi\{R_i\}$].cardinality := $|\sigma_\psi(R_i)|$;
planTable[$\sigma_\psi\{R_i\}$].leftSide := $\emptyset$;
planTable[$\sigma_\psi\{R_i\}$].cost := $\kappa(\sigma_\psi(R_i))$;
planTable[$\sigma_\psi\{R_i\}$].selectivity := 1;
planTable[$\sigma_\psi\{R_i\}$].selections := S;
}
}

Nachdem die Arbeitsumgebung erzeugt wurde, kann jetzt die eigentliche Optimierung stattfinden:

for (jede Teilmenge $\widetilde{R} := \widetilde{R}_1 \cup \widetilde{R}_2, \widetilde{R}_1 \cap \widetilde{R}_2 = \emptyset$, für die zu
$\widetilde{R}_1$ und $\widetilde{R}_2$ Einträge in der planTable existieren) do {
planTable[$\widetilde{R}$].cardinality := $|\widetilde{R}|$;
planTable[$\widetilde{R}$].leftSide := find_best_split($\widetilde{R}$);
planTable[$\widetilde{R}$].cost := planTable[τ_{left}].cost + planTable[τ_{right}].cost + $\kappa(\widetilde{R}_1 \bowtie_\psi \widetilde{R}_2)$;

Danach definieren wir uns ein $\widetilde{P}$, welches alle natürlichen Verbundbedingungen für $\widetilde{R}$ bereitstellt.

planTable[$\widetilde{R}$].selectivity := compute_selectivity($\widetilde{P}$);
planTable[$\widetilde{R}$].selections := $\emptyset$;
$S := \{\varphi \mid (\widetilde{R}, \varphi) \in \mathcal{P}$ and φ kommt in der Selektivität $\widetilde{R}$ noch nicht vor$\}$;

Wiederum definieren wir S, welches anschließend alle weiteren Bedingungen, die auf $\widetilde{R}$ agieren, enthält und ein wiederholtes Mal ψ als deren Konjunktion.

if (Eintrag planTable[$\widetilde{R}$] ist optimierbar mit der L2-Regel) {
remove($\widetilde{R}$, planTable);
}
}

An dieser Stelle wird mit Hilfe der L2-Regel eine Selektion entfernt, welche zu „tief“ im Baum ist. Beispielsweise wird $\widetilde{R} = \{R, \sigma_\varphi\{S, T\}\}$ daraufhin überprüft, ob sich der Term $R \bowtie \sigma_\varphi(S \bowtie T)$ nach der L2-Regel zu $\sigma_\varphi(R \bowtie (S \bowtie T))$ verändert. Falls dies der Fall sein sollte, kann der Eintrag gelöscht werden, da $\sigma_\varphi\{R, S, T\}$, bedingt durch das Paradigma der Dynamischen Programmierung, schon an einer anderen Stelle berücksichtigt wurde.

Anschließend folgt die Betrachtung für aufsteigende Selektionen:

if ($S \neq \emptyset$) {

```
        planTable[σ_ψ{R̃}].cardinality := |σ_ψ{R̃}|;
        planTable[σ_ψ{R̃}].leftSide := planTable[R̃].leftSide;
        planTable[σ_ψ{R̃}].cost := κ(σ_ψ{R̃}) + planTable[R̃].cost;
        planTable[σ_ψ{R̃}].selectivity := 1;
        planTable[σ_ψ{R̃}].selections := S;
```

Jetzt wird mit Hilfe der L2-Regel überprüft, ob eine Selektion zu „hoch“ im Baum ist. Dies wird untersucht, indem z. B. beim Term $\sigma_\psi(\mathsf{R} \bowtie \sigma_\varphi(\mathsf{S} \bowtie \mathsf{T}))$ theoretisch die Selektion nach unten verschoben wird. Falls das zu Kosteneinsparungen führt, existiert wiederum durch das gegebene Optimierungskonzept ein anderer Eintrag in der `plantable`, so dass der aktuelle Eintrag gelöscht werden kann.

```
        if (item σ_ψ{R̃} ist optimierbar mit der L2-Regel) {
            remove(σ_ψ{R̃},planTable);
        }
    }
return get_result(R, planTable);
}
```

Die schon in optimize verwendete compute_selectivity Methode schätzt für beliebige Joins deren Selektivitäten ab:

Eingabe: Menge aller Verbundbedingungen $\widetilde{\mathsf{P}}$, die auf $\mathcal{R}$ agieren
Ausgabe: Die Selektivität $\widetilde{\mathsf{P}}$ des Joins

```
compute_selectivity(join conditions P̃) {
 s := 1;
 for each φ_1 ∈ P̃ do {
  s = s · sel(φ_1);
 }
 return s;
}
```

Anschließend sei mit der Methode `compute_cost` die Möglichkeit gegeben, Kosten für ganze Teilbäume abzuschätzen:

Eingabe: Relationenmenge τ_{left} (linker Joinpartner),
Relationenmenge τ_{right} (rechter Joinpartner)
Ausgabe: die Kosten des Verbundes $\tau_{\mathsf{left}} \bowtie \tau_{\mathsf{right}}$

```
compute_cost(Relationenmenge τ_left, Relationenmenge τ_right) {
 return planTable[τ_left].cost + planTable[τ_right].cost + κ(τ_left, τ_right)
}
```

Schließlich wird der von Vance und Maier angegebene Algorithmus zur Rückgabe der kostengünstigsten Variante von möglichen Kombinationen nicht-leerer Teilmengen auf Basis der ihm übergebenen Relationenmenge dargestellt:

Eingabe: Relationenmenge $\widetilde{\mathsf{R}}$
Ausgabe: Relationenmenge $\widetilde{\mathsf{R}}_{\mathsf{best}}$, die beste linke Seite des Verbundes leftSide

```
find_best_split(Relationenmenge R̃) {
R̃_best := ∅;
```

```
 best_cost := ∞;
 for each R̃1, ∅ ⊂ R̃1 ⊂ R̃ do {
  R̃2 := R̃ \ R̃1;
  cost = compute_cost(R̃1, R̃2);
  if (cost ≤ best_cost) {
   best_cost = cost;
   R̃best = R̃;
  }
 }
 return R̃best;
}
```

3.5.1 Abstraktes Beispiel für den bedingten Join-Ordering-Algorithmus

Gegeben seien zwei Relationen R_1 und R_2 mit jeweils einer darauf agierenden Selektion σ_{φ_1} und σ_{φ_2}. Beide Relationen werden per Joinbedingung φ_3 verbunden. Die Aufgabe besteht darin, die beste Anordnung der drei Operatoren per Dynamischer Programmierung zu finden.

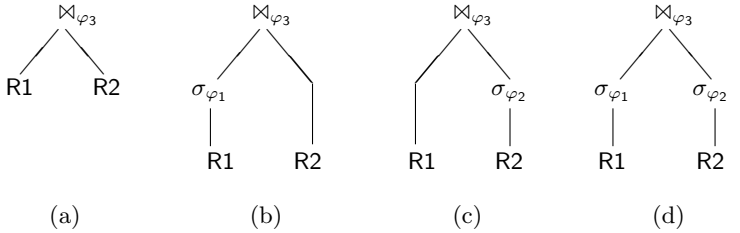
Der von uns vorgestellte Algorithmus füllt anfangs die Optimierungstabelle mit allen Basisrelationen auf, in unserem Fall R_1 und R_2:

R_1	...
R_2	...

Danach übt er alle möglichen Selektionen auf Basisrelationen aus und schreibt auch diese Terme in die Tabelle `planTable`:

R_1	...
R_2	...
$\sigma_{\varphi_1}(R_1)$	...
$\sigma_{\varphi_2}(R_2)$	...

Jetzt mit Hilfe der `for`-Schleife die nachfolgenden vier Varianten a) - d) erzeugt und in die `planTable` eingefügt:



(a) (b) (c) (d)

	R_1	...
	R_2	...
	$\sigma_{\varphi_1}(R_1)$	...
	$\sigma_{\varphi_2}(R_2)$	...
a)	$R_1 \bowtie_{\varphi_3} R_2$	...
b)	$\sigma_{\varphi_1}(R_1) \bowtie_{\varphi_3} R_2$	...
c)	$R_1 \bowtie_{\varphi_3} \sigma_{\varphi_2}(R_2)$	...
d)	$\sigma_{\varphi_1}(R_1) \bowtie_{\varphi_3} \sigma_{\varphi_2}(R_2)$	...

Im nächsten Schritt versucht die L2-Regel, die erste Selektion σ_{φ_1} oberhalb des Joins $\sigma_{\varphi_1}(R_1 \bowtie_{\varphi_3} R_2)$ anzuordnen. In unserem Beispiel steigt die Selektion bei der ersten Variante a) ab. Daraus folgt, dass diese Variante gelöscht wird. Demnach entscheidet man zwischen Variante a) und b), welche von beiden für den weiteren Optimierungsprozess verwendet wird.

Danach wird Variante b) genommen, um dort die zweite Selektion σ_{φ_2} oberhalb des Joins anzuordnen, da die erste schon verwendet wurde. Auch dort gibt die L2-Regel das „Absinken“ der Selektion an. Somit wird auch diese Variante aus der `planTable` entfernt.

Diesmal ist die Entscheidung zwischen Variante b) und d) gefallen. Als nächstes wird Variante c) verwendet um dort die erste Selektion σ_{φ_1} nach dem Join anzuordnen. Diesmal übt die L2-Regel keine Veränderung aus, so dass die Variante in der Tabelle bleibt. Bei der letzten Variante d) braucht keine Selektion mehr angeordnet zu werden und sie bleibt ebenfalls in der `planTable`.

	R_1	...
	R_2	...
	$\sigma_{\varphi_1}(R_1)$	...
	$\sigma_{\varphi_2}(R_2)$	...
c)	$R_1 \bowtie_{\varphi_3} \sigma_{\varphi_2}(R_2)$	...
d)	$\sigma_{\varphi_1}(R_1) \bowtie_{\varphi_3} \sigma_{\varphi_2}(R_2)$	...

In der zweiten Phase des Algorithmus wird die L2-Regel für „aufsteigende“ Selektionen aufgerufen. Auf die Variante c) angewendet, verändert sich nichts. Aber bei d) steigt σ_{φ_1} auf, so dass diese Variante erst jetzt in der zweiten Stufe entfernt wird. Hierbei entscheidet man zwischen c) und d). Somit liefert der Algorithmus die Anordnung $\sigma_{\varphi_1}(R_1 \bowtie_{\varphi_3} \sigma_{\varphi_2}(R_2))$ als besten Ausführungsplan. Wie man leicht sieht, ist es egal, mit welcher Variante generell gestartet wird, da immer alle vier Varianten mit der Indikatorfunktion abgeschätzt werden.

Wäre dieser Baum nur ein Teilbaum eines größeren Ausführungsplanes, dann würde für die weitere Optimierung nur noch Variante c) zur Neugenerierung von optimalen Teilmengen durch `find_best_split` verwendet. Damit wird eine Kosteneinsparung bei der Ausführung, im Gegensatz zur heuristischen Anordnung von Selektionen, gewonnen.

Wie an diesem Beispiel vorgeführt, können Kostenbedingungen und Indikatorfunktionen somit während der Dynamischen Programmierung verwendet werden, um zusätzliche Operatoren optimal anzuordnen. Dazu gehören auch Projektionsoperatoren, deren Anordnung analog zu dem oben vorgestellten Verfahren ist.

Eigene Implementierungen und Tests (siehe konkretes Beispiel im Anhang B) zeigten, dass mit Hilfe des vorgestellten Algorithmus Joinanordnungen mit exponentiellem Zeitaufwand betrieben werden können. Beim Verbund von mehr als 8 Basisrelationen verschlechterten sich die Anwortzeiten so stark, dass – genau wie bei kommerziellen Datenbanken – von einer Verwendung abgesehen werden sollte.

3.5.2 Laufzeitbetrachtung des erweiterten Join-Ordering-Algorithmus

Es ist aus Jo et al. [65] bekannt, dass der Join-Ordering-Algorithmus eine Laufzeit von $O(3^n)$ bei n Basisrelationen besitzt. Durch verschiedene Optimierungsmöglichkeiten und Einschränkungen des untersuchten Anfrageraumes kann die exponentielle Laufzeit auf eine polynomielle ($O(n^3)$) verbessert werden (siehe Moerkotte et al. [64]). Die Anwendung der in dieser Arbeit vorgestellten Indikatorfunktionen auf jede neu generierte Teilmenge entscheidet lokal zwischen zwei Varianten und speichert demzufolge nur eine der beiden ab. Hiermit ist die Berechnung der Anordnung für die Selektionen in linearer Zeit ($O(n)$) möglich, so dass sich letztlich nichts an der gesamten Laufzeit, verglichen mit der des Basisalgorithmus, ändert. Demzufolge ist der von uns erweiterte Algorithmus schneller als die in Moerkotte et al. (1997) [76], (1998) [77] und (1997) [82] vorgeschlagene Optimierung mit Hilfe von Ranking-Verfahren.

3.6 Fazit der bedingten Termersetzungsregeln

Mit der Formalisierung von implementierungsabhängigen, -unabhängigen und erzeugenden Regeln hin zu generalisierten Regeln konnte in diesem Kapitel der Grundstein für ein Termersetzungssystem zur Optimierung von Anfragen gelegt werden. Wurde bislang der Versuch unternommen, die Anzahl solcher Regeln so gering wie möglich zu halten, wird in unserem Fall

das genaue Gegenteil angestrebt. Mehr Regeln bedeuten auch mehr Situationen, in denen der zu optimierende Term verbessert werden kann. Um willkürliche Ersetzungen zu unterbinden, wurden semantische, syntaktische und kostenbasierte Bedingungen definiert. Mit ihrer Hilfe ist es jetzt möglich, die Termersetzung zielgerichteter und optimierter „ablaufen“ zu lassen. Die dafür benötigte Datenstruktur wurde mit Hilfe einer Attributierten Grammatik in Form eines generalisierten Baumes definiert. Durchgeführte Tests (siehe Anhang D) zeigen die Effizienz des Ansatzes, mit der Ausführungspläne optimiert werden können.

Des Weiteren wurde eine Methodik vorgestellt, mit der man Kostenbedingungen vereinfachen und ihre Anwendung beschleunigen kann. Durch Benutzung der entwickelten Indikatorfunktionen kann z. B. das bisher bekannte Join-Ordering Verfahren mit Hilfe der Dynamischen Optimierung um die Anordnung von beliebigen Operatoren erweitert werden. Dabei werden nur lineare Berechnungslaufzeiten hinzugefügt, welche die Größenordnung der Gesamtlaufzeit des Basisalgorithmus nicht verschlechtern.

Da bisherige Kostenformeln keine Unterscheidung für besonders kostenintensive Berechnungen – wie sie bei objektrelationalen Operatoren benötigt werden – ermöglichen, werden im anschließenden Kapitel diese Operatortypen gesondert betrachtet, um danach auch sie während der Anfrageoptimierung adäquat berücksichtigen zu können.

Persönlichkeiten werden nicht durch schöne Reden geformt,
sondern durch Arbeit und Leistung.

Albert Einstein 1879-1955



4 Erweiterung auf objektrelationale Operatoren

In diesem Kapitel wird die Erweiterbarkeit des vorgestellten Modells für Anfrageoptimierung in relationalen Datenbanken (siehe 2. Kapitel) auf objektrelationale Operatoren erläutert. Als Beispiel für ein objektrelationales Datenbanksystem wird ein räumliches Datenbanksystem angenommen, welches ein relationales Modell um geometrische Datentypen (z. B. Punkt, Linie, Polygon) und Operationen (z. B. Bereichsanfragen) erweitert.

Räumliche Datenbanken bieten nach [72] folgende vier Erweiterungen gegenüber relationalen Datenbanken:

(a) Ein erweitertes Schema zur Beschreibung der Speicherung, der Syntax und der Semantik geometrischer Datentypen;

(b) Operatoren, Funktionen und Prozeduren für räumliche Selektionen und Verbunde mit mengenorientierten, topologischen, metrischen und direktionalen Prädikaten;

(c) Räumliche Indizierungsmethoden;

(d) Tools zur Administration der Datenbank.[1]

Um auch in diesen Umgebungen sinnvolle Kostenabschätzungen machen zu können, werden nicht nur neue Abschätzungen für die Kardinalitäten und Seitenanzahlen der Ergebnisrelationen benötigt, sondern auch neue Formeln für Selektivitätsabschätzungen, Attributwertanzahlen und Tupellängen. Als Basis dafür dient das Entwicklungsprinzip der Kostenfunktionen für relationale Operatoren, so dass zusätzlich die einfache Erweiterbarkeit des vorgestellten Kostenmodells verdeutlicht wird.

Die zuvor genannten Erweiterungen wurden als sinnvoll angesehen, da bisherige Datenbanksysteme die Optimierung von Ausführungsplänen mit objektrelationalen Operatoren nicht unterstützen und dem Benutzer deren Anordnung überlassen [43]. Dabei können bei falscher Anwendung, z. B. bei gleichzeitiger Verwendung verschiedener Indexarten (räumliche und relationale), sogar falsche Resultate erzeugt werden:

„[For example] Oracle Spatial does not provide cost and selectivity estimates for spatial operators that are comparable to other operators in SQL. As a result, the choice made by the optimizer to use or not to use the spatial index may be incorrect. [...] [Sometimes Oracle] gives incorrect results (in fact, it may give different results at different times too).[2]“

Aus der Sicht des Datenbankoptimierers unterscheidet sich ein relationaler von einem räumlichen Anfrageoptimierer in drei Punkten (siehe Kriegel et al. [45] (1993)):

(a) Im Gegensatz zu relationalen Datenbanken besitzen räumliche keine bestimmten Wiederholungen von Anfrageblöcken und keine begrenzte Operatorenmenge, sondern können beliebig erweitert werden.

[1] Auf diese Komponente wird in dieser Arbeit nicht näher eingegangen.

[2] Eine Sammlung von Beispielen, bei denen inkorrekte Ergebnisse generiert werden, kann man bei Kothuri et al. [43] (S. 241 / 267-270) finden.

(b) Räumliche Datenbanken müssen mit extrem großen Datenmengen und dementsprechend großen Objekten umgehen können. Diese Objekte haben verschiedene räumliche Erweiterungen und besitzen keine intuitive Ordnung für eine Sortierung.

(c) Extrem komplexe und damit teure Operationen zur Berechnung von räumlichen Prädikaten widerlegen die simple Annahme, dass I/O-Kosten die CPU-Kosten dominieren.

Nachfolgend wird eine kurze Einführung in den Bereich räumliche Datenbanken gegeben. Eine ausführlichere Darstellung bieten Rigaux et al. [71] (2003).

4.1 Räumliche Datentypen

Relationen in räumlichen Datenbanksystemen können Standardattribute wie alphanumerische Werte (z. B. Postleitzahlen, Strassennamen) und geometrische Ausprägungen von modellierten räumlichen Objekten (Geometrien) beinhalten.

Das verwendete räumliche Datenmodell besitzt eine hierarchische Struktur, welche aus Elementen, Geometrien und Layern besteht. Dabei werden Layer (Schichten) aus Geometrien und diese wiederum aus primitiven Grundformen, den Elementen, zusammengesetzt (siehe [71],[84]). Elemente sind die Basisformen einer Geometrie. Es werden Punkte, Linien und unterschiedliche Flächen als Elementtypen unterstützt, wie in Abbildung 4.1 (siehe [7]) schematisch dargestellt.

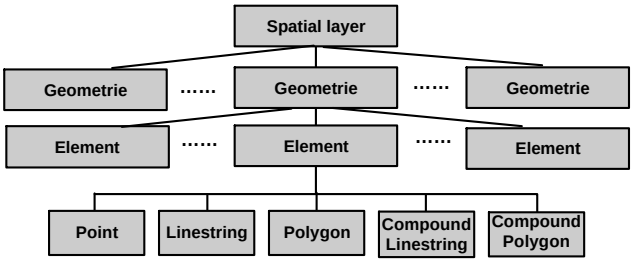


Abbildung 4.1: Räumliches Datenmodell

Ein oder mehrere Elemente können Repräsentanten eines räumlichen Objektes sein, z. B. kann eine Linie einen Strassenabschnitt, eine Punktmenge eine Sternenkonstellation darstellen. Geometrien sind eine geordnete Anzahl von Grundelementen, die entweder nur einen Elementtyp (homogene Geometriemenge) oder unterschiedliche Typen aufweisen (heterogene Geometriemenge). Ein Layer ist eine Sammlung von Geometrien, z. B. stellt ein Ortslayer eine Stadt dar, ein anderer Layer, aus Punkten bestehend, die Bevölkerungsdichte. Dabei kann ein Layer auch aus verschiedenen Geometrien definiert werden und er wird durch Elemente und Geometrien dargestellt. Die Geometrien eines gegebenen Layers und die dazugehörigen Indexe werden in objektrelationalen Datenbanken in relationalen Tabellen gespeichert.

4.2 Räumliche Operatoren

Räumliche Operatoren können nach Gaede und Gunther [22] (1998) in vier Gruppen aufgeteilt werden:

(a) Update-Operationen, zu denen herkömmliche Standardoperationen zählen, z. B. die Erzeugung und die Veränderung von Geometrien.

(b) Selektionsoperatoren, die wiederum in zwei Untergruppen gegliedert werden:

(1) Punktanfragen (engl. point query) (PQ): Gegeben sei ein Punkt p; finde alle räumlichen Objekte O, die diesen Punkt beinhalten:

$$\mathsf{PQ}(\mathsf{p}) = \{\mathsf{O} | \mathsf{p} \in \mathsf{O.G}\},$$

wobei O.G die Geometrie des Objektes O ist.

(2) Bereichsanfragen (engl. range or regional query) (RQ): Gegeben sei ein Anfragepolygon P; finde alle räumlichen Objekte O, die P schneiden. Falls P als ein Rechteck vorliegt, dann wird diese Anfrage auch als so genannte Fensteranfrage definiert (bzgl. weiterer Bedingungen siehe Abschnitt 4.6):

$$\mathsf{RQ}(\mathsf{P}) = \{\mathsf{O} | \mathsf{O} \in \mathsf{R} \wedge \varphi_{\mathsf{geo}}(\mathsf{O}, \mathsf{P})\} \text{ mit } \varphi_{\mathsf{geo}} := \{\mathsf{O.G} \cap \mathsf{P} \neq \emptyset\}.$$

Hierbei ist wiederum O.G die Geometrie des Objektes O.

(c) Räumliche Verbunde (engl. spatial joins): Falls zwei Tabellen R_1 und R_2 über ein räumliches Prädikat φ_{geo} verbunden werden, spricht man von einem Spatial-Join. Beispiele für solche Verbundbedingungen werden im Abschnitt 4.6 gegeben.

$$\mathsf{R}_1 \bowtie_{\varphi_{\mathsf{geo}}} \mathsf{R}_2 = \{\mathsf{O}, \tilde{\mathsf{O}} | \mathsf{O} \in \mathsf{R}_1 \wedge \tilde{\mathsf{O}} \in \mathsf{R}_2 \wedge \varphi_{\mathsf{geo}}(\mathsf{O}, \tilde{\mathsf{O}})\} \text{ mit } \varphi_{\mathsf{geo}} := \{\mathsf{O.G} \cap \tilde{\mathsf{O}}.\mathsf{G} \neq \emptyset\}$$

(d) Räumliche Aggregation (engl. spatial aggregate): dieser Operatortyp wird normalerweise bei der Nächsten-Nachbar-Suche (engl. nearest neighbor query (NNQ)) eingesetzt[3]:

Gegeben sei ein Objekt $\tilde{\mathsf{O}}$; finde alle Objekte O, die die minimale Distanz zu diesem haben:

$$\mathsf{NNQ}(\tilde{\mathsf{O}}) = \{\mathsf{O} \,|\, \forall\, \tilde{\tilde{\mathsf{O}}} : \mathsf{distance}(\tilde{\mathsf{O}}.\mathsf{G}, \mathsf{O.G}) \leq \mathsf{distance}(\tilde{\mathsf{O}}.\mathsf{G}, \tilde{\tilde{\mathsf{O}}}.\mathsf{G})\}$$

4.3 Beispiel für eine objektrelationale Anfrage

Um den Unterschied zu den relationalen Operatoren deutlich zu machen, wird im Folgenden ein naives Beispiel für eine objektrelationale Anfrage in einer räumlichen Datenbank mit Hilfe einer Bereichsanfrage näher erläutert.

Gegeben sei eine Relation mit räumlichem Datenbestand $\mathsf{R}(\underline{\mathsf{id}} : \mathsf{int}, \mathsf{A} : \mathsf{int}, \mathsf{G} : \mathsf{polygon})$, dessen Primärschlüsselattribut id ist. Darüber hinaus beinhaltet sie ein thematisches Attribut A und räumliche Daten im Attribut G (siehe Statistiktabelle 4.1). Darauf agierend seien zwei einfache Selektionen gegeben $\sigma^{\mathsf{Rel}}_{\mathsf{A} > 300 \wedge \mathsf{G\ inside\ W}}(\mathsf{R})$ (siehe Abbildung 4.2).[4]

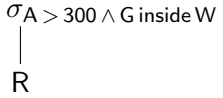


Abbildung 4.2: Beispiel für einen objektrelationalen Anfragebaum

[3] Eine differenziertere Betrachtung der verschiedenen räumlichen Aggregationsfunktionen wird in dieser Arbeit nicht vorgenommen.

[4] An dieser Stelle sei vorgreifend der topologische, boolesche Operator A inside B genannt. Er überprüft, ob eine Geometrie A in einer anderen Geometrie B liegt.

W = Fenster, z. B. linker unterer Quadrant des räumlichen Datenbestandes, wie in Abbildung 4.3

A = Werte zwischen 1 und 600

Statistiken	Statistiken
$\mathsf{sel}(\mathsf{A} > 300, \mathsf{R}) = \frac{1}{2}$	$\mathsf{pg}(\mathsf{R}) = 60$
$\mathsf{sel}(\mathsf{G}\ \mathsf{inside}\ \mathsf{W}, \mathsf{R}) = \frac{1}{4}$	$\mathsf{w}_1 = 0,02$
$\lvert\mathsf{R}\rvert = 6.000$	$\mathsf{w}_2 = 0$

Tabelle 4.1: Statistiken zur räumlichen Beispielanfrage



Abbildung 4.3: Beispiel für eine Fensteranfrage

Eine klassische heuristische Strategie ordnet die Selektionsbedingungen nach ihrer Selektivität an (siehe 3. Kapitel). Somit sollte folglich diejenige Selektion zuerst ausgeführt werden, deren Selektivität am kleinsten ist:

$$\sigma_{\varphi_1}(\sigma_{\varphi_2}(\mathsf{R})) \overset{1}{\underset{2}{\rightleftharpoons}} \sigma_{\varphi_2}(\sigma_{\varphi_1}(\mathsf{R})),$$

1) falls $\mathsf{sel}(\varphi_1, \mathsf{R}) \leq \mathsf{sel}(\varphi_2, \mathsf{R})$
2) falls $\mathsf{sel}(\varphi_2, \mathsf{R}) \leq \mathsf{sel}(\varphi_1, \mathsf{R})$

Infolgedessen würde ein relationaler Anfrageoptimierer mit den im Kapitel 2.3 vorgestellten Funktionen den Anfrageterm wie folgt ersetzen und zu folgender Kostenabschätzung kommen,

$$\sigma_{\mathsf{A}>300 \wedge \mathsf{G}\ \mathsf{inside}\ \mathsf{W}}(\mathsf{R}) \rightarrow \sigma_{\mathsf{A}>300}(\sigma_{\mathsf{G}\ \mathsf{inside}\ \mathsf{W}}(\mathsf{R})) =: \tau_1$$

wobei $\overline{\mathsf{R}} := \sigma_{\mathsf{G}\ \mathsf{inside}\ \mathsf{W}}(\mathsf{R})$:

(a) $\mathsf{cost}_{\mathsf{op}}(\sigma^{\mathsf{Rel}}_{\mathsf{G}\ \mathsf{inside}\ \mathsf{W}}(\mathsf{R})) = \mathsf{pg}(\mathsf{R}) + \mathsf{w}_1 \cdot \lvert\mathsf{R}\rvert = 60 + 0,02 \cdot 6.000 = 180$

(b) $\mathsf{cost}_{\mathsf{op}}(\sigma^{\mathsf{Rel}}_{\mathsf{A}>300}(\overline{\mathsf{R}})) = \mathsf{pg}(\overline{\mathsf{R}}) + \mathsf{w}_1 \cdot \lvert\overline{\mathsf{R}}\rvert = \mathsf{sel}(\mathsf{G}\ \mathsf{inside}\ \mathsf{W},\ \mathsf{R}) \cdot \mathsf{pg}(\mathsf{R}) +$

$\mathsf{w}_1 \cdot \mathsf{sel}(\mathsf{G}\ \mathsf{inside}\ \mathsf{W},\ \mathsf{R}) \cdot \lvert\mathsf{R}\rvert = \frac{1}{4} \cdot 60 + 0,02 \cdot \frac{1}{4} \cdot 6.000 = 45$

$\Rightarrow \mathsf{cost}(\tau_1) = 180 + 45 = \mathbf{225}$

$\tau_1 :=$ $\sigma^{\mathsf{Rel}}_{\mathsf{A}>300}$ / $\sigma^{\mathsf{Rel}}_{\mathsf{G}\ \mathsf{inside}\ \mathsf{W}}$ / R

Dreht man die Reihenfolge um, so ergeben sich folgende Kosten:

$$\sigma_{\mathsf{A}>300 \wedge \mathsf{G}\ \mathsf{inside}\ \mathsf{W}}(\mathsf{R}) \rightarrow \sigma_{\mathsf{G}\ \mathsf{inside}\ \mathsf{W}}(\sigma_{\mathsf{A}>300}(\mathsf{R})) =: \tau_2$$

Mit $\overline{\mathsf{R}} := \sigma_{\mathsf{A}>300}(\mathsf{R})$ folgt:

(a) $\mathsf{cost}_{\mathsf{op}}(\sigma^{\mathsf{Rel}}_{\mathsf{A}>300}(\mathsf{R})) = \mathsf{pg}(\mathsf{R}) + \mathsf{w}_1 \cdot \lvert\mathsf{R}\rvert = 60 + 0,02 \cdot 6.000 = 180$

(b) $\mathsf{cost}_{\mathsf{op}}(\sigma^{\mathsf{Rel}}_{\mathsf{G}\ \mathsf{inside}\ \mathsf{W}}(\overline{\mathsf{R}})) = \mathsf{pg}(\overline{\mathsf{R}}) + \mathsf{w}_1 \cdot \lvert\overline{\mathsf{R}}\rvert = \mathsf{sel}(\mathsf{A} > 300, \mathsf{R}) \cdot \mathsf{pg}(\mathsf{R}) +$

$\mathsf{w}_1 \cdot \mathsf{sel}(\mathsf{A} > 300, \mathsf{R}) \cdot \lvert\mathsf{R}\rvert = \frac{1}{2} \cdot 60 + 0,02 \cdot \frac{1}{2} \cdot 6.000 = 90$

$\Rightarrow \mathsf{cost}(\tau_2) = 180 + 90 = \mathbf{270}$

$\tau_2 :=$ $\sigma^{\mathsf{Rel}}_{\mathsf{G}\ \mathsf{inside}\ \mathsf{W}}$ / $\sigma^{\mathsf{Rel}}_{\mathsf{A}>300}$ / R

Daraus folgt die Wahl von τ_1 als bester Ausführungsplan. Obwohl τ_1 im relationalen Fall richtig wäre, ist seine Ausführung für den objektrelationalen Fall falsch. Hier entwickeln Selektionsoperatoren verschiedene Kosten hinsichtlich alphanumerischer und räumlicher Prädikate. Diese Feststellung kann berücksichtigt werden, indem man räumliche Selektionen verteuert, d. h.

man gewichtet räumliche Selektionen in der Kostenberechnung zusätzlich mit einem Proportionalitätsfaktor ω. In den Fällen, in denen wir wissen, dass ein objektrelationaler Operator im Schnitt zehn Mal länger zur Bearbeitung benötigt als eine einfache Integer-Selektion, dreht sich schließlich die Präferenz bezüglich eines Ausführungsplanes um:

$$\Rightarrow \mathsf{cost}(\tau_1) = \omega \cdot 180 + 45 = 10 \cdot 180 + 45 = \mathbf{1.845}$$

$$\Rightarrow \mathsf{cost}(\tau_2) = 180 + \omega \cdot 90 = 180 + 10 \cdot 90 = \mathbf{1.080}$$

Zu klären bleibt, wie der operatorabhängige Gewichtungsfaktor ω zu wählen ist. Eine exakte Definition dieses Gewichtungsfaktors würde uns die Möglichkeit bieten, objektrelationale Operatoren mit relationalen Operatoren ins Verhältnis zu setzen. Nachfolgende Betrachtungen beschäftigen sich mit diesem Problem.

4.4 Motivation zur differenzierten Betrachtung von Geometrietypen und Prädikatsfunktionen

Komplexe Datentypen erfordern laufzeitbedingt einen höheren Berechnungsaufwand. Wird bislang bei Optimierern in relationalen Datenbanksystemen keine Differenzierung bezüglich der Datentypen vorgenommen, sollte sie bei räumlichen Datenbanken die Hauptgrundlage der Optimierung sein. Grund dafür ist die Erkenntnis dieser Arbeit, dass sich keine Kostenabschätzungen ohne sie durchführen lassen. Beispielsweise verursacht ein räumlicher Vergleich zweier Geometrien (innerhalb eines Tupel- und Seitenzugriffs) höhere Kosten als ein einfacher numerischer (z. B. $\mathsf{R}_1.\mathsf{Nr} = \mathsf{R}_2.\mathsf{Nr}$) Abgleich (siehe Abschnitt 4.5).

Darüber hinaus sollte eine Unterscheidung der verwendeten Funktionen in den räumlichen Prädikaten angestrebt werden. Die aus den relationalen Datenbanken bekannten Unterschiede im Laufzeitverhalten von verschiedenen Funktionen bieten zusätzliches Optimierungspotential, z. B. ist ein Abgleich in relationalen Datenbanken schneller zu berechnen als eine Like-Funktion auf demselben Attribut einer Relation.[5] Aus diesem Grund sollte die Reduzierung der Zugriffe auf Geometrietupeln im Verhältnis zu alphanumerischen Tupeln unter Berücksichtigung der darauf agierenden Funktionen angestrebt werden (siehe Abschnitt 4.7).

4.5 Geometrietypen und ihre Verteilung

Um die Komplexität einer Operation auf einem beliebigen Datentyp berechnen zu können, müssen wir die Komplexitat des Datentyps mit in Betracht ziehen. Im relationalen Modell besitzt jedes Attribut die gleiche Komplexität, einen vorher fest definierten Typ (z. B. numerisch (Integer) oder Zeichenkette (String)) und eine gegebene Attributwertanzahl (siehe 2.5.2).

Bei geometrischen Attributen existieren eigentlich unterschiedliche Geometrietypen, der in räumlichen Datenbanksystemen üblicherweise verwendete Datentyp „Geometrie“ ist nur eine Generalisierung. In Tabelle 4.2 listen wir sieben verschiedene Geometrietypen (auch primitive Grundformen genannt) und eine Untergruppe COLLECTION auf. Angelehnt an Oracle 10g kürzen

[5] Dieses wurde schon bei relationalen Datenbanken erkannt (siehe Gultzan et al. [31]), da auch dort der Zugriff auf Attribute, die als Strings vorliegen, meistens langsamer vonstatten geht, als auf solche, die als Integer abgespeichert werden. Jedoch sind die Geschwindigkeitsunterschiede nicht so extrem, dass sich der zusätzliche Aufwand für eine differenziertere Betrachtung lohnen würde.

wir sie mit $(\mathsf{dl0j}), \mathsf{j} = 0, ..., 7$ ab. Die Tabellen im Anhang C setzen die Geometrietypen ins Verhältnis mit den darauf agierenden räumlichen Operatoren.

ID	Geometrietyp	Beschreibung
dl00	UNKNOWN_GEOMETRY	Eine `unbekannte Geometrie`, die zu keiner anderen Menge gehört.
dl01	POINT	Die Geometrie `Punkt` besteht aus einem einzelnen Koordinatenpaar.
dl02	LINE / CURVE	Eine `Linien`- oder `Kurven`-Geometrie besteht aus einer Liste von Koordinatenpaaren, die Liniensegmente darstellen. (Linien und Kurven werden hier als gleichwertig betrachtet.)
dl03	POLYGON	Die `Polygon`-Geometrie beinhaltet ein Polygon mit oder ohne Löcher.
dl04	COLLECTION	Diese Geometrie ist eine heterogene `Sammlung` von Geometrieelementen. COLLECTION ist eine Obermenge, die alle anderen Geometrien beinhalten kann, außer dl00 Geometrien.
dl05	MULTIPOINT	Ein oder `mehrere Punkte` gehören zur Geometrie MULTIPOINT, einer Obermenge von POINT.
dl06	MULTILINE / MULTICURVE	MULTILINE und MULTICURVE sind Obermengen von LINE und CURVE, werden hier als gleichwertig betrachtet und beinhalten eine oder mehrere `Linien` oder `Kurven`.
dl07	MULTIPOLYGON	Die `Multipolygon`-Geometrie kann aus mehreren disjunkten Polygonen bestehen. MULTIPOLYGON ist eine Obermenge von POLYGON.

Tabelle 4.2: Geometrietypen

Nachfolgende Abbildung 4.4 verdeutlicht die vorhandenen Zusammenhänge im geometrischen Datenmodell, z. B. gibt die Kombination von verschiedenen Geometrien eine COLLECTION an. Die hierarchische Struktur der Geometrien sieht folgendermaßen aus:

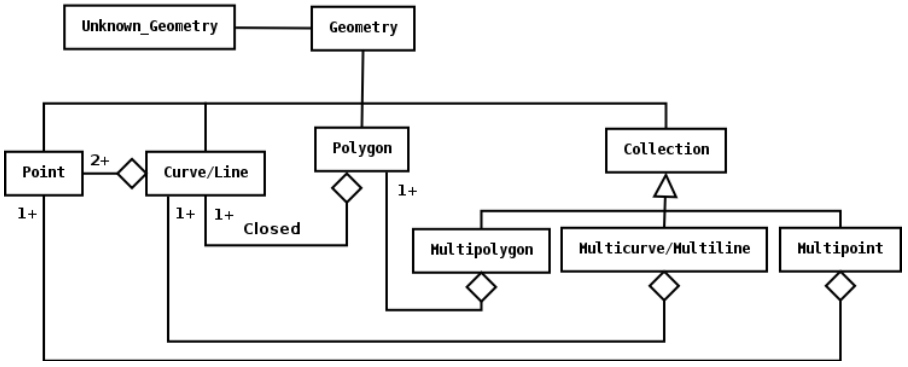


Abbildung 4.4: Geometrienmodell

Bisherige Datenbankmanagementsysteme halten keine Statistiken über die Verteilung der Geometrietypen bereit. Dabei sollte man gerade im geometrischen Fall für die Berechnung der Kosten nicht von einer Gleichverteilung der Geometrietypen ausgehen, da die Geometrietypen unterschiedlich stark kostenintensive Operatoren haben.

Der in dieser Arbeit vorgestellte Einphasenoptimierer kann solche wichtigen Informationen in den Optimierungsprozess mit einfließen lassen. Dazu müssen diese Statistiken vor einer Optimierung zusätzlich abgefragt werden.

4.6 Topologische Beziehungsoperatoren

Topologische Beziehungsoperatoren (boolesche Funktionen zwischen Geometrien) sind Bestandteile von räumlichen Analysefunktionen eines räumlichen Datenbanksystems. Sie geben Auskunft über die räumlichen und strukturellen Eigenschaften von geometrischen Objekten unabhängig von ihrer Ausdehnung bzw. geometrischen Form. Dabei gibt es verschiedene Modelle und Implementierungsarten (siehe [84]). In dieser Arbeit beziehen sich die Aussagen auf das 9-Intersection-Modell von Egenhofer [18] (1993). Bei diesem Modell wird zwischen dem Inneren A°, dem Rand $\partial\mathsf{A}$, und dem Äußeren A^- eines räumlichen Objektes unterschieden. Die topologische Beziehung zwischen zwei Objekten A und B wird deshalb durch eine 3x3-Matrix beschrieben:

$$\mathsf{R}_{9\mathsf{IM}}(\mathsf{A},\mathsf{B}) = \begin{pmatrix} \mathsf{A}^\circ \cap \mathsf{B}^\circ & \mathsf{A}^\circ \cap \partial\mathsf{B} & \mathsf{A}^\circ \cap \mathsf{B}^- \\ \partial\mathsf{A} \cap \mathsf{B}^\circ & \partial\mathsf{A} \cap \partial\mathsf{B} & \partial\mathsf{A} \cap \mathsf{B}^- \\ \mathsf{A}^- \cap \mathsf{B}^\circ & \mathsf{A}^- \cap \partial\mathsf{B} & \mathsf{A}^- \cap \mathsf{B}^- \end{pmatrix}$$

Die Einträge in der Matrix geben jeweils an, ob die Schnittmenge zwischen den entsprechenden Teilen der Objekte leer ist $\emptyset$ oder nicht $\neg\emptyset$. Für ein Polygon entsprechen die Definitionen von Innerem, Rand und Äußerem den intuitiven Begriffen. Bei Linien werden die beiden Endpunkte als Rand aufgefasst, das Innere sind alle anderen Punkte. Eine geschlossene Linie hat keinen Rand. Bei Multilinien zählen alle Punkte, die in einer ungeraden Anzahl von Linien als Endpunkte auftauchen, als Rand.

Abbildung 4.5 zeigt exemplarisch die topologische Beziehung TOUCH (meet) zwischen den Geometrien A und B. Dabei liegt ein Teil des Randes von B auf dem Rand von A, d. h. der Schnitt zwischen dem Äußeren von A und dem Rand von B ist nicht leer $(\mathsf{A}^- \cap \partial\mathsf{B}) = \neg\emptyset$. Die dazugehörige Matrix lautet somit:

$$\begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$$

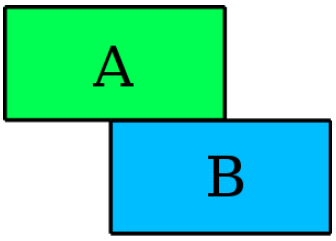


Abbildung 4.5: Topologisches Beziehungsbeispiel zweier Geometrien

Theoretisch existieren im 9IM-Modell $2^9 = 512$ verschiedene Beziehungsmatrizen, die aber in Abhängigkeit davon, welchen Beschränkungen die Objekte unterliegen, nicht alle angenommen werden können. In dieser Arbeit beziehen wir uns auf die 12 gebräuchlichsten zwei-dimensionalen Beziehungstypen, welche in der Tabelle 4.3 aufgeführt sind. Spielt die Reihenfolge der Parameter eine Rolle, so sind die Attribute A und B angegeben.

Topologische Beziehungen können als räumliche Selektionsbedingungen verwendet werden. Beispielsweise kann eine Geometrie A mit Geometrie B aus einer Relation R wie folgt auf Gleichheit abgefragt werden:

$$\sigma_{\mathsf{A\ EQUAL\ B}}(\mathsf{R})$$

Natürlich können topologische Beziehungen auch als Joinbedingungen verwendet werden. Dabei ist A ein geometrisches Attribut in R_1 und B eines aus R_2:

$$\sigma_{\mathsf{A\ EQUAL\ B}}(\mathsf{R}_1 \times \mathsf{R}_2) \rightleftharpoons \mathsf{R}_1 \bowtie_{\mathsf{A\ EQUAL\ B}} \mathsf{R}_2$$

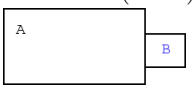
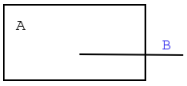
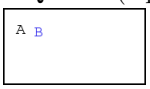
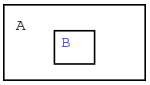
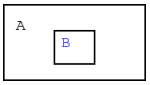
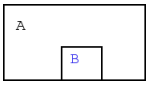
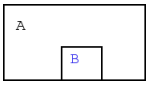
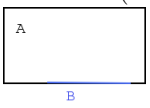
Funktion (Topologie)	Beschreibung
DISJOINT (disjoint) 	Weder die Ränder noch das Innere der beiden Objekte überschneiden sich.
TOUCH (meet) 	Es überschneiden sich nur die Ränder.
OVERLAPBDYDISJOINT  (overlap)	Das Innere eines Objektes schneidet sowohl den Rand als auch das Innere des anderen Objektes. Die beiden Ränder schneiden sich jedoch nicht.
OVERLAPBDYINTERSECT  (intersect)	Sowohl die Ränder als auch das Innere der beiden Objekte überschneiden sich.
EQUAL (equal) 	Die Ränder und das Innere der beiden Objekte sind gleich.
A CONTAINS B (contains) 	Das Innere und der Rand des zweiten Objektes B sind komplett im Inneren des ersten Objekts A enthalten.
B INSIDE A (covers) 	entspricht A CONTAINS B
A COVERS B (inside) 	Das Innere des zweiten Objektes B ist komplett in dem Inneren und/oder dem Rand des ersten Objektes A enthalten, und die beiden Ränder überschneiden sich.
B COVEREDBY A  (covered by)	entspricht A COVERS B
B ON A (on) 	Das Innere und der Rand des ersten Objektes B liegen komplett auf dem Rand des zweiten Objektes A.
ANYINTERACT (anyinteract)	Die Objekte sind nicht DISJOINT.
A DETERMINE B (determine)	Bestimmt, in welcher topologischen Beziehung A zu B steht.

Tabelle 4.3: Topologische Beziehungsoperatoren

4.7 Vorselektierung bei topologischen Beziehungsanfragen

Mit den zusätzlichen Statistiken aus den Geometrietypenverteilungen besitzen wir hinsichtlich der topologischen Beziehungen einen wichtigen Vorteil: Das Wissen, wann eine Vorfilterung der Argumentrelationen durch eine zusätzliche Selektion nach Geometrietypen sinnvoll ist. Diese redundante Operation kann letztlich Gesamtkosten einsparen und per Kostenmodell abgeschätzt werden[6].

Im Folgenden werden Termersetzungsregeln zur Vorselektierung für topologische Operatoren sowie ihre Bedingungen aufgeführt. Seien A und B wiederum Geometrien:

EQUAL: Die zu vergleichenden Geometrien müssen denselben Geometrietyp besitzen, $\mathsf{gt}(\mathsf{A}) = \mathsf{gt}(\mathsf{B})$:

$$\sigma_{\mathsf{A\ EQUAL\ B}}(\mathsf{R}) \Longrightarrow \sigma_{\mathsf{A\ EQUAL\ B}}\left(\sigma_{\mathsf{gt(A)=gt(B)}}(\mathsf{R})\right) \tag{4.1}$$

COVERS: Punkte können keine Geometrien enthalten:

$$\sigma_{\mathsf{A\ COVERS\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\Longrightarrow}} \emptyset \tag{4.2}$$

$$1)\ \text{falls } \mathsf{gt}(\mathsf{A}) \in \{\mathsf{point}, \mathsf{multipoint}\}$$

Linien können keine Flächen enthalten:

$$\sigma_{\mathsf{A\ COVERS\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\Longrightarrow}} \sigma_{\mathsf{A\ COVERS\ B}}\left(\sigma_{\mathsf{gt(B)}\notin\{\mathsf{polygon,multipolygon}\}}(\mathsf{R})\right) \tag{4.3}$$

$$1)\ \text{falls } \mathsf{gt}(\mathsf{A}) \in \{\mathsf{line}, \mathsf{multiline}\}$$

INSIDE: Flächen können nicht in Punkten und Linien enthalten sein:

$$\sigma_{\mathsf{A\ INSIDE\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\Longrightarrow}} \sigma_{\mathsf{A\ INSIDE\ B}}\left(\sigma_{\mathsf{gt(B)}\notin\{\mathsf{line,multiline,point,multipoint}\}}(\mathsf{R})\right) \tag{4.4}$$

$$1)\ \text{falls } \mathsf{gt}(\mathsf{A}) \in \{\mathsf{polygon}, \mathsf{multipolygon}\}$$

Anderenfalls können auch keine Geometrien in Punkten enthalten sein:

$$\sigma_{\mathsf{A\ INSIDE\ B}}(\mathsf{R}) \Longrightarrow \sigma_{\mathsf{A\ INSIDE\ B}}\left(\sigma_{\mathsf{gt(B)}\notin\{\mathsf{point,multipoint}\}}(\mathsf{R})\right) \tag{4.5}$$

CONTAINS: Punkte können keine Geometrien enthalten:

$$\sigma_{\mathsf{A\ CONTAINS\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\Longrightarrow}} \emptyset \tag{4.6}$$

$$1)\ \text{falls } \mathsf{gt}(\mathsf{A}) \in \{\mathsf{point}, \mathsf{multipoint}\}$$

Linien können keine Flächen enthalten, da das Innere einer Fläche nicht im Inneren einer Linie liegen kann:

$$\sigma_{\mathsf{A\ CONTAINS\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\Longrightarrow}} \sigma_{\mathsf{A\ CONTAINS\ B}}\left(\sigma_{\mathsf{gt(B)}\notin\{\mathsf{polygon,multipolygon}\}}(\mathsf{R})\right) \tag{4.7}$$

$$1)\ \text{falls } \mathsf{gt}(\mathsf{A}) \in \{\mathsf{line}, \mathsf{multiline}\}$$

COVEREDBY: Geometrien können nicht von Punkten überdeckt werden:

$$\sigma_{\mathsf{A\ COVEREDBY\ B}}(\mathsf{R}) \Longrightarrow \sigma_{\mathsf{A\ COVEREDBY\ B}}\left(\sigma_{\mathsf{gt(B)}\notin\{\mathsf{point,multipoint}\}}(\mathsf{R})\right) \tag{4.8}$$

Flächen können nicht von Linien überdeckt werden, da das Innere einer Linie nicht im Inneren einer Fläche liegen kann:

$$\sigma_{\mathsf{A\ COVEREDBY\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\Longrightarrow}} \sigma_{\mathsf{A\ COVEREDBY\ B}}\left(\sigma_{\mathsf{type(B)}\notin\{\mathsf{line,multiline,point,multipoint}\}}(\mathsf{R})\right) \tag{4.9}$$

[6] Da der Geometrietyp für die Prädikatsgewichtung verarbeitet werden muss, kann auch der Fall einer zusätzlichen Selektion abgeschätzt werden.

$$1) \text{ falls } \mathsf{gt}(\mathsf{A}) \in \{\mathsf{polygon}, \mathsf{multipolygon}\}$$

OVERLAPBDYDISJOINT: Punkte können keine Geometrien schneiden:

$$\sigma_{\mathsf{A\ OVERLAPBDYDISJOINT\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\rightleftharpoons}} \emptyset \tag{4.10}$$

$$1) \text{ falls } \mathsf{gt}(\mathsf{A}) = \mathsf{point}$$

Flächen können keine Flächen schneiden, ohne dass sich die Ränder schneiden:

$$\sigma_{\mathsf{A\ OVERLAPBDYDISJOINT\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\rightleftharpoons}} \sigma_{\mathsf{A\ OVERLAPBDYDISJOINT\ B}}\left(\sigma_{\mathsf{gt(B)} \notin \{\mathsf{polygon},\mathsf{multipolygon}\}}(\mathsf{R})\right) \tag{4.11}$$

$$1) \text{ falls } \mathsf{gt}(\mathsf{A}) = \{\mathsf{polygon}, \mathsf{multipolygon}\}$$

ON: Eine Fläche kann nie auf dem Rand einer anderen Geometrie liegen:

$$\sigma_{\mathsf{A\ ON\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\rightleftharpoons}} \emptyset \tag{4.12}$$

$$1) \text{ falls } \mathsf{gt}(\mathsf{A}) \notin \{\mathsf{polygon}, \mathsf{multipolygon}\}$$

Geometrien können nicht auf einem Punkt liegen:

$$\sigma_{\mathsf{A\ ON\ B}}(\mathsf{R}) \Rightleftharpoons \sigma_{\mathsf{A\ ON\ B}}\left(\sigma_{\mathsf{gt(B)} \notin \{\mathsf{point},\mathsf{multipoint}\}}(\mathsf{R})\right) \tag{4.13}$$

$$\sigma_{\mathsf{A\ ON\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\rightleftharpoons}} \emptyset \tag{4.14}$$

$$1) \text{ falls } \mathsf{gt}(\mathsf{B}) \in \{\mathsf{point}, \mathsf{multipoint}\}$$

TOUCH: Punkte können keine anderen Punkte berühren:

$$\sigma_{\mathsf{A\ TOUCH\ B}}(\mathsf{R}) \overset{1}{\underset{1}{\rightleftharpoons}} \sigma_{\mathsf{A\ TOUCH\ B}}\left(\sigma_{\mathsf{gt(B)} \notin \{\mathsf{point},\mathsf{multipoint}\}}(\mathsf{R})\right) \tag{4.15}$$

$$1) \text{ falls } \mathsf{gt}(\mathsf{A}) \in \{\mathsf{point}, \mathsf{multipoint}\}$$

4.8 Funktionsweise von räumlichen Operatoren

Räumliche Datenbanksysteme benutzen ein zweistufiges Modell, um Anfragen zu bearbeiten. Zunächst werden durch einen primären Filter Obermengen errechnet, die auf Basis von geometrischen Näherungswerten (z. B. mit Hilfe der Minimal Bounding Box (MBR)) für die ursprünglichen Geometrien berechnet werden (siehe Abbildung 4.6 aus Belina et al. [7] (2004)).

Anschließend wird für diese Menge im zweiten Filterprozess genau überprüft, ob die geforderten Bedingungen wirklich erfüllt sind oder nicht. Beim ersten Filter kann ein R-Baum-basierter räumlicher Index verwendet werden. Somit werden nur TIDs verarbeitet, die schließlich die Tupelmenge beschreiben, auf der anschließend die räumlichen Operatoren angewendet werden. Detaillierte Darstellungen können bei Shekhar et al. [80] nachgelesen werden.

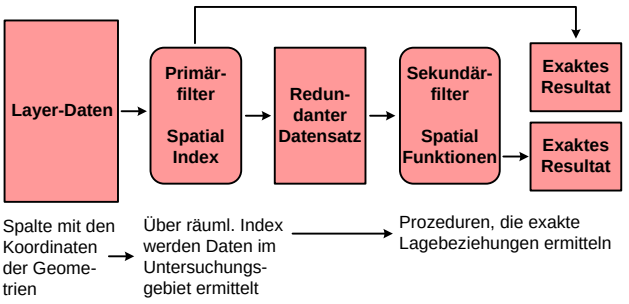


Abbildung 4.6: Anfragemodell von räumlichen Datenbanken

Die Konsequenz aus diesem Ablauf ist, dass man im Gegensatz zu relationalen Indexen, aus denen exakte Ergebnisse gewonnen werden können, für Ergebnisse aus räumlichen Indexen einen

zusätzlichen Selektionsprozess mit denselben Bedingungen (φ_{geo}) benötigt. Nicht immer werden beide Filter benötigt. Beim einfachen Zoomen einer Landkarte wird nur der erste Filter verwendet. Er liefert die in diesem Fall ausreichende Obermenge von noch zu verarbeitenden Tupeln, aus welcher dann – durch ein einfaches Clipping – die Zielfläche berechnet werden kann. Aus den oben genannten Gründen sind räumliche Operatoren in räumlichen Datenbanken keine einfachen Selektionen oder Joins, sondern zusammengesetzte Operationen. Wir bauen deshalb den räumlichen Operator je nach Anzahl der verwendeten Filterfunktionen zusammen. In der Tabelle 4.4 ist die Primärfilterfunktion durch ein p in der Implementierungsvariante gekennzeicht. Falls noch zusätzlich oder ausschließlich ein Sekundärfilter benötigt wird, fügen wir eine weitere relationale Selektion s ein.

räumlicher Operator	relationale Repräsentanten	Beschreibung
$\sigma^{\mathsf{Rel}}_{\varphi_{\mathsf{geo}}}(\mathsf{R})$ $\sigma^{\mathsf{Index}}_{\varphi_{\mathsf{geo}}}(\mathsf{R})$	$\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}(\mathsf{R})$ $\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}\left(\sigma^{\mathsf{Index_p}}_{\varphi_{\mathsf{geo}}}(\mathsf{R})\right)$	Im Gegensatz zur relationalen Selektion, muss bei der indexbasierten eine zweite Sekundärfilterung vorgenommen werden, da durch die erste Operation nur eine Obermenge der Ergebnisrelation zurückgeliefert wird.
$\mathsf{R}_1 \bowtie^{\mathsf{Rel,Rel}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	$\mathsf{R}_1 \bowtie^{\mathsf{Rel,Rel_s}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	Dieser Join-Operator kann beispielsweise dafür verwendet werden, um festzustellen, ob zwei Geometrien sich innerhalb einer bestimmten Distanz befinden. In diesem Fall wird keine zweite Filterung benötigt.
$\mathsf{R}_1 \bowtie^{\mathsf{Sort,Sort}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	$\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}\left(\mathsf{R}_1 \bowtie^{\mathsf{Sort,Sort_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2\right)$	Eine Merge-Joinvariante über eine vorherige Partitionierung des zu selektierenden Raumes (siehe DeWitt et al. [67] (1996)).[7]
$\mathsf{R}_1 \bowtie^{\mathsf{Rel,Index}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	$\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}(\mathsf{R}_1 \bowtie^{\mathsf{Rel,Index_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2)$	Verbindet zwei Relationen über einen Indexzugriff und liefert Tupel zurück.
$\mathsf{R}_1 \bowtie^{\mathsf{Index,Index}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	$\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}(\mathsf{R}_1 \bowtie^{\mathsf{Index,Index_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2)$	Hier werden zwei Relationen über den jeweiligen Indexzugriff verbunden und Ergebnistupel zurückgeliefert.
$\mathsf{R}_1 \bowtie^{\mathsf{Hash,Hash}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	$\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}\left(\mathsf{R}_1 \bowtie^{\mathsf{Hash,Hash_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2\right)$	Räumliche Hash-Joinvariante (siehe Lo et al. [55] (1996) und Fornari et al. [21] (2004).[7]
$\Gamma_{\overline{\mathsf{F}}\#\overline{\mathsf{G}}}(\mathsf{R})$	$\Gamma_{\overline{\mathsf{F}}\#\overline{\mathsf{G}}}(\mathsf{R})$	Räumlicher Aggregationsoperator, der beliebige Geometrien verknüpft und eine Ergebnisgeometrie zurückliefert.

Tabelle 4.4: Relationale Repräsentanten räumlicher Operatoren

4.9 Erweiterung des relationalen Kostenmodells auf objektrelationale Operatoren

Im Folgenden werden die bisherigen Abschätzungen für die Berechnung von Kosten für relationale Operatoren auf objektrelationale erweitert. Zunächst wird die Anwendung der klassischen

[7] Dieser Operator wird bislang in kommerziellen Datenbanken nicht unterstützt.

Selektivitätsformel aus Abschnitt 2.5 für räumliche Bereichsanfragen vorgestellt. Genaue Herleitungen und erweiterte Darstellungen zur Selektivität können bei Papadias et al. [60] nachgelesen werden.

4.9.1 Selektivitätsabschätzungen von räumlichen Prädikaten

Gegeben sei eine Relation R mit k-dimensionalen, gleichverteilten Geometrien G in einem rechteckigen Wertebereich $\mathsf{r}(\mathsf{G},\mathsf{R})$ und ein Anfragefenster W. Für die Selektivitätsberechnung einer gegebenen räumlichen Bedingung $\varphi_{\mathsf{geo}} := \mathsf{G} \cap \mathsf{W} \neq \emptyset$ auf der Relation R werden die MBRs der Geometrien verwendet. Die Selektivität von φ_{geo} im Anfragefenster W ergibt sich zu:

$$\mathsf{sel}\left(\varphi_{\mathsf{geo}},\mathsf{R}\right) := \prod_{\mathsf{d}=1}^{\mathsf{k}} \mathsf{min}\left(1, \frac{\overline{\mathsf{s_d}}(\mathsf{G},\mathsf{R}) + \overline{\mathsf{W_d}}}{\overline{\mathsf{r_d}}(\mathsf{G},\mathsf{R})}\right)$$

Dabei ist $\overline{\mathsf{s_d}}(\mathsf{G},\mathsf{R})$ die Durchschnittslänge der Projektionen aller MBRs s von $\mathsf{G} \in \mathsf{R}$ auf die Dimension d. $\overline{\mathsf{W_d}}$ und $\overline{\mathsf{r_d}}(\mathsf{G},\mathsf{R})$ sind korrespondierende Projektionen für W und $\mathsf{r}(\mathsf{G},\mathsf{R})$. Das Produkt stellt die Abschätzung der Selektivität einer Fensteranfrage, d. h. die Wahrscheinlichkeit, dass ein beliebiges MBR aus R das Fenster W schneidet, dar. Diese Wahrscheinlichkeit für die Dimension d gleicht der Summe der Projektionen von $\overline{\mathsf{s_d}}(\mathsf{G},\mathsf{R})$ und $\overline{\mathsf{W_d}}$, normalisiert auf den Wertebereich.

Verallgemeinert auf den Verbund zweier Relationen $\mathsf{R_1}$ mit den Geometrien $\mathsf{G_1}$, $\mathsf{R_2}$ mit den Geometrien $\mathsf{G_2}$ und der Bedingung $\varphi_{\mathsf{geo}} := \mathsf{G_1} \cap \mathsf{G_2} \neq \emptyset$, sowie dem Wertebereich $\mathsf{r}(\mathsf{G_1} \cup \mathsf{G_2}, \mathsf{R_1} \times \mathsf{R_2}) = \mathsf{r}(\mathsf{G_1},\mathsf{R_1}) \cup \mathsf{r}(\mathsf{G_2},\mathsf{R_2})$ ergibt sich:

$$\mathsf{sel}\left(\varphi_{\mathsf{geo}},\mathsf{R_1} \times \mathsf{R_2}\right) := \prod_{\mathsf{d}=1}^{\mathsf{k}} \mathsf{min}\left(1, \frac{\overline{\mathsf{s_d}}(\mathsf{G_1},\mathsf{R_1}) + \overline{\mathsf{s_d}}(\mathsf{G_2},\mathsf{R_2})}{\overline{\mathsf{r_d}}(\mathsf{G_1} \cup \mathsf{G_2}, \mathsf{R_1} \times \mathsf{R_2})}\right)$$

Durch die Annahme der Gleichverteilung der MBRs werden diese Abschätzungen mit steigender Anzahl der Dimensionen schlechter. Deswegen sollten histogrammbasierte Abschätzungen wie in Acharya et al. [2] (1999) oder Aboulnaga et al. [1] (2000) verwendet werden.

Schließlich lassen sich alle topologischen Beziehungsoperatoren auf den Schnitt zweier Geometrien zurückführen, siehe dazu Papadias et al. [66] (2001).

4.9.2 Abschätzung von Kardinalitäten und Seitenanzahlen

Die Kardinalität und Seitenanzahl einer Ergebnisrelation ergeben sich bekannterweise aus der Selektivität des Operators multipliziert mit der Kardinalität bzw. Seitenanzahl der Argumentrelation. Demzufolge verwendet man die Abschätzungen aus dem Abschnitt 4.9.1, um analog zu 2.5.1 diese Werte zu ermitteln.

4.9.3 Schätzungen für Attributwertanzahlen

Es wird davon ausgegangen, dass jede Geometrie nur jeweils einmal in der Relation vorkommt. Die Attributwertanzahl ist dann gleich der Kardinalität der Relation (siehe Tabelle 4.5).

Attributwertanzahl	Größe
$\mathsf{n}(\mathsf{geo},\mathsf{R})$	$\lvert\mathsf{R}\rvert$

Tabelle 4.5: Attributwertverteilungen bei geometrischen Attributen

4.9.4 Abschätzungen von Tupellängen

Da geometrische Attribute in Relationen abgespeichert werden, wird auch hier von gleich langen Attributlängen ausgegangen. Somit können auch für räumliche Datenbanken die Abschätzungen aus Kapitel 2.6 verwendet werden.

4.10 Räumliche Kostenfunktionen

Nachfolgend werden die Kostenfunktionen für räumliche Operatoren vorgestellt. Allgemein kann man sagen, dass die neuen Operatoren in Beziehung mit den bisherigen Standardoperatoren gesetzt werden müssen, wenn beide Operatortypen verwendet und abgeschätzt werden sollen. Dies geschieht, indem teure räumliche Prädikate speziell gewichtet werden. Dementsprechend werden räumliche Selektionen und Joins, wie in Tabelle 4.4 abgebildet, durch entsprechende relationale Repräsentanten ersetzt und ihr Aufwand mit Hilfe der in Kapitel 2 vorgestellten Kostenformeln abgeschätzt.

4.10.1 Gewichtungsfaktor für objektrelationale Operatoren

Durch die Erweiterung der Datentypen der Argumentrelationen ist die Einführung eines Gewichtungsfaktors zwingend erforderlich. Da diese nicht mehr nur aus rein alphanumerischen Werten bestehen, sondern Geometrien sein können, kann man keine einfachen Aussagen mehr bezüglich der Kosten treffen, sondern muss für jedes Prädikat diesen Faktor neu abschätzen. Generell kann man sagen, dass die Ausführungszeiten proportional mit der Erhöhung der Komplexität der Attribute (im räumlichen Fall der Geometrieattribute) steigen. Zusätzlich müssen die unterschiedlichen Laufzeiten von verschiedenen räumlichen Prädikaten berücksichtigt werden. Auch sie beeinflussen stark die Ausführungszeit eines räumlichen Operators.

Definition 4.10.1 Wir unterscheiden folgende drei Gewichtungsfaktoren:

1) y_{ij}^{top} gewichtet die Komplexität der Argumentgeometrien bei gleicher topologischer Operation

2) x^{top} schätzt die verschiedenen Komplexitäten zwischen den topologischen Operatoren ab

3) s^{e} gibt das systembedingte Verhältnis zwischen einem alphanumerischen und einem geometrischen Attributzugriff an

Um den Gewichtungsfaktor y_{ij}^{top} zu bestimmen, wurden jeweils für jede topologische Funktion und jede Kombination von Geometrietypen Anfragen gestellt und die dazugehörige Laufzeit gemessen. Dabei wurde eine Normierung hinsichtlich beliebiger Paare von Punktgeometrien vorgenommen[8], indem wir die sich ergebende Bearbeitungsdauer als Grundeinheit nahmen und dazu den Abgleich aller weiteren möglichen Geometrietypen ins Verhältnis setzten. Damit ergeben sich beispielsweise auf Basis unserer räumlichen ATKIS Daten (siehe Abschnitt 6.3.3) die in Tabelle 4.6 aufgeführten Werte für y_{ij}^{EQUAL}. Somit dauert der Abgleich eines beliebigen Polygons mit einem Multipolygon 3,2-mal länger als der Vergleich zweier Punkte. Die Tabellen für alle in dieser Arbeit berücksichtigten topologischen Funktionen sind im Anhang C zu finden.

Obwohl die topologische Anfrage A CONTAINS B der Anfrage B INSIDE A entspricht, sind die getesteten Laufzeiten mit identischen Daten sichtbar unterschiedlich. Somit ist also schon die Wahl

[8] Es wurde der Geometrietyp Punkt gewählt, da es sich hierbei um den am wenigsten komplexen Typ handelt.

der Funktion und deren Ersetzung durch eine äquivalente Variante eine Optimierungsmöglichkeit.[9]

EQUAL	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,3	1,2	1,1	1,3	1,8
line	1,1	1,2	1,3	1,2	1,1	1,3	1,7
polygon	1,2	1,7	1,9	1,9	1,4	2,3	3,2
collection	1,1	1,2	1,4	1,3	1,2	1,5	1,9
multipoint	1,2	1,3	1,4	1,3	1,2	1,4	1,8
multiline	1,3	1,4	1,6	1,6	1,4	1,7	2,0
multipolygon	1,2	1,3	1,5	1,4	1,3	1,5	1,9

Tabelle 4.6: Standardwerte für die topologische Funktion **EQUAL**

Zur Bestimmung des Gewichtungsfaktors $\mathsf{x}^{\mathsf{top}}$ für topologische Beziehungsoperatoren wurde die topologische Funktion EQUAL verwendet. Dafür wurde auf Basis von Punktpaaren für jede topologische Beziehung die Laufzeit ins Verhältnis gesetzt (siehe Tabelle 4.7). Somit dauerte das Testen zweier beliebiger Punkte A und B bezüglich TOUCH 1,28-mal länger als die Prüfung auf Gleicheit mit EQUAL.

Mit Hilfe des systembedingten Gewichtungsfaktors s^{ϵ} können unterschiedliche Zugriffsmöglichkeiten auf ein räumliches Attribut abgeschätzt werden. Folgende Unterteilungen werden in dieser Arbeit vorgenommen:

Funktion (Topologie)	Standardwert $\mathsf{x}^{\mathsf{top}}$
DISJOINT (disjoint)	1,60
TOUCH (meet)	1,28
OVERLAPBDYDISJOINT	1,08
OVERLAPBDYINTERSECT	1,08
EQUAL (equal)	1,00
CONTAINS (contains)	1,15
INSIDE (covers)	1,00
COVERS (inside)	1,05
COVEREDBY	0,74
ON	0,98
ANYINTERACT	1,29
DETERMINE	1,48

Tabelle 4.7: Topologische Beziehungsoperatoren

$\mathsf{s}^{\mathsf{Rel}}$	Systemabhängige Geometriezugriffsgewichtung für einen relationalen Scan (z. B. $\mathsf{s}^{\mathsf{Rel}} = 33$, siehe Abschnitt 6.3.3)
$\mathsf{s}^{\mathsf{Sort}}$	Systemabhängige Geometriezugriffsgewichtung für einen relationalen Scan über eine sortierte Argumentrelation
$\mathsf{s}^{\mathsf{Index}}$	Systemabhängige Geometriezugriffsgewichtung für einen indexbasierten Scan (z. B. $\mathsf{s}^{\mathsf{Index}} = 11$, siehe Abschnitt 6.3.3)
$\mathsf{s}^{\mathsf{Hash}}$	Systemabhängige Geometriezugriffsgewichtung für einen hashbasierten Scan
$\mathsf{s}^{\mathsf{Index,Rel}}$	Systemabhängige Geometriezugriffsgewichtung für einen indexbasierten Scan auf die erste und einen relationalen auf die zweite Argumentrelation (z. B. $\mathsf{s}^{\mathsf{Index,Rel}} = 11$, siehe Abschnitt 6.3.3)

Mit diesen Informationen besitzen wir alle Faktoren, um $\omega^{\epsilon}_{(\varphi_{\mathsf{geo}},\mathsf{R})}$ für objektrelationale Operatoren definieren zu können:

Definition 4.10.2 Der Proportionalitätsfaktor für topologische Beziehungsfunktionen berechnet sich abhängig von der Anzahl der Argumentrelationen wie folgt:

[9] Beim Versuch, Kostenformeln anhand der Algorithmenlaufzeiten für topologische Funktionen (z. B. aus [51]) abzuleiten, kamen widersprüchliche Aussagen zu den durch Stichproben und ausgiebigen Tests (siehe Abschnitt 6.3.3) ermittelten Gewichtungsfaktoren heraus. Somit wurde dieser Ansatz verworfen.

$$\omega^{\epsilon}_{(\varphi_{\mathsf{geo}},\mathsf{R}_1\times\mathsf{R}_2)} := \mathsf{s}^{\epsilon}\cdot \mathsf{x}^{\mathsf{top}}\cdot \sum_{\mathsf{i}=1}^{7}\mathsf{sel}(\mathsf{gt}(\mathsf{A})=\mathsf{dl0i},\mathsf{R}_1)\cdot\sum_{\mathsf{j}=1}^{7}\mathsf{sel}(\mathsf{gt}(\mathsf{B})=\mathsf{dl0j},\mathsf{R}_2)\cdot \mathsf{y}^{\mathsf{top}}_{\mathsf{ij}}$$

mit $\epsilon \in \{\mathsf{Rel},\mathsf{Index},\mathsf{Hash},\{\mathsf{Rel},\mathsf{Index}\}\}$

$$\omega^{\epsilon}_{(\varphi_{\mathsf{geo}},\mathsf{R})} := \mathsf{s}^{\epsilon}\cdot \mathsf{x}^{\mathsf{top}}\cdot \sum_{\mathsf{i}=1}^{7}\mathsf{sel}(\mathsf{gt}(\mathsf{A})=\mathsf{dl0i},\mathsf{R})\cdot \mathsf{y}^{\mathsf{top}}_{\mathsf{ij}}$$

mit $\epsilon \in \{\mathsf{Rel},\mathsf{Index}\}$

mit j als gegebener Geometrietyp

Mit Hilfe des jetzt definierten Gewichtungsfaktors $\omega^{\epsilon}_{(\varphi_{\mathsf{geo}},\mathsf{R})}$ ist es jetzt möglich, Zugriffskosten für relationale Operatoren ins Verhältnis mit räumlichen zu setzen.

Definition 4.10.3 Die Kosten für einen räumlichen Operatorknoten τ ergeben sich aus der Gewichtung des räumlichen Operators $\omega^{\epsilon}_{(\varphi_{\mathsf{geo}},\mathsf{R})}$ multipliziert mit den Kosten des relationalen Operators:

$$\mathsf{cost}^{+}_{\mathsf{op}}(\tau) := \omega^{\epsilon}_{(\varphi_{\mathsf{geo}},\mathsf{R})}\cdot \mathsf{cost}_{\mathsf{op}}(\tau)$$

4.10.2 Räumliche Selektion

In der Tabelle 4.8 werden zwei räumliche Zugriffsvarianten für den Selektionsoperator abgeschätzt. Dabei kann man per relationalem Scan $\sigma^{\mathsf{Rel}}(\mathsf{R})$ die komplette Relation R einmal tupelweise abarbeiten. Mit Hilfe eines Indexes lassen sich je nach Existenz ebenfalls Kosten gegenüber dem kompletten Zugriff einsparen. Die Art des räumlichen Indexes kann durch Anpassung der Werte für den systembedingten Zugriffsfaktor s und daraus folgernd $\omega^{\mathsf{Index}}_{(\varphi_{\mathsf{geo}},\mathsf{R})}$ berücksichtigt werden.

τ	$\mathsf{cost}^{+}_{\mathsf{op}}(\tau) = \omega^{\epsilon}_{(\varphi_{\mathsf{geo}},\mathsf{R})}\cdot \mathsf{cost}_{\mathsf{op}}(\tau)$
$\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}(\mathsf{R})$	$\omega^{\mathsf{Rel}}_{(\varphi_{\mathsf{geo}},\mathsf{R})}\cdot \mathsf{cost}_{\mathsf{op}}\left(\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}(\mathsf{R})\right)$
$\sigma^{\mathsf{Index_p}}_{\varphi_{\mathsf{geo}}}(\mathsf{R})$	$\omega^{\mathsf{Index}}_{(\varphi_{\mathsf{geo}},\mathsf{R})}\cdot \mathsf{cost}_{\mathsf{op}}\left(\sigma^{\mathsf{Index_p}}_{\varphi_{\mathsf{geo}}}(\mathsf{R})\right)$

Tabelle 4.8: Selektionskosten

4.10.3 Räumlicher Join

Im Gegensatz zu einem relationalen Verbund ist der räumliche Join durch die verschiedenen Filterstufen unterschiedlich komplex aufgebaut (siehe Tabelle 4.4). Beispielsweise muss bei der Nutzung eines räumlichen Indexes immer eine zweite Selektion nachgeschaltet werden, da beim Zugriff darüber keine exakten Ergebnisse zurückgeliefert werden.

τ	$\mathsf{cost}^{+}_{\mathsf{op}}(\tau) = \omega^{\epsilon}_{(\varphi_{\mathsf{geo}},\mathsf{R}_1\times\mathsf{R}_2)}\cdot \mathsf{cost}_{\mathsf{op}}(\tau)$
$\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	$\omega^{\mathsf{Rel,Rel}}_{(\varphi_{\mathsf{geo}},\mathsf{R}_1\times\mathsf{R}_2)}\cdot \mathsf{cost}_{\mathsf{op}}\left(\mathsf{R}_1 \bowtie^{\mathsf{Rel},\,\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2\right)$
$\mathsf{R}_1 \bowtie^{\mathsf{Index},\,\mathsf{Rel_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	$\omega^{\mathsf{Index,Rel}}_{(\varphi_{\mathsf{geo}},\mathsf{R}_1\times\mathsf{R}_2)}\cdot \mathsf{cost}_{\mathsf{op}}\left(\mathsf{R}_1 \bowtie^{\mathsf{Index},\,\mathsf{Rel_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2\right)$
$\mathsf{R}_1 \bowtie^{\mathsf{Index},\,\mathsf{Index_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2$	$\omega^{\mathsf{Index,Index}}_{(\varphi_{\mathsf{geo}},\mathsf{R}_1\times\mathsf{R}_2)}\cdot \mathsf{cost}_{\mathsf{op}}\left(\mathsf{R}_1 \bowtie^{\mathsf{Index},\,\mathsf{Index_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2\right)$

Tabelle 4.9: Kosten von Joins

Diese Definition der Kostenformeln für räumliche Operatoren fügt sich nahtlos an die bisherigen Kostenformeln an. Möchte man z. B. die Kosten eines räumlichen Index-Index-Joins abschätzen, folgt:

$$\mathsf{cost}^{+}\left(\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}\left(\mathsf{R}_1 \bowtie^{\mathsf{Index,Index_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2\right)\right) = \omega^{\mathsf{Rel}}_{(\varphi_{\mathsf{geo}},\mathsf{R}_1\times\mathsf{R}_2)}\cdot \mathsf{cost}_{\mathsf{op}}\left(\sigma^{\mathsf{Rel_s}}_{\varphi_{\mathsf{geo}}}\left(\mathsf{R}_1 \bowtie^{\mathsf{Index,Index_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2\right)\right) +$$
$$\omega^{\mathsf{Index,Index}}_{(\varphi_{\mathsf{geo}},\mathsf{R}_1\times\mathsf{R}_2)}\cdot \mathsf{cost}_{\mathsf{op}}\left(\mathsf{R}_1 \bowtie^{\mathsf{Index,Index_p}}_{\varphi_{\mathsf{geo}}} \mathsf{R}_2\right)$$

4.10.4 Pipelining

Räumliche Operatoren müssen als „blocking“-Operatoren betrachtet werden, da in unserem Kostenmodell nur Input-Kosten eine Rolle spielen. Wäre dies nicht der Fall, dann würde z. B. das „unnötige“ Einfügen einer relationalen Selektion die I/O-Kosten des nachfolgenden räumlichen Operators neutralisieren können. Aus diesem Grund wird generell kein Pipelining von relationalen zu räumlichen Operatoren ermöglicht. Ausgehende Tupel können natürlich weitergeleitet werden.

4.10.5 Vererbung von Sortierungen und Indexen

Es existiert bislang keine von kommerziellen Datenbanken unterstützte Ordnung für Geometrien. Somit werden in dieser Arbeit keine Sortierungen darauf aufgebaut, die demzufolge auch nicht vererbt werden können. Wie im relationalen Fall werden auch hier keine Indexe vererbt, da verwendete Indexe sich auf obsolete Relationen beziehen würden.

4.11 Beispiel für eine objektrelationale Kostenberechnung

Um die einfache Anbindung der Kostenfunktionen für objektrelationale Operatoren an die relationalen zu verdeutlichen, wird auch in diesem Beispiel die Anordnung einer Selektion vor und nach einem Join untersucht. Diesmal wird neben einem numerischen Join auch eine geometrische Selektion durchgeführt. Mit einem gegebenen Rechteck W und einem vorhandenen räumlichen Attribut G aus R_2 folgt:[10]

$$\sigma_{\varphi_1}(R_2) \bowtie_{\varphi_2} R_1 \text{ mit } \varphi_1 := \mathsf{W\ ANYINTERACT\ G} \text{ und } \varphi_2 := R_1.\mathsf{objektid} = R_2.\mathsf{objektid}$$

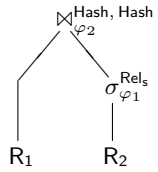
Statistiken:

$\lvert R_1\rvert$	$= 15.285$	$\mathsf{sel}(\mathsf{gt(L)} = \mathsf{line}, R_1)$	$= 1$	$\overline{r_1}(G, R_2)$	$= 22.479,75$
$\mathsf{pg}(R_1)$	$= 668$	$\mathsf{sel}(\mathsf{gt(G)} = \mathsf{point}, R_2)$	$= 0,012$	$\overline{r_2}(G, R_2)$	$= 22.514,254$
$\mathsf{n}(\mathsf{objektid}, R_1)$	$= 15.285$	$\mathsf{sel}(\mathsf{gt(G)} = \mathsf{line}, R_2)$	$= 0,495$	$\overline{w_1}$	$= 1.972$
$\mathsf{n}(L, R_1)$	$= 15.285$	$\mathsf{sel}(\mathsf{gt(G)} = \mathsf{polygon}, R_2)$	$= 0,492$	$\overline{w_2}$	$= 3.650$
$\lvert R_2\rvert$	$= 30.891$	$\mathsf{sel}(\mathsf{gt(G)} = \mathsf{multipoint}, R_2)$	$= 0$	$\mathsf{gt(P)}$	$=$ polygon
$\mathsf{pg}(R_2)$	$= 2.363$	$\mathsf{sel}(\mathsf{gt(G)} = \mathsf{multiline}, R_2)$	$= 0$	s^{Rel}	$= 33$
$\mathsf{n}(\mathsf{objektid}, R_2)$	$= 30.818$	$\mathsf{sel}(\mathsf{gt(G)} = \mathsf{multipolygon}, R_2)$	$= 0,001$	x	$= 1,29$
$\mathsf{n}(\mathsf{objektnr}, R_2)$	$= 24.755$	$\mathsf{sel}(\mathsf{gt(G)} = \mathsf{collection}, R_2)$	$= 0$	w_1	$= 0,02$
$\mathsf{n}(G, R_2)$	$= 30.891$	$\overline{s_1}(G, R_2)$	$= 189,968$	w_2	$= 0$
$\mathsf{n}(\mathsf{objektart}, R_2)$	$= 54$	$\overline{s_2}(G, R_2)$	$= 177,782$		

In diesem Fall handelt es sich um einen Gleichheitsverbund mit Hilfe eines Hash-Joins über den Primärschlüssel, so dass aus dem Join maximal die Anzahl von Tupeln herauskommt, die R_1 besitzt. Die benötigten Werte für $y_{ij}^{\mathsf{ANYINTERACT}}$ befinden sich in den Tabellen im Anhang C.

1. Der heuristisch optimierte Ausführungsplan τ_1 und seine Berechnung in zwei Schritten:

[10] Das nachfolgende Beispiel entspricht einer realen Anfrage mit ATKIS-Daten vom Raum Buchholz bei Hannover, Deutschland in der Auflösung 1:25.000.



$$\begin{aligned}\omega^{\{Rel\}}_{(\varphi_1,R_2)} &= s \cdot x^{ANYINTERACT} \cdot \sum_{i=1}^{7} sel(gt(G) = dl0i, R_2) \cdot y^{ANYINTERACT}_{i\,POLYGON} \\ &= 33 \cdot 1,29 \cdot (1,2 \cdot 0,012 + 1,3 \cdot 0,495 + 1,4 \cdot 0,492 + 2,0 \cdot 0,001) \\ &= 33 \cdot 1,29 \cdot 1,3487 \\ &\approx 57,414\end{aligned}$$

Berechnung der Selektionskosten mit $\tau_{1_1} := \sigma^{Rel_s}_{\varphi_1}(R_2)$:

$$\begin{aligned}cost_{op}(\tau_{1_1}) &= pg(R_2) + w_1 \cdot |R_2| \\ &= 2.363 + 0,02 \cdot 30.891 \\ &\approx 2.980,82\end{aligned} \qquad \Rightarrow cost^{+}_{op}(\tau_{1_1}) = \omega^{\{Rel\}}_{(\varphi_1,R_2)} \cdot cost_{op}(\tau_{1_1}) = 57,414 \cdot 2.980,82 \approx 171.140,799$$

Berechnung der Selektivität $sel(\varphi_1, R_2)$:

$$\begin{aligned}sel(\varphi_1, R_2) &= \min\left(1, \frac{\overline{s_1}(G,R_2) + \overline{W_1}}{\overline{r_1}(G,R_2)}\right) \cdot \min\left(1, \frac{\overline{s_2}(G,R_2) + \overline{W_2}}{\overline{r_2}(G,R_2)}\right) \\ &= \min\left(1, \frac{189,968 + 1.972}{22.479,75}\right) \cdot \min\left(1, \frac{177,782 + 3.650}{22.514,254}\right) \\ &= 0,096174023 \cdot 0,170015937 \\ &= 0,016351117\end{aligned}$$

Berechnung der Kardinalität, Seitengröße und relevanter Attributwertanzahlen von τ_{1_1};

$$\begin{aligned}|\tau_{1_1}| &= sel(\varphi_1, R_2) \cdot |R_2| \\ &= 0,016351117 \cdot 30.891 \\ &\approx 505,102\end{aligned} \qquad \begin{aligned}pg(\tau_{1_1}) &= \frac{pg(R_2) \cdot |\tau_{1_1}|}{|R_2|} = \frac{2.363 \cdot 505,102}{30.891} = 38,638 \\ n(objektid, \tau_{1_1}) &= 505,102\end{aligned}$$

Kosten für den nachfolgenden Hash-Join mit $\tau_{1_2} := R_1 \bowtie^{Hash,\,Hash}_{\varphi_2} \tau_{1_1}$:

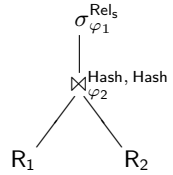
$$\begin{aligned}cost_{op}(\tau_{1_2}) &= pg(R_1) + pg(\tau_{1_1}) + |R_1| \cdot \frac{pg(\tau_{1_1})}{n(objektid, \tau_{1_1})} \\ &+ w_1 \cdot (n(objektid, R_1) + n(objektid, R_2) + |R_1| \cdot \frac{|\tau_{1_1}|}{n(objektid, \tau_{1_1})} + |R_1| + |\tau_{1_1}|) \\ &= 668 + 38,638 + 15.285 \cdot \frac{38,638}{505,102} \\ &+ 0,02 \cdot (15.285 + 505,102 + 15.285 \cdot \frac{505,102}{505,102} + 15.285 + 505,102) \\ &\approx 2.813,175\end{aligned}$$

Es ergeben sich die Gesamtkosten zu:

$$cost(\tau_1) = cost^{+}_{op}(\tau_{1_1}) + cost_{op}(\tau_{1_2}) = 171.140,799 + 2.813,175 = \mathbf{173.953,974}$$

2. Nicht heuristisch optimierter Ausführungsplan τ_2 mit Berechnung:

Berechnung der Kosten für den anfänglichen Hash-Join mit $\tau_{2_1} := R_1 \bowtie^{Hash,\,Hash}_{\varphi_2} R_2$:



$$\begin{aligned}cost_{op}(\tau_{2_1}) &= pg(R_1) + pg(R_2) + |R_1| \cdot \frac{pg(R_2)}{n(objektid, R_2)} \\ &+ w_1 \cdot (n(objektid, R_1) + n(objektid, R_2) + |R_1| \cdot \frac{|R_2|}{n(objektid, R_2)} + |R_1| + |R_2|) \\ &= 668 + 2.363 + 15.285 \cdot \frac{2.363}{30.891} \\ &+ 0,02(\cdot 15.285 + 30.891 + 15.285 \cdot \frac{30.891}{30.891} + 15.285 + 30.891) \\ &\approx 6.352,963\end{aligned}$$

$$|\tau_{2_1}| = |R_1| = 15.285$$

$$pg(\tau_{2_1}) = pg(R_1) + \frac{pg(R_2)\cdot|\tau_{2_1}|}{|R_2|} = 1.837,222$$

$$\begin{aligned}\omega^{\{Rel\}}_{(\varphi_1,R_2)} &= s \cdot x^{ANYINTERACT} \cdot \sum_{i=1}^{7} sel(gt(G) = dl0i, R_2)\\ &= 33 \cdot 1,29 \cdot (1,3 \cdot 1)\\ &\approx 55,341\end{aligned}$$

Berechnung der Kosten für die räumliche Selektion mit $\tau_{2_2} := \sigma^{Rel_s}_{\varphi_1}(R_1 \bowtie_{\varphi_2} R_2)$:

$$\begin{aligned}cost_{op}(\tau_{2_2}) &= pg(\tau_{2_1}) + w_1 \cdot |\tau_{2_1}|\\ &= 1837,222 + 0,02 \cdot 15.285\\ &= 2142,922\end{aligned}$$

$$\begin{aligned}\Rightarrow cost^{+}_{op}(\tau_{2_2}) &= \omega^{\{Rel\}}_{(\varphi_1,\tau_{2_1})} \cdot cost_{op}(\tau_{2_2})\\ &= 55,341 \cdot 2.142,922 \approx 118.591,446\end{aligned}$$

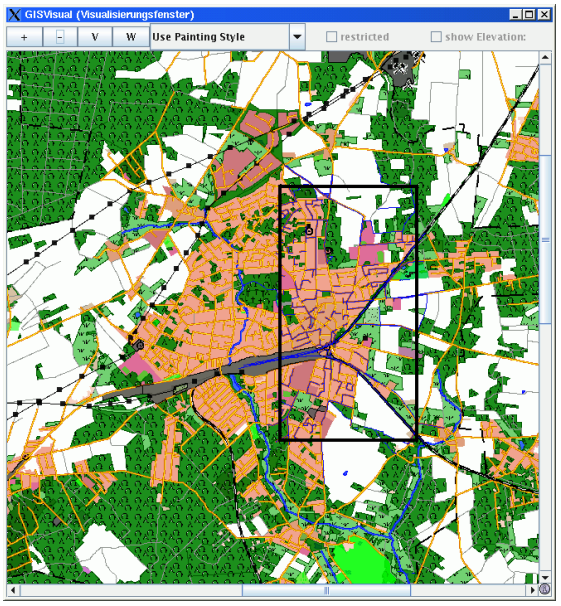


Abbildung 4.7: Visuelle Darstellung der räumlichen Beispielanfrage mit Hilfe von GisVisual 7.0[5]

Die Gesamtkosten addieren sich folglich zu:

$$cost(\tau_2) = cost_{op}(\tau_{2_1}) + cost^{+}_{op}(\tau_{2_2}) = 6.352,963 + 118.591,446 = \mathbf{124.944,409}$$

Somit ist der zweite Ausführungsplan rund 28% schneller zu berechnen als der erste. Bei einem Test auf realen Daten (siehe Tabellen im Anhang D.2.9) ergaben sich vergleichbare Performancegewinne wie bei diesem theoretischen Beispiel.

4.12 Fazit der Erweiterung des Kostenmodells auf räumliche Operatoren

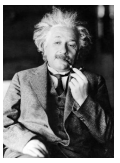
Die leichte Erweiterbarkeit des vorgestellten Optimierungskonzeptes wurde mit den in diesem Kapitel aufgelisteten räumlichen Operatoren und den dazugehörigen Abschätzungen verdeutlicht. Als besondere Erweiterung wurden Proportionalitätsfaktoren für Berechnungskosten eingeführt, die relationale Operatoren mit Repräsentanten räumlicher Operatoren in Beziehung setzen können. Somit besitzen wir jetzt die Möglichkeit, auch Ausführungspläne zu bewerten, die sowohl räumliche als auch relationale Operatoren verwenden.

Darüber hinaus wurde das zusätzliche Selektieren von Geometrietypen als weitere Regeln eingeführt, der mit Hilfe von Termersetzungsregeln die Gesamtkosten eines Ausführungsplanes senken kann, obwohl sie unter Umständen lokal steigen. Diese Vorfilterung ist nur bei räumlichen Operatoren sinnvoll und bietet demzufolge, vergleichbar mit den bisherigen Operatoren wie der Sortierung oder Indexbildung, weiteres Optimierungspotential für räumliche Anfragen.

[11] WiPKA–Entwicklung eines Konzeptes und eines Prototyps für die Modellierung und Verwaltung raumbezogener Daten mit multiplen Repräsentationen (`http://www.dbs.uni-hannover.de/wipka.0.html`)

Der Horizont vieler Menschen ist ein Kreis mit Radius Null - und das nennen sie ihren Standpunkt.

Albert Einstein 1879-1955



5 Der bedingte Termersetzungsoptimierer

Dieses Kapitel beschäftigt sich mit den bisher nicht näher beschriebenen Aspekten des Termersetzungsoptimierers. Es wird eine Regelsprache definiert, die flexibel ist und jede beliebige Anordnung von Regeln erlaubt. Die Steuerung des Optimierers und dessen Variantengenerierung bilden den zweiten Teil dieses Kapitels. Des Weiteren bekommt auch der modulare Aufbau der einzelnen Optimierungseinheiten seine Darstellung.

Ausgehend von einer Menge von Regeln und einer Ausgangsvariante des zu optimierenden Ausführungsplanes, ist es jetzt die Aufgabe des Termersetzungsoptimierers, auf Basis dieser Regeln eine Optimierung durchzuführen. Dafür werden im Folgenden die einzelnen Komponenten des Optimierungsablaufes beschrieben.

5.1 Das Matching von generalisierten Regeln

Da generalisierte Regeln (siehe Abschnitt 3.3) als einfache generalisierte Bäume (siehe Abschnitt 2.10) abgespeichert werden, wird während des Untersuchens auf mögliche Anwendbarkeit (Matching) auf gleiche Baumstrukturen verglichen. Technisch wird eine Unifikation vorgenommen, wie z. B. von der Programmiersprache Prolog her bekannt. Dabei werden alle Knoten des generalisierten Baumes einzeln mit der Wurzel der Ersetzungsregel verglichen. Als Suchrichtung wird ein simpler bottom-up Durchlauf innerhalb des Ausführungsplanes vorgenommen. Falls beide Bäume den gleichen Operator beschreiben, wird zusätzlich auf die physische Implementierung geachtet. Liegt auch hier Gleichheit vor, dann werden die einzelnen Kinderknoten rekursiv betrachtet, bis eine strukturelle Übereinstimmung beim Erreichen des letzten zu vergleichenden Knotens festgestellt worden ist.

Ist der syntaktische Abgleich erfolgreich verlaufen, werden die Bäume semantisch verglichen. Dabei werden gleiche Variablennamen auf Gleichheit der Terme überprüft. Damit kann man Ersetzungsregeln, die mehrfach denselben Teilbaum erwarten, von den Regeln, die beliebige Teilbäume verarbeiten können, beim Matching unterscheiden.

Wir sprechen im weiteren Text von einem Iterationsschritt, wenn die gesamte anzuwendende Regelmenge (Regelliste) benutzt wurde. Für jede Regel aus dieser Liste durchwandert der Termersetzer einen Ausführungsplan per bottom-up Suche, um dabei herauszufinden, ob eine Regel von der vorher festgelegten Regelliste angewendet werden kann. Falls für eine Regel alle noch zu überprüfenden Bedingungen erfüllt sind und mit ihrer Hilfe ein neuer, bisher unbekannter Ausführungsplan generiert werden kann, wird er zusätzlich abgespeichert. Eine in einem Schritt erzeugte Variante eines gegebenen Terms wird erst in der nächsten Iteration weiter betrachtet, um sicherzustellen, dass alle möglichen Varianten erzeugt werden.

Eine nicht im Sinne von Regelbedingungen, aber dennoch zu kontrollierende Bedingung, ist die Konsistenzprüfung der zu speichernden Ausführungspläne. Hauptsächlich muss sichergestellt werden, dass noch alle verwendeten Implementierungsarten gültig sind. Wurden z. B. unterhalb eines Operatorknotens Sortierungen oder Indexe zerstört, können sie natürlich nicht nur vom direkt „darüberliegenden“ Knoten, sondern auch für alle anderen nachfolgenden Operatoren nicht

mehr genutzt werden. In solchen Fällen kommt es zu einer kostenbasierten Korrektur des Planes, die der generalisierte Baum (siehe Abschnitt 2.10) autonom auf Basis der gegeben Statistiken durchführt. Grund dafür ist die Erkenntnis, dass die Konsistenzprüfung eines Ausführungsplanes durch einfache lokale Regeln nicht durchgeführt werden kann. Falls man dies nicht berücksichtigt und Varianten von Ausführungsplänen als ungültig markiert und „wegwirft“, ist wegen des top-down-Durchlaufes des Baumes nicht mehr sichergestellt, dass jeder äquivalente Ausführungsplan generiert wird.

5.1.1 Auswerten von Bedingungen: pre- und postconditions

Definition 5.1.1 Es wird zwischen zwei verschiedenen Arten von Bedingungen unterschieden: den Vorbedingungen bzw. Prämissen (engl. *preconditions*), die vor einer Termersetzung überprüft werden können, und den nach der Substitution überprüfbaren Nachbedingungen (engl. *postconditions*).

Eine Termersetzung wird erst dann ausgeführt, wenn alle Vorbedingungen (preconditions) einer Regel erfüllt wurden. Liegt ein Matching der Regel mit dem Ausführungsplan vor, dann werden mögliche vorhandene Variablen in den Blättern der Bäume, die die Terme darstellen, initialisiert. Bei der anschließenden Auswertung der Bedingungen beziehen sich die Variablen der Bedingungen auf die zugewiesenen Werte.

Zusätzlich dazu existieren Bedingungen, die sich auf das Ergebnis von Ersetzungen beziehen (z. B. $\mathsf{cost}(\tau_1) < \mathsf{cost}(\tau_2)$), die nicht vor der Erzeugung von τ_2 überprüft werden können (postconditions). Dabei liegt das größere Optimierungspotential eindeutig bei den Vorbedingungen, da sie die Erzeugung von neuen Ausführungsplänen schon im Vorfeld unterbinden.

5.1.2 Definition von Varianten eines Ausführungsplanes

Jede Termersetzungsregel, die auf einen Ausführungsplan angewendet werden kann, erzeugt eine neue äquivalente Variante des Ausführungsplanes. Zusätzlich zu dieser wird eine Liste von Referenzen auf Regeln und deren Ersetzungsrichtung abgespeichert, aus denen sie erzeugt werden konnte, damit sinnlose Ersetzungen a priori unterbunden werden. Ist z. B. Variante B mit der Regel i aus A entstanden, dann darf die Regel i nicht mehr in entgegengesetzter Richtung am selben Knoten auf die Variante B angewendet werden. Diese Erzeugungsinformation kann rekursiv für die gesamten Ersetzungsgraphen, für jede Variante in Form eines mehrstufigen Ableitungsgraphen, abgespeichert werden und dient dabei als eine der wichtigsten impliziten Vorbedingungen.

Definition 5.1.2 Varianten, die erzeugt wurden, aber nicht mehr weiter untersucht werden, da sie mindestens einer Bedingung nicht genügen, werden deaktiviert. Damit besitzt der Termersetzer zwei Sorten von Varianten:

(a) *aktive Varianten*: Diese Varianten werden vom Termersetzer im nächsten Iterationsschritt weiter betrachtet.

(b) *deaktivierte Varianten*: Hierbei handelt es sich um Varianten, die im Zuge der Termersetzung erzeugt, aber nicht weiter betrachtet werden. Zwar können sie zwischenzeitlich wieder generiert und erneut erkannt werden, sollen aber nicht aktiv zur Variantenneubildung genutzt werden. Beispielsweise könnten Varianten erzeugt werden, die mit ihren Kosten oberhalb einer gesetzten Grenze liegen und demzufolge deaktiviert werden müssen.[1]

[1] Ob eine Variante deaktiviert werden muss oder nicht (Baumvergleiche), kann effizient durch Hashing der Kosten (Hashschlüssel) eines Ausführungsplanes festgestellt werden.

Im Folgenden werden aktive Varianten mit Kreisen und deaktivierte mit Quadraten als Knoten eines Entstehungsgraphen bildlich dargestellt. Dabei entsprechen die durchgezogenen gerichteten Kanten durchgeführten und gestrichelte möglichen Regelanwendungen (siehe Abbildung 5.1).

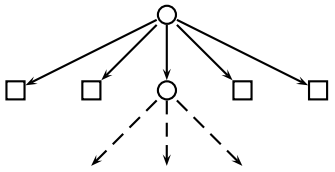


Abbildung 5.1: Deaktivierte (□) bzw. aktivierte (◯) Varianten

5.1.3 Generierung von neuen Varianten

Falls eine generalisierte Regel auf einem Teilbaum eines generalisierten Baumes angewendet werden kann, wird daraus eine neue Variante erzeugt, die anschließend auf Bekanntheit geprüft wird. Da eine Regel auf unterschiedlichen Teilbäumen eines generalisierten Baumes angewendet werden kann, werden alle Möglichkeiten der Anwendung in allen Teilbäumen in einem Iterationsschritt generiert. Damit wird gewährleistet, dass in einem Iterationsschritt keine Variante die anderen beeinflusst bzw. deren Erzeugung verhindert.

Klar ist, dass eine Umformung des Baumes neue Anwendungen erst ermöglichen oder verhindern kann, so dass dort eine strikte Trennung von bekannten und neuen Varianten nötig ist. Da nach Entstehung einer neuen Variante immer der Ausgangsbaum zur weiteren Generierung herangezogen wird, unterscheiden sich alle nachfolgenden Varianten, die in einem Iterationsschritt entstehen, nur genau in einem Teilbaum von ihrem Ausgangsbaum. Dieses bietet die Möglichkeit, eine Ordnung im Anfrageraum zu definieren, wofür wir ein Abstandsmaß zwischen den generierten Varianten benötigen:

Definition 5.1.3 Der Abstand zwischen zwei Varianten wird als die minimale Länge einer Folge von Regelanwendungen angesehen, um die eine Variante in die andere zu überführen. Dabei wird der Abstand hinsichtlich einer fest vorgegebenen Regelmenge berechnet, die für die gesamte Optimierungsphase gilt.

Somit haben alle neu generierten Varianten zu ihrer Ausgangsvariante den Abstand 1.

Satz 5.1.4 *Die maximale Anzahl von in einer Iteration neu generierten Varianten ist begrenzt durch die Anzahl von Knoten des Ausgangsbaumes* B *minus aller verwendeten Basisrelationen, multipliziert mit der Anzahl der Regeln, die auf diese Variante angewendet werden können:*

$$\mathsf{Varianten}_{\mathsf{max}}(\mathsf{B}) = |\mathsf{anwendbare\,Regeln}| \cdot (|\mathsf{Knoten}(\mathsf{B})| - |\mathsf{Basisrelationen}|)$$

Beweis: Da jede Regel an jedem Operatorknoten maximal genau einmal in einem Iterationsschritt angewendet werden kann, und wir keine Transformation der Basisrelationen vornehmen, resultiert die Anzahl der Anwendungsmöglichkeiten aus der Anzahl der Regeln |Regeln| multipliziert mit der Anzahl an Operatorknoten (|Knoten| − |Basisrelationen|).

5.2 Definition einer Regelsteuerungssprache

Wurden die Abläufe von Regeln früher fest programmiert (siehe Abschnitt 1.6), so soll mit der nachfolgenden Regelsteuerungssprache ihre Darstellung, Gruppierung und Anordnung flexibler und einfacher gemacht werden.

Bisher werden Termersetzungsregeln mit den dazugehörigen Bedingungen stets manuell definiert. Dabei müssen auch Optimierungsstrategien mit eingegeben werden können. Diese Strategien spiegeln sich in der Regelanordnung wider. Durch den Benutzer kann damit Einfluss auf die zu erzeugende Menge von Varianten in einem Iterationsschritt vorgenommen werden.

Bisherige Optimierungen beinhalten z. B. die „Push-Down Selections“ Strategie (siehe Abschnitt 3.3), welche darin besteht, dass Selektionsverschiebungsregeln so lange angewendet werden, bis Selektionen so nah wie nur möglich an den Argumentrelationen ausgeführt werden. Somit können Tupel schon frühzeitig eliminiert werden, so dass sie in der weiteren Bearbeitung von nachfolgenden Operatoren nicht betrachtet werden müssen, da sie letztlich nicht für das Ergebnis relevant sind. Klar ist, dass man mit diesem Vorgehen nicht alle möglichen Varianten eines Ausführungsplanes erzeugt. Jedoch wurde davon ausgegangen, dass eine Strategie zur Optimierung, welche die Erzeugung von kostenintensiven Ausführungsplänen nur verhindert, nicht den besten finden muss.

Eine Steuerungssprache sollte in der Lage sein, beliebige Regeln für eine unbestimmte Anzahl von Iterationsschritten anordnen zu können. Die Benutzung der nächsten Regel erfolgt, wenn entweder keine neuen Varianten durch die momentane Regel erzeugt werden, oder eine spezifizierte Anzahl von Regelanwendungen durchgeführt wurde. Als Basis für eine Regelsteuerungssprache werden hier reguläre Ausdrücke verwendet. Mit spezifischen Erweiterungen, z. B. der Terminierung von Erzeugungen von Varianten, oder ihren darauf folgenden Auswertungen, kann der benötigte Sprachumfang erreicht werden.

Definition 5.2.1 Die Reihenfolge, in welcher Regeln nacheinander auf einen gegebenen Term angewandt werden sollen, wird durch die nachfolgende Regelsteuerungssprache dargestellt:

(a) Alternative Ausführungen von Regeln werden durch einen vertikalen Strich dargestellt, z. B. $\mathsf{r}_1|\mathsf{r}_2$.

(b) Gruppen von Regeln werden durch runde Klammern zusammengefasst, z. B. $(\mathsf{r}_1|\mathsf{r}_2)|\mathsf{r}_3$.

(c) Quantoren nach einer Regel oder einer Gruppe geben an, wie oft sie angewandt werden soll:

  (1) Bei der Verwendung eines Ausrufezeichens '!' wird angezeigt, dass mindestens eine der vorherigen Regeln angewandt werden muss.

  (2) Ein Stern '*' zeigt an, dass die vorgestellten Regeln so oft angewandt werden sollen, bis sich keine neuen Varianten mehr generieren lassen.

  (3) Ein Fragezeichen '?' zeigt an, dass die vorgestellten Regeln höchstens einmal auf den Baum angewandt werden sollen.

Zusätzlich wird eine Darstellung zum Terminieren der Generierung von Varianten benötigt:

(d) Durch die Verwendung von eckigen Klammern $[...]_\mathsf{n}$ mit $\mathsf{n} \in \mathbb{N}$ kann die Bewertung und Speicherung der n-besten bisherigen Varianten festgelegt werden. Beispielsweise bekommen wir mit Hilfe des Kostenmodells bei der Angabe von $[(\mathsf{r}_1|\mathsf{r}_2|\mathsf{r}_3)^*]_{100}$ die 100 besten Varianten. Um genau einen Ausführungsplan festzulegen, wird $[...]_1$ festgesetzt.

Beispiel 5.2.2 Durch die oben definierte Regelsteuerungssprache ist eine strikte Phaseneinteilung, wie sie z. B. in [57] verwendet wird, weiterhin möglich:

$$[(\langle \mathsf{Algebraische\ Regeln} \rangle)^*]_1[(\langle \mathsf{Physische\ Regeln} \rangle)^*]_1.$$

Entsprechend dem Beispiel in Abbildung 5.2 können wir bei der Wahl einer brute-force Strategie (der alternativen Ausführung aller vier bekannten Regeln $\mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_3$ und r_4), einfach folgenden Term der Regelsteuerungssprache angeben $[(\mathsf{r}_1|\mathsf{r}_2|\mathsf{r}_3|\mathsf{r}_4)^*]_1$. Dieses wird solange ausgeführt, bis keine neue, noch unbekannte Ausführungsvariante mehr erzeugt wird.

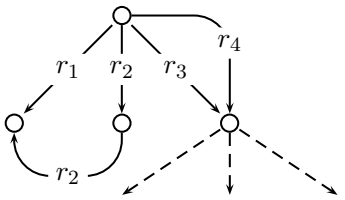


Abbildung 5.2: Variantenerzeugung

5.3 Beschränkung der Möglichkeiten zur Variantenerzeugung

Würde man bei der Optimierung von Ausführungsplänen nur algebraische Regeln benutzen, dann müsste man sich in jedem Iterationsschritt mit weniger als 30 Regeln beschäftigen (siehe Abschnitt 3.2.1). Durch die Erzeugungsregeln (siehe Abschnitt 3.2.2) und die physischen Implementierungsarten (siehe Abschnitt 3.2.3) explodiert die Regelmenge auf ein Vielfaches, so dass man zwangsläufig dieselben Probleme bekommt, die schon beim Projekt COKO-KOLA [13] (1998) eine sinnvolle Nutzung verhinderten (bei 60 Operatoren 600 zu betrachtende Regeln). Demzufolge reichen preconditions definitiv nicht aus und müssen durch sinnvolle, nach einem Iterationsschritt anwendbare Strategien, ergänzt werden.

Die nachfolgend vorgestellten Strategien geben uns die Möglichkeit, den Suchraum so einzuschränken, dass wir immer noch das globale Kostenminimum finden können. Dadurch, dass sie nicht auf einzelne Regelanordnungen festgelegt sind und kombiniert verwendet werden können, bieten sie ein Maximum an Flexibilität (siehe Kapitel 6).

5.3.1 Dynamische Kostengrenze

Da zu Anfang der Optimierung die Kosten des initialen Ausführungsplans bekannt sind, kann eine obere Schranke für die Ausführungskosten neuer Pläne angegeben werden. Darauf aufbauend können weitere Kostenschranken definiert werden, die von neu generierten oder von allen bekannten Varianten eingehalten werden müssen. Beispielsweise könnten bei gegebenen Kosten von $\mathsf{cost}(\tau)$ nur jene Varianten von τ weiter berücksichtigt werden, die geringere Kosten verursachen. Dieses Vorgehen würde der bekannten Heuristischen Suche des Hill-Climbing entsprechen. Eine weitere Alternative wäre es, darauf zu achten, dass die Kostengrenze $\mathsf{cost}(\tau) \cdot \log(\mathsf{cost}(\tau))$ nicht überstiegen wird (siehe Abschnitt 2.7.6).

$\tau_1 \overset{1}{\underset{2}{\rightleftharpoons}} \tau_2$ 1) wenn $\mathsf{cost}(\tau_2) \leq \mathsf{cost}(\tau_1) \cdot \mathsf{d}$ 2) false	$\mathsf{d} = 1$ entspricht Hill-Climbing $\mathsf{d} > 1$ (z. B. 1,1 oder $\log(\mathsf{cost}(\tau_1))$) entspricht Simulated-Annealing

Die Grundidee hinter dieser heuristischen Grenze liegt darin (mit $\mathsf{d} > 1$), dass z. B. das einmalige Anlegen von Sortierungen mehrmalige Nutzung ermöglichen kann und somit die Kosten lokal zwar steigen, aber global zu Einsparungen führen können (siehe Abbildung 5.3). Dabei können auch intelligentere Strategien abgebildet werden, die, wie z. B. beim Simulated-Annealing, eine Toleranz des Kostenwertes mit berücksichtigen.

Die dynamische Grenze hat einen entscheidenden Vorteil gegenüber einer vorher hypothetisch gesetzten und wieder verworfenen statischen Grenze: das adaptive Verhalten hinsichtlich der Generierung neuer Varianten. Wird in einem Iterationsschritt eine Variante erzeugt, die günstiger ist als eine der vorher generierten Varianten, passt sich die Kostengrenze an die neuen Kosten an und schränkt den Suchraum damit ein. Je tiefer die Grenze liegt, desto höher ist die Wahrscheinlichkeit, dass neue Varianten über dem Limit liegen. Diese Varianten werden dementsprechend deaktiviert und müssen im nächsten Iterationsschritt nicht mehr berücksichtigt werden. Sind hingegen alle neuen Varianten teurer, dann verändert sich das Filterkriterium nicht und lässt damit eine breitere Suche zu.

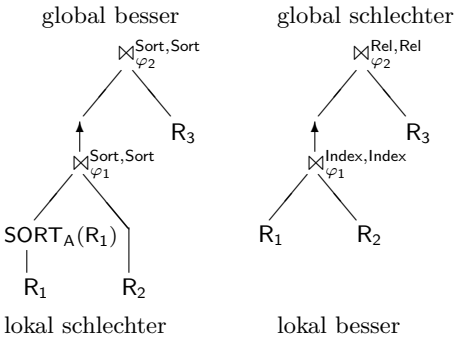


Abbildung 5.3: Mehrmalige Nutzung einer Sortierung vs. einmaliger Indexnutzung

In Abbildung 5.4 sind Varianten und Iterationen in einer Ebene kombiniert dargestellt. Dabei werden nur neu generierte Varianten abgebildet. Falls keine mehr erzeugt werden, wird die Optimierung eingestellt. Eine Möglichkeit den Suchraum zusätzlich einzuschränken ist der Abbruch der Optimierung nach einer vorgegebenen Anzahl von Iterationsschritten. Allerdings können die besten Varianten auch erst im letzten Iterationsschritt vorkommen, so dass das Abbrechen zu einem schlechten Ergebnis führt. Eine besseres Vorgehen ist es, eine dynamische Grenze mit der Regelanordnung in einem Iterationsschritt zu kombinieren, so dass man potentielle Regeln dementsprechend früh breit gefächert anwendet und damit möglichst frühzeitig Minima erreicht.

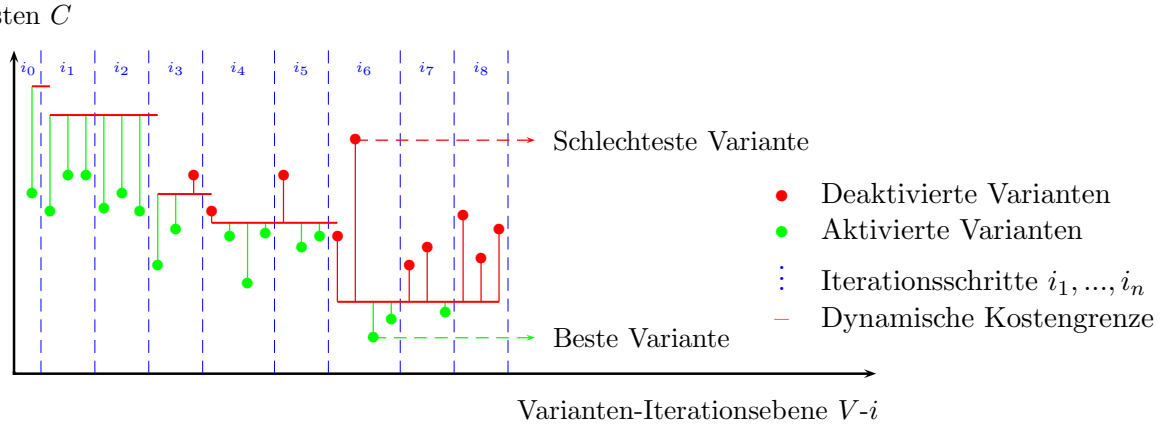


Abbildung 5.4: Dynamische Kostenentwicklung

5.3.2 Statische Variantengrenze

Neben einer Kostengrenze muss das System auch in der Lage sein, die Anzahl von zu berücksichtigenden Varianten zu limitieren. Da Systeme nur begrenzten Hauptspeicher aufweisen und jede Variante einen festen Wert an Speicherplatz benötigt, kann die maximale Anzahl x vor dem Beginn der Optimierung statisch festgelegt werden.

Dabei gibt es zwei verschiedene Möglichkeiten (siehe z. B. Abbildung 5.5):

(a) Es werden maximal nur die x besten Varianten am Ende eines Iterationsschrittes abgespeichert. Dafür wird in jedem Schritt eine Auswertung aller bisher bekannten Varianten ausgeführt, $[...]_{\mathsf{I}}$ mit $\mathsf{I} \in \mathbb{N}$.

(b) In jedem Iterationsschritt werden wir nur die x besten neuen Varianten zusätzlich abspeichern.

Bei der ersten Beschränkung berücksichtigen wir nur die x besten Varianten (vergleiche Hill-Climbing). Beim zweiten Ansatz limitieren wir ein exponentielles Wachstum hin zu einem linearen.

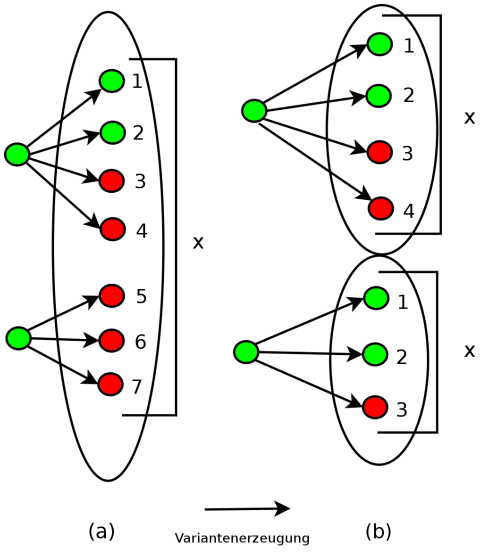


Abbildung 5.5: Mögliche statische Variantengrenze

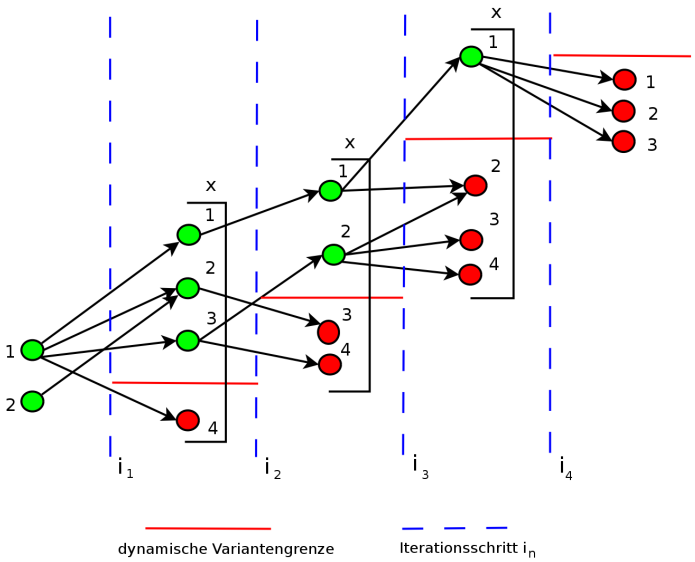


Abbildung 5.6: Mögliche dynamische Variantengrenze

5.3.3 Dynamische Variantengrenze

Eine dynamische Variantengrenze limitiert im Gegensatz zu einer statischen Grenze die Anzahl von neuen Varianten dahingehend, dass für jeden Iterationsschritt unterschiedlich viele neue Varianten abgespeichert werden können. Somit kann in Schritten, in denen viele Varianten erzeugt werden, mehr abgespeichert werden als bei Schritten, in denen insgesamt nur wenig Neues generiert wird.

Es werden somit maximal nur die x besten neuen Varianten am Ende eines Iterationsschrittes abgespeichert, wobei x in diesem Fall kein statischer Wert ist, sondern sich zu der erzeugten Menge prozentual definieren lässt. Dafür muss erneut in jedem Schritt eine Bewertung für alle bisher bekannten Varianten $[...]_l$ mit $l \in \mathbb{N}$ ausgeführt werden. Beispielsweise könnten im ersten Schritt 75% und in den darauf folgenden Schritten immer 25% weniger Varianten übernommen werden. Somit wird die Variantensuche spätestens nach 4 Iterationen gestoppt (siehe Abbildung 5.6).

5.4 Aufbau des bedingten Termersetzungsoptimierungssimulators

Der Termersetzungsoptimierungssimulator, der im Zuge dieser Arbeit entwickelt worden ist, besteht aus drei Modulen und einer grafischen DBMS-Schnittstelle, über die der Benutzer die deklarative Anfrage eingeben und optimieren lassen kann (siehe Abbildung 5.7).

Das erste Modul ist ein Parser für die Eingabe von Anfragen und Regeln. Modul 2 koordiniert alle Abläufe und steuert somit das dritte Modul, welches die eigentliche Optimierung von Termen vornimmt. Im Folgenden werden die einzelnen Module vorgestellt.

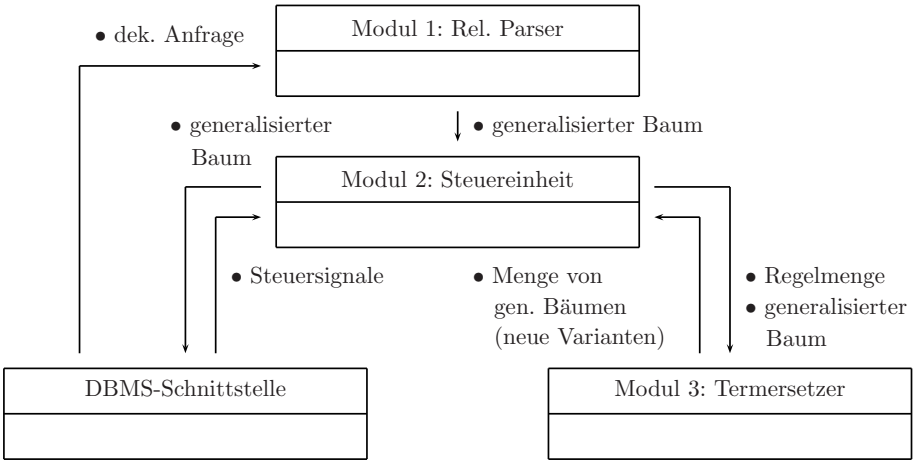


Abbildung 5.7: Module des Termersetzungsoptimierungssimulators

5.4.1 Modul 1: Der relationale Parser

Die Aufgabe des relationalen Parsers besteht aus zwei verschiedenen Bereichen: Als erstes wird eine Anfrage aus der relationalen Algebra in einen generalisierten Baum umgewandelt. Als Basis für diese Erzeugung wird eine Attributierte Grammatik definiert, die in den Attributen Bedingungsfunktionen enthält, die während des Parsens berechnet werden müssen. Der so entstandene generalisierte Baum besitzt relationale bzw. objektrelationale Operatoren mit den zugehörigen Funktionen zur Berechnung in den einzelnen Knoten. Jeder Baumknoten im generalisierten Baum enthält eine Reihe von relationenbezogenen Daten und Metainformationen, sowie Funktionen, die daraus berechnet werden. Hierzu gehören die physischen Umsetzungen, Attribute der Ergebnisrelation des Knotens, Attributwertanzahlen, Kardinalitäten, Pagegrößen, Kosten, Sortierungen und mögliche Pipelines.

Die zweite Aufgabe des Parsers ist es, Termersetzungsregeln abzuspeichern, wofür eine vereinfachte Form des generalisierten Baumes verwendet wird. Eine Ersetzungsregel besteht immer aus zwei Seiten, wobei jede einzelne als einfacher generalisierter Baum abgespeichert wird. Allerdings werden dabei keine konkreten Relationen festgelegt, sondern entsprechende Variablen für relationale Terme bzw. für Ausführungspläne.

5.4.2 Modul 2: Die Steuereinheit

Die Steuereinheit kontrolliert den Optimierungsablauf, indem sie entscheidet, wann welche Regeln mit welchen generalisierten Bäumen zur Optimierung weiterbetrachtet werden müssen (siehe Rudnicki [73]). Somit stellt sie die Kommunikation zwischen den beiden anderen Modulen her. Die manuellen Eingaben über die DBMS-Schnittstelle werden an die Steuereinheit weitergeleitet und dort koordiniert. Darüber hinaus hat sie die Aufgabe, die Generierung von gleichen Ausführungsplänen zu unterbinden. Zusätzlich ist sie für die Einschränkung des Suchraumes zuständig, den sie je nach gewählter Strategie (z. B. durch die Anordnung von Regelgruppen) anders durchsuchen lässt (siehe Abschnitt 5.3). Liegen nach Abschluss der Berechnungen in diesem Modul die Varianten und die anzuwendenen Regeln vor, wird die Termersetzungsphase eingeleitet.

Nach der Termersetzung wird der Steuereinheit die Menge von neuen Varianten geliefert. Es

hat sich als sinnvoll erwiesen, dem Termersetzer die dynamische Kostengrenze mit zu übergeben, damit Varianten, die diese postcondition nicht erfüllen, sofort verworfen werden können. Dadurch werden alle erzeugten Varianten nicht mehr mindestens einmal abgespeichert, sondern schon bei der Entstehung klassifiziert. Grund dafür war die Beobachtung, dass das Wachstum der Variantenanzahlen genau an dieser Stelle am sinnvollsten kontrolliert werden kann. Wird eine neue Variante erkannt, deren Kosten die dynamische Kostengrenze weiter nach unten senken, dann werden die bislang bekannten Varianten hinsichtlich ihrer Kosten überprüft und falls nötig, deaktiviert. Die Steuereinheit untersucht zusätzlich, ob gleiche Varianten entstanden sind. Natürlich wäre es wenig effizient, wenn jede bekannte Variante mit den neuen verglichen würde. Deswegen verwendet man zum einen den vorher definierten Ableitungsgraphen (siehe Abschnitt 5.1.2) und zum anderen die Ausführungskosten als gehashten Schlüssel.

Algorithmisch entstehende Varianten werden nach der Generierung auf Gleichheit der Hashwerte ihrer Kosten verglichen. Es ist nämlich unwahrscheinlich, dass zwei verschiedene Varianten denselben Kostenhashwert haben. Falls es doch einmal eintritt, dann werden die einzelnen Varianten knotenweise durch ein Matching verglichen. Liegt wiederum eine Gleichheit vor, werden die neu erzeugten Informationen zu der bekannten Variante hinzugefügt: Dazu gehört der erweiterte Entstehungsgraph inklusive der Daten zum Iterationsschritt und der verwendeten Regeln. Handelt es sich jedoch um eine neue Variante, so wird ein neuer Eintrag in die Variantentabelle eingefügt.

5.4.3 Modul 3: Das Termersetzungssystem

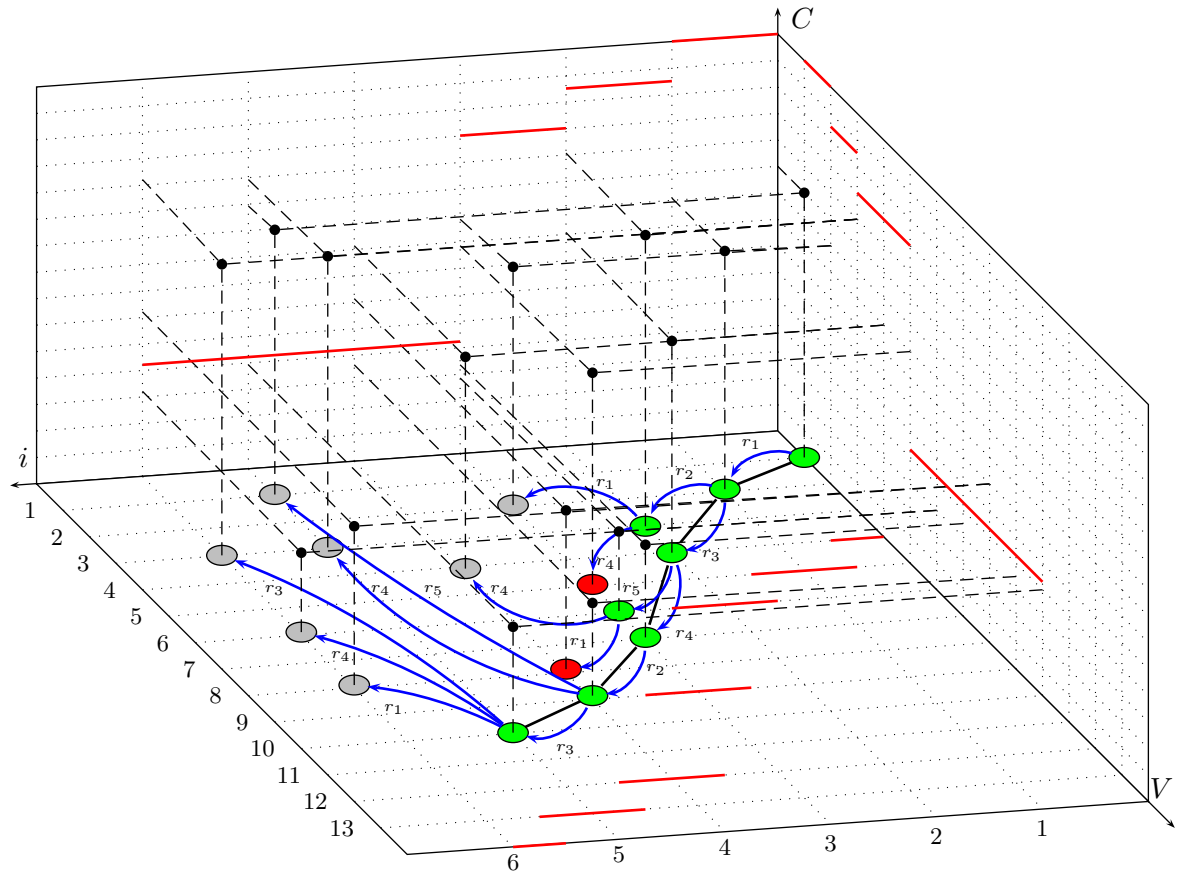
Die Steuereinheit übergibt Modul 3, dem Termersetzungssystem, alle anzuwendenden Regeln inklusive eines initialen Ausführungsplanes. Das System wiederum liefert die daraus generierten neuen Ausführungspläne zurück. Dieser Schritt wird so lange wiederholt, bis keine Ausführungspläne mehr erzeugt werden.

5.5 Ein Beispiel für den Suchraum des Termersetzers

Die nachfolgende Abbildung 5.8 zeigt einen Suchraum aus der Sicht des Termersetzers. Dabei steht die Dimension i für die Iterationsschritte, V gibt die unterschiedlichen Varianten und C die dazugehörigen Kosten an. Man kann die steigende Anzahl von Varianten im Laufe der Iterationen erkennen. Doppelte Varianten werden in der Abbildung durch gleiche Knoten mit mehr als einer gerichteten Kante dargestellt. Die Kosten der bisher besten Variante bestimmt die dynamische Grenze, welche man in der $i - C$ und der $V - C$-Ebene durch rote Linien wiederfindet. Die statische Variantengrenze ($i - V$-Ebene) lässt in jedem Schritt nur maximal 4 neue Varianten zu.

5.6 Fazit der Termersetzung

In diesem Kapitel wurden neben den lokalen Restriktionen, unter denen eine Regel angewandt werden kann, spezifizierte Regelabfolgen (dargestellt durch eine eigene Regelsprache) und als dritte Komponente globale Strategien zur kontrollierten Variantenerzeugung vorgestellt. Die Nutzung dieser diversen Bedingungen bringt den Vorteil, dass es nicht zwangsweise zu einer explosionsartigen Vermehrung der Varianten kommen muss. Diese stark eingrenzenden Faktoren sind in der Lage, die Anzahl an Varianten sehr niedrig zu halten. Dadurch wird eine gute



Neue Variante
Bekannte Variante
Deaktivierte Variante
r_n Entwicklung einer Variante aus einer anderen unter Benutzung der Regel r_n
Beschreibt die Entwicklung der Variantenanzahl V
Orientierungslinie einer Variante
Grenze: dynamisch in den Ebenen $i-C$ und $V-C$; statisch in der $i-V$-Ebene

Abbildung 5.8: Der Suchraum des Termersetzers

Optimierungszeit gewährleistet, obwohl die Zahl der anzuwendenden Regeln im Vergleich zur Verwendung von reinen algebraischen Regeln immens gestiegen ist.

Man darf nicht das, was uns unwahrscheinlich und unnatürlich erscheint,
mit dem verwechseln, was absolut unmöglich ist.

Carl Friedrich Gauss 1777-1855



6 Entwicklung von Optimierungsstrategien

Die konzeptionellen Aussagen und Erkentnisse aus den vorherigen Kapiteln wurden in praktischen Anwendungen realisiert, um die theoretischen Schlussfolgerungen zu bestätigen und zu verdeutlichen. Die durchgeführten Tests dienten dazu, die Qualität und Optimierungsleistung verschiedener Strategien bezüglich relationaler und objektrelationaler Daten vergleichen zu können. Es sollte die Entwicklung der Anzahl von Varianten untersucht werden, indem die Regelanordnung und -anwendung permutiert wird. Diese Variationen werden als Darstellungen von Optimierungsstrategien angesehen. Bekanntermaßen gibt es für algebraische Heuristiken auch eine Regelanordnung (Strategie). In unserem Fall können wir sie durch die bedingte Ersetzung verfeinern.

So weit dies möglich war, wurde ein international anerkannter Benchmark-Test benutzt, um eine objektive Vergleichbarkeit zu garantieren. In den anderen Fällen wurden möglichst repräsentative Beispielanfragen erstellt. Die Messungen und Auswertungen waren auf zwei Bereiche gerichtet: zum Einen sollte das Optimierungspotential einer Anfrage herausgefunden werden, d. h. ob und inwiefern sich die Kosten eines optimierten Ausführungsplanes im Vergleich zu einer nicht optimierten Ausgangsvariante[1] verbessern. Normalerweise können Aussagen und Vergleiche bezüglich des Opimierungspotentials nur dann getroffen werden, wenn man die verwendeten Übersetzungsregeln und Heuristiken kennt. Da kommerzielle Datenbanksysteme dem Benutzer keine Möglichkeit bieten, den direkt übersetzten Ausführungsplan mit dem optimierten zu vergleichen, haben wir uns auf die Informationen aus `RELOpt` beschränkt. Zum Anderen wurde im Zuge dieser Messungen auch ein Vergleich zwischen den verschiedenen Strategien zur Optimierung durchgeführt. Des Weiteren ging es um zeitliche Differenzen in der Ausführung der durch Oracle 10g und `RELOpt` optimierten Pläne, was einen direkten Vergleich der Fähigkeiten der beiden Optimierer ermöglicht.

Das Kapitel gliedert sich wie folgt: Nach einer einleitenden Betrachtung der Strategien werden Ergebnisse der Untersuchungen auf relationalen Daten vorgestellt. Daran anschließend werden die Tests und Ergebnisse auf objektrelationalen Daten erläutert. Zum Ende stellen wir die Erfahrungen und Schlussfolgerungen bezüglich der zu Beginn diskutierten Strategien vor.

6.1 Einführung in die Strategiebetrachtung

Die Optimierung beginnt mit dem ersten generierten Ausführungsplan, den wir vom Parser bereitgestellt bekommen. Da nur eine endliche Anzahl von kombinatorischen Möglichkeiten für den Ausführungsplan existiert, wissen wir, dass die zu betrachtende Menge beschränkt ist. Daraus folgt, dass, obwohl ein exponentielles Wachstum der Variantenanzahl durch die Regeln zu erwarten ist, die Termersetzung in jedem Fall stoppt.

Das Problem, welches wir bekämpfen müssen, ist aber nicht das Generieren von neuen Varianten, sondern das wiederholte Erzeugen von schon bekannten. Je mehr Varianten gefunden

[1] Diese Option steht bei `RELOpt` zur Verfügung und stellt den Ausführungsplan aus der naiven Übersetzung einer SQL-Anfrage dar.

wurden, desto seltener werden neue, bisher noch unbekannte Varianten entdeckt. Der Aufwand, diese neuen Varianten aus den schon bekannten zu finden, wächst exponentiell, wenn wir nicht Varianten deaktivieren. Daraus folgt, dass sinnvolle Deaktivierung das Hauptkriterium für die Qualität der Optimierung darstellt.

Das maßgeblich wichtigste Kriterium zur Deaktivierung wird durch die Erzeugungsinformationen („ausgehend von welcher Variante wurde die aktuelle Variante unter Benutzung welcher Regel erzeugt?“) in den einzelnen Varianten abgespeichert. Es wird somit eine Liste von verwendeten Regeln und Richtungen in jeder bekannten Variante geführt, mit deren Hilfe man herausfinden kann, ob eine Variante deaktiviert werden müsste. Dieses ist dann der Fall, wenn keine anwendbaren Regeln aus der ursprünglichen Regelmenge mehr benutzt werden können, ohne dass schon bekannte Varianten entstehen.

Bei der Erzeugung neuer Varianten gibt es folgende Deaktivierungskriterien:

(a) *Preconditions:* Bedingungen, die vor der Termersetzung geprüft werden. Die Reihenfolge der Bedingungsprüfung sollte so angeordnet sein, dass Kriterien, die die Differenzierung stark vereinfachen, als erstes abgefragt werden. Durch dieses Vorgehen ist es wahrscheinlicher, möglichst viele Regeln, die nicht angewandt werden können, zu überspringen.

(b) *Postconditions:* Bedingungen, die nach der Ersetzung geprüft werden, z. B. Kostenbedingungen. Dabei wurden im Kapitel 3.4 Vereinfachungen vorgestellt, so dass nicht erst die Gesamtkosten eines Ausführungsplanes berechnet werden müssen, bevor man ihn auf Grund von Nicht-Anwendbarkeit verwerfen kann. Im Abschnitt 6.2 werden mögliche Strategien aufgeführt.

Ein allgemeiner Nachteil ist, dass durch das Deaktivieren von zu vielen Varianten der Suchraum so stark eingegrenzt werden kann, dass das globale Minimum nicht mehr erreicht wird. Daher ist es wichtig, Anpassungen der Einschränkungen, wie im nachfolgendem Abschnitt 6.3 vorgestellt, vorzunehmen.

6.2 Optimierungsstrategien

Wie im Kapitel 5 beschrieben, werden Strategien durch Kombinationen und Reihenfolgen von verschiedenen Bedingungen dargestellt. Sie geben Auskunft darüber, wann und unter welchen Umständen eine Variante deaktiviert wird. Veränderungen können sowohl durch kostenbedingte als auch durch Regelanordnungsstrategien erwirkt werden, die in den nachfolgenden Unterkapiteln behandelt werden.

6.2.1 Kostenbedingte Strategien

Im Folgenden betrachten wir zur Einführung drei gängige kostenbedingte Strategien auf Basis eines Beispiels mit 4 Relationen:

Gehen wir davon aus, dass wir vom Parser eine Variante (0,0) mit Kosten in Höhe von 47.235 bekommen. Darauf lassen wir alle bekannten Regeln anwenden, wobei nur Varianten weiterverfolgt werden sollen, die weniger Kosten verursachen als das bisherige Minimum (Hill-Climbing Strategie). In unserem Beispiel ergeben sich daraus 16 Varianten in sechs Iterationsschritten (siehe Abbildung 6.1[2]).

Hill–Climbing Strategie

$\tau_1 \overset{1}{\underset{2}{\rightleftharpoons}} \tau_2$

1) wenn $\mathsf{cost}(\tau_2) \leq \mathsf{cost}(\tau_1)$

2) false

[2]Hierbei sind rote Varianten deaktivierte, grüne aktivierte und blaue das bisherige Kostenminimum.

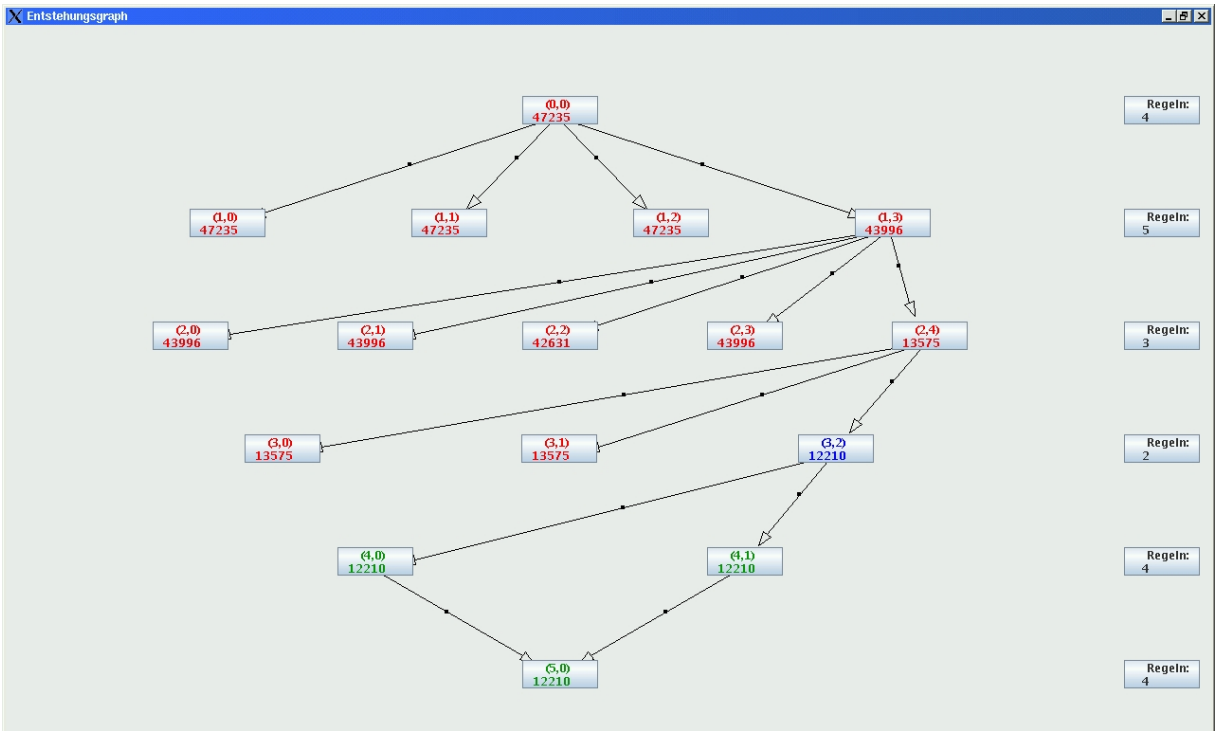


Abbildung 6.1: Beispiel eines Entstehungsgraphen - Hill-Climbing Strategie

Schon im vierten Iterationsschritt wird eine Variante mit minimalen Kosten von 12.210 gefunden, wobei diese Variante rund drei Viertel der Ausführungszeit der Variante $(0, 0)$ einspart. Dabei kann aber keine Aussage darüber gemacht werden, ob es einen Ausführungsplan geben kann, der billiger wäre als das bisher gefundene Minimum. Das Einzige, was wir wissen, ist, dass die Variante (3,2) durch Anwendung von drei Regeln aus der Urspungsvariante generiert werden konnte ($(0, 0) \rightarrow (1, 3) \rightarrow (2, 4) \rightarrow (3, 2)$).

Verändern wir die Strategie für dasselbe Beispiel hin zu einer Simulated-Annealing-Suche, so erhalten wir das globale Minimum, da wir nicht mehr im lokalen Minimum verharren müssen (siehe Abbildung 6.2).

Simulated-Annealing Strategie

$$\tau_1 \underset{2}{\overset{1}{\rightleftharpoons}} \tau_2$$

1) wenn $\mathsf{cost}(\tau_2) \leq \mathsf{cost}(\tau_1) \cdot \log(\mathsf{cost}(\tau_1))$
2) false

Im dritten Iterationsschritt wurde nicht nur die bisher günstigste Variante weiterverfolgt, sondern auch die teurere $(1, 3)$. Erst dadurch war es möglich, das globale Minimum $(3, 2)$ zu finden (Kosten 4.325). Da in diesem Beispiel alle Varianten, die im Hill-Climbing gefunden werden, auch eine Teilmenge des Simulated-Annealing sind, können wir davon ausgehen, dass die dynamische Grenze sinnvollerweise auf einen Wert proportional zu dem bisherigen Minimum gestellt werden sollte. Berücksichtigen wir das Beispiel aus 5.3.1 und wählen als Grenze $\mathsf{cost}(\tau) \cdot \log(\mathsf{cost}(\tau))$, erreichen wir zwar das globale Minimum, müssen aber mehr als doppelt so viele Varianten im Vergleich zur Hill-Climbing Strategie, nämlich 34, untersuchen. Dafür ist die gefundene Variante nur noch ein Zehntel so kostenintensiv wie die Ursprungsvariante.

Da wir jeweils die dynamische und statische Grenze für den Optimierungsprozess ausschalten können, besitzen wir dadurch auch die Möglichkeit der vollständigen Suche (engl. british museum search bzw. exhaustive search), um damit das globale Minimum finden zu können. Eine Anwendung ist notwendig, falls die zu optimierenden Anfragen sich nicht verbessern liessen und ihr Optimierungspotential untersucht werden soll.

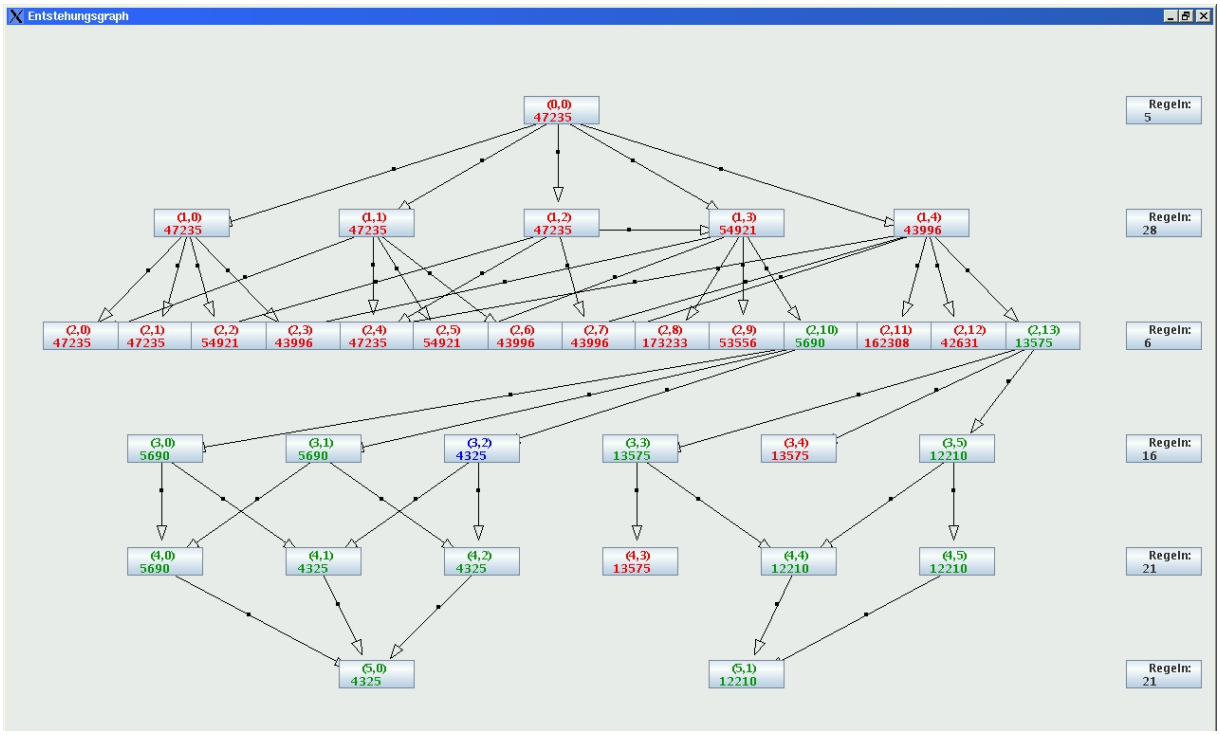


Abbildung 6.2: Beispiel eines Entstehungsgraphen – Simulated-Annealing Strategie

Aus dem Abschalten der Kostengrenzen resultiert ein Klassifikator für Optimierungsstrategien, der zu einer Menge von Regeln, Regelanordnungen und gegebenen Grenzen die Frage beantwortet, ob mit der angestrebten Strategie das Minimum überhaupt gefunden werden kann (vorausgesetzt die Suchzeit spielt keine Rolle und es existiert genügend Speicherplatz[3]).

British Museum Suche

$\tau_1 \underset{1}{\rightleftharpoons} \tau_2$

1) false

6.2.2 Strategien mit Hilfe von Variantengrenzen

Mit Hilfe von Variantengrenzen können Korridore im Suchraum definiert werden. Dabei muss bei der Wahl der Grenzen der zur Verfügung stehende Hauptspeicher berücksichtigt werden. In Kombination mit Kostengrenzen sind sie dafür verantwortlich, dass bei einer „weiteren“ Korridorbreite kostengünstigere Varianten frühzeitig gefunden werden und sich die dynamische Kostengrenze somit schneller senkt. Damit müssen die gefundenen Ausführungspläne in „schmaleren“ Korridoren nicht Bestandteil von „breiteren“ sein, da sich zwangsläufig eine andere Tiefensuche ergibt. Schließlich sollte die Korridorbreite abhängig vom System und der Anfrage definiert werden. Dafür kann man den Speicherplatzverbrauch des direkt übersetzten Ausführungsplanes nehmen und ihn in Beziehung mit dem Hauptspeicher setzen, d. h. je höher der Speicherplatzverbrauch einer Variante ist, desto kleiner sollte die Suchbreite sein.

6.3 Ergebnisse der Experimente

Alle Anfragen wurden auf einem intel Pentium IV System mit 3.0 GHz (Linux 2.6.13) und 1 GB Hauptspeicher optimiert (siehe detaillierte Auflistung der Testsystemspezifikationen im

[3]Schon in unserem kleinen obigen Beispiel reichte der Speicherplatz des Testsystems für die komplette Suche nicht aus.

Anhang D). Für die RELOpt-Optimierung wurden verschiedene dynamische (V1.1,...,V3.2) und statische Grenzen getestet (siehe Tabelle 6.1). Reichte der Arbeitsspeicher für die Berechnung der Anfrage nicht aus, wurde die statische Grenze jeweils um eine Stufe gesenkt. In einigen Fällen ergab es sich, dass Strategien teilweise nicht angewendet werden konnten, da für eine Erhöhung der Suchbreite der Hauptspeicher nicht mehr genügte.

Das Vorgehen für die Arbeit mit allen Anfragen war wie folgt: Zunächst wurde die Anfrage in SQL eingegeben und von Oracle 10g optimiert. Die dabei entstandenen Kosten und benötigten Ausführungszeiten wurden festgehalten. Als nächstes wurde die Anfrage manuell in die relationale Sprache von RELOpt überführt und von diesem System optimiert.

Der so entstandene Ausführungsplan musste wiederum durch den Sprachumfang von SQL plus Optimierungshints ausgedrückt und durch Oracle 10g ausgeführt werden. Dabei war es in manchen Fällen nötig, die Anfrage in Teilanfragen zu separieren, um den Ausführungsplan tatsächlich in der gewünschten Form bearbeiten zu lassen. Diese Notwendigkeit entstand durch die Einschränkungen der Oracle 10g-Datenbank, da es weder möglich ist Indexe auf Zwischenergebnissen zu erzeugen noch das automatische Verändern der Ausführungspläne in allen Fällen zu unterbinden. Die Kosten und Ausführungszeiten der von RELOpt optimierten Anfrage wurden für die Auswertung ebenfalls festgehalten.

Strategie	dynamische Kostengrenze	statische Variantengrenze
1:1-Übersetzung	0,01	1
Hill-Climbing V1.1	1	≤ 5
Hill-Climbing V1.2	1	≤ 10
Simulated-Annealing V1.1	1,1	≤ 5
Simulated-Annealing V1.2	1,1	≤ 10
Simulated-Annealing V2.1	$\log_{10}$	≤ 5
Simulated-Annealing V2.2	$\log_{10}$	≤ 10
Simulated-Annealing V3.1	$\log_e$	≤ 5
Simulated-Annealing V3.2	$\log_e$	≤ 10

Tabelle 6.1: Optimierungsstrategien

6.3.1 Resultate der relationalen Optimierung mit RELOpt

Die Ergebnisse hinsichtlich 22 Anfragen aus dem TPC-H Benchmark Version 2.3.0[4] werden nachfolgend vorgestellt. Die Zielsetzung dabei war es, gleiche Ausführungspläne wie kommerzielle Systeme bei der Optimierung von deklarativen Anfragen zu erreichen.

Es wurde die folgende Regelanordnung mit der statischen Variantengrenze x (siehe Kapitel 3) verwendet:

$$[[(\text{Alle impl.-unabh.- Regeln})^*]_x[(\text{Alle Erzeugungsregeln})^*]_x[\text{Alle Implementierungsvarianten})^*]_x]_1$$

Somit wurden alle in dieser Arbeit aufgeführten implementierungsunabhängigen Regeln (siehe Abschnitt 3.2) parallel angewendet und anschließend ausgewertet. Als nächster Schritt folgten alle erzeugenden Regeln gruppiert mit der Kontrolle, ob dadurch günstigere Implementierungen für die Operatoren generiert wurden (siehe Abschnitt 3.2.2 und 3.2.3).

Da RELOpt im relationalen Fall für manche Attributwertverteilungen auf Standardwerte zurückgreift und kommerzielle Systeme durch Benutzung von Histogrammen diese Werte besser abschätzen können, konnten nur in wenigen Fällen signifikante Performancegewinne erreicht werden. Eine ausführliche Betrachtung der Ergebnisse inklusive der Kosten und der Optimierungsdauer für die jeweiligen Strategien ist in Tabelle D.2 des Anhangs D zu finden. Im Folgenden wird eine Zusammenfassung und Diskussion der Resultate präsentiert.

[4] http://www.tpc.org/tpch/default.asp

Für Aussagen bezüglich des Optimierungspotentials wurden nach der Optimierung mit `RELOpt` die Kosten der jeweiligen optimierten Variante mit denen der Ausgangsvariante ins Verhältnis (y) gesetzt (siehe Tabelle D.3 auf Seite 136). Dabei konnte festgestellt werden, bei welchen Anfragen die verwendeten Regeln eine Optimierung durch Neuanordnung der Operatoren oder Neuerzeugung von Indexen bzw. Sortierungen erzielen konnten. Falls keine Transformation des Ausführungsplanes durchgeführt werden konnte, war der Baum „zu einfach“: beispielsweise ein Join mit anschließender Gruppierung und Projektion. Mit den entwickelten Regeln lassen sich die Anfragen hinsichtlich Optimierbarkeit in vier Gruppen einteilen, wobei x das Optimierungspotential $(1 - y)$ darstellt (siehe Abbildung 6.3):

(a) *Signifikant optimierbare* Anfragen ($x \geq 99\%$): Die Ausführung dieser Gruppe der Anfragen (Q5, Q16, Q17 und Q21 (4 Stück)) konnte durch die Termersetzung bis zum Faktor 10^7 beschleunigt werden.

(b) *Optimierbare* Anfragen ($99\% > x \geq 1\%$): Bei den Anfragen Q2, Q7, Q8, Q14, Q20 und Q22 (6 Stück) wurde die Ausführungszeit so weit verbessert, dass sich die zusätzlich investierte Optimierungszeit lohnte.

(c) *Ungenügend optimierbare* Anfragen ($1\% > x > 0\%$): Bei diesen Anfragen (7 Stück) lohnen sich die zusätzlichen Investitionen für die Optimierung nicht zur signifikanten Beschleunigung der Ausführung. Bei Q3, Q4, Q9, Q10, Q11, Q12 und Q18 sind nur kleine Teile der Anfrage sinnvoll optimierbar, so dass sich der Mehraufwand zu den bisher verwendeten Heuristiken nicht rentiert.

(d) *Nicht optimierbare* Anfragen ($x = 0\%$): Bei den Anfragen Q1, Q6, Q13, Q15 und Q19 (5 Stück) konnten wir mit den in Kapitel 3 aufgeführten Regeln keine Verringerung der Kosten erreichen. Grund dafür war die Einfachheit der Anfragen, die kein Optimierungspotential besaßen.

Generell kann gesagt werden, dass sich das Optimierungspotential steigert, je mehr Basisrelationen und darauf agierende Joins vorhanden sind und benutzt werden. Bekanntermaßen ist die Anordnung der Joins eine der wichtigsten Optimierungsstrategien, was sich auch während der Tests widergespiegelt hat. Zur Übersicht über die Anfragen wurde in Tabelle 6.2 jeweils die Anzahl der Joins und verschiedener Relationen aufgeführt. In Tabelle 6.3 wurden die Anfragen nach der Anzahl der Joins gruppiert.

Anfrage	Anzahl der Joins	Anzahl verschiedener Relationen
Q1	0	1
Q2	4	5
Q3	2	3
Q4	1	2
Q5	5	6
Q6	0	1
Q7	5	5
Q8	7	7
Q9	5	6
Q10	3	4
Q11	5	6
Q12	1	2
Q13	1	2
Q14	1	2
Q15	2	2
Q16	3	3
Q17	1	2
Q18	3	3
Q19	1	2
Q20	5	5
Q21	5	4
Q22	1	2

Tabelle 6.2: Anzahl der vorkommenden Joins mit den dabei verwendeten Basisrelationen in den TPC-H Version 2.3.0 Anfragen

Das Optimierungspotential der Benchmark-Anfragen ist in Abbildung 6.3 dargestellt.

Anzahl der Joins							
0	1	2	3	4	5	6	7
Q1	Q4	Q3	Q10	Q2	Q5	-	Q8
Q6	Q12	Q15	Q16		Q7		
	Q13		Q18		Q9		
	Q14				Q11		
	Q17				Q20		
	Q19				Q21		
	Q22						

Tabelle 6.3: Anzahl der vorkommenden Joins in den TPC-H Version 2.3.0 Anfragen

Während bei drei Joins nur ein Drittel der Anfragen zu der signifikant bzw. optimierbaren Gruppe gehört, sind es bei fünf Joins schon zwei Drittel.

Da üblicherweise bei größeren Anzahlen von Joins ($\geq$ 14 Relationen) keine Optimierung stattfindet, sondern sich ein kommerzieller Optimierer nur noch heuristisch orientiert, könnten gerade dort Optimierungspotentiale mit Hilfe des in dieser Arbeit entwickelten Optimierungskonzeptes gefunden werden (siehe dazu Lawson [49]).
Bezüglich der Ausführungszeiten der jeweiligen optimierten Anfragen kann als Ergebnis der Tests zusammenfassend die Aussage getroffen werden, dass sich die Hill-Climbing Suche als beste Strategie bewährt hat, da sich die Vergrößerung des untersuchten Anfrageraumes in keinem einzelnen Fall „richtig“ lohnte (siehe dazu die ausführlichen Daten im Anhang D). Zwar wurden mit Hilfe von Simulated-Annealing bei etwa der Hälfte der Anfragen bessere Varianten gefunden (siehe Q3, Q5, Q7, Q9, Q11, Q12, Q18, Q20 und Q21), doch stand die Verbesserung in keinem Verhältnis zur Optimierungszeit.

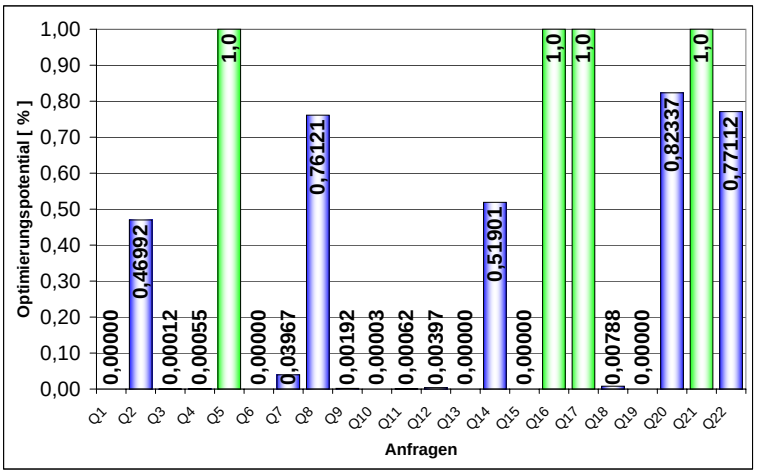


Abbildung 6.3: Optimierungspotential der TPC-H Version 2.3.0 Anfragen

6.3.2 Fazit der relationalen Optimierung

Die Tests auf relationalen Datenbeständen haben mehrere verschiedene Ziele verfolgt und entsprechende Ergebnisse gebracht: Zum Einen dienten die Tests, besonders zum Anfang unserer Untersuchungen, dazu, den nötigen Sprachumfang der relationalen Algebra zu testen und zu überprüfen. Wir haben, auch für spätere Arbeiten, Informationen zu den benötigten und verbrauchte Ressourcen gewonnen, z. B. bezüglich der Speichernutzung und CPU-Auslastungszeit. Dadurch konnte das System auf zukünftige Anforderungen durch Tests mit größeren und komplexeren Datenbeständen und objektrelationalen Anfragen vorbereitet werden.

In einigen Fällen der Optimierung empfahl `RELOpt` die Erzeugung von Indexen auf Teilmengen der Anfragen. Wie sich herausstellte, ist die Realisierung solcher optimierten Anfragen nicht ohne Umwege in Oracle 10g möglich. In solchen Fällen muss man die Anfrage aufteilen, die einzelnen Teilergebnisse materialisieren, darauf Indexe erzeugen und die restliche Anfrage beantworten lassen. Die Ausführungszeit für diese drei Komponenten muss zur Bewertung addiert werden, was das gesamte Vorgehen bislang nur manuell in Tests realisierbar macht. Benutzerfreundliche Anwendungen werden zur Zeit durch Einschränkungen der kommerziellen Optimierer erschwert. Die Tests haben uns also ein besseres Verständnis der Arbeitsweise des Oracle 10g-Optimierers gegeben.

Aus den Untersuchungsergebnissen von Haritsa et al. (2005) [70] bei einem Vergleich von heutigen kommerziellen Datenbankanfrageoptimierern ist bekannt, dass wir heutzutage zu viel Zeit in die Optimierung von relationalen Operatoren investieren, wobei große Performancegewinne nicht mehr zu erwarten sind:

„Overall, this leads us to the hypothesis that current optimizers may perhaps be over-sophisticated in that they are "doing too good a job", not merited by the coarseness of the underlying cost space. Moreover, if it were possible to simplify the optimizer to produce only reduced plan diagrams, it is plausible that the considerable processing overheads typically associated with query optimization could be significantly lowered.“

Daher ist es nicht verwunderlich, dass die Messergebnisse bezüglich der Ausführungszeiten von `RELOpt` und Oracle 10g im relationalen Fall entweder identisch oder mit solch einem minimalen Unterschied waren, dass auf die Wiedergabe verzichtet wurde.

Bezüglich des Optimierungspotentials sei gesagt, dass mit Hilfe von `RELOpt` bei 10 von 22 Anfragen, also knapp der Hälfte, eine rentable Optimierung mit den in dieser Arbeit vorgestellten Ersetzungsregeln durchgeführt werden konnte. Dies bestätigt die Annahme, dass eine Verbesserung noch möglich ist, und dass sie mit unseren Theorien erreicht werden kann.

Des Weiteren hat sich gezeigt, dass eine Verdoppelung der Suchbreite oberhalb einer systembedingten Grenze kaum Einfluss auf die Ergebnisse hat. Vielmehr waren es kostenbedingte Variantengrenzen, welche die Qualität und Laufzeit der Suche stark dominierten.

6.3.3 Objektrelationale Tests mit ATKIS-Daten

Da bisherige Benchmark-Tests für objektrelationale Datenbanken keine international spezifizierten Richtlinien für räumliche Operatoren vorsehen, wurden die bekannten Anfragetypen vergleichbar mit den TPC-H Version 2.3.0 Benchmark-Anfragen selbstständig auf 8 neue erweitert. Als Basisdatenstamm wurden Relationen aus dem ATKIS-Modell (Amtliches Topographisch-Kartographisches Informationssystem)[5] verwendet, die geometrische und alphanumerische Attribute beinhalten.

Als kommerzielle Datenbank wurde wiederum die zu Beginn des Kapitels vorgestellte Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - 64bit Production verwendet. Da RELOpt als ein sehr flexibles System entwickelt worden ist, bietet es die Möglichkeit der Anpassung an vorhandene Datenbestände. Die spezifischen Einstellungen für die von uns durchgeführten Experimente auf objektrelationalen Objekten werden im Folgenden vorgestellt.

Die für die Berechnung des Proportionalitätsfaktors $\omega(\varphi_{\mathsf{geo}}, \mathsf{R})$ benötigten Werte der in Abschnitt 4.10.1 vorgestellten Variablen sind in der Tabelle 6.4 angegeben. Diese wurden ermittelt, indem die Ausführungszeiten von relationalen zu räumlichen Selektionen und Joins mit jeweils gleicher Selektivität ins Verhältnis gesetzt worden sind. Da Oracle 10g weder einen räumlichen Hash- noch einen räumlichen Merge-Join unterstützt, konnten die Werte für die entsprechenden Variablen nicht bestimmt werden.

Faktor		Wert
s^{Rel}	:=	33
s^{Sort}	:=	nicht vorhanden
s^{Index}	:=	11
s^{Hash}	:=	nicht vorhanden
$s^{\mathsf{Index,Rel}}$	:=	11

Tabelle 6.4: Werte der Variablen

Folgende Regelanordnung mit der statischen Variantengrenze x wurde zur Optimierung objektrelationaler Anfragen verwendet:

$$[[(\text{Vorselektierungen})^*]_x[(\text{Alle impl.-unabh.-Regeln})^*]_x$$
$$[(\text{Alle räumlichen Erzeugungsregeln})^*|(\text{Alle räumlichen Impl.-varianten})^*]_x$$
$$[(\text{Alle relationalen Erzeugungsregeln})^*]_x[(\text{Alle relationalen Impl.-varianten})^*]_x]_1$$

[5] `http://www.atkis.de`

In diesem Fall werden räumliche Erzeugungsregeln und Implementierungsvarianten in einer Gruppe zusammengefasst, da z. B. das Erzeugen eines Indexes die Kosten so stark fallen lässt, dass durch die dynamische Kostengrenze keine anderen Varianten mehr generiert werden. Somit müssen entweder diese Regeln parallel ausgeführt oder mit Hilfe eines Simulated-Annealing angegangen werden. In dieser Arbeit wurde die Gruppierung gewählt, da sich damit bessere Suchzeiten ergaben.

	Benutzung von							
Anfrage	σ	σ_{geo}	$\bowtie$	$\bowtie_{\mathsf{geo}}$	$\overline{\ltimes}_{\mathsf{geo}}$	π	Γ	Sort
oQ1		x	x			x		
oQ2	x		x	x			x	
oQ3	x		x	x		x		
oQ4		x	x	x				x
oQ5	x	x	x		x	x		
oQ6		x	x	x			x	
oQ7	x		x	x		x	x	
oQ8	x	x	x	x		x		

Tabelle 6.5: Verteilung der Operatoren in den Anfragebäumen

Die neu konzipierten Anfragen oQ1 bis oQ8 wurden an den uns zur Verfügung stehenden räumlichen Datenbestand angepasst, so dass möglichst viele Variationen von Kombinationen der benutzten Operatoren verwendet wurden. Eine vollständige Darstellung der Anfragen, sowie der Ergebnisse der Optimierungen ist im Anhang ab Abschnitt D.2.1 aufgeführt.

Im Folgenden werden wir uns mit charakteristischen Eigenschaften der Anfragen befassen, deren Auflistung und Verteilung in den Anfragebäumen in Tabelle 6.5 zu finden ist. Dabei haben wir uns an den Baumstrukturen von TPC-H Version 2.3.0 orientiert, um eine angemessene Aussagekraft der Tests zu gewährleisten. Somit wurden vergleichbare Baumstrukturen mit den dazugehörigen Anordnungen von Joins, Selektionen und Aggregationen angestrebt.

	Benutzung von								
Anfrage	σ^{Rel}	$\sigma^{\mathsf{Rel}}_{\mathsf{geo}}$	$\sigma^{\mathsf{Index}}_{\mathsf{geo}}$	$\bowtie^{\mathsf{Rel,Rel}}$	$\bowtie^{\mathsf{Hash,Hash}}$	$\bowtie^{\mathsf{Rel,Index}}_{\mathsf{geo}}$	σ^{Index}	$\bowtie^{\mathsf{Sort,Sort}}$	Sort
oQ1		x			x				
oQ2	x				x	x			
oQ3	x			x	x	x	x	x	
oQ4		x	x		x	x			x
oQ5	x	x			x				
oQ6		x			x	x			
oQ7	x				x	x			
oQ8	x	x			x	x			

Tabelle 6.6: Verteilung der physischen Operatoren in den Ausführungsplänen I

	Benutzung von						
Anfrage	$\ltimes^{\mathsf{Hash,Hash}}$	$\overline{\ltimes}^{\mathsf{Index,Index}}_{\mathsf{geo}}$	INDEX	Pipelining	π^{Rel}	Γ^{Rel}	SORT
oQ1					x		
oQ2			x	x		x	
oQ3			x	x	x		x
oQ4			x	x			
oQ5	x	x	x	x	x		
oQ6						x	
oQ7			x	x	x	x	
oQ8			x	x	x		

Tabelle 6.7: Verteilung der physischen Operatoren in den Ausführungsplänen II

Während der Optimierung durch `RELOpt` wurden die besten Implementierungsarten für die benötigten Operatoren, ihre Anordnung sowie teilweise zusätzliche Erzeugungen von Indexen, Sortierungen und/oder Selektionen herausgearbeitet.

Bei der Konzeption der Anfragepläne lag der Fokus darauf, keine äquivalenten Beispiele zu erzeugen, sondern verschiedene, möglichst viele unterschiedliche Operator-Kombinationen beinhaltende Anfragen, die damit eine möglichst repräsentative Übersicht der Optimierungsmöglichkeiten bieten. Mit diesen Informationen werden die Ausführungspläne erstellt, die die in den Tabellen 6.6 und 6.7 aufgeführten physischen Operatoren beinhalten.

6.3.4 Resultate der objektrelationalen Optimierung mit `RELOpt`

Im Gegensatz zu den Ergebnissen der Experimente für relationale Anfragen bieten räumliche Datenbestände sehr viel mehr Optimierungspotential. Tabelle 6.8 gibt einen Überblick über Kosten der optimierten Varianten, die ins Verhältnis mit den Kosten der Ausgangsvarianten gesetzt wurden. Für die Wahl der Ausgangsvarianten wurden keine räumlichen Indexe genutzt, da sie erst durch die Optimierung berücksichtigt oder erzeugt werden sollten.

Anfrage	cost(optimierte Variante)	/	cost(Ausgangsvariante)	=	Verhältnis y
oQ1	12698	/	129539	=	$9.8024533 \cdot 10^{-2}$
oQ2	62836990	/	5130632283	=	$1.2247416 \cdot 10^{-2}$
oQ3	1828507	/	93398371	=	$1.9577504 \cdot 10^{-2}$
oQ4	100550	/	88870358	=	$1.131423 \cdot 10^{-3}$
oQ5	152189	/	97571780	=	$1.559765 \cdot 10^{-3}$
oQ6	301451	/	97943979	=	$3.07779 \cdot 10^{-3}$
oQ7	16464	/	2607657663	=	$6.314 \cdot 10^{-6}$
oQ8	5935	/	2997419078	=	$1.98 \cdot 10^{-6}$

Tabelle 6.8: Vergleich der Ergebnisse auf räumlichen Daten

Wieder ergibt sich das Optimierungspotential der einzelnen Anfragen als (1 - y) mit y als Wert des Verhältnisses der Kosten. Abbildung 6.4 stellt diese bildlich dar mit der aus Abschnitt 6.3.1 bekannten Gruppierung der Anfragen.

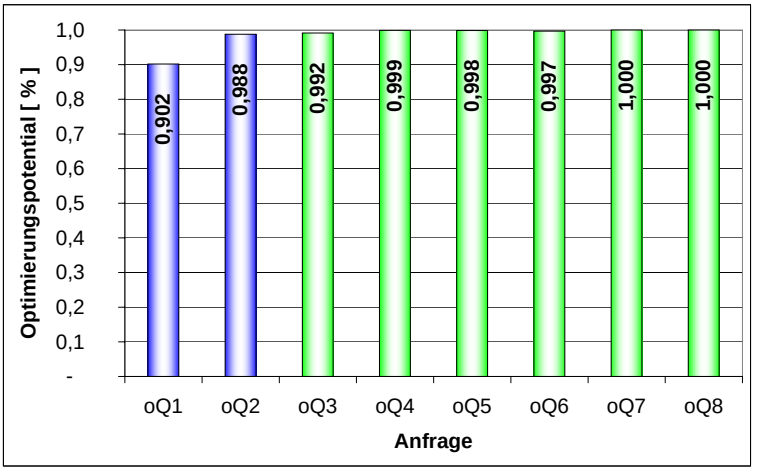


Abbildung 6.4: Optimierungspotential räumlicher Anfragen

Betrachtet und vergleicht man die Ausführungszeiten der beiden Optimierer in Tabelle D.5 des Anhanges D, so werden die signifikanten Verbesserungen mit dem von uns vogeschlagenen Optimierungskonzept deutlich.

Abbildung 6.5 zeigt zusammengefasst die Laufzeiten der 8 räumlichen Anfragen, die zum Testen des Optimierungskonzeptes verwendet wurden. In dem besten Fall konnte eine von `RELOpt` optimierte Anfrage von demselben System bis zu 2141-mal schneller bearbeitet werden als mit einer kommerziellen Datenbank wie Oracle 10g (siehe räumliche Beispielanfrage 3).[6]

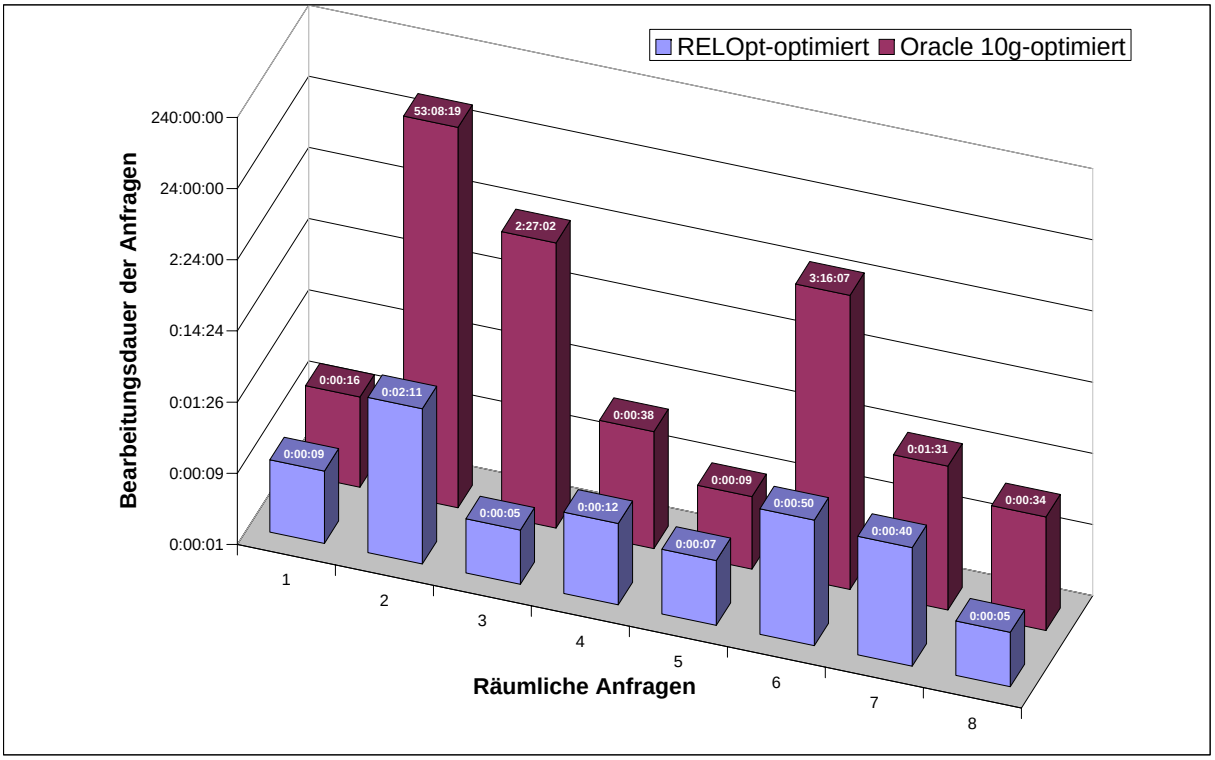


Abbildung 6.5: Dauer der Testanfragen jeweils mit Oracle 10g und `RELOpt` optimiert

6.3.5 Fazit der objektrelationalen Optimierung

Für die Strategiebetrachtungen hat sich auch bei räumlichen Anfragen, wie auch schon im relationalen Fall, Hill-Climbing als schnellste und effizienteste Suchstrategie herausgestellt (siehe Tabelle D.4 auf Seite 162). Für dieses Ergebnis muss man die Operatoren, die den größten Einfluss haben könnten, betrachten: Indexe und Sortierungen. Resultierend aus der Strategie des Hill-Climbings muss sich das Einfügen eines Indexes sofort lohnen, da er nur einmal verwendet wird und sich nicht vererben lässt. In sehr seltenen Fällen amortisierte sich der Mehraufwand einer ausgiebigeren Suche durch das Simulated-Annealing, da sich dort durch das Einfügen von Vorselektierungen nochmals Kosteneinsparungen realisieren ließen. Sortierungen dagegen machen unter den Bedingungen, dass es keine allgemein gültige Ordnung von Geometrien und daraus folgernd keine kommerzielle Implementierung gibt, grundsätzlich keinen Sinn.

Auch bei den Optimierungszeiten schneidet Simulated-Annealing schlechter ab als Hill-Climbing, da diese sogar extrem steigen. In den Tests ergaben sich bei der Erweiterung der Suche durchschnittlich 10- bis 20-mal längere Laufzeiten.

Mit der besten Strategie Hill-Climbing und der Optimierung durch `RELOpt` konnten Anfragen sehr viel schneller bearbeitet werden als mit Oracle 10g. Hier zeigt sich die Stärke unseres Konzeptes und die zukünftigen Einsatzmöglichkeiten, die durch bessere Optimierung und Flexibilität gekennzeichnet sind.

[6] Um eine kompaktere Darstellung zu erreichen, wurde die Achse der Bearbeitungsdauer logarithmisch skaliert.

6.4 Erfahrungen bei der Verwendung der Regeln

Nach der Betrachtung der kostenbedingten Suchstrategien und ihren Einflüssen auf das Endergebnis der Optimierung werden in den folgenden zwei Abschnitten Erfahrungen bezüglich der Anwendung und Kombination von Regeln beschrieben. Da wir fast alle Regeln parallel anwenden konnten, wurden viele Erfahrungen nicht in die Regelanordnung übernommen. Dennoch sollen sie im folgenden Abschnitt präsentiert werden, da sie in anderen Situationen eingesetzt werden können.

6.4.1 Verschmelzung von Regeln am Beispiel der Anordnung von Joins

Schaut man sich TPC-H Anfragen an (siehe Abschnitt 6.3.1), dann erkennt man, dass das größte Optimierungspotential bei der Anordnung von Joins und Selektionen vorhanden ist. Anders gesagt erzeugen Join-Ordering Regeln die meisten neuen Varianten mit den günstigsten Nachfolgevarianten.

Man kann mit Hilfe der Kommutativitätsregel F) (siehe Kapitel 3), z. B. bei einem Nested-Loop-Join, die „kleinere“ Relation als innere nehmen und somit die Kosten lokal immer senken. Unter Verwendung solcher Regeln könnte man jetzt im generalisierten Baum die Anordnung der Relationen strikt vorgeben. Diese Art von Regeln dienen dann als *Metaregeln*, da sie implizit ohne expliziten Aufruf angewendet werden können (siehe Abbildung 6.6).

$$\mathsf{R}_1 \bowtie\!\triangleleft_{\varphi} \mathsf{R}_2 \overset{1}{\underset{2}{\rightleftharpoons}} \mathsf{R}_2 \bowtie\!\triangleleft_{\varphi} \mathsf{R}_1,$$
$$1)\ \text{falls}\ |\mathsf{R}_2| < |\mathsf{R}_1|$$
$$2)\ \text{falls}\ |\mathsf{R}_2| \geq |\mathsf{R}_1|$$

Abbildung 6.6: Kommutativitätsregel F) mit erweiterten Bedingungen

Eine solche Optimierung mit *Metaregeln* ließe sich auch mit einer generalisierten Regel durchführen, würde aber den Aufwand eines zusätzlichen Iterationsschrittes bedeuten. Somit liefert die Idee der Anordnung von Regeln aufbauend auf vorherigem Wissen über die Kostenentwicklung die Strategie, zwei Regeln zu einer zusammenzufassen.

Allein solch ein Ausschließen von Varianten, die sich nur durch gespiegelte Teilbäume unterscheiden, reduziert die Anzahl von möglichen Kombinationen für das Join-Ordering durch Termersetzungsregeln. Das einzige Problem ist, dass durch die Einschränkung der Kommutativität nicht mehr alle Varianten erzeugt werden. Grund dafür ist wiederum die lokale Optimierung, die global falsch sein kann. Um dieses Manko auszumerzen, fasst man die Assoziativitätsregeln (Regel G) mit den Kommutativitätsregeln (Regel F)) zusammen:

FG1)
$$(\mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_2) \bowtie_{\varphi_2}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_3 \overset{1}{\underset{2}{\rightleftharpoons}} \mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel},\mathsf{Rel}} (\mathsf{R}_2 \bowtie_{\varphi_2}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_3)$$
$$1)\ \text{falls}\ \mathsf{attr}(\varphi_2) \cap \mathbf{sch}(\mathsf{R}_1) = \emptyset$$
$$2)\ \text{falls}\ \mathsf{attr}(\varphi_1) \cap \mathbf{sch}(\mathsf{R}_3) = \emptyset$$

FG2)
$$(\mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_2) \bowtie_{\varphi_2}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_3 \overset{1}{\underset{2}{\rightleftharpoons}} (\mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_3) \bowtie_{\varphi_2}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_2$$
$$1)\ \text{falls}\ \mathsf{attr}(\varphi_2) \cap \mathbf{sch}(\mathsf{R}_2) = \emptyset$$
$$2)\ \mathsf{false}$$

FG3)
$$(\mathsf{R}_1 \bowtie_{\varphi_1}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_2) \bowtie_{\varphi_2}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_3 \overset{1}{\underset{2}{\rightleftharpoons}} (\mathsf{R}_3 \bowtie_{\varphi_1}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_2) \bowtie_{\varphi_2}^{\mathsf{Rel},\mathsf{Rel}} \mathsf{R}_1$$
$$1)\ \text{falls}\ \mathsf{attr}(\varphi_2) \cap \mathbf{sch}(\mathsf{R}_1) = \emptyset$$
$$2)\ \mathsf{false}$$

FG4)

$$\mathsf{R}_1 \bowtie^{\mathsf{Rel},\mathsf{Rel}}_{\varphi_1} (\mathsf{R}_2 \bowtie^{\mathsf{Rel},\mathsf{Rel}}_{\varphi_2} \mathsf{R}_3) \overset{1}{\underset{2}{\rightleftharpoons}} (\mathsf{R}_2 \bowtie^{\mathsf{Rel},\mathsf{Rel}}_{\varphi_1} (\mathsf{R}_1 \bowtie^{\mathsf{Rel},\mathsf{Rel}}_{\varphi_2} \mathsf{R}_3)$$

1) falls $\mathsf{attr}(\varphi_1) \cap \mathbf{sch}(\mathsf{R}_2) = \emptyset$
2) false

FG5)

$$\mathsf{R}_1 \bowtie^{\mathsf{Rel},\mathsf{Rel}}_{\varphi_1} (\mathsf{R}_2 \bowtie^{\mathsf{Rel},\mathsf{Rel}}_{\varphi_2} \mathsf{R}_3) \overset{1}{\underset{2}{\rightleftharpoons}} (\mathsf{R}_3 \bowtie^{\mathsf{Rel},\mathsf{Rel}}_{\varphi_1} (\mathsf{R}_2 \bowtie^{\mathsf{Rel},\mathsf{Rel}}_{\varphi_2} \mathsf{R}_1)$$

1) falls $\mathsf{attr}(\varphi_1) \cap \mathbf{sch}(\mathsf{R}_3) = \emptyset$
2) false

Schließlich sind die Regeln FG1)- FG5) nachweisbar äquivalent zu der Anwendung von einzelnen Kommutativitäts- und Assoziativitätsregeln (siehe [73]). Dies führte mit Hilfe der anfangs genannten Metaregel dazu, dass sie in dieser Arbeit die herkömmlichen einfachen Regeln F) und G) ersetzten.

6.4.2 Strategien durch Präferierung von Ersetzungsregeln

Eine weitere Strategie besteht darin, Regeln mit „größerem“ Optimierungspotential früher als andere, sowie gruppiert anzuwenden. Dabei liegt der Hauptgewinn darin, dass die Anzahl von auszuführenden Regeln von vorneherein eingeschränkt wird, da schon frühzeitig Varianten erzeugt werden, die geringere Kosten verursachen. Optimierungsregeln wie ein Join-Ordering oder das Pushen von Selektionen sind Bestandteil dieser Gruppierungsidee. Man findet solche Regelabläufe schon bei der bisherigen algebraischen Optimierung. Dort hatten sich heuristische Anordnungen von Regeln (siehe Beispiele für Regelanordnungen bei [38], [57] und [78] und in Abbildung 6.9) als sinnvoll herausgestellt. In unseren Experimenten konnten durch die Erweiterung der Bedingungsmenge Regeln sogar fast ausschließlich parallel ausgeführt werden.

Folgende Auflistung von in dieser Arbeit gewonnen Erfahrungswerten bei der Verwendung von Regeln aus dem 3. Kapitel wird als Grundlage verwendet:

- Man kann leicht erkennen, dass mit Verwendung der Regel H) aus Abbildung 6.7, eine Optimierung der Anfrage in diesem Fall durch die Wahl der restriktivsten Selektion als innerste Selektion erreicht wird.

$$\sigma_{\varphi_1}(\sigma_{\varphi_2}(\mathsf{R})) \overset{1}{\underset{2}{\rightleftharpoons}} \sigma_{\varphi_2}(\sigma_{\varphi_1}(\mathsf{R}))$$

1) falls $\mathsf{sel}(\varphi_1, \mathsf{R}) \leq \mathsf{sel}(\varphi_2, \mathsf{R})$
2) falls $\mathsf{sel}(\varphi_1, \mathsf{R}) \geq \mathsf{sel}(\varphi_2, \mathsf{R})$

Abbildung 6.7: Regel H) mit Selektivitätsbedingungen

Vorsicht sollte man bei der zusätzlichen Kombination von räumlichen mit alphanumerischen Selektionsbedingungen walten lassen, da es sein kann, dass das System keine sinnvolle Anordnung der Selektionen vornehmen kann, da bisherige Optimierer beide Bedingungen schlecht in Beziehung setzen können (siehe Beispiel im Abschnitt 4.3 auf Seite 61).

Regel I ←
Regel K →
Regel L →
Regel N →
Regel D →
Regel I ←
Regel P →
Regel M ←
Regel G →
Regel M →
Regel O →
Regel Q →
Regel E →
Regel J →
Regel M ←
Regel O ←
Regel B →
Regel R →
Regel C →
Regel F →

Tabelle 6.9: heuristische algebraische Regelanordnung

Dagegen ist die Anordnung der Selektion nach ihrer Selektivität mit ausschließlich alphanumerischen Attributen eine der schnellsten und aussichtsreichsten Heuristiken.

- Beim Pushen von Projektionen unter eine Selektion sollte darauf geachtet werden, dass sich durch das Wegfiltern von Attributen auch die flexible Anordnung der Operatoren verändert. Allgemein gesprochen wird bei der Benutzung der Regel K) letztlich die Anzahl von Ausführungsvarianten stark eingeschränkt. Demzufolge sollte die Anordnung der Projektionen während eines Join-Ordering in gleicher Weise betrachtet werden, wie es beim Selektionsoperator der Fall ist.
- Die wohl wichtigste Optimierungsregel ist die Regel P), die Kartesische Produkte verhindert und zu kostengünstigeren Verbundoperationen führt. Da auch hier eine gleichzeitige Anordnung von Selektionen stattfindet, sollte bei der Anordnung die gesamte Selektionsmenge betrachtet werden. Erst dann kann die maximale Flexibilität bei der Anordnung der Operatoren erreicht werden.
- Die Regel T) hängt, genau wie Regel L), von der Richtung der Anwendung bzw. von der Selektivität des Joinprädikates und der Größe des daraus resultierenden Ergebnisses ab, verglichen mit der Anwendung auf einer der Argumentrelationen. Generell gesagt verringert eine frühzeitige Gruppierung die Anzahl von Tupeln und somit auch die Anzahl von belegten Seiten. Damit könnten für diese Regeln genauso Indikatorfunktionen entwickelt werden, wie für die Regeln L) und M) (siehe Abschnitt 3.4).
- Die physischen Regeln, z. B. (a) aus Abschnitt 3.2.3, haben alle etwas gemeinsam: bei der Wahl eines Indexes sollte diejenige physische Operatorvariante benutzt werden, welche die geringsten Kosten verursacht; z. B. lohnt sich der Zugriff über einen Index nur für stark selektive Selektionen. Ausnahmen bilden räumliche Operatoren, bei denen die Selektivität bisweilen auf bis zu 0,5 steigen kann, ohne dass sich der Indexzugriff als teurere Operation herausstellt.
- Rechenregeln für Mengenoperationen und Selektionen, wie Regel B) aus Abbildung 6.8, bieten die Möglichkeit, mehrattributige Indexe für Selektionen zu nutzen, falls mit deren Hilfe die Selektionen zusammengefasst werden können.

$$\sigma_{\varphi_1}(R) \cup \sigma_{\varphi_2}(R) \rightleftharpoons \sigma_{\varphi_1 \vee \varphi_2}(R)$$
$$\sigma_{\varphi_1}(R) \cap \sigma_{\varphi_2}(R) \rightleftharpoons \sigma_{\varphi_1 \wedge \varphi_2}(R)$$
$$\sigma_{\varphi_1}(R) - \sigma_{\varphi_2}(R) \rightleftharpoons \sigma_{\varphi_1 \wedge \neg\varphi_2}(R)$$

Abbildung 6.8: Regel B)

Durch die vorgestellten Erfahrungen konnten Gruppierungen von Regeln, wie sie in früheren Arbeiten, wie z. B. Volcano [9], [14], [29], [61], [89], vorgeschlagen wurden, dahingehend verbessert werden, den Regeln ein maximales Wissen zu übergeben, so dass sie nur noch stark restriktiv angewendet werden. Eine Regel sollte nur dann eine neue Variante erzeugen, wenn dies sinnvoll sein könnte. Damit spricht nichts dagegen, die Erzeugung einer Variante so früh wie möglich zu verhindern, wenn aus ihr durchweg nur teurere Varianten erzeugt werden.

6.5 Zusammenfassung der Ergebnisse

Generell besteht während des Optimierungsprozesses ein Konflikt: Je mehr Varianten betrachtet werden, um so höher ist die Wahrscheinlichkeit, dass das globale Minimum gefunden wird. Viele Varianten bedeuten aber auch lange Optimierungszeiten. Demzufolge muss ein Kriterium gefunden werden, welches die Optimierungsdauer ins Verhältnis mit der Ausführungszeit setzt.

In unserem Fall besitzen wir einen Vorteil gegenüber anderen Optimierungsstrategien (z. B. dem Join-Ordering mit Hilfe von Dynamischer Programmierung): es existiert zu jeder Zeit ein

Ausführungsplan, der verwendet werden kann. Somit ist eine Maxime, dass wir keine weitere Ersetzung vornehmen, wenn die Optimierung länger dauern würde als das sofortige Ausführen.

Wie schon vorgestellt, sollte als Suchstrategie für die Auswahl der Varianten, die weiterverfolgt werden, der Ansatz von Hill-Climbing genutzt werden.

Für die Regelanwendung und -anordnung gab es zu den zuvor genannten Erfahrungen zwei wichtige Ergebnisse: zum Einen sollten „Super-Regeln“ wie das Join-Ordering mit Hilfe der Dynamischen Programmierung oder der Minimum-Selectivity Algorithmus nicht verwendet werden. Diese Optimierungsverfahren, die gegenüber einfachen Transformationsregeln den gesamten Ausführungsplan erst zerlegen müssen, um ihn dann wieder zusammenzubauen, kosten zu viel Berechnungszeit. Da mit der im Kapitel 3 vorgestellten Menge an generalisierten Regeln jede relationale und in unserem Fall auch jede räumliche Anfrage optimiert werden konnte, gibt es also eine sich lohnende Alternative zu den vorgestellten „Super-Regeln“. Für keine einzige Anfrage wurde in zusätzlichen Experimenten ein schlechteres Ergebnis als mit den „Super-Regeln“ gefunden, so dass sie weggelassen werden können.

Zum Anderen zeigte sich bezüglich der Kombination von Regelanordnungstrategien und kostenbedingten Strategien für die Suchkorridore, dass der Einfluss der ersten Gruppe sehr viel kleiner ist als der der zweiten. Grund dafür war die parallele Ausführung der generalisierten Regeln, welche die Regelanordnung nicht notwendig machte.

Insgesamt konnten bei allen Tests für räumliche Daten sehr viel bessere Ausführungszeiten durch den geringeren Berechnungsaufwand festgestellt werden. Damit haben wir gezeigt, dass das vorgestellte Optimierungsverfahren für objektrelationale Datenbanken mit den angegebenen Strategien und Regelanordnungen den bisher etablierten Anfrageoptimierungskonzepten überlegen ist.

Nos mathematici sumus isti veri poetae
sed quod fingimus nos et probare decet.
Leopold Kronecker 1823-1891



7 Implementierung

In diesem Kapitel wird die Implementierung von `RELOpt` und den dazu gehörenden Modulen, welche in Java 1.5 realisiert worden sind, vorgestellt. Dabei werden die einzelnen Module klassifiziert und ihre Nutzung näher erläutert. Nach einem kurzen Überblick der Pakete folgt eine ausführliche Darstellung des Termersetzers, wie er im Zuge dieser Arbeit realisiert worden ist. Des Weiteren wird die in Kapitel 5 vorgestellte Einphasenoptimierung mit einem Sequenzdiagramm genau erläutert, so dass der zeitliche Ablauf der Optimierung verdeutlicht werden kann.

7.1 Das System des Anfragensimulators RELOpt

Der Anfragenoptimierungssimulator `RELOpt` besteht aus zwei Optimierungsmodulen. Die Dreiphasenoptimierung verbessert einen Ausführungsplan per algebraischer, physischer und nachfolgender kostenbasierter Optimierung. Dabei wird der Ausführungsbaum rekursiv durchlaufen und schrittweise transformiert. Für eine genaue Darstellung der Implementierung wird auf die vorherige Arbeit [57] verwiesen. Eine allgemeine Darstellung der Paketstruktur befindet sich in Abbildung 7.1.

Als einziges externes Modul beinhaltet das Paket *dbs.unihannover.db.oracle* alle Klassen, die zur Kommunikation mit einer Oracle Datenbank benötigt werden. Das Paket *dbs.unihannover.sopt.gui* enthält die zur Darstellung des Startfensters und aller weiteren grafischen Masken verwendeten Klassen. Eine Ausnahme bildet das Paket *dbs.unihannover.sopt.tree*, welches als externes Paket die Visualisierung eines Ausführungsplanes übernimmt. Die Hauptklasse `RELOpt` befindet sich im Paket *dbs.unihannover.sopt.test*. Die erste Realisierung eines relationalen Parsers findet sich im Paket *dbs.unihannover.sopt.sqlp*. Dieser Parser wird nur während einer Dreiphasenoptimierung genutzt. Der weitaus mächtigere und später entwickelte Parser befindet sich im Paket *dbs.unihannover.sopt.struc.general*, der auch die für einen generalisierten Baum benötigten Klassen vorhält. Im weiteren Verlauf des Kapitels werden die wichtigsten Klassen und Methoden beschrieben. Eine grafische UML-Übersicht wird in der Abbildung 7.2 gegeben.

7.1.1 Package *dbs.unihannover.sopt.struc.general*

Die Klasse *GeneralRelationalParser* verfügt über folgende Methoden:

- **parseTree**

  Diese Methode erwartet als Parameter eine relationale Anfrage als Zeichenkette, eine Liste der Basisrelationen und die zugehörigen Metadaten. Die Ausgabe ist ein generalisierter Baum.

- **parseRule**

  Die Übergabewerte an diese Methode sind jeweils zwei Zeichenketten für die Regelterme und die Regelbedingungen. Als Ausgabe erhält man eine komplette Termersetzungsregel.

Die Klasse *RuleConditionParser* wird von *GeneralRelationalParser* verwendet und verfügt über folgende Methoden:

- **parse**

  Diese Methode erwartet als Parameter eine Regelbedingung als Zeichenkette. Die Ausgabe ist ein Objekt der Klasse *dbs.unihannover.sopt.struc.general.opitmizer.GeneralRule*.

- **getTokens**

  Nachdem *parse* ausgeführt wurde, ist die Ausgabe dieser Funktion eine Liste der einzelnen terminalen Symbole einer Regelbedingung.

Die Klasse *RuleCreater* erzeugt eine GUI zur Bearbeitung und Erzeugung von Regeln und Regelbäumen. Sie kann mit Hilfe des Konstruktors oder direkt über die Methode *main* gestartet werden.

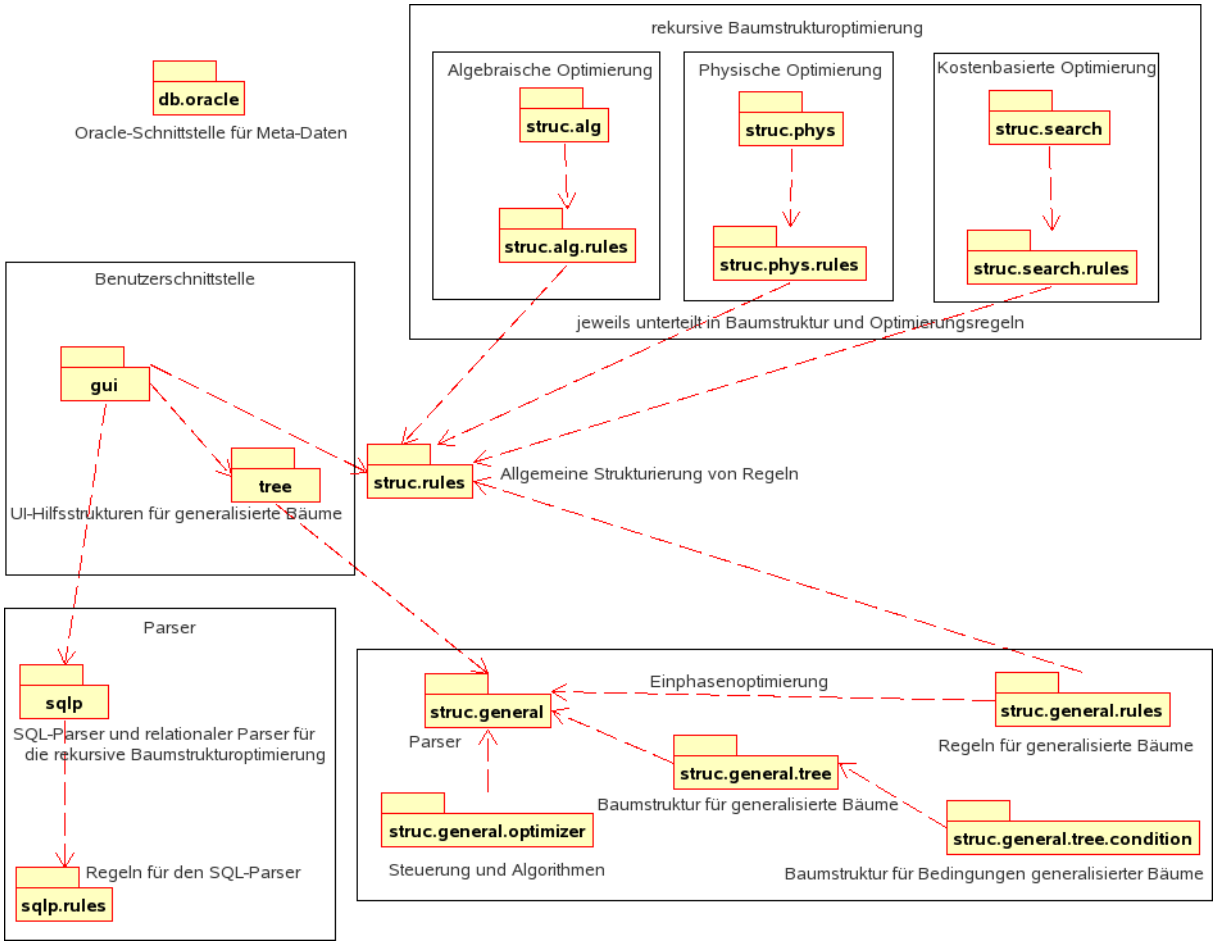


Abbildung 7.1: Paketstruktur von `RELOpt`

7.1.2 Package *dbs.unihannover.sopt.struc.general.optimizer*

In der Klasse *ControlSystem* befindet sich die Steuerung der Einphasenoptimierung. Eine Instanz dieser Klasse startet den Optimierungsprozess. In ihr werden alle Varianten als Listen von Listen gespeichert. Dabei wird in jedem Iterationsschritt eine Liste von Varianten erzeugt, die in diesem Schritt generiert worden sind. Eine eindeutige Kennzeichnung der Varianten erfolgt über ihre Position in der Liste der Listen. Eine Identifikation einer bestimmten Variante ergibt sich aus dem i-ten Iterationsschritt, in dem die Variante zum ersten Mal generiert worden ist und der Anzahl der bereits in diesem Schritt gefundenen weiteren Varianten. Zusätzlich dazu wird

für jede Variante eine Liste geführt, in der ihre Ursprungsvarianten und die darauf angewandten Regeln gespeichert werden.

Aus Effizienzgründen wurde eine Hashmap zur Speicherung der Kosten für jede neue Variante gewählt. Dabei dient der Kostenwert als Schlüssel für die Hashmap. Dieses ermöglicht die schnelle Duplikatenfindung, da neue Schlüssel auch neue Varianten bedingen.

- **countVariants**

  Diese Funktion gibt am Ende einer Optimierung die Anzahl der entstandenen Varianten zurück.

- **getBestTree**

  Es wird am Ende einer Optimierung der beste Ausführungsplan zurückgegeben.

- **getConvertStatus**

  Falls ein Anordnen der Regelgruppen fehlschlägt, wird von dieser Methode der Rückgabewert $false$ geliefert. Damit kann überprüft werden, ob die Regeln in einer für die Steuereinheit nutzbaren Form vorliegen.

- **getMaxIterations**

  Diese Methode liefert die Einstellungen für die Begrenzung der Iterationen.

- **getProtocol**

  Nach einer erfolgreichen Optimierung kann mit dieser Funktion ein Protokoll dieser abgefragt werden.

- **getVariants**

  Mit dieser Funktion kann nach einer erfolgreichen Optimierung eine Liste der entstandenen Varianten ausgegeben werden.

Die Klasse *GeneralRule* dient dem Speichern von Termersetzungsregeln für die generalisierte Einphasenoptimierung. Dabei werden jeweils zwei Terme und die dazugehörigen Listen mit Bedingungen gespeichert. Dadurch kann für jede Richtung, in der man die Regel lesen und ausführen kann, die Bedingungsmenge zurückgegeben werden.

- **copy**

  Mit dieser Funktion wird eine Regel kopiert.

- **getLeftConditions**

  Diese Methode liefert eine Liste der Regelbedingungen für die linke Regelseite.

- **getRightConditions**

  Es wird eine Liste der Regelbedingungen für die rechte Regelseite geliefert.

- **getLeftTree**

  Diese Methode liefert den Regelbaum der linken Regelseite.

- **getName**

  Der Name einer Regel kann hiermit abgefragt werden. Unter anderem kann diese Methode dazu genutzt werden, um innerhalb von Varianten festzustellen, unter Benutzung welcher Regel sie entstanden ist.

- **getRightTree**

  Es erfolgt die Ausgabe des Regelbaumes der rechten Regelseite.

- **setDescription**

  Mit dieser Funktion wird einer Regel eine Beschreibung hinzugefügt.

- **setRuleName**

  Mit dieser Funktion wird der Regelname gesetzt.

Die Klasse *Pointer* erzeugt einen Zeiger (Iterationsschritt i, Variante v) auf eine Variante im Entstehungsgraph. Dadurch wird das mehrfache Abspeichern einer Variante unterbunden.

- **clone**

  Erzeugt eine tiefe Kopie eines Pointers.

- **getIPosition**

  Diese Methode liefert den Positionsanteil i.

- **getVPosition**

  Diese Methode liefert den Positionsanteil v.

- **setPointer**

  Mit dieser Funktion werden i und v gesetzt.

Das Termersetzungssystem der Einphasenoptimierung befindet sich in der Klasse $ReWriter$. Mit der Methode $startReWriting$ wird der Ersetzungsprozess gestartet.

Mit Hilfe der Klasse *RuleCondition* wird eine Ersetzungsbedingung einer Termersetzungsregel gespeichert.

- **getTokens**

  Die Ausgabe dieser Funktion ist eine Liste der einzelnen terminalen Wörter einer Regelbedingung.

Die Klasse *RuleConditionAnalyzer* wird während des Optimierungsprozesses genutzt, um die Gültigkeit einer Regelbedingung zu überprüfen. An dieser Stelle sei erwähnt, dass der hauptsächliche Zeitverbrauch während der Optimierung durch die Methode *checkCondition* verursacht wird, da sie die mögliche Anwendung einer Regel untersucht.

Mit Hilfe der Klasse *RuleGroups* können Termersetzungsregeln in Gruppen zusammengefasst werden.

- **addRulesToGroups**

  Diese Funktion fügt eine Liste von Regeln zu einzelnen Gruppen hinzu.

- **addRuleToAGroup**

  Diese Methode fügt eine einzelne Regel zu einer Gruppe hinzu.

- **getGroup**

  Die Rückgabe dieser Methode ist eine Regelgruppe.

- **showRowsAndCols**

  Mit dieser Methode hat man die Möglichkeit, sich die Zeilen- und Spaltennamen der Gruppierungsmatrix anzuschauen.

Die Klasse *Variant* dient der Speicherung von Varianten, welche während der Optimierung entstanden sind. Darüber hinaus wird zu jeder einzelnen Variante ihre „Entstehungsgeschichte“ in Listen gespeichert. Außerdem besitzt eine Variante einen Aktivitätsstatus, der die Nutzung dieser Variante für die Generierung weiterer Varianten steuert.

- **addIterationStep**

  Diese Funktion fügt den Iterationsschritt hinzu, in dem die betrachtete Variante entstanden ist.

- **addPreCursor**

  Diese Methode erzeugt ein Objekt der Klasse *Pointer*. Dieser Zeiger repräsentiert eine Variante, aus der die aktuelle hervorgegangen ist.

- **addRule**

  Eine Regel, aus der die betrachtete Variante hervorgegangen ist, wird mit dieser Funktion gespeichert.

- **getActivity**

  Diese Methode gibt während des Optimierungsprozesses an, ob eine Variante weiterhin aktiv ist.

- **getDeactivationStep**

  Mit dieser Methode wird der Iterationsschritt ausgegeben, in dem diese Variante deaktiviert wurde.

- **getGeneralTree**

  Diese Funktion gibt den gespeicherten generalisierten Baum aus.

- **getIterationStepVec**

  Die Ausgabe dieser Funktion ist eine Liste der Iterationsschritte, in denen diese Variante entstanden ist.

- **getOwnPosition**

  Mit dieser Methode wird der Zeiger für diese Variante zurückgegeben.

- **getPreCursorVec**

  Es wird durch diese Methode eine Liste von Zeigern der Ursprungsvarianten ausgegeben.

- **getRuleVec**

  Diese Funktion gibt eine Liste von Ursprungsvarianten aus.

- **setActivity**

  Mit dieser Funktion kann eine Variante aktiviert oder deaktiviert werden.

- **setGeneralTree**

  Zum Setzen des generalisierten Baumes wird diese Methode verwendet.

- **setOwnPosition**

  Diese Funktion setzt den Zeiger für diese Variante.

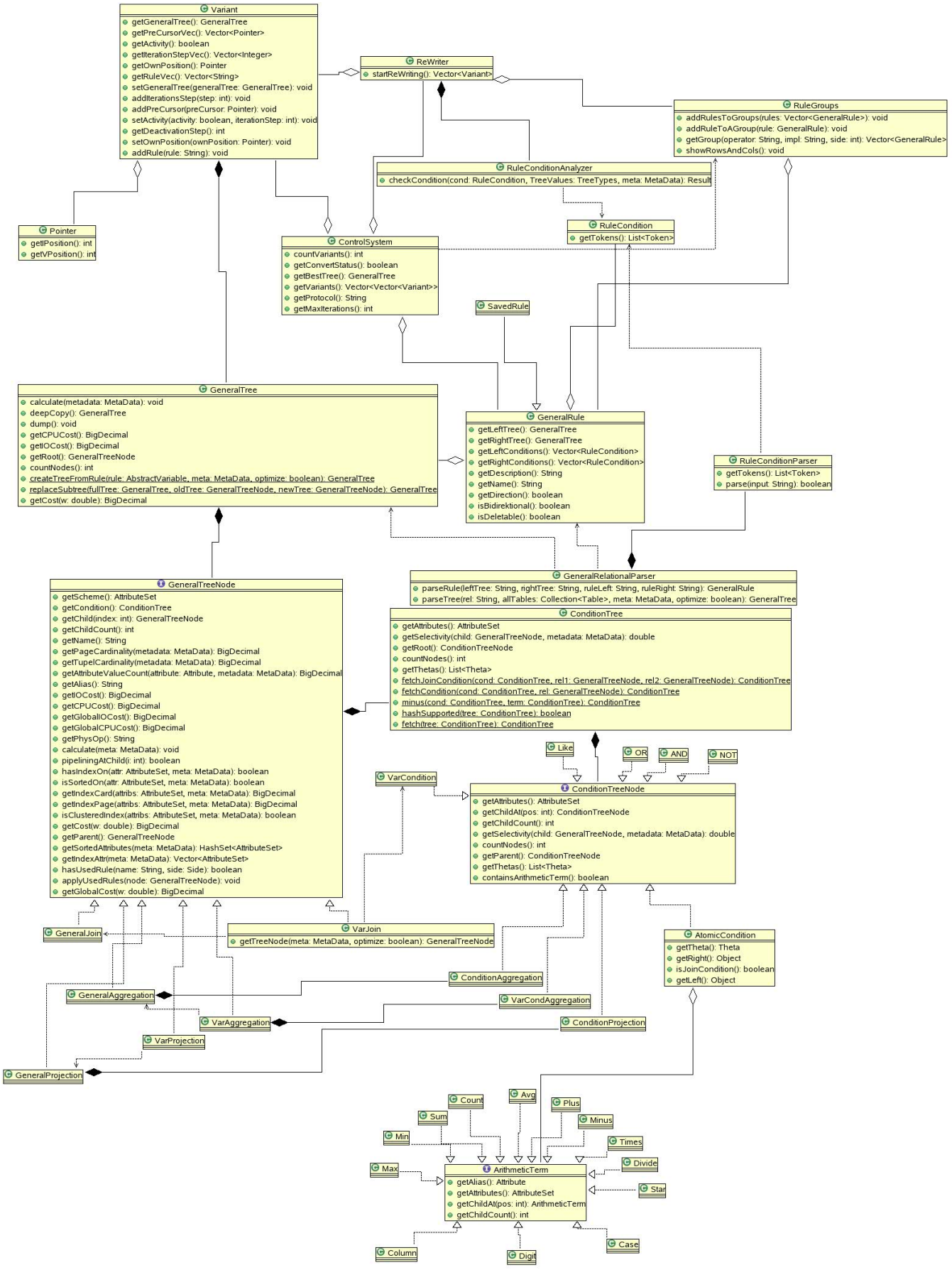


Abbildung 7.2: Package *dbs.unihannover.sopt.struc.general*

7.1.3 Package *dbs.unihannover.sopt.struc.general.tree*

In diesem Paket befinden sich die Klassen für den Aufbau von generalisierten Bäumen (siehe Abbildung 7.3).

Die Klasse *GeneralTree* speichert die Knoten des Baumes und bietet folgende Funktionen:

- **calculate**

  Mit dieser Methode kann ein Baum neu berechnet werden, falls sich z. B. die Metadaten geändert haben.

- **countNodes**

  Die Ausgabe dieser Methode ist die Anzahl der Knoten und Blätter im Baum.

- **createTreeFromRule**

  Dies ist eine Funktion zum Erstellen eines generalisierten Baumes aus einem Regelterm.

- **deepCopy**

  Um eine Kopie des Baumes zu erhalten, muss diese Methode aufgerufen werden.

- **dump**

  Mit dieser Funktion kann der Aufbau des Baumes als Zeichenkette ausgegeben werden.

- **equals**

  Mit *equals* kann überprüft werden, ob zwei Bäume gleich sind.

- **getCost**

  Die Gesamtkosten des Ausführungsplanes werden mit dieser Methode ausgegeben.

- **getIOCost**

  Die I/O-Kosten des Ausführungsplanes werden mit dieser Methode ausgegeben.

- **getCPUCost**

  Die ungewichteten CPU-Kosten werden mit dieser Methode ausgegeben.

- **getRoot**

  Diese Funktion liefert die Wurzel des gespeicherten Baumes.

- **mergeSupported**

  Mit dieser Funktion wird geprüft, ob bei einem Join eine vorhandene Sortierung genutzt werden kann.

- **postOrderIterator**

  Iterator zum Durchlaufen eines Baumes.

- **postOrderHasNext**

  Es findet eine Überprüfung statt, ob ein Post-Order-Iterator weitere Elemente enthält.

- **postOrderNext**

  Die Ausgabe des nächsten Elementes im Baum geschieht mit dieser Funktion.

- **replaceSubTree**

  Um einen Teilbaum in einem Ausführungsplan austauschen zu können, existiert diese Methode.

Um zukünftlig Erweiterungen einfach vornehmen zu können, wurde ein Interface entwickelt, welches jeder Knoten in einem generalisierten Baum implementieren muss. Dieses Interface heißt *GeneralTreeNode*. Es fordert folgende Methoden:

- **addUsedRule**

  fügt einem Knoten eine Regel hinzu, aus der dieser hervorgegangen ist

- **applyUsedRules**

  fügt einem Knoten eine Menge von Regeln hinzu, welche er von einem übergebenen Knoten erhält

- **calculate**

  berechnet die Attribute eines Knotens neu

- **countNodes**

  gibt die Anzahl der Knoten in diesem Teilbaum aus

- **deepCopy**

  liefert eine Kopie des Teilbaumes, mit diesem Knoten als Wurzel

- **dump**

  gibt den Aufbau des Teilbaumes als Zeichenkette aus

- **equals**

  überprüft, ob zwei Teilbäume gleich sind

- **getAlias**

  liefert den Alias einer Operation

- **getAttributeValueCount**

  gibt die Attributwertanzahl zu einem Attribut aus

- **getChild**

  liefert den Kindknoten an der übergebenen Position

- **getChildCount**

  gibt die Anzahl der Kinder eines Knotens an

- **getCondition**

  gibt die zu der Operation gehörige Bedingungsparametrisierung aus

- **getCost**

  liefert die lokalen Gesamtkosten

- **getGlobalCost**

  liefert die globalen Gesamtkosten

- **getCPUCost**

  gibt die lokalen CPU-Kosten aus

- **getGlobalCPUCost**

  informiert über globale CPU-Kosten

- **getIOCost**

  liefert die entstandenen I/O-Kosten an diesem Knoten

- **getGlobalIOCost**

  liefert die bisher entstandenen I/O-Kosten des Teilbaumes

- **getIndexAttr**

  gibt eine Liste mit Attributmengen, auf denen ein Index existiert, zurück

- **getIndexCard**

  liefert die Kardinaliät eines Index

- **getIndexPage**

  liefert die Seitengröße eines Index

- **getName**

  gibt den Namen einer Operation oder Relation aus

- **getPageCardinality**

  liefert die Pagegröße einer Relation

- **getParent**

  liefert den übergeordneten Knoten

- **getPhysOp**

  gibt die physische Umsetzung einer Operation aus

- **getScheme**

  liefert das Schema einer Relation

- **getSortedAttributes**

  gibt eine Liste von Attributmengen aus, auf denen eine Sortierung existiert

- **getTupelCardinality**

  liefert die Kardinalität einer Relation

- **hasUsedRule**

  prüft, ob die übergebene Regel bereits auf diesen Knoten angewandt wurde

- **hasIndexOn**

  sagt aus, ob auf einer Attributmenge ein Index existiert

- **isClusteredIndex**

  informiert, ob ein Index geclustert ist

- **isSortedOn**

  bestimmt, ob auf einer Attributmenge eine Sortierung existiert

- **pipeliningAtChild**

  gibt an, ob bei der Relation an der übergebenen Position Pipelining angewandt werden kann

- **setAlias**

  setzt bei einer Relation einen Alias

- **setChild**

  setzt das Kind an der übergebenen Position

- **setParent**

  setzt den übergeordneten Knoten

- **setPhysOp**

  setzt die physische Umsetzung

Dieses Interface wird direkt in der Klasse *GeneralTable* umgesetzt, welche eine Basisrelation darstellt. Für die anderen Operatoren wurden zur Unterscheidung von variablen und normalen Knoten abstrakte Oberklassen implementiert, welche das Interface realisieren.

Für die normalen Operatoren existieren die abstrakten Klassen *AbstractBinaryOperation* für binäre Operatoren und *AbstractUnaryOperation* für Operatoren mit nur einer Argumentrelation. Diese Klassen implementieren bereits einen Großteil der von *GeneralTreeNode* geforderten Methoden.

Für variable Knoten wurde die abstrakte Klasse *AbstractVariable* eingeführt. Sie implementiert einen großen Teil der Methoden von dem Interface *GeneralTreeNode*, fügt aber weitere hinzu, welche von variablen Knoten umgesetzt werden müssen:

- **getTreeNode**
  liefert einen konkreten Knoten, indem sie die Werte des Regelknotens verwendet
- **getRealCondition**
  gibt bei einem gefüllten Knoten den Wert der Variablen aus
- **setRealCondition**
  setzt den Wert für die variable Parametrisierung

Zur Unterscheidung von binären und einfachen variablen Operationen wurden ebenfalls zwei Klassen, *AbstractUnaryVariable* und *AbstractBinaryVariable*, eingeführt.

7.1.4 Package *dbs.unihannover.sopt.struc.general.tree.condition*

Die Klassen zur Erstellung von Parametrisierungen für Operationen sind in dem Package *.tree.condition* zu finden. Um arithmetische Ausdrücke speichern zu können, wurde eine Baumstruktur für die Operatoren und Operanden entworfen. Jeder Knoten in dieser Struktur muss das Interface *ArithmeticTerm* implementieren.

Die Klassen **Plus**, **Minus**, **Times** und **Divide** stellen die Operatoren $+, -, \cdot$ und / dar. Die Klasse **Column** wird zum Speichern von Attributen und die Klasse **Digit** zum Speichern von Zahlen genutzt. Die Aggregationsfunktionen **Count**, **Avg**, **Min**, **Max**, **Sum** und ihre Parameter werden ebenfalls als arithmetische Ausdrücke angesehen und gespeichert. Folgende Methoden werden von dem Interface *ArithmeticTerm* verlangt:

- **addChildAt**
  fügt einen bestimmten Kindknoten hinzu
- **deepCopy**
  liefert eine Kopie des Baumes ab diesem Knoten
- **getAlias**
  liefert den Alias für einen arithmetischen Term
- **getAttributes**
  gibt die in dem Teilbaum vorhandenen Attribute aus
- **getChildAt**
  liefert den Kindknoten an der übergebenen Position
- **getChildCount**
  gibt die Anzahl der Kinder zurück
- **removeChildAt**
  entfernt einen bestimmten Kindknoten
- **setAlias**
  setzt den Alias für einen arithmetischen Ausdruck

Da Selektions- und Verbundbedingungen logisch verknüpft werden können, wurde zur einheitlichen Speicherung der Parametrisierungen auch eine Baumstruktur verwendet.

Jeder Knoten in solch einem Baum muss das Interface *ConditionTreeNode* implementieren. Dazu werden folgende Methoden benötigt:

- **addChild**
  setzt den Kindknoten bei Verknüpfungen wie z. B. AND
- **containsArithmeticTerm**
  prüft, ob in der Bedingung ein arithmetischer Term vorhanden ist
- **countNodes**
  zählt die Knoten in einem Teilbaum
- **deepCopy**
  kopiert einen Teilbaum

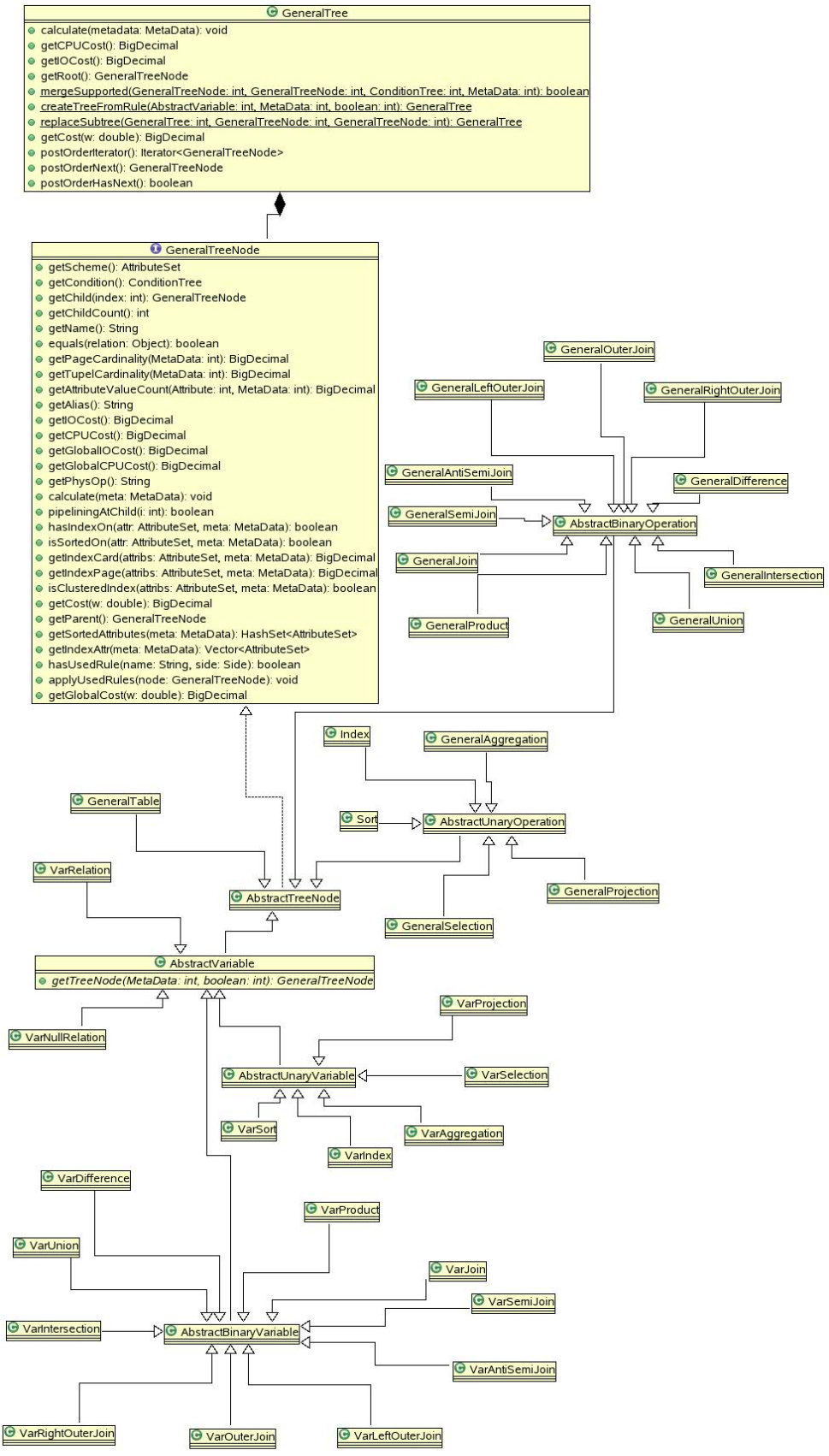


Abbildung 7.3: Package *dbs.unihannover.sopt.struc.general.tree*

- **getAttributes**
  liefert die Attribute, auf die sich eine Bedingungsparametrisierung bezieht
- **getChildAt**
  liefert den Kindknoten an der übergebenen Position
- **getChildCount**
  gibt die Anzahl der Kindknoten aus
- **getParent**
  liefert den übergeordneten Knoten
- **removeChildAt**
  entfernt einen Kindknoten
- **replaceChildAt**
  tauscht einen Kindknoten aus
- **setParent**
  setzt den übergeordneten Knoten

Die Klasse *AtomicCondition* wird zum Speichern von Selektions- und Verbundbedingungen verwendet. Objekte dieser Klasse können durch AND, OR und NOT verknüpft werden. Dasselbe gilt für die Klasse $VarCondition$, welche für variable Parametrisierungen in Regeltermen verwendet wird. Dabei sind AND und OR nicht auf zwei Argumente beschränkt, sondern können beliebig viele Kinder haben.

Die Klassen *ConditionProjection* und *ConditionAggregation* dienen der Speicherung von Projektions- bzw. Aggregationsparametrisierungen. Objekte dieser Klassen können nur als einzelne Elemente verwendet und nicht durch AND, OR oder NOT verknüpft werden.

Die Knoten werden in der Klasse $ConditionTree$ gespeichert. Diese Klasse stellt folgende Methoden zur Verfügung:

- **countNodes**
  zählt die Knoten in einem Teilbaum
- **deepCopy**
  kopiert einen Teilbaum
- **fetch**
  extrahiert aus einer verknüpften Selektionsbedingung die Prädikate, die sich auf dasselbe Attribut beziehen
- **fetchCondition**
  extrahiert aus einer verknüpften Selektionsbedingung die Prädikate, die sich auf dieselbe Relation beziehen
- **fetchJoinCondition**
  extrahiert aus einer verknüpften Selektionsbedingung die Prädikate, die als Joinbedingung genutzt werden können
- **hashSupported**
  prüft, ob eine Joinbedingung die nötige Struktur für ein Hash-Join hat
- **minus**
  entfernt Teile einer Parametrisierung
- **getAttributes**
  liefert die Attribute auf die sich eine Parametrisierung bezieht
- **getRoot**
  gibt die Wurzel des Baumes zurück
- **setRoot**
  setzt die Wurzel eines Baumes

Das Paket *.condition* enthält außerdem die Klassen für das Verfahren von Quine-McCluskey zur Minimierung von logisch verknüpften Selektionsbedingungen.

(a) **ValueTable**
Mit dieser Klasse wird eine Wertetabelle dargestellt. Sie bietet folgende Methoden:

- **fill**: Dieser Methode wird eine Selektionsbedingung in disjunktiver Form übergeben, aus der die Tabelle aufgebaut wird.
- **getMinTerms**: Die Minterme der Wertetabelle können mit dieser Funktion ausgegeben werden.

(b) **FirstQuineTable**

Diese Klasse repräsentiert die erste Quine'sche Tabelle. Bei der Erzeugung eines Objektes dieser Klasse werden die Minterme, welche man aus *ValueTable* erhält, übergeben. Mit der Funktion **getPrimterms** können anschließend die gefundenen Primterme ausgegeben werden.

(c) **SecondQuineTable**

Mit den Primtermen aus *FirstQuineTable* kann die zweite Quine'sche Tabelle aufgebaut werden. Dazu wird die Klasse *SecondQuineTable* verwendet. Die Funktion *getDMF* wird genutzt, um abschließend die disjunktive Minimalform ausgeben zu lassen.

(d) **ConditionMinimizer**

Die Klasse *ConditionMinimizer* bekommt bei der Erzeugung eines Objektes eine logisch verknüpfte Selektionsbedingung übergeben, welche von ihr in eine disjunktive Form gebracht wird. Anschließend werden Objekte der Klassen *ValueTable*, *FirstQuineTable* und *SecondQuineTable* erzeugt, um eine disjunktive Minimalform zu erhalten.

7.2 Sequenzablaufdiagramm der Einphasenoptimierung

Das folgende Sequenzdiagramm zeigt den Ablauf der Einphasenoptimierung (siehe Abbildung 7.4). Nachdem der Benutzer durch die deklarative Eingabesprache seine Anfrage gestellt hat, wird diese vom System geparst und in einen generalisierten Baum übersetzt. Danach wird der Termersetzungsoptimierer mit diesem Baum und einer Liste von anzuwendenden Regeln initialisiert. Dementsprechend wird auf jede aktive Variante dieser Regelstamm angewendet. Dabei wird jeweils jede Precondition auf den zu verändernden Teilbaum geprüft und zurückgegeben. Gespeichert wird eine Variante erst dann, wenn zusätzlich auch alle Postconditions erfüllt werden konnten.

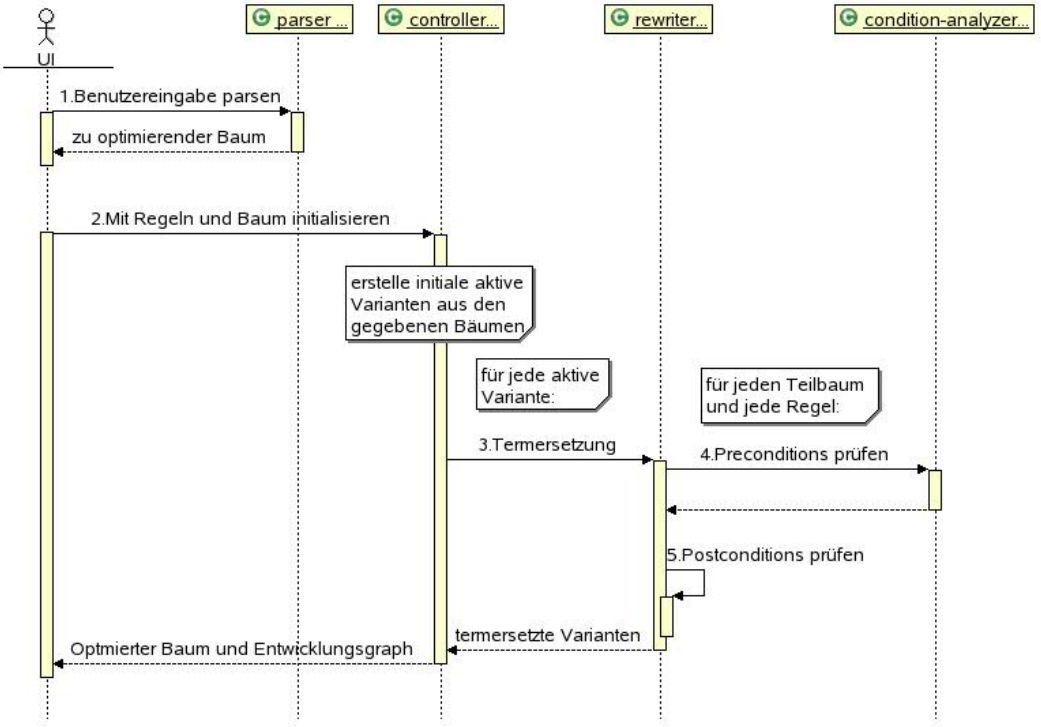


Abbildung 7.4: Sequenzablaufdiagramm der Einphasenoptimierung

7.3 Komponentendiagramm für die Einphasenoptimierung

Das nachfolgende Komponentendiagramm zeigt abstrakt den Aufbau des Termersetzungsoptimierers aus der Sicht eines Benutzers. Die Interaktion startet mit der Eingabe eines zu optimierenden Ausdrucks. Dazu wird vom System ein Satz von angeordneten Regeln erwartet, die durch den Regelparser auf Konformität untersucht werden. Danach übernimmt das Kontrollsystem die Abarbeitung, dessen Auftrag die Steuerung des Termersetzers ist. Dabei ist die Hauptaufgabe des Steuerungsmoduls, die Aktivierung bzw. Deaktivierung aller bekannten Varianten, indem neue mit alten Varianten in Beziehung gesetzt werden. Letztlich stoppt das Kontrollsystem die Termersetzung und liefert dem Benutzer eine nach Iterationen geordnete Liste von möglichen Ausführungsplänen zurück.

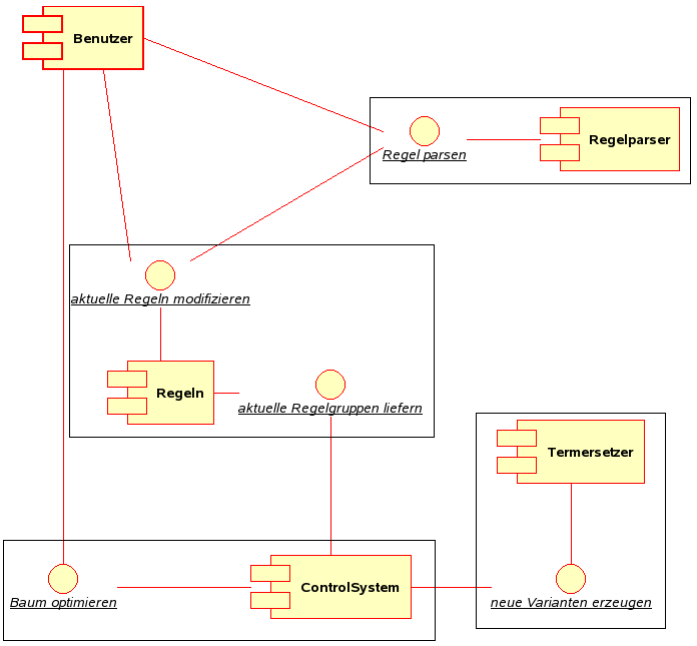


Abbildung 7.5: Komponentendiagramm der Einphasenoptimierung

7.4 Fazit der Implementierung

Durch den modularen Aufbau von `RELOpt` stellte die Erweiterung des relationalen Anfragesimulators hin zu einem objektrelationalen eine gut lösbare Aufgabe dar. Es konnte nachgewiesen werden, dass das Konzept der Termersetzung nicht nur in der Lage ist, beliebige Strategien abzubilden, sondern auch die Implementierung von beliebigen Operatoren, in unserem Fall von räumlichen, abzudecken. Nachdem zu Beginn dieser Arbeit die Rahmenbedingungen für relationale Tests gelegt wurden, konnten auch Aussagen gemacht werden, welche Punkte für neue, objektrelationale Operatoren fehlten. Diese Lücke, welche bei heutigen kommerziellen Anfragenoptimierern weiterhin existiert, wurde mit den in Kapitel 4 vorgestellten Abschätzungen gefüllt und ihre Implementierung in diesem Kapitel beschrieben.

Er ist ein Mathematiker und also hartnäckig.
Johann Wolfgang von Goethe 1749-1832



8 Ausblick

Im Zuge dieser Arbeit wurden bisherige Optimierungsstrategien für regelbasierte Optimierer um physische Implementierungen erweitert und mit Hilfe eines flexiblen Termersetzungssystems definiert. Die daraus möglichen Kontrollsysteme, bestehend aus lokalen Bedingungen, globalen Strategien und untersuchten Regelanordnungen, schaffen ein bislang unerreichtes Potential zur Erweiterung der Optimierbarkeit für beliebige objektrelationale Operatoren.

Durch die Einschränkung auf räumliche Operatoren als Beispiel für objektrelationale Operatoren konnten einige Aspekte der Optimierung in dieser Arbeit nicht betrachtet werden. Daher werden im Folgenden weitere Ideen und Forschungsansätze aufgeführt, mit denen das vorgestellte Optimierungskonzept erweitert werden kann:

- **Erweiterung der Menge der objektrelationalen Operatoren**

  Es liegt natürlich nahe, die bisherige Menge an objektrelationalen um neue Operatoren zu erweitern, z. B. auf Präferenzoperatoren (siehe Kießling et al. [40] (2002)).

- **Implementierung des vorgestellten Optimierungskonzeptes für reale Datenbanken**

  Aufbauend auf bisherigen Arbeiten sollte die vorgestellte Implementierung eines Anfrageoptimierungssimulators `RELOpt` hin zu einem echten Optimierer angegangen werden. Als Basis dafür könnten Open-Source Datenbanken verwendet werden, die eine leichte Ersetzung des Optimierers mit eigenen Modulen anbieten, wie z. B. die PostgreSQL[1] Datenbank.

- **Histogrammbasierte Abschätzungen für Attributwertverteilungen**

  In dieser Arbeit wurde die Annahme getroffen, dass alphanumerische Werte für Attribute gleichverteilt vorkommen. Als eine bekannte Verbesserung wären histogrammbasierte Abschätzungen von Attributwertverteilungen einzupflegen, mit deren Hilfe Berechnungen deutlich besser und exakter vorgenommen werden können.

- **Verwendung von oberen Schranken für Attributlängen**

  Anstatt von gleichlangen Attributlängen auszugehen, sollte man obere Schranken bzw. Mittelwerte für einzelne Attributlängen verwenden. Damit könnten Geometrien, die um ein vielfaches größere Seitenbelegungen verursachen, verglichen mit einfachen alphanumerischen Attributen, exakter abgeschätzt werden. Diese zusätzlichen Statistiken liegen immer vor, da sie bei der Erzeugung der Relationen definiert werden müssen.

- **Erweiterung auf n-näre Operatoren**

  Das Konzept der Termersetzung ist so flexibel, dass es nicht auf binäre Operatoren eingeschränkt ist, sondern hinsichtlich n-nären Operatoren (z. B. Multi-Joins) erweitert werden kann.

[1] `http://www.postgresql.org/`

- **Kostenmodell für Föderierte Datenbanken**

  Wurden bislang nur zentrale Datenbanken simuliert, so kann das Kostenmodell auch für Föderierte Datenbanken erweitert und getestet werden.

- **Erweiterung der Regelsprache**

  Als weiterer Punkt sollte die Erweiterung der Regelsprache angegangen werden. Beispielsweise könnte der Zugriff auf Teilbäume eines Ausführungsplanes ermöglicht werden, so dass Kostenbedingungen sich nicht mehr auf den gesamten Baum beziehen müssen.

- **Erzeugung von Materialisierten Sichten**

  Materialisierte Sichten können nach einmaliger Erzeugung für die mehrmalige Nutzung während eines oder mehrerer Ausführungspläne verwendet werden. Den strategisch größten Vorteil bieten sie bei der Betrachtung eines gesamten Workloads einer Datenbank, bei der schließlich festgestellt werden kann, ob einzelne Sichten sehr oft verwendet und aus diesem Grund materialisert werden sollten. Da wir uns in unserem Optimierungskonzept auf die Betrachtung einzelner Anfragen spezialisiert haben, kommen ihre Vorteile hier nicht so stark zum Vorschein. Es können jedoch Materialisierungsregeln – vergleichbar mit den Sortierungs- und Indexierungsregeln – definiert werden, die zukünftige föderierte Strategien berücksichtigen.

- **Anfrageoptimierung von XML-Daten und den dazugehörigen speziellen Operatoren**

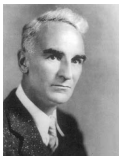
  Die wohl interessanteste Frage für ein objektrelationales Optimierungssystem ist wahrscheinlich, ob andere Datenstrukturen, wie z. B. XML-Daten, welche in einem relationalen Modell abgespeichert und durch eigene Operatoren (z. B. Staircase-Join (siehe Grust et al. [30] (2003)) angegangen werden, auch mit Ersetzungsregeln optimiert werden können. Effektiv gesehen könnte somit die Tragweite des vorgestellten Optimierungskonzeptes verdeutlicht werden.

- **Erweiterte Tests mit realen Datenbanken und Workload**

  Bislang wurden der TPC-H Version 2.3.0 Benchmark für relationale Daten und ATKIS für räumliche zum Testen verwendet. Wahrscheinlich würden weitere reale Datenbanken, wie z. B. die iMDb[2], neue Erkenntnisse bieten.

[2] iMDb – International Movie Database (`www.imdb.de`)

„Offensichtlich“ ist das gefährlichste Wort in der Mathematik.
Eric Temple Bell 1883-1960



A Leistungsstatistiken

Im Anhang befinden sich alle zusätzlichen Statistiken und Informationen, welche im Zuge dieser Arbeit gesammelt bzw. verwendet worden sind.

Die Tabelle A.1 gibt Aufschluss über Leistungsdaten von heutigen Prozessoren und Festplatten. Sie stammt aus der öffentlich zugänglichen Online-quelle `www.de.tomshardware.com` vom 22.08.2005. Die aufgeführten MIPS-Werte (Dhrystone-Werte) in den Abbildungen A.2 und A.3 wurden mit dem Freeware Programm Sisoft Sandra Benchmark 2004/05[1] vorgenommen. I/O-Durchsatzwerte stammen vom Open Source Projekt I/O-Meter[2].

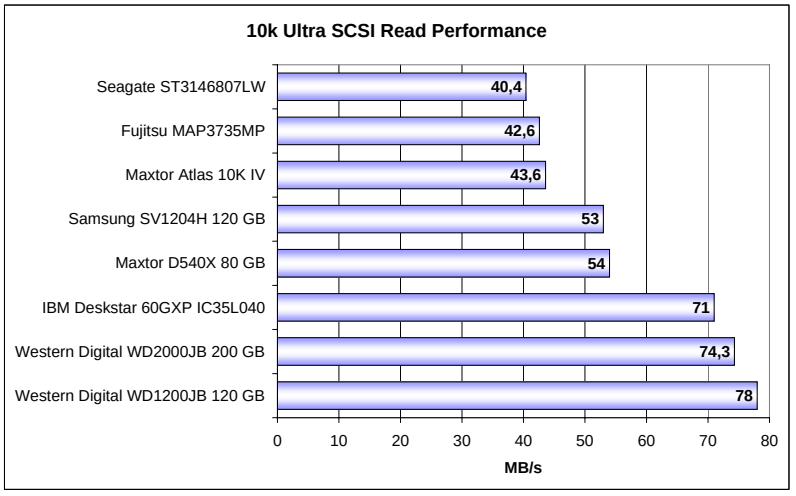


Abbildung A.1: IO-Meter Festplatten

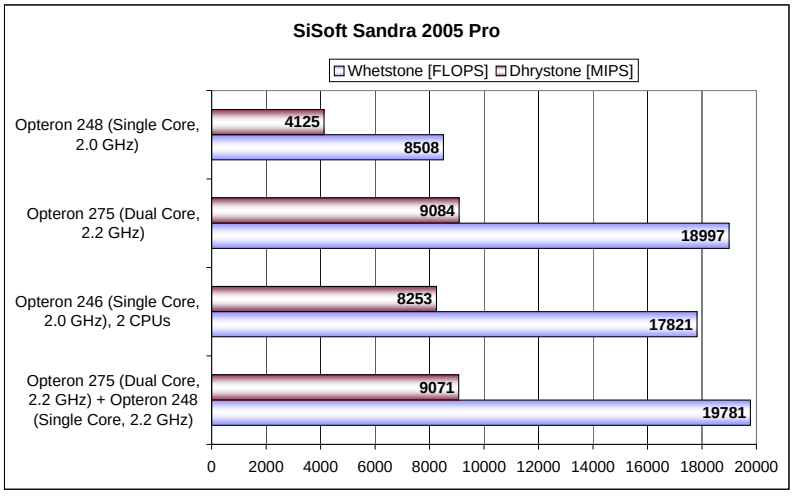


Abbildung A.2: CPU-MIPS Tabelle I

[1] `http://www.sisoftware.net/`
[2] `http://www.iometer.org/`

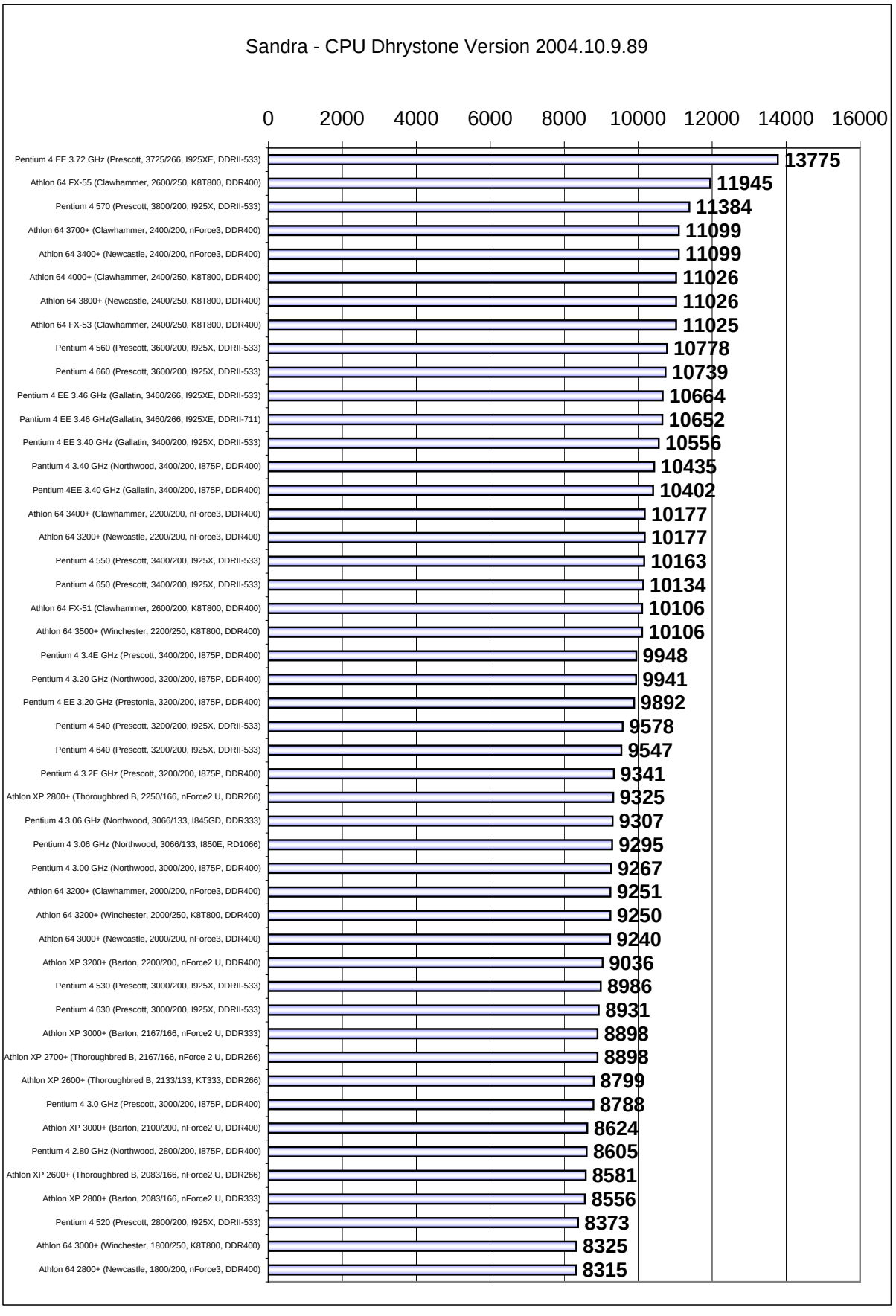


Abbildung A.3: CPU-MIPS Tabelle II

Beware of bugs in the above code;
I have only proved it correct, not tried it.

Donald Ervin Knuth *1938



B Konkretes Beispiel für den bedingten Join-Ordering-Algorithmus

In diesem Abschnitt wird ein konkretes Beispiel für den in Kapitel 3.5 vorgestellten Join-Ordering-Algorithmus vorgeführt. Gegeben seien folgende Relationen:

V(courseNo, reading)	Größe
$\|V\|$	300
pg(V)	30
n(courseNo, V)	300
n(reading, V)	100
$\|\text{Index}(\text{courseNo}, V)\|_l$	2

P(ID, name)	Größe
$\|P\|$	100
pg(P)	10
n(ID, P)	100
n(name, P)	75
$\|\text{Index}(\text{ID}, P)\|_l$	2

H(courseNo, studentID)	Größe
$\|H\|$	10000
pg(H)	10
n(studentID, H)	2000
n(courseNo, H)	300
$\|\text{Index}(\text{studentID}, H)\|_l$	3
$\|\text{Index}(\text{courseNo}, H)\|_l$	3

S(studentID, semester)	Größe
$\|S\|$	2000
pg(S)	200
n(studentID, S)	2000
n(semester, S)	20
$\|\text{Index}(\text{studentID}, S)\|_l$	3
$\|\text{Index}(\text{semester}, S)\|_l$	3

Die darauf agierende Anfrage sei:

$$\pi_{\text{P.ID}}\left(\sigma_{\text{S.semester}='5'}\left(\left(\text{H} \bowtie_{\text{H.studentID}=\text{S.studentID}} \text{S}\right) \bowtie_{\text{H.courseNo}=\text{V.courseNo}} \left(\text{P} \bowtie_{\text{P.ID}=\text{V.reading}} \text{V}\right)\right)\right)$$

Es wird von einer Gleichverteilung der Attributwerte ausgegangen. Zur Berechnung der Kosten werden die Formeln und Abschätzungen aus dem 2. Kapitel benutzt.

In der nachfolgenden `planTable` sind die erzeugten Einträge schwarz, die durch die L2-Regel verworfenen Teilmengen rot markiert. Alle Teilmengen, welche nach dem Entfernen der roten Teilmengen ($\tau_{\text{left}}, \tau_{\text{right}}$ existieren nicht) gar nicht mehr erzeugt werden können, sind blau gekennzeichnet.

Relationenmenge	τ_{left}	cost(τ)	Typ	sel(φ, $R_1 \times R_2$)	$g_{\text{sel}}^{\{\sigma^{\text{Rel}}, \{\bowtie, \rtimes, \overline{\rtimes}\}\}}$
{H}	∅	0	Basisrelation	1	0.0
{P}	∅	0	Basisrelation	1	0.0
{S}	∅	0	Basisrelation	1	0.0
{V}	∅	0	Basisrelation	1	0.0
{σ(S)}	∅	240	$\sigma_{\varphi}^{\text{Rel}}$	0.0	0.0
{H, P}	{H}	120210.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{H, P}	{P}	21012.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{H, S}	{S}	740.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, S}	{H}	740.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V}	{H}	468.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V}	{V}	468.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, σ(S)}	{σ(S)}	274.5	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	1.1999854801756901E-5
{H, σ(S)}	{H}	274.5	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	1.1999854801756901E-5
{P, S}	{P}	24012.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{P, S}	{S}	24240.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{P, V}	{V}	88.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0

Relationenmenge	τ_{left}	cost(τ)	Typ	sel(φ, $R_1 \times R_2$)	$g_{\text{sel}}^{\{\sigma^{\text{Rel}},\{\bowtie,\triangleright,\overline{\triangleright}\}\}}$
{P, V}	{P}	88.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, σ(S)}	{σ(S)}	1452.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	0.0011973657952504492
{P, σ(S)}	{P}	1452.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	0.0011973657952504492
{S, V}	{S}	72240.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{S, V}	{V}	72036.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{V, σ(S)}	{V}	3876.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	3.997068816201452E-4
{V, σ(S)}	{σ(S)}	3852.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	3.997068816201452E-4
{σ(H, S)}	{H}	750.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.1139866207618888
{σ(H, S)}	{S}	750.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.1139866207618888
{σ(P, S)}	{P}	24212.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	-0.11255238475354222
{σ(P, S)}	{S}	24440.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	-0.11255238475354222
{σ(S, V)}	{S}	72840.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	-0.11351675438012125
{σ(S, V)}	{V}	72636.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	-0.11351675438012125
{H, P, S}	{S}	241320.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, S}	{H, S}	120940.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{H, P, S}	{H}	65290.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, S}	{P, S}	65290.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, S}	{P}	121752.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{H, P, S}	{H, P}	125320.0	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V}	{H, P}	121648.0	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V}	{H, V}	2427.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V}	{H}	496.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V}	{P, V}	496.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V}	{P}	2427.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V}	{V}	241048.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, σ(S)}	{H, P}	123154.0	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	0.01	1.1999997348000587E-7
{H, P, σ(S)}	{H, σ(S)}	6284.5	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{H, P, σ(S)}	{H}	3502.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, σ(S)}	{P, σ(S)}	3502.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, σ(S)}	{P}	6386.5	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{H, P, σ(S)}	{σ(S)}	127104.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	1.1999997348000587E-7
{H, S, V}	{H, S}	2458.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, S, V}	{H, V}	2731.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, S, V}	{H}	552222.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, S, V}	{S, V}	195290.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, S, V}	{S}	2731.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, S, V}	{V}	2458.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(S)}	{σ(S)}	1325.55	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	1.1999734685866095E-5
{H, V, σ(S)}	{H, σ(S)}	168.34	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(S)}	{H}	27822.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(S)}	{V, σ(S)}	27822.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(S)}	{V}	168.34	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(S)}	{H, V}	1325.55	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	1.1999734685866095E-5
{H, σ(P, S)}	{H}	3540.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	5.0E-4	2.1999513810744782E-5
{H, σ(P, S)}	{σ(P, S)}	3540.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	5.0E-4	2.1999513810744782E-5
{H, σ(S, V)}	{H}	27822.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	7.4906367041198485E-6	2.1999513810744782E-5
{H, σ(S, V)}	{σ(S, V)}	10040.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	7.4906367041198485E-6	2.1999513810744782E-5
{P, S, V}	{P, S}	116040.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, S, V}	{P, V}	72094.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{P, S, V}	{P}	204016.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, S, V}	{S, V}	204016.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, S, V}	{S}	132328.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{P, S, V}	{V}	116040.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(S)}	{P, V}	3934.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	3.995737879595099E-4
{P, V, σ(S)}	{P, σ(S)}	5840.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(S)}	{P}	10216.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(S)}	{V, σ(S)}	10216.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(S)}	{V}	5840.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(S)}	{σ(S)}	6940.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	3.995737879595099E-4
{P, σ(H, S)}	{P}	6862.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	0.0012172975993294886
{P, σ(H, S)}	{σ(H, S)}	6760.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	0.0012172975993294886
{P, σ(S, V)}	{P}	10216.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	0.002192982456140351
{P, σ(S, V)}	{σ(S, V)}	10216.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	0.002192982456140351
{V, σ(H, S)}	{V}	168.34	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.003745318352059925	4.0636595585933076E-4
{V, σ(H, S)}	{σ(H, S)}	168.34	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.003745318352059925	4.0636595585933076E-4
{V, σ(P, S)}	{V}	5840.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	7.325519445924348E-4
{V, σ(P, S)}	{σ(P, S)}	5840.0	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	7.325519445924348E-4
{σ(H, P, S)}	{H, P}	126320.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.11399985480603209
{σ(H, P, S)}	{H, S}	121940.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	-0.11348918889254747

Relationenmenge	τ_{left}	cost(τ)	Typ	sel(φ, $R_1 \times R_2$)	$g_{\text{sel}}^{\{\sigma^{\text{Rel}}, \{\bowtie, \rtimes, \overline{\rtimes}\}\}}$
{σ(H, P, S)}	{H}	66290.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.2089733816882647
{σ(H, P, S)}	{P, S}	66290.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.2089733816882647
{σ(H, P, S)}	{P}	122752.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	-0.11348918889254747
{σ(H, P, S)}	{S}	242320.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.11399985480603209
{σ(H, S, V)}	{H, S}	2468.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0033333333333333335	-0.11446234607172717
{σ(H, S, V)}	{H, V}	2741.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.11398547978104205
{σ(H, S, V)}	{H}	552232.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.6666666666666669E-6	-0.2089733816882647
{σ(H, S, V)}	{S, V}	195300.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.6666666666666669E-6	-0.2089733816882647
{σ(H, S, V)}	{S}	2741.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.11398547978104205
{σ(H, S, V)}	{V}	2468.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0033333333333333335	-0.11446234607172717
{σ(P, S, V)}	{P, S}	116640.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.01	-0.20804475226425145
{σ(P, S, V)}	{P, V}	72694.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	-0.11347895578050082
{σ(P, S, V)}	{P}	204616.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.01	-0.20614035087719298
{σ(P, S, V)}	{S, V}	204616.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.01	-0.20614035087719298
{σ(P, S, V)}	{S}	132928.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.0	-0.11347895578050082
{σ(P, S, V)}	{V}	116640.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.01	-0.20804475226425145
{H, P, V, σ(S)}	{H, P, V}	4965.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	1.1999614812364523E-5
{H, P, V, σ(S)}	{H, P, σ(S)}	12365.36	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{H, P}	128863.02	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{H, V, σ(S)}	877.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{H, V}	6153.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{H, σ(S)}	1278.15775	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{H, P, V, σ(S)}	{H}	10971.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{P, V, σ(S)}	10971.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{P, V}	1278.15775	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{P, σ(S)}	4215.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{P}	877.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{V, σ(S)}	128763.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{V}	12315.36	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, V, σ(S)}	{σ(S)}	4965.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	1.1999614812364523E-5
{H, P, σ(S, V)}	{H, P}	128863.02	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	7.490636704119851E-8	2.1999992938002267E-7
{H, P, σ(S, V)}	{H, σ(S, V)}	878.02	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0	0.0
{H, P, σ(S, V)}	{H}	11073.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, σ(S, V)}	{P, σ(S, V)}	14541.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, σ(S, V)}	{P}	878.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, P, σ(S, V)}	{σ(S, V)}	132231.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	7.490636704119851E-8	2.1999992938002267E-7
{H, V, σ(P, S)}	{H, V}	8091.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	2.0E-5	2.199929360268242E-5
{H, V, σ(P, S)}	{H, σ(P, S)}	12349.02	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(P, S)}	{H}	11073.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(P, S)}	{V, σ(P, S)}	11073.02	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(P, S)}	{V}	12299.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{H, V, σ(P, S)}	{σ(P, S)}	4215.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	2.0E-5	2.199929360268242E-5
{H, σ(P, S, V)}	{H}	11073.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	7.4906367041198485E-6	3.1998972832972065E-5
{H, σ(P, S, V)}	{σ(P, S, V}	14541.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	7.4906367041198485E-6	3.1998972832972065E-5
{P, V, σ(H, S)}	{P, V}	1278.15775	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0050	4.062306457735364E-4
{P, V, σ(H, S)}	{P, σ(H, S)}	11941.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(H, S)}	{P}	878.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(H, S)}	{V, σ(H, S)	878.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(H, S)}	{V}	12315.36	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0	0.0
{P, V, σ(H, S)}	{σ(H, S)}	1278.15775	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.0050	4.062306457735364E-4
{P, σ(H, S, V)}	{P}	877.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	0.0022128745439684216
{P, σ(H, S, V)}	{σ(H, S, V)}	877.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	0.01	0.0022128745439684216
{V, σ(H, P, S)}	{V}	12299.02	$\bowtie_{\varphi}^{\text{Hash,Hash}}$	3.745318352059925E-5	7.358792758415162E-4
{V, σ(H, P, S)}	{σ(H, P, S)}	12349.02	$\rtimes_{\varphi}^{\text{Hash,Hash}}$	3.745318352059925E-5	7.358792758415162E-4
{σ(H, P, S, V)}	{H, P, S}	262381.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	3.3333333333333335E-5	-0.2089897143389906
{σ(H, P, S, V)}	{H, P, V}	44464.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.11398434110265061
{σ(H, P, S, V)}	{H, P}	294345.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.666666666666667E-8	-0.20899971291109215
{σ(H, P, S, V)}	{H, S, V}	44160	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.01	-0.2071687810440675
{σ(H, P, S, V)}	{H, S}	43229	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0050	-0.1144242327376374
{σ(H, P, S, V)}	{H, V}	207124.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-6	-0.20897128993188033
{σ(H, P, S, V)}	{H}	652955.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.6666666666666669E-6	-0.3039582429404016
{σ(H, P, S, V)}	{P, S, V}	296023.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.6666666666666669E-6	-0.3039582429404016
{σ(H, P, S, V)}	{P, S}	2507948.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-6	-0.20897128993188033
{σ(H, P, S, V)}	{P, V}	43229.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.0050	-0.1144242327376374
{σ(H, P, S, V)}	{P}	44160.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	0.01	-0.2071687810440675
{σ(H, P, S, V)}	{S, V}	3.6593813E7	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	1.666666666666667E-8	-0.20899971291109215
{σ(H, P, S, V)}	{S}	44464.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	5.0E-4	-0.11398434110265061
{σ(H, P, S, V)}	{V}	381781.0	$\bowtie_{\varphi}^{\text{Rel,Rel}}$	3.3333333333333335E-5	-0.2089897143389906

In unserem Beispiel werden mit 4 Basisrelationen und einer Basisselektion 153 Teilmengen durch den Join-Ordering Algorithmus erzeugt (siehe Tabelle B.2).

Natürlich liefern beide Join-Ordering-Algorithmen (mit und ohne Verwendung der L2-Regel) denselben Ausführungsplan. Aber in unserem Fall werden 46,41% weniger Teilmengen betrachtet. Zwar muss zusätzlicher Aufwand für die Anordnung der Selektionen betrieben werden, dennoch rentiert sich diese Investition schon ab einer geringen Anzahl von zu joinenden Relationen.

Anzahl der Elemente in den Teilmengen	1	2	3	4	Gesamt
Algorithmus ohne L2-Regel					
Teilmengen	5	24	72	52	153
Algorithmus mit L2-Regel					
mehrmals betrachtete Teilmengen	5	12	24	15	56
einmal betrachtete Teilmengen	0	12	12	1	25
nicht erzeugte Teilmengen	0	0	36	36	72
Einsparung					46,41%

Tabelle B.2: Vergleich der Join-Ordering-Algorithmen

In der Informatik geht es genau so wenig um Computer,
wie in der Astronomie um Teleskope.

Edsger W. Dijkstra 1930-2002



C Gewichtungsfaktoren für topologische Funktionen

Nachfolgend die Liste aller in Oracle 10g unterstützten räumlichen Funktionen mit den dazugehörigen Gewichten bei Verwendung des ersten Geometrietypes (Spalten in nachfolgenden Tabellen) und des zweiten (Zeilen).

Die Standardwerte wurden durch Verwendung von ATKIS-Daten (siehe Abschnitt 6.3.3) und eigenen Abwandlungen errechnet. Da sie leicht schwanken und sich proportional zur Komplexität der Geometrien verhalten, werden die nachfolgenden Werte als grobe Abschätzungen gesehen.

Eigene Tests haben aber bestätigt, dass diese Werte hinsichtlich einer Anordnung von objektrelationalen mit gleichzeitiger Verwendung von relationalen Operatoren für eine sinnvolle Optimierung vollkommen ausreichen und sich in sie leicht einfügen lassen. Dennoch sollten die nachfolgenden Werte ebenfalls an die jeweilige Datenbank durch Tests angepasst werden.

OVERLAPBDY DISJOINT	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,2	1,1	1,1	1,2	1,6
line	1,1	1,4	1,5	1,4	1,1	1,5	1,8
polygon	1,2	1,7	1,8	2,1	1,6	2,4	3,2
collection	1,2	1,5	1,6	1,5	1,2	1,6	2,1
multipoint	1,1	1,3	1,6	1,4	1,4	1,4	1,9
multiline	1,3	1,6	1,7	1,7	1,3	1,9	2,1
multipolygon	1,2	1,5	1,4	1,6	1,5	1,7	1,8

Tabelle C.1: Standardwerte für die topologische Funktion **OVERLAPBDYDISJOINT**

OVERLAPBDY INTERSECT	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,2	1,1	1,1	1,2	1,5
line	1,1	1,2	1,5	1,2	1,1	1,5	1,9
polygon	1,2	1,8	1,9	2,0	1,3	2,4	3,0
collection	1,1	1,4	1,5	1,5	1,2	1,6	2,0
multipoint	1,0	1,1	1,2	1,1	1,1	1,2	1,6
multiline	1,1	1,5	1,6	1,6	1,2	1,7	2,0
multipolygon	1,1	1,4	1,5	1,4	1,1	1,5	1,8

Tabelle C.2: Standardwerte für die topologische Funktion **OVERLAPBDYINTERSECT**

EQUAL	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,3	1,2	1,1	1,3	1,8
line	1,1	1,2	1,3	1,2	1,1	1,3	1,7
polygon	1,2	1,7	1,9	1,9	1,4	2,3	3,2
collection	1,1	1,2	1,4	1,3	1,2	1,5	1,9
multipoint	1,2	1,3	1,4	1,3	1,2	1,4	1,8
multiline	1,3	1,4	1,6	1,6	1,4	1,7	2,0
multipolygon	1,2	1,3	1,5	1,4	1,3	1,5	1,9

Tabelle C.3: Standardwerte für die topologische Funktion **EQUAL**

INSIDE	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,3	1,2	1,1	1,3	1,6
line	1,1	1,2	1,3	1,2	1,1	1,3	1,7
polygon	1,5	1,6	1,7	1,7	1,5	1,7	2,1
collection	1,1	1,1	1,4	1,3	1,2	1,5	1,7
multipoint	1,1	1,2	1,4	1,2	1,2	1,3	1,7
multiline	1,2	1,4	1,5	1,4	1,3	1,5	1,9
multipolygon	1,4	1,5	1,6	1,4	1,4	1,6	1,7

Tabelle C.4: Standardwerte für die topologische Funktion **INSIDE**

TOUCH	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,0	0,9	0,8	0,8	0,9	1,2
line	1,0	1,1	1,0	1,1	0,8	1,2	1,2
polygon	1,0	1,2	1,6	1,4	1,2	1,7	2,5
collection	1,0	1,1	1,2	1,1	1,1	1,2	1,5
multipoint	0,8	1,1	1,2	0,9	0,8	1,2	1,5
multiline	0,9	1,2	1,2	1,3	1,0	1,4	1,5
multipolygon	1,0	0,9	1,1	1,0	1,1	1,0	1,3

Tabelle C.5: Standardwerte für die topologische Funktion **TOUCH**

DISJOINT	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,1	1,1	1,0	1,1	1,3
line	1,0	1,1	1,2	1,1	1,1	1,2	1,4
polygon	1,1	1,3	1,4	1,5	1,2	1,7	2,2
collection	1,1	1,1	1,2	1,2	1,1	1,2	1,4
multipoint	1,0	1,1	1,1	1,1	1,1	1,2	1,4
multiline	1,1	1,2	1,3	1,3	1,2	1,4	1,5
multipolygon	1,1	1,1	1,2	1,2	1,1	1,2	1,4

Tabelle C.6: Standardwerte für die topologische Funktion **DISJOINT**

CONTAINS	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,2	1,1	1,1	1,2	1,5
line	1,1	1,2	1,3	1,2	1,1	1,3	1,6
polygon	0,9	1,0	1,2	1,5	1,0	1,1	2,5
collection	1,0	1,1	1,4	1,1	1,0	1,1	1,7
multipoint	0,9	1,0	1,3	1,0	0,9	1,1	1,6
multiline	1,0	1,1	1,2	1,2	1,0	1,2	1,6
multipolygon	0,9	1,0	1,2	1,1	1,0	1,1	1,5

Tabelle C.7: Standardwerte für die topologische Funktion **CONTAINS**

ON	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,3	1,2	1,1	1,3	1,6
line	1,1	1,2	1,3	1,2	0,8	1,3	1,7
polygon	1,2	1,7	1,9	1,9	1,3	2,3	3,2
collection	1,2	1,3	1,5	1,4	1,2	1,5	1,8
multipoint	1,1	1,2	1,4	1,3	1,2	1,3	1,7
multiline	1,3	1,4	1,5	1,5	1,3	1,7	2,0
multipolygon	1,2	1,3	1,4	1,3	1,2	1,4	1,8

Tabelle C.8: Standardwerte für die topologische Funktion **ON**

COVERS	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	0,8	1,1	1,1	0,9	1,2	1,8
line	0,9	1,1	1,4	1,1	1,0	1,4	1,7
polygon	1,0	1,1	1,5	1,1	1,1	1,2	1,8
collection	1,0	1,1	1,4	0,9	1,1	1,3	1,9
multipoint	0,9	1,0	1,4	1,1	1,0	1,2	1,8
multiline	1,1	1,2	1,3	1,2	1,1	1,3	1,6
multipolygon	1,0	1,1	1,5	1,1	1,1	1,2	1,8

Tabelle C.9: Standardwerte für die topologische Funktion **COVERS**

COVEREDBY	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,9	2,1	1,5	1,3	1,7	2,4
line	1,3	1,5	1,6	1,6	1,4	1,7	2,2
polygon	1,5	2,3	2,7	2,6	2,0	2,9	4,0
collection	1,5	1,6	1,8	1,7	1,6	1,8	2,3
multipoint	1,4	1,6	1,8	1,6	1,5	1,8	2,3
multiline	1,6	1,8	2,0	1,8	1,7	2,0	2,5
multipolygon	1,5	1,9	2,1	1,7	1,9	2,1	2,7

Tabelle C.10: Standardwerte für die topologische Funktion **COVEREDBY**

DETERMINE	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	1,1	1,1	1,1	1,0	1,1	1,3
line	1,0	1,1	1,2	1,1	1,1	1,2	1,4
polygon	1,1	1,4	1,5	1,6	1,2	1,8	2,3
collection	1,1	1,1	1,2	1,2	1,1	1,3	1,5
multipoint	1,0	1,1	1,2	1,1	1,1	1,2	1,4
multiline	1,2	1,2	1,3	1,3	1,2	1,4	1,6
multipolygon	1,1	1,2	1,2	1,2	1,1	1,2	1,4

Tabelle C.11: Standardwerte für die topologische Funktion **DETERMINE**

ANYINTERACT	point	line	polygon	collection	multipoint	multiline	multipolygon
point	1,0	0,9	1,2	0,9	1,1	1,0	1,4
line	0,9	1,1	1,2	1,2	0,9	1,2	1,5
polygon	1,2	1,3	1,4	1,7	1,2	1,5	2,0
collection	1,0	1,2	1,3	1,3	1,2	1,3	1,6
multipoint	1,0	1,1	1,3	1,1	1,1	1,2	1,5
multiline	1,0	1,2	1,3	1,3	1,0	1,4	1,6
multipolygon	1,1	1,2	1,2	1,2	1,1	1,3	1,5

Tabelle C.12: Standardwerte für die topologische Funktion **ANYINTERACT**

Früher hatten wir die spanische Inquisition.
Jetzt haben wir die Kompatibilität.

Herbert Klaeren *1950



D Tabellarische Übersichten der Optimierungsresultate

D.1 Relationale Tests mit der TPC-H Version 2.3.0 Datenbank

Für alle von uns durchgeführten Experimente wurde eine zentrale Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - 64bit Production in einer 10-Gigabytes Variante aufgesetzt und verwendet. Die Oracle 10g Datenbank lief auf einem AMD Opteron 246 mit 2,0 GHz (8253 MIPS (siehe Tabelle A.3)) mit 8 GB an DDR-400 RAM. Als I/O-Speichermedium kamen 8 Seagate ST3146807LC Cheetah 10K.6 Festplatten im Raid 0+1 mit jeweils 40,4 IOPS zum Einsatz (siehe Tabelle A.1).[1]

Datenbank:	Oracle 10g R10.2.0.1.0 - 64bit
CPU:	AMD Opteron 246, 2,0 GHz (8253 MIPS)
RAM:	8 GB DDR-400
HD:	8 x 146 GB Seagate ST3146807LC Cheetah 10K.6, Ultra320, SCSI, 10.000 U/min, 8MB Cache, Raid 0+1 je (40,4 IOPS)
Betriebssystem:	Gentoo Linux (Kernel 2.6.13)

Tabelle D.1: Spezifikation des Datenbankrechners

Aus den oben genannten Spezifikationen ergab sich folgender CPU-I/O Gewichtungsfaktor[2]:

$$w_1 = \frac{4 \cdot 40,4 \frac{\text{MInstr.}}{\text{s}}}{8253 \frac{\text{MInstr.}}{\text{s}}} \approx 0,02$$

Nachfolgend ein Beispiel für eine relationale TPC-H Benchmark Anfrage, deren Ausführungsplan durch `RELOpt` verbessert werden konnte. Die Anfrage ausgedrückt in SQL ist in Abbildung D.1 gegeben.

[1] Da die Festplatten im Raid 0+1 gefahren werden und somit die Hälfte der additiven Bandbreite genutzt werden konnte, wird in der nachfolgenden Rechnung mit 4 Festplatten gerechnet.
[2] Hierbei wurde von einem System mit CPU-Engpass ausgegangen.

```
select  n_name,sum(l_extendedprice * (1 - l_discount)) as revenue
from    customer,
        orders,
        lineitem,
        supplier,
        nation,
        region
where   c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and l_suppkey = s_suppkey
        and c_nationkey = s_nationkey
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'ASIA'
        and o_orderdate >= date '1994-01-01'
        and o_orderdate < date '1994-01-01' + interval '1' year
group by n_name
order by revenue desc;
```

Abbildung D.1: TPC-H Version 2.3.0 Query 5

Nach 2 Minuten und 53,28 Sekunden lieferte die Datenbank das Ergebnis:

```
N_NAME                         REVENUE
------------------------- ----------
INDIA                          536862588
CHINA                          535350830
VIETNAM                        532269389
JAPAN                          526766837
INDONESIA                      523176852

Elapsed: 00:02:53.28
```

Die gleiche Anfrage in der relationalen Sprache für `RELOpt`:

```
(SORT (R.revenue)
 (AGGREGATION (nation.n_name#nation.n_name,
               sum(lineitem.l_extendedprice*(1-lineitem.l_discount)) as R.revenue)
  (SELECTION (AND(region.r_name='ASIA',orders.o_orderdate>='1994-01-01',orders.o_orderdate<'1994-08-01'))
   (JOIN (nation.n_regionkey=region.r_regionkey) region,
    (JOIN (supplier.s_nationkey=nation.n_nationkey) nation,
     (JOIN (AND(lineitem.l_orderkey=orders.o_orderkey,customer.c_nationkey=supplier.s_nationkey))
      (JOIN (customer.c_custkey=orders.o_custkey) customer,orders)
       (JOIN (lineitem.l_suppkey=supplier.s_suppkey) lineitem,supplier)))))))
```

Als Optimierungsergebnis wird folgender Ausführungsplan von `RELOpt` zurückgeliefert:

```
(SORT (R.REVENUE)
  (AGGREGATION rel (NATION.N_NAME#NATION.N_NAME,
                    sum((LINEITEM.L_EXTENDEDPRICE*(1-LINEITEM.L_DISCOUNT))) as R.REVENUE)
    (JOIN rel-rel SUPPLIER.S_NATIONKEY=NATION.N_NATIONKEY
      (JOIN rel-rel NATION.N_REGIONKEY=REGION.R_REGIONKEY
        (SELECTION rel REGION.R_NAME='ASIA'
          REGION)
        NATION)
      (JOIN ind-rel (AND(CUSTOMER.C_NATIONKEY=SUPPLIER.S_NATIONKEY,LINEITEM.L_ORDERKEY=ORDERS.O_ORDERKEY))
        (INDEX (CUSTOMER.C_NATIONKEY,ORDERS.O_ORDERKEY)
          (SELECTION rel (AND(ORDERS.O_ORDERDATE>='1994-01-01',ORDERS.O_ORDERDATE<'1994-08-01'))
            (JOIN ind-rel CUSTOMER.C_CUSTKEY=ORDERS.O_CUSTKEY
              CUSTOMER
              ORDERS)))
        (JOIN rel-ind LINEITEM.L_SUPPKEY=SUPPLIER.S_SUPPKEY
          LINEITEM
          SUPPLIER)))))
```

Hiernach wurde die von `RELOpt` vorgeschlagene äquivalente Anfrage an die Datenbank gestellt:

```
select  n_name,sum(l_extendedprice * (1 - l_discount)) as revenue
from   (select * from
                (select *
                 from    customer,
                         orders
                 where   c_custkey = o_custkey
                         and o_orderdate >= date '1994-01-01'
                         and o_orderdate <
                          date '1994-01-01' + interval '1' year),
                (select *
                 from    lineitem,
                         supplier
                 where   l_suppkey = s_suppkey)
        where  l_orderkey = o_orderkey
               and c_nationkey = s_nationkey),
       (select *
        from    nation,
                region
        where   n_regionkey = r_regionkey
                and r_name = 'ASIA')
where s_nationkey = n_nationkey
group by
        n_name
order by
        revenue desc;
```

Nach 2 Minuten und 7,72 Sekunden lieferte die Datenbank das Ergebnis:

```
N_NAME                         REVENUE
------------------------- ----------
INDIA                          536862588
CHINA                          535350830
VIETNAM                        532269389
JAPAN                          526766837
INDONESIA                      523176852

Elapsed: 00:02:07.72
```

Durch den Rewrite konnte der Optimierer die Anfrage rund 30% schneller beantworten. Die per `RELOpt` optimierte Anfrage wird nach einmaliger Ausführung von Oracle 10g als der bessere Ausführungsplan erkannt und gecached, so dass zukünftige äquivalente Anfragen denselben Plan besitzen. Erst durch eine manuelle Löschung des Caches kann dieser Zustand wieder zurückgesetzt werden.

In der nachfolgenden Tabelle D.2 sind jeweils die verschiedenen Anfragen mit ihren Abkürzungen, die unterschiedlichen anfallenden Kosten, die Anzahl berechneter Varianten und der Iterationen, die Werte für die beiden Grenzen, sowie die Dauer für die Optimierung durch `RELOpt` zu finden. Durch die Grenzangaben ergeben sich die entsprechenden verschiedenen Optimierungsstrategien.

Nach der Optimierung mit `RELOpt` wurde die optimierte Variante mit der Ausgangsvariante ins Verhältnis `y` gesetzt. Dabei konnte festgestellt werden, bei welchen Anfragen die verwendeten Regeln eine Optimierung durch Neuanordnung der Operatoren oder Neuerzeugung von Indexen bzw. Sortierungen vornehmen konnten (siehe Tabelle D.3 auf Seite 136). Die farbliche Markierung der Anfragen entspricht dabei der Gruppierung nach Abschnitt 6.3.1.

Tabelle D.2: Relationale Anfragen und Testergebnisse

Anfrage	Gesamtkosten	$\mathsf{cost}_{\mathsf{I/O}}$	$\mathsf{cost}_{\mathsf{CPU}}$	Anz. Varianten	Iterationen	Grenzen dyn./stat.	Optimierungsdauer
Q1	7403558104769	3412695680345	199543121221232	1	-	0.01/1	-
	7403558104769	3412695680345	199543121221232	1	4	1/5	62 ms
	7403558104769	3412695680345	199543121221232	1	4	1/10	62 ms
	7403558104769	3412695680345	199543121221232	5	6	1.1/5	146 ms
	7403558104769	3412695680345	199543121221232	5	6	1.1/10	146 ms
	7403558104769	3412695680345	199543121221232	5	6	$\log_{10}/5$	61 ms
	7403558104769	3412695680345	199543121221232	5	6	$\log_{10}/10$	61 ms
	7403558104769	3412695680345	199543121221232	5	6	$\log_e/5$	54 ms
	7403558104769	3412695680345	199543121221232	5	6	$\log_e/10$	54 ms
Q2	2671620025457	2347935207012	16184240922266	1	-	0.01/1	-
	1416162271013	1362604768159	2677875142715	68	13	1/5	5964 ms
	1416162271013	1362604768159	2677875142715	68	13	1/10	5964 ms
	1416162271013	1362604768159	2677875142715	168	15X	1.1/1	457447 ms
	1416162271013	1362604768159	2677875142715	152	12X	$\log_{10}/1$	242046 ms
	1416162271013	1362604768159	2677875142715	152	12X	$\log_e/1$	238363 ms
Q3	64342085874	47386779751	847765306210	1	-	0.01/1	-
	64338817213	47383511090	847765306210	4	6	1/5	769 ms
	64338817213	47383511090	847765306210	4	6	1/10	769 ms
	64334333418	47382365724	847598384785	461	18	1.1/5	236460 ms
	64334333418	47382365724	847598384785	535	24	$\log_{10}/3$	399948 ms
	64334333418	47382365724	847598384785	416	24	$\log_e/2$	276577 ms
Q4	49690245189	21904841362	1389270191384	1	-	0.01/1	-
	49662743845	21877610911	1389256646737	2	5	1/5	405 ms
	49662743845	21877610911	1389256646737	2	5	1/10	405 ms
	49662743845	21877610911	1389256646737	79	10	1.1/5	2538 ms
	49662743845	21877610911	1389256646737	135	10	1.1/10	3361 ms
	49662743845	21877610911	1389256646737	79	10	$\log_{10}/5$	2580 ms
	49662743845	21877610911	1389256646737	135	10	$\log_{10}/10$	3432 ms
	49662743845	21877610911	1389256646737	115	12	$\log_e/5$	3770 ms
	49662743845	21877610911	1389256646737	172	11	$\log_e/10$	4453 ms

Anfrage	Gesamtkosten	$\text{cost}_{\text{I/O}}$	cost_{CPU}	Anz. Varianten	Iterationen	Grenzen dyn./stat.	Optimierungsdauer
Q5	5316670316290	34969490613497	909860627470633	1	-	0.01/1	-
	1495486484	1454709895	2038829542	27	9	1/5	5702 ms
	1495486484	1454709895	2038829542	27	9	1/10	4449 ms
	1477657793	1443941335	1685823012	778	36	1.1/3	366012 ms
	1477657793	1443941335	1685823012	644	50	$\log_{10}/1$	1111219 ms
	1477657793	1443941335	1685823012	644	50	$\log_e/1$	1099072 ms
Q6	2232753	1029192	60178047	1	-	0.01/1	-
	2232753	1029192	60178047	1	4	1/5	140 ms
	2232753	1029192	60178047	1	4	1/10	140 ms
	2232753	1029192	60178047	9	9	1.1/5	12968 ms
	2232753	1029192	60178047	9	9	1.1/10	12968 ms
	2232753	1029192	60178047	64	11	$\log_{10}/5$	19634 ms
	2232753	1029192	60178047	85	11	$\log_{10}/10$	26026 ms
	2232753	1029192	60178047	70	11	$\log_e/5$	19653 ms
	2232753	1029192	60178047	91	11	$\log_e/10$	26235 ms
Q7	1296694609	764588650	26605298012	1	-	0.01/1	-
	1245285508	723667139	26080918638	40	13	1/5	34930 ms
	1245285508	723667139	26080918638	40	13	1/10	34584 ms
	1245253600	723667139	26079323218	608	23	1.1/2	1609335 ms
	1245253600	723667139	26079323218	671	25	$\log_{10}/2$	1637471 ms
	1245253600	723667139	26079323218	671	25	$\log_e/2$	1664274 ms
Q8	431712865	413003445	935470986	1	-	0.01/1	-
	103089560	99285837	190186175	81	17	1/5	5520 ms
	103089560	99285837	190186175	81	17	1/10	5227 ms
	103089560	99285837	190186175	1218	43	1.1/2	688044 ms
	103089560	99285837	190186175	1102	58	$\log_{10}/1$	564538 ms
	103089560	99285837	190186175	1108	58	$\log_e/1$	560938 ms
Q9	92881925377262111	313047853166769 3	307885700302925957	1	-	0.01/1	-
	92704037790288889	312227876824164 0	307406250539362514	24	9	1/5	3780 ms
	92704037790288889	312227876824164 0	307406250539362514	24	9	1/10	3780 ms

Anfrage	Gesamtkosten	$\mathsf{cost}_{\mathsf{I/O}}$	$\mathsf{cost}_{\mathsf{CPU}}$	Anz. Varianten	Iterationen	Grenzen dyn./stat.	Optimierungsdauer
Q9	9288192537725376	3130478531666858	307885700302925957	178	38X	1.1/1	996710 ms
	9288192537725376	3130478531666858	307885700302925957	178	38X	$\log_{10}/1$	1000830 ms
	9288192537725376	3130478531666858	307885700302925957	178	38X	$\log_e/1$	1004110 ms
Q10	555336276277	508248564360	2354385595895	1	-	0.01/1	-
	555318879198	508237800907	2354053914603	19	10	1/5	956 ms
	555318879198	508237800907	2354053914603	19	10	1/10	956 ms
	555318852267	508237800907	2354052568041	593	17	1.1/5	265995 ms
	555318852267	508237800907	2354052568041	636	24	$\log_{10}/4$	195986 ms
	555318852267	508237800907	2354052568041	671	26	$\log_e/3$	341494 ms
Q11	10860498567	9881873734	48931241543	1	-	0.01/1	-
	10858650350	9880341142	48915460260	24	11	1/5	1694 ms
	10858650350	9880341142	48915460260	24	11	1/10	1694 ms
	10858650350	9880341142	48915460260	1814	56	1.1/5	649942 ms
	10858650350	9880341142	48915460260	1814	56	1.1/10	649942 ms
	10853789964	9876101496	48884423253	1187	56	$\log_{10}/2$	518833 ms
	10853789964	9876101496	48884423253	1188	56	$\log_e/2$	516116 ms
Q12	938208886	581310558	17844916413	1	-	0.01/1	-
	938208886	581310558	17844916413	2	5	1/5	1394 ms
	938208886	581310558	17844916413	2	5	1/10	1394 ms
	934487786	580133681	17717705264	251	33	1.1/1	1870601 ms
	934487786	580133681	17717705264	300	33	$\log_{10}/1$	1889542 ms
	934487786	580133681	17717705264	312	33	$\log_e/1$	1910144 ms
Q13	1614497	624430	49503396	1	-	0.01/1	-
	1614497	624430	49503396	1	4	1/5	7 ms
	1614497	624430	49503396	1	4	1/10	7 ms
	1614497	624430	49503396	1	4	1.1/5	163 ms
	1614497	624430	49503396	1	4	1.1/10	163 ms
	1614497	624430	49503396	1	4	$\log_{10}/5$	18 ms
	1614497	624430	49503396	1	4	$\log_{10}/10$	18 ms
	1614497	624430	49503396	5	6	$\log_e/5$	19 ms

Anfrage	Gesamtkosten	$cost_{I/O}$	$cost_{CPU}$	Anz. Varianten	Iterationen	Grenzen dyn./stat.	Optimierungsdauer
Q13	1614497	624430	49503396	5	6	$\log_e/10$	19 ms
Q14	6211588	2344463	193356213	1	-	0.01/1	-
	2987704	1267943	85988042	3	5	1/5	246 ms
	2987704	1267943	85988042	3	5	1/10	246 ms
	2987704	1267943	85988042	4	6	1.1/5	222 ms
	2987704	1267943	85988042	4	6	1.1/10	222 ms
	2987704	1267943	85988042	14	8	$\log_{10}/5$	1642 ms
	2987704	1267943	85988042	14	8	$\log_{10}/10$	1642 ms
	2987704	1267943	85988042	14	8	$\log_e/5$	1009 ms
	2987704	1267943	85988042	14	8	$\log_e/10$	1009 ms
Q15	82624008508	37919374039 8	2217151340552 0	1	-	0.01/1	-
	82624008508	37919374039 8	2217151340552 0	1	4	1/5	35 ms
	82624008508	37919374039 8	2217151340552 0	1	4	1/10	35 ms
	82624008508	37919374039 8	2217151340552 0	5	6	1.1/5	168 ms
	82624008508	37919374039 8	2217151340552 0	5	6	1.1/10	168 ms
	82624008508	37919374039 8	2217151340552 0	5	6	$\log_{10}/5$	59 ms
	82624008508	37919374039 8	2217151340552 0	5	6	$\log_{10}/10$	59 ms
	82624008508	37919374039 8	2217151340552 0	5	6	$\log_e/5$	56 ms
	82624008508	37919374039 8	2217151340552 0	5	6	$\log_e/10$	56 ms
Q16	2656833406	2556187964	5032272122	1	-	0.01/1	-
	168018	86649	4068494	7	8	1/5	664 ms
	168018	86649	4068494	7	8	1/10	664 ms
	166020	86650	3968497	399	19	1.1/4	374553 ms
	166020	86650	3968497	492	22	$\log_{10}/4$	352440 ms
	168018	86649	4068494	211	24	$\log_e/1$	159612 ms
Q17	2048103536530	2048099658831	193884933	1	-	0.01/1	-
	1381719	40573	67057345	8	8	1/5	356 ms
	1381719	40573	67057345	8	8	1/10	356 ms
	1381719	40573	67057345	28	12	1.1/5	1066 ms
	1381719	40573	67057345	31	12	1.1/10	2751 ms

Anfrage	Gesamtkosten	$\mathsf{cost}_{\mathsf{I/O}}$	$\mathsf{cost}_{\mathsf{CPU}}$	Anz. Varianten	Iterationen	Grenzen dyn./stat.	Optimierungsdauer
Q17	1381719	40573	67057345	106	13	$\log_{10}/5$	28470 ms
	1381719	40573	67057345	139	13	$\log_{10}/10$	31432 ms
	1381719	40573	67057345	108	13	$\log_e/5$	33485 ms
	1381719	40573	67057345	141	13	$\log_e/10$	28781 ms
Q18	83109015597512	43022705975952	2004315481077976	1	-	0.01/1	-
	82454196394188	42367884438959	2004315597761448	12	8	1/5	805 ms
	82454196394188	42367884438959	2004315597761448	12	8	1/10	365 ms
	82454193910208	42367884288648	2004315481077976	351	22	1.1/5	23158 ms
	82454193910208	42367884288648	2004315481077976	689	22	1.1/10	67698 ms
	82454193910208	42367884288648	2004315481077976	435	23	$\log_{10}/5$	38857 ms
	82454193910208	42367884288648	2004315481077976	802	23	$\log_{10}/10$	82160 ms
	82454193910208	42367884288648	2004315481077976	435	23	$\log_e/5$	39252 ms
	82454193910208	42367884288648	2004315481077976	802	23	$\log_e/10$	82065 ms
Q19	5834501	2101515	186649301	1	-	0.01/1	-
	5834501	2101515	186649301	2	5	1/5	1293169 ms
	5834501	2101515	186649301	2	5	1/10	1293169 ms
	5834501	2101515	186649301	2	5	1.1/5	1277440 ms
	5834501	2101515	186649301	2	5	1.1/10	1277440 ms
	5834501	2101515	186649301	2	5	$\log_{10}/5$	1273378 ms
	5834501	2101515	186649301	2	5	$\log_{10}/10$	1273378 ms
	5834501	2101515	186649301	14	7	$\log_e/5$	2290891 ms
	5834501	2101515	186649301	15	7	$\log_e/10$	2536674 ms
Q20	4657469308787	1423427793861	161702075746356	1	-	0.01/1	-
	822636143488	379197580952	22171928126870	36	12	1/5	3129 ms
	822636143488	379197580952	22171928126870	36	12	1/10	3129 ms
	822636135697	379197576687	22171927950556	1748	34	1.1/5	219823 ms
	822636135697	379197576687	22171927950556	3026	34	1.1/10	411716 ms
	822636135697	379197576687	22171927950556	2358	47	$\log_{10}/5$	444775 ms
	822636135697	379197576687	22171927950556	4144	47	$\log_{10}/10$	848806 ms
	822636135697	379197576687	22171927950556	2358	47	$\log_e/5$	440765 ms
	822636135697	379197576687	22171927950556	4144	47	$\log_e/10$	922406 ms

Anfrage	Gesamtkosten	$\mathsf{cost}_{\mathsf{I/O}}$	$\mathsf{cost}_{\mathsf{CPU}}$	Anz. Varianten	Iterationen	Grenzen dyn./stat.	Optimierungsdauer
Q21	250437439189 33	1154928704 3505	6747228437 71403	1	-	0.01/1	-
	23079663	8871406	710412744	36	14	1/5	8139 ms
	23079663	8871406	710412744	36	14	1/10	7890 ms
	3825784	9528334	1214872395	587	22	1.1/3	282518 ms
	49769042	27160348	1130434650	718	40	$\log_{10}/1$	1352204 ms
	49769042	27160348	1130434650	738	41	$\log_e/1$	1238268 ms
Q22	1006200	926995	3960233	1	-	0.01/1	-
	230296	108160	6106803	4	6	1/5	97 ms
	230296	108160	6106803	4	6	1/10	166 ms
	230296	108160	6106803	6	4	1.1/5	84 ms
	230296	108160	6106803	6	4	1.1/10	62 ms
	230296	108160	6106803	9	3	$\log_{10}/5$	133 ms
	230296	108160	6106803	9	3	$\log_{10}/10$	156 ms
	230296	108160	6106803	11	3	$\log_e/5$	120 ms
	230296	108160	6106803	11	3	$\log_e/10$	190 ms

Anfrage	cost(optimierte Variante)	/	cost(Ausgangsvariante)	=	Verhältnis y
Q1	7403558104769	/	7403558104769	=	1
Q2	1416162271013	/	2671620025457	=	$5.30076230 \cdot 10^{-1}$
Q3	64334333418	/	64342085874	=	$9.99879512 \cdot 10^{-1}$
Q4	49662743845	/	49690245189	=	$9.99446544 \cdot 10^{-1}$
Q5	1477657793	/	53166703162908	=	$2.77929175 \cdot 10^{-5}$
Q6	2232753	/	2232753	=	1
Q7	1245253600	/	1296694609	=	$9.60329126 \cdot 10^{-1}$
Q8	103089560	/	431712865	=	$2.38791957 \cdot 10^{-1}$
Q9	9270403779028889	/	9288192537726211	=	$9.98084799 \cdot 10^{-1}$
Q10	555318852267	/	555336276277	=	$9.99968624 \cdot 10^{-1}$
Q11	10853789964	/	10860498567	=	$9.99382293 \cdot 10^{-1}$
Q12	934487786	/	938208886	=	$9.96033826 \cdot 10^{-1}$
Q13	1614497	/	1614497	=	1
Q14	2987704	/	6211588	=	$4.80988758 \cdot 10^{-1}$
Q15	822624008508	/	822624008508	=	1
Q16	166020	/	2656833406	=	$6.24879225 \cdot 10^{-5}$
Q17	1381719	/	2048103536530	=	$6.74633374 \cdot 10^{-7}$
Q18	82454193910208	/	83109015597512	=	$9.92120931 \cdot 10^{-1}$
Q19	5834501	/	5834501	=	1
Q20	822636135697	/	4657469308787	=	$1.76627280 \cdot 10^{-1}$
Q21	23079663	/	25043743918933	=	$9.21573990 \cdot 10^{-7}$
Q22	230296	/	1006200	=	$2.28876963 \cdot 10^{-1}$

Tabelle D.3: TPC-H Version 2.3.0 Benchmarkergebnisse

D.2 Objektrelationale Tests mit ATKIS-Daten

In diesem Abschnitt gibt die erste Tabelle die Statistiken zu den 10 Relationen der räumlichen Daten wieder. Es ist jeweils die Kardinalität, die Pagegröße, die Attributwertanzahl und Geometrietypenverteilungen, die durchschnittliche Ausdehnung der vorhandenen Geometrien in x- und y-Richtung, und schließlich die Kardinalität und Seitengröße vorhandener Indexe aufgeführt.

In den Abschnitten D.2.1 bis D.2.8 sind alle räumlichen Anfragen sowie Informationen zu ihrer Optimierung durch `RELOpt` und Oracle 10g beschrieben. Da es in Oracle 10g für alle in dieser Arbeit verwendeten Anfragen nicht möglich war, die von `RELOpt` erstellten Ausführungspläne direkt einzugeben und berechnen zu lassen, mussten diese in Teilanfragen zerlegt werden, die separat aufgeführt sind.

Es folgt die Präsentation der Optimierungsergebnisse in Tabelle D.4. Die Darstellung der Resultate wird abgeschlossen durch die Tabelle D.5. Dort sind die einzelnen Ausführungszeiten, die sich durch die Zerlegung ergeben, in den Tabellenspalten I - VI angegeben. Ihre Summe entspricht der Ausführungszeit der von `RELOpt` empfohlenen Variante. Durch die nötige Zerlegung der Anfrage wird zusätzlicher, nicht benötigter Aufwand betrieben (z. B. werden Terminalausgaben mitberechnet), so dass es sich hier nur um eine obere Schranke für die Laufzeit handelt. Das Verhältnis der gesamten Ausführungszeiten zeigt das Potential des Prinzips der Optimierung durch bedingte Termersetzungen.

Statistiken

|bholz_25_geo|	=	30891
pg(bholz_25_geo)	=	2363
n(objektid, bholz_25_geo)	=	30818
n(objektnr, bholz_25_geo)	=	24755
n(geo, bholz_25_geo)	=	30891
n(objektart, bholz_25_geo)	=	54
sel(gt(geo) = point, bholz_25_geo)	=	0,012
sel(gt(geo) = line, bholz_25_geo)	=	0,495
sel(gt(geo) = polygon, bholz_25_geo)	=	0,492
sel(gt(geo) = collection, bholz_25_geo)	=	0
sel(gt(geo) = multipoint, bholz_25_geo)	=	0
sel(gt(geo) = multiline, bholz_25_geo)	=	0
sel(gt(geo) = multipolygon, bholz_25_geo)	=	0,001
$\overline{r_1}$(geo, bholz_25_geo)	=	22479,75
$\overline{r_2}$(geo, bholz_25_geo)	=	22514,254
$\overline{s_1}$(geo, bholz_25_geo)	=	189,968
$\overline{s_2}$(geo, bholz_25_geo)	=	177,782
|Index(geo, bholz_25_geo)|$_l$	=	4
pg(Index(geo, bholz_25_geo))	=	496
|Index(objektid, bholz_25_geo)|$_l$	=	1
pg(Index(objektid, bholz_25_geo))	=	91

|bholz_25_geo_l|	=	15285
pg(bholz_25_geo_l)	=	668
n(objektid, bholz_25_geo_l)	=	15285
n(geo, bholz_25_geo_l)	=	15285
n(objectid, bholz_25_geo_l)	=	15285
sel(gt(geo) = line, bholz_25_geo_l)	=	1
$\overline{r_1}$(geo, bholz_25_geo_l)	=	22470,676
$\overline{r_2}$(geo, bholz_25_geo_l)	=	22507,095
$\overline{s_1}$(geo, bholz_25_geo_l)	=	144,491
$\overline{s_2}$(geo, bholz_25_geo_l)	=	132,82
|Index(objektid, bholz_25_geo_l)|$_l$	=	1
pg(Index(objektid, bholz_25_geo_l))	=	31
|Index(objectid, bholz_25_geo_l)|$_l$	=	1
pg(Index(objectid, bholz_25_geo_l))	=	45

|bholz_25_geo_f|	=	15243
pg(bholz_25_geo_f)	=	1702
n(objektid, bholz_25_geo_f)	=	15243
n(geo, bholz_25_geo_f)	=	15243
n(objectid, bholz_25_geo_f)	=	15243
sel(gt(geo) = polygon, bholz_25_geo_f)	=	0,997
sel(gt(geo) = multipolygon, bholz_25_geo_f)	=	0,003
$\overline{r_1}$(geo, bholz_25_geo_f)	=	22479,75
$\overline{r_2}$(geo, bholz_25_geo_f)	=	22514,254
$\overline{s_1}$(geo, (bholz_25_geo_f)	=	240,095
$\overline{s_2}$(geo, bholz_25_geo_f)	=	227,101
|Index(objektid, bholz_25_geo_f)|$_l$	=	1
pg(Index(objektid, bholz_25_geo_f))	=	31
|Index(objectid, bholz_25_geo_f)|$_l$	=	1
pg(Index(objectid, bholz_25_geo_f))	=	45

|bholz_50_geo|	=	19017
pg(bholz_50_geo)	=	1177
n(objektid, bholz_50_geo)	=	19017
n(objektnr, bholz_50_geo)	=	12907
n(geo, bholz_50_geo)	=	19017
n(objektart, bholz_50_geo)	=	51
n(aggr, bholz_50_geo)	=	1
sel(gt(geo) = point, bholz_50_geo)	=	0,002
sel(gt(geo) = line, bholz_50_geo)	=	0,704
sel(gt(geo) = polygon, bholz_50_geo)	=	0,293
sel(gt(geo) = collection, bholz_50_geo)	=	0
sel(gt(geo) = multipoint, bholz_50_geo)	=	0
sel(gt(geo) = multiline, bholz_50_geo)	=	0
sel(gt(geo) = multipolygon, bholz_50_geo)	=	0,001
$\overline{r_1}$(geo, bholz_50_geo)	=	27395,54
$\overline{r_2}$(geo, bholz_50_geo)	=	25523,48
$\overline{s_1}$(geo, bholz_50_geo)	=	216,134
$\overline{s_2}$(geo, bholz_50_geo)	=	201,432
|Index(geo, bholz_50_geo)|$_l$	=	4
pg(Index(geo, bholz_50_geo))	=	306
|Index(objektid, bholz_50_geo)|$_l$	=	1
pg(Index(objektid, bholz_50_geo))	=	56

|bholz_250_geo|	=	701
pg(bholz_250_geo)	=	51
n(objektid, bholz_250_geo)	=	701
n(objektnr, bholz_250_geo)	=	701
n(geo, bholz_250_geo)	=	701
n(objektart, bholz_250_geo)	=	11
sel(gt(geo) = point, bholz_250_geo)	=	0,101
sel(gt(geo) = line, bholz_250_geo)	=	0,817
sel(gt(geo) = polygon, bholz_250_geo)	=	0,066
sel(gt(geo) = collection, bholz_250_geo)	=	0
sel(gt(geo) = multipoint, bholz_250_geo)	=	0
sel(gt(geo) = multiline, bholz_250_geo)	=	0,001
sel(gt(geo) = multipolygon, bholz_250_geo)	=	0,014
$\overline{r_1}$(geo, bholz_250_geo)	=	22479,757
$\overline{r_2}$(geo, bholz_250_geo)	=	22514,255
$\overline{s_1}$(geo, bholz_250_geo)	=	745,354
$\overline{s_2}$(geo, bholz_250_geo)	=	733,652
|Index(geo, bholz_250_geo)|$_l$	=	2
pg(Index(geo, bholz_250_geo))	=	13
|Index(objektid, bholz_250_geo)|$_l$	=	1
pg(Index(objektid, bholz_250_geo))	=	3

|bholz_25_namen|	=	23793
pg(bholz_25_namen)	=	76
n(objektnr, bholz_25_namen)	=	14429
n(typ, bholz_25_namen)	=	3
n(name, bholz_25_namen)	=	999
|Index(objektnr, bholz_25_namen)|$_l$	=	1
pg(Index(objektnr, bholz_25_namen))	=	70
|Index(typ, bholz_25_namen)|$_l$	=	1
pg(Index(typ, bholz_25_namen))	=	47

|bholz_25_objekte|	=	24390
pg(bholz_25_objekte)	=	94
n(objektnr, bholz_25_objekte)	=	24390
n(objektart, bholz_25_objekte)	=	58
n(ebene, bholz_25_objekte)	=	28
|Index(objektnr, bholz_25_objekte)|$_l$	=	1
pg(Index(objektnr, bholz_25_objekte))	=	63

|objektarten|	=	193
pg(objektarten)	=	4
n(objektart, objektarten)	=	193
n(name, objektarten)	=	192
|Index(objektart, objektarten)|$_l$	=	1
pg(Index(objektart, objektarten))	=	1

|bholz_25_attribute|	=	72771
pg(bholz_25_attribute)	=	228
n(objektid, bholz_25_attribute)	=	20767
n(typ, bholz_25_attribute)	=	24
n(wert, bholz_25_attribute)	=	57
|Index(objektid, bholz_25_attribute)|$_l$	=	1
pg(Index(objektid, bholz_25_attribute))	=	243
|Index(typ, bholz_25_attribute)|$_l$	=	1
pg(Index(typ, bholz_25_attribute))	=	248

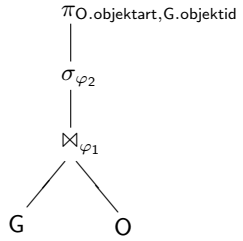
|attribute|	=	130
pg(attribute)	=	4
n(typ, attribute)	=	3
n(name, attribute)	=	128
n(attribut, attribute)	=	130
|Index(attribut, attribute)|$_l$	=	1
pg(Index(attribut, attribute))	=	1

D.2.1 oQuery 1

Diese Anfrage besteht aus einem einfachen Join von bholz_25_geo mit bholz_25_geo_l und einer anschließenden räumlichen Selektion. Der Index auf dem geometrischen Attribut von bholz_25_geo wurde bei dieser Anfrage nicht berücksichtigt:

$$\pi_{\text{O.objektart,G.objektid}}$$
$$(\sigma_{\text{G.geo ANYINTERACT GEOMETRY[POLYGON;1972;3650]}}$$
$$((\text{bholz_25_geo_l as O}) \bowtie_{\text{O.objektid=G.objektid}} (\text{bholz_25_geo as G})))$$

Der dazugehörige, leichter lesbare Operatorbaum:



$G := \text{bholz_25_geo}$
$O := \text{bholz_25_geo_l}$

$\varphi_1 := \text{O.objektid} = \text{G.objektid}$
$\varphi_2 := \text{G.geo ANYINTERACT GEOMETRY[POLYGON; 1972; 3650]}$

Abbildung D.2: räumliche Testanfrage 1

Nachfolgend die Anfrage in SQL, sowie der Ausführungsplan und die Statistiken von Oracle 10g:

```
select
    g.objektart,
    o.objektid
from
    bholz_25_geo g,
    bholz_25_geo_l o
where
    o.objektid=g.objektid
and SDO_GEOM.RELATE(
        g.geo,
        'ANYINTERACT',
        SDO_GEOMETRY(2003,NULL,NULL,
        SDO_ELEM_INFO_ARRAY(1,3,3),
        SDO_ORDINATE_ARRAY(
                  3558063.86015625,
                  5909408.354227405,
                  3560035.920703125,
                  5913058.421282799)),
                        0.05)='TRUE';
```

```
Elapsed: 00:00:16:40

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS
                              (Cost=527 Card=309 Bytes=11742)
   1    0   NESTED LOOPS (Cost=527 Card=309 Bytes=11742)
   2    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO' (TABLE)
                              (Cost=527 Card=309 Bytes=8343)
   3    1     INDEX (UNIQUE SCAN) OF 'PK_BUCHHOLZ_25_OBJGEO_L'
                       (INDEX(UNIQUE)) (Cost=0 Card=1 Bytes=11)

Statistics
----------------------------------------------------------
      31205  recursive calls
          0  db block gets
     336635  consistent gets
          0  physical reads
          0  redo size
      11756  bytes sent via SQL*Net to client
        793  bytes received via SQL*Net from client
         35  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        507  rows processed
```

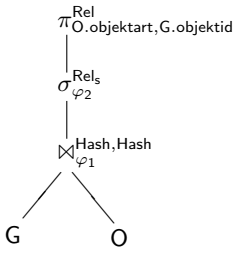
Die gleiche Anfrage nach einer Übersetzung für `RELOpt`:

```
(PROJECTION G.objektart,O.objektid
  (SELECTION G.geo_ANYINTERACT_GEOMETRY[POLYGON;1972;3650]
   (JOIN G.objektid=O.objektid (bholz_25_geo_l as O)(bholz_25_geo as G))))
```

Ergebnis der `RELOpt`-Optimierung:

```
(PROJECTION rel (G.OBJEKTART,O.OBJEKTID)
  (SELECTION rel G.GEO_ANYINTERACT_GEOMETRY[POLYGON;1972.0;3650.0]
    (JOIN hash-hash G.OBJEKTID=O.OBJEKTID
      (BHOLZ_25_GEO AS G)(BHOLZ_25_GEO_L AS O))))
```

mit dem dazugehörigen Operatorbaum:



G := bholz_25_geo
O := bholz_25_geo_l

φ_1 := O.objektid = G.objektid
φ_2 := G.geo
ANYINTERACT GEOMETRY[POLYGON; 1972; 3650]

Abbildung D.3: optimierte räumliche Testanfrage 1

Um die von `RELOpt` empfohlene Reihenfolge der Operatoren in Oracle 10g umsetzen zu können, wird die Anfrage in zwei Teile zerlegt. Anfangs erzeugen wir eine Sicht aus dem Verbund der Relationen bholz_25_geo und bholz_25_geo_l:

```
create
       materialized view x as
 select
        g.geo,
        g.objektart,
        n.objektid
 from
        bholz_25_geo g,
        bholz_25_geo_l n
 where
        n.objektid=g.objektid;
```

```
Elapsed: 00:00:03.97

Execution Plan
----------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS
                                    (Cost=524 Card=15321 Bytes=413667)
   1    0   NESTED LOOPS (Cost=524 Card=15321 Bytes=413667)
   2    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO' (TABLE)
                                    (Cost=521 Card=30891 Bytes=494256)
   3    1     INDEX (UNIQUE SCAN) OF 'PK_BUCHHOLZ_25_OBJGEO_L'
                              (INDEX(UNIQUE)) (Cost=0 Card=1 Bytes=11)
Statistics
----------------------------------------------------------------------
    1 recursive calls     363024 bytes sent via SQL*Net to client
    0 db block gets        11628 bytes received via SQL*Net from client
35204 consistent gets       1020 SQL*Net roundtrips to/from client
    0 physical reads           0 sorts (memory)
    0 redo size                0 sorts (disk)
                           15285 rows processed
```

um daraufhin die räumliche Selektion auszuüben:

```
select
  objektid,
  objektart
from x
where
  SDO_GEOM.RELATE(
    geo, 'ANYINTERACT',
    SDO_GEOMETRY(
        2003,NULL,NULL,
        SDO_ELEM_INFO_ARRAY
                      (1,3,3),
        SDO_ORDINATE_ARRAY(
             3558063.86015625,
            5909408.354227405,
            3560035.920703125,
          5913058.421282799)),
                 0.05)='TRUE';
```

```
Elapsed: 00:00:05.19

Execution Plan
----------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS
                                    (Cost=148 Card=127 Bytes=485902)
   1    0   MAT_VIEW ACCESS (FULL) OF 'X' (MAT_VIEW)
                                    (Cost=148 Card=127 Bytes=485902)

Statistics
----------------------------------------------------------------------
 15352 recursive calls     11756 bytes sent via SQL*Net to client
     0 db block gets         793 bytes received via SQL*Net from client
123236 consistent gets        35 SQL*Net roundtrips to/from client
   648 physical reads          0 sorts (memory)
     0 redo size               0 sorts (disk)
                             507  rows processed
```

Diese Anfrage konnte 1,79-mal schneller ausgeführt werden als mit einer Optimierung durch Oracle 10g.

D.2.2 oQuery 2

Im Folgenden die zweite räumliche Testanfrage mit zwei räumlichen und zwei relationalen Joins, sowie zwei relationalen Selektionen und anschließender relationaler Aggregation:

$\Gamma_{\text{O.objektart,O.name\#O.name,(count(G1.objektid))}}$
$(\sigma_{\text{G3.objektart=5101} \wedge \text{N.name} \neq \text{`NNNN`}}$[3]
$((\text{objektarten as O}) \bowtie_{\text{O.objektart=G1.objektart}}$
$((\text{bholz_25_namen as N}) \bowtie_{\text{N.objektnr=G1.objektnr}}$
$((\text{bholz_25_geo_f as G2}) \bowtie_{\text{G1.geo EQUAL G2.geo}}$
$((\text{bholz_25_geo as G1}) \bowtie_{\text{G1.geo ANYINTERACT G3.geo}} (\text{bholz_250_geo as G3}))))))$

mit dem dazugehörigen, leichter lesbaren Operatorbaum:

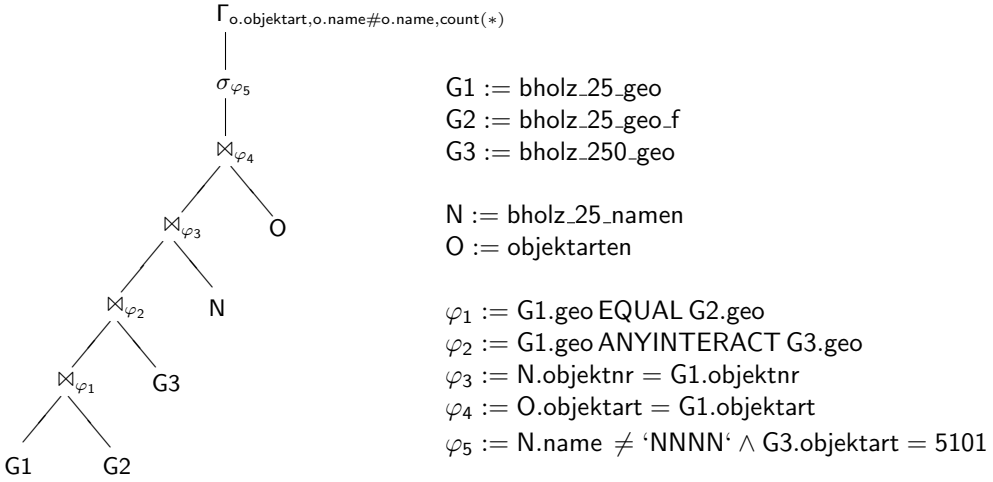


Abbildung D.4: räumliche Testanfrage 2

und der passenden SQL-Anfrage für die Datenbank:

```
select o.name, count(*)
from   bholz_25_geo g1, bholz_25_geo_f g2, bholz_250_geo g3,
       bholz_25_namen n1, objektarten o
where  SDO_RELATE(g1.geo,g2.geo,
                  'MASK=EQUAL QUERYTYPE=WINDOW')='TRUE'
and    SDO_RELATE(g3.geo,g1.geo,
                'MASK=ANYINTERACT QUERYTYPE=WINDOW')='TRUE'
and    n1.objektnr=g1.objektnr   and    o.objektart=g1.objektart
and    n1.name <> 'NNNN'         and    g3.objektart=5101
group by o.objektart,o.name;
```

Da diese Anfrage nach mehr als zwei Tagen kein Ergebnis geliefert hat, wurde sie ressourcenbedingt abgebrochen:

```
Elapsed: >53:08:19.49
```

Gleiche Anfrage, nur für `RELOpt` übersetzt:

```
(AGGREGATION (O.objektart,O.name#O.name,(count(G1.objektid)) as X.objectcount)
 (SELECTION (AND(N1.name<> 'NNNN',G3.objektart='5101'))
  (JOIN O.objektart=G1.objektart (objektarten as O)
   (JOIN N1.objektnr=G1.objektnr (bholz_25_namen as N1)
    (JOIN G1.geo_EQUAL_G2.geo (bholz_25_geo_f as G2)
     (JOIN G1.geo_ANYINTERACT_G3.geo(bholz_25_geo as G1)(bholz_250_geo as G3)))))))
```

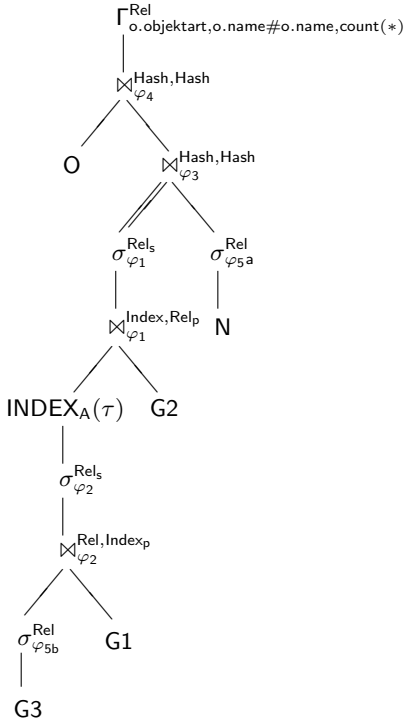
Ergebnis Optimierung durch `RELOpt`:

```
(AGGREGATION rel (O.OBJEKTART,O.NAME#O.NAME,count(G1.OBJEKTID) AS X.OBJECTCOUNT)
  (JOIN hash-hash O.OBJEKTART=G1.OBJEKTART (OBJEKTARTEN AS O)
    (JOIN hash-hash N1.OBJEKTNR=G1.OBJEKTNR
      (SECONDARY_FILTER rel G1.GEO_EQUAL_G2.GEO
        (JOIN ind-rel G1.GEO_EQUAL_G2.GEO
          (INDEX (G1.GEO)
```

[3] 'NNNN' ist als Name eingetragen, wenn der richtige Name unbekannt ist.

```
          (SECONDARY_FILTER rel G1.GEO_ANYINTERACT_G3.GEO
            (JOIN rel-ind G1.GEO_ANYINTERACT_G3.GEO
              (SELECTION rel G3.OBJEKTART='5101'(BHOLZ_250_GEO AS G3))
              (BHOLZ_25_GEO AS G1))))
        (BHOLZ_25_GEO_F AS G2)))
    (SELECTION rel N1.NAME<>'NNNN'(BHOLZ_25_NAMEN AS N1)))))
```

mit dem dazugehörigen Operatorbaum:



G1 := bholz_25_geo
G2 := bholz_25_geo_f
G3 := bholz_250_geo

N := bholz_25_namen
O := objektarten

φ_1 := G1.geo EQUAL G2.geo
φ_2 := G1.geo ANYINTERACT G3.geo
φ_3 := N.objektnr = G1.objektnr
φ_4 := O.objektart = G1.objektart
φ_{5a} := N.name $\neq$ 'NNNN'
φ_{5b} := G3.objektart = 5101

$\tau := \sigma^{\text{Rel}_s}_{\varphi_2}\left(\sigma^{\text{Rel}}_{\varphi_{5b}}(\text{G3}) \bowtie^{\text{Rel,Index}}_{\varphi_2} \text{G1}\right)$
A := geo

Abbildung D.5: optimierte räumliche Testanfrage 2

Wir zerlegen die Anfrage in vier Teile:

```
create
 materialized view tmp as
select
   b.objektid as id_b,
   b.objektnr as nr_b,
   b.geo as geo_b,
   b.objektart as art_b,
   a.objektid as id_a,
   a.objektnr as nr_a,
   a.geo as geo_a,
   a.objektart as art_a
from
   bholz_250_geo b,
   bholz_25_geo a
where
   b.objektart=5101
and SDO_RELATE(a.geo,
     b.geo,
     'MASK=ANYINTERACT
     QUERYTYPE=WINDOW')='TRUE';
```

```
Elapsed: 00:00:04.28

Execution Plan
----------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS
                                  (Cost=4784 Card=12974 Bytes=713570)
   1    0   NESTED LOOPS (Cost=4784 Card=12974 Bytes=713570)
   2    1     TABLE ACCESS (FULL) OF 'BHOLZ_250_GEO' (TABLE)
                                        (Cost=13 Card=42 Bytes=1176)
   3    1     TABLE ACCESS (BY INDEX ROWID) OF 'BHOLZ_25_GEO' (TABLE)
                                      (Cost=4784 Card=309 Bytes=8343)
   4    3       DOMAIN INDEX OF 'BHOLZ_25_GEOM_IDX' (INDEX (DOMAIN))
Statistics
----------------------------------------------------------------------
 7540 recursive calls     5611791 bytes sent via SQL*Net to client
   84 db block gets          1858 bytes received via SQL*Net from client
14546 consistent gets         104 SQL*Net roundtrips to/from client
    0 physical reads           86 sorts (memory)
    0 redo size                 0 sorts (disk)
                             1448 rows processed
```

Erzeugung eines Indexes über das Attribut geo_A:

```
create index tmp_idx on tmp(geo_A) indextype is MDSYS.SPATIAL_INDEX;
```

```
Elapsed: 00:00:00.67
```

Mit Hilfe des ersten Ergebnisses und dem Index wird das dritte Teilergebnis erzeugt:

```
create
  materialized view
   tmp2 as
select
  g1.art_a,
  g1.nr_a
from
  tmp g1,
  bholz_25_geo_f g2
where
  SDO_RELATE(
  g1.geo_a,
  g2.geo,
  'MASK=EQUAL
  QUERYTYPE=WINDOW')
            ='TRUE';
```

```
Elapsed: 00:02:06.24

Execution Plan
----------------------------------------------------------------------------------
 0     SELECT STATEMENT Optimizer=ALL_ROWS(Cost=30900 Card=191931 Bytes=741429453)
 1    0  NESTED LOOPS (Cost=30900 Card=191931 Bytes=741429453)
 2    1    TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_F' (TABLE)
                                              (Cost=375 Card=15243 Bytes=411561)
 3    1    MAT_VIEW ACCESS (BY INDEX ROWID) OF 'TMP'(MAT_VIEW)
                                               (Cost=30900 Card=13 Bytes=49868)
 4    3      DOMAIN INDEX OF 'TMP_IDX' (INDEX (DOMAIN))

Statistics
----------------------------------------------------------------------------------
 248688 recursive calls              21619 bytes sent via SQL*Net to client
  30486 db block gets                 1222 bytes received via SQL*Net from client
1174138 consistent gets                 74 SQL*Net roundtrips to/from client
      0 physical reads                30488 sorts (memory)
      0 redo size                         0 sorts (disk)
                                       1088 rows processed
```

Anschließend ein abschließender Verbund für die Erzeugung des Endergebnisses:

```
select
 /*+ ORDERED */
 o.name,
 count(*)
from
 tmp2 g1,
 bholz_25_namen n1,
 objektarten o
where
 n1.name<>'NNNN'
and
 n1.objektnr=g1.nr_a
and
 o.objektart=g1.art_a
group by
    o.objektart,
    o.name;
```

```
Elapsed: 00:00:00.02

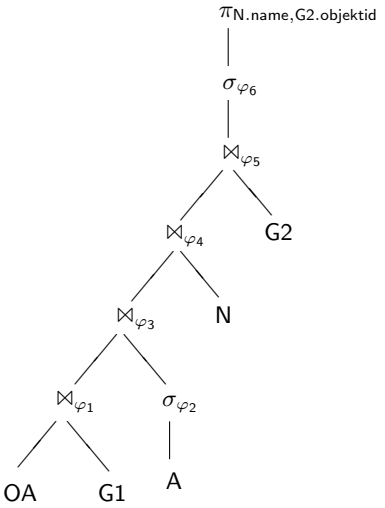
Execution Plan
----------------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=25 Card=1792 Bytes=75264)
   1    0   HASH (GROUP BY) (Cost=25 Card=1792 Bytes=75264)
   2    1     NESTED LOOPS (Cost=24 Card=1792 Bytes=75264)
   3    2       HASH JOIN (Cost=22 Card=1792 Bytes=39424)
   4    3         MAT_VIEW ACCESS (FULL) OF 'TMP2' (MAT_VIEW)
                                                 (Cost=3 Card=1088 Bytes=9792)
   5    3         TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN' (TABLE)
                                               (Cost=18 Card=23769 Bytes=308997)
   6    2       TABLE ACCESS (BY INDEX ROWID) OF 'OBJEKTARTEN'(TABLE)
                                                    (Cost=1 Card=1 Bytes=20)
   7    6         INDEX (UNIQUE SCAN) OF 'PK_OBJEKTARTEN'(INDEX (UNIQUE))
                                                          (Cost=0 Card=1)
Statistics
----------------------------------------------------------------------------------
        4  recursive calls        813  bytes sent via SQL*Net to client
        0  db block gets          430  bytes received via SQL*Net from client
      457  consistent gets          2  SQL*Net roundtrips to/from client
        0  physical reads           1  sorts (memory)
        0  redo size                0  sorts (disk)
                                   15  rows processed
```

D.2.3 oQuery 3

Im Folgenden die dritte Testanfrage mit drei relationalen sowie einem räumlichen Join und zwei relationalen Selektionen:

$$\pi_{\text{N.name,G2.objektid}}$$
$$(\sigma_{\text{G1.objektart}<3000}$$
$$((\text{bholz_25_geo_f as G2}) \bowtie_{\text{G1.geo INSIDE G2.geo}}$$
$$((\text{bholz_25_namen as N}) \bowtie_{\text{G1.objektnr=N.objektnr}}$$
$$((\sigma_{\text{A.attribut='a'}}(\text{attribute as A})) \bowtie_{\text{OA.typ=A.attribut}}$$
$$((\text{bholz_25_attribute as OA}) \bowtie_{\text{OA.objektid=G1.objektid}} (\text{bholz_25_geo as G1}))))))$$

Der entsprechende Operatorbaum ist:



G1 := bholz_25_geo
G2 := bholz_25_geo_f
N := bholz_25_namen
OA := bholz_25_attribute
A := attribute

φ_1 := OA.objektid = G1.objektid
φ_2 := A.attribut = 'a'
φ_3 := OA.typ = A.attribut
φ_4 := G1.objektnr = N.objektnr
φ_5 := G1.geo INSIDE G2.geo
φ_6 := G1.objektart < 3000

Abbildung D.6: räumliche Testanfrage 3

Nachfolgend die Anfrage in SQL:

```
select
    n.name,
    g2.objektid
from
    bholz_25_geo g1,
    bholz_25_attribute oa,
    attribute a,
    bholz_25_namen n,
    bholz_25_geo_f g2
where
    oa.objektid=g1.objektid
and oa.typ=a.attribut
and g1.objektnr=n.objektnr
and a.typ='a'
and g1.objektart<3000
and SDO_RELATE(
        g1.geo,
        g2.geo,
        'MASK=INSIDE
        QUERYTYPE=WINDOW')
                ='TRUE';
```

```
Elapsed: 02:27:01.58

Execution Plan
----------------------------------------------------------------------------
 0     SELECT STATEMENT Optimizer=ALL_ROWS
                                (Cost=28219 Card=1339798 Bytes=117902224)
 1    0  HASH JOIN (Cost=28219 Card=1339798 Bytes=117902224)
 2    1    TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN' (TABLE)
                                     (Cost=18 Card=23793 Bytes=309309)
 3    1    HASH JOIN (Cost=28192 Card=812506 Bytes=60937950)
 4    3      TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO' (TABLE)
                                      (Cost=521 Card=1545 Bytes=41715)
 5    3      HASH JOIN(Cost=20752 Card=1092441560 Bytes=52437194880)
 6    5        INDEX (FAST FULL SCAN) OF 'PK_BUCHHOLZ_25_OBJATTR'
                        (INDEX (UNIQUE))(Cost=62 Card=72771 Bytes=1091565)
 7    5        MERGE JOIN (CARTESIAN)(Cost=13772 Card=660530 Bytes=21797490)
 8    7          TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_F' (TABLE)
                                     (Cost=375 Card=15243 Bytes=411561)
 9    7          BUFFER (SORT) (Cost=13397 Card=43 Bytes=258)
10    9            TABLE ACCESS (FULL) OF 'ATTRIBUTE' (TABLE)
                                           (Cost=1 Card=43 Bytes=258)

Statistics
----------------------------------------------------------------------------
 2591429  recursive calls       3489  bytes sent via SQL*Net to client
       2  db block gets          529  bytes received via SQL*Net from client
46720679  consistent gets          11  SQL*Net roundtrips to/from client
       0  physical reads     1295656  sorts (memory)
       0  redo size                0  sorts (disk)
                                 139  rows processed
```

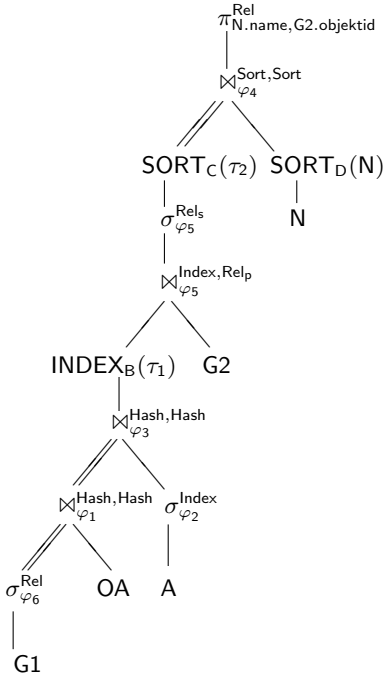
und nach der Übersetzung für `RELOpt`:

```
(PROJECTION (N.name, G2.objektid)
  (SELECTION G1.objektart<3000
    (JOIN G1.geo_INSIDE_G2.geo (bholz_25_geo_f as G2)
      (JOIN G1.objektnr=N.objektnr (bholz_25_namen as N)
        (JOIN OA.typ=A.attribut (SELECTION A.attribut='a' (attribute as A))
          (JOIN OA.objektid=G1.objektid(bholz_25_attribute as OA) (bholz_25_geo as G1)))))))
```

Ergebnis der `RELOpt`-Optimierung:

```
(PROJECTION rel (N.NAME,G2.OBJEKTID)
  (JOIN sort-sort G1.OBJEKTNR=N.OBJEKTNR
    (SORT (G1.OBJEKTNR)
      (SECONDARY_FILTER rel G1.GEO_INSIDE_G2.GEO
        (JOIN ind-rel G1.GEO_INSIDE_G2.GEO
```

```
          (INDEX (G1.GEO)
            (JOIN hash-hash OA.TYP=A.ATTRIBUT
              (JOIN hash-hash OA.OBJEKTID=G1.OBJEKTID
                (SELECTION rel G1.OBJEKTART<3000
                  (BHOLZ_25_GEO AS G1))
                (BHOLZ_25_ATTRIBUTE AS OA))
              (SELECTION ind A.ATTRIBUT='a'
                (ATTRIBUTE AS A))))
          (BHOLZ_25_GEOM_F AS G2))))
    (SORT (N.OBJEKTNR)(BHOLZ_25_NAMEN AS N))))
```



$G1 := bholz_25_geo$
$G2 := bholz_25_geo_f$
$N := bholz_25_namen$
$OA := bholz_25_attribute$
$A := attribute$

$\varphi_1 := OA.objektid = G1.objektid$
$\varphi_2 := A.attribut = \text{‘a‘}$
$\varphi_3 := OA.typ = A.attribut$
$\varphi_4 := G1.objektnr = N.objektnr$
$\varphi_5 := G1.geo\ \text{INSIDE}\ G2.geo$
$\varphi_6 := G1.objektart < 3000$

$\tau_1 := \sigma^{Rel}_{\varphi_6}\left(\sigma^{Index}_{\varphi_2}(A) \bowtie^{Rel,Rel}_{\varphi_3} (G1 \bowtie^{Hash,Hash}_{\varphi_1} OA)\right)$
$\tau_2 := \sigma_{\varphi_5}(\tau_1 \bowtie_{\varphi_5} G2)$
$B := geo$
$C := objektnr$
$D := objektnr$

Abbildung D.7: optimierte räumliche Testanfrage 3

Um die von `RELOpt` empfohlene Reihenfolge der Operatoren in Oracle 10g umsetzen zu können, wird die Anfrage in drei Teilanfragen sowie die Erzeugung eines zusätzlichen Indexes über das Attribut geo getrennt.

```
create table
   tmp as
select
   g1.objektid as objektid_g,
   objektnr,
   geo,
   objektart,
   attribut,
   name,
   a.typ as typ_a,
   oa.objektid as objektid_oa,
   oa.typ as typ_oa,
   wert
from
   bholz_25_geo g1,
   attribute a,
   bholz_25_attribute oa
where
   oa.objektid=g1.objektid
and oa.typ=a.attribut
and a.typ='a'
and g1.objektart<3000
```

```
Elapsed: 00:00:00.12

Execution Plan
----------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS
                                        (Cost=587 Card=5330 Bytes=197210)
   1    0   HASH JOIN (Cost=587 Card=5330 Bytes=197210)
   2    1     TABLE ACCESS (FULL) OF 'ATTRIBUTE' (TABLE)
                                          (Cost=3 Card=43 Bytes=258)
   3    1     HASH JOIN (Cost=584 Card=5412 Bytes=167772)
   4    3       TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO' (TABLE)
                                           (Cost=521 Card=1545 Bytes=24720)
   5    3       INDEX (FAST FULL SCAN) OF 'PK_BUCHHOLZ_25_OBJATTR'
                      (INDEX (UNIQUE))(Cost=62 Card=72771 Bytes=1091565)

Statistics
----------------------------------------------------------------------------
    1 recursive calls         1862 bytes sent via SQL*Net to client
    0 db block gets            485 bytes received via SQL*Net from client
 2608 consistent gets            7 SQL*Net roundtrips to/from client
    0 physical reads             0 sorts (memory)
    0 redo size                  0 sorts (disk)
                                85 rows processed
```

```
create index tmp_idx on bholz_25_geo_f(geo) indextype is MDSYS.SPATIAL_INDEX;
```

```
Elapsed: 00:00:03.18
```

```
create
  materialized view
   tmp2 as
select
   g2.objektid,
   g1.objektnr
from
   bholz_25_geo_f g2,
   tmp g1
where
   SDO_RELATE(
      g2.geo,
      g1.geo,
      'MASK=CONTAINS
   QUERYTYPE=WINDOW')
            ='TRUE';
```

```
Elapsed: 00:00:01.31

Execution Plan
----------------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=174 Card=12957 Bytes=49845579)
   1    0   NESTED LOOPS (Cost=174 Card=12957 Bytes=49845579)
   2    1     TABLE ACCESS (FULL) OF 'TMP' (TABLE)(Cost=4 Card=85 Bytes=324700)
   3    1     TABLE ACCESS (BY INDEX ROWID)OF 'BHOLZ_25_GEO_F'(TABLE)
                                                 (Cost=174 Card=152 Bytes=4104)
   4    3       DOMAIN INDEX OF 'TMP_IDX' (INDEX (DOMAIN))

Statistics
----------------------------------------------------------------------------------
       2533  recursive calls          3902  bytes sent via SQL*Net to client
        172  db block gets             529  bytes received via SQL*Net from client
       9251  consistent gets            11  SQL*Net roundtrips to/from client
          0  physical reads            170  sorts (memory)
          0  redo size                   0  sorts (disk)
                                       139  rows processed
```

```
select
  /*+ USE_MERGE
         (g1, n)*/
  n.name,
  g1.objektid
from
  tmp2 g1,
  bholz_25_namen n
where
  g1.objektnr
       =n.objektnr;
```

```
Elapsed: 00:00:00.03

Execution Plan
----------------------------------------------------------------------------------
 0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=24 Card=229 Bytes=5725)
 1    0   MERGE JOIN (Cost=24 Card=229 Bytes=5725)
 2    1     SORT (JOIN) (Cost=4 Card=139 Bytes=1668)
 3    2       MAT_VIEW ACCESS (FULL) OF 'TMP2'(MAT_VIEW)(Cost=3 Card=139 Bytes=1668)
 4    1     SORT (JOIN) (Cost=20 Card=23793 Bytes=309309)
 5    4       TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN'(TABLE)
                                              (Cost=18 Card=23793 Bytes=309309)

Statistics
----------------------------------------------------------------------------------
          4  recursive calls          3313  bytes sent via SQL*Net to client
          0  db block gets             529  bytes received via SQL*Net from client
         79  consistent gets            11  SQL*Net roundtrips to/from client
          0  physical reads              3  sorts (memory)
          0  redo size                   0  sorts (disk)
                                       139  rows processed
```

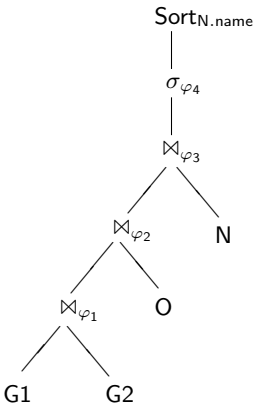
Insgesamt konnte die dritte räumliche Anfrage mit dem vorgeschlagenen Optimierungskonzept fast 1901, 20-mal schneller beantwortet werden.

D.2.4 oQuery 4

Im Folgenden die vierte räumliche Testanfrage mit einem räumlichen und zwei relationalen Joins sowie einer räumlichen Selektion:

$$
\begin{aligned}
&\mathsf{Sort}_{\mathsf{N.name}}\\
&\quad(\sigma_{\mathsf{G2.geo\ ANYINTERACT\ GEOMETRY[POLYGON;7622;6568]}}\\
&\qquad((\mathsf{bholz_25_namen\ as\ N}) \bowtie_{\mathsf{N.objektnr=G2.objektnr}}\\
&\qquad\quad((\mathsf{objektarten\ as\ O}) \bowtie_{\mathsf{O.objektart=G2.objektart}}\\
&\qquad\qquad((\mathsf{bholz_250_geo\ as\ G1}) \bowtie_{\mathsf{G1.geo\ COVERS\ G2.geo}}\\
&\qquad\qquad\quad(\mathsf{bholz_25_geo\ as\ G2})))))
\end{aligned}
$$

mit dem dazugehörigen, leichter lesbaren Operatorbaum:



G1 := bholz_250_geo
G2 := bholz_25_geo
N := bholz_25_namen
O := objektarten

φ_1 := G1.geo COVERS G2.geo
φ_2 := O.objektart = G2.objektart
φ_3 := N.objektnr = G2.objektnr
φ_4 := G2.geo
ANYINTERACT GEOMETRY[POLYGON; 7622; 6568]

Abbildung D.8: räumliche Testanfrage 4

Nachfolgend die Anfrage in SQL:

```
select
  o.name, n.name, g2.objektid
from
  bholz_25_namen n,
  bholz_25_geo g2,
  bholz_250_geo g1,
  objektarten o
where
  n.objektnr=g2.objektnr
and
  o.objektart=g2.objektart
and
  SDO_RELATE(g2.geo,
   SDO_GEOMETRY(2003,NULL,NULL,
   SDO_ELEM_INFO_ARRAY(1,3,3),
   SDO_ORDINATE_ARRAY(
              3546445.2265625,
              5898233.66224649,
             3554067.266796875,
          5904801.7090483615)),
          'MASK=ANYINTERACT
              QUERYTYPE=WINDOW')
                          ='TRUE'
and SDO_RELATE(g1.geo,
                g2.geo,
                'MASK=COVERS
              QUERYTYPE=WINDOW')
                          ='TRUE'
order by n.name;
```

```
Elapsed: 00:00:37.76

Execution Plan
----------------------------------------------------------------------
 0     SELECT STATEMENT Optimizer=ALL_ROWS
                                    (Cost=754 Card=3571 Bytes=314248)
 1    0  SORT (ORDER BY) (Cost=754 Card=3571 Bytes=314248)
 2    1    HASH JOIN (Cost=753 Card=3571 Bytes=314248)
 3    2      NESTED LOOPS (Cost=734 Card=2165 Bytes=162375)
 4    3        NESTED LOOPS (Cost=115 Card=309 Bytes=14523)
 5    4          TABLE ACCESS (BY INDEX ROWID)OF 'BHOLZ_25_GEO' (TABLE)
                                      (Cost=114 Card=309 Bytes=8343)
 6    5            DOMAIN INDEX OF 'BHOLZ_25_GEOM_IDX'(INDEX (DOMAIN))
 7    4          TABLE ACCESS (BY INDEX ROWID) OF 'OBJEKTARTEN'(TABLE)
                                             (Cost=1 Card=1 Bytes=20)
 8    7            INDEX (UNIQUE SCAN) OF 'PK_OBJEKTARTEN'
                                           (INDEX (UNIQUE)) (Cost=0 Card=1)
 9    3        DOMAIN INDEX OF 'BHOLZ_250_GEOM_IDX'(INDEX (DOMAIN))
10    2      TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN' (TABLE)
                                    (Cost=18 Card=23793 Bytes=309309)

Statistics
----------------------------------------------------------------------
 63022 recursive calls       541 bytes sent via SQL*Net to client
  5442 db block gets         430 bytes received via SQL*Net from client
267450 consistent gets         2 SQL*Net roundtrips to/from client
     0 physical reads       5443 sorts (memory)
     0 redo size               0 sorts (disk)
                               2 rows processed
```

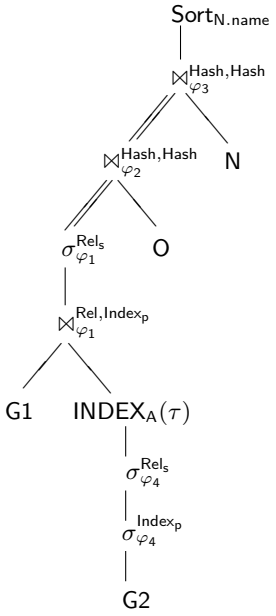
Gleiche Anfrage in der `RELOpt`-Sprache:

```
(SORT (N.name)
 (SELECTION G2.geo_ANYINTERACT_GEOMETRY[POLYGON;7622;6568]
   (JOIN N.objektnr=G2.objektnr (bholz_25_namen as N)
    (JOIN O.objektart=G2.objektart (objektarten as O)
     (JOIN G1.geo_COVERS_G2.geo(bholz_250_geo as G1) (bholz_25_geo as G2))))))
```

Ergebnis der `RELOpt`-Optimierung:

```
(SORT (N.NAME)
  (JOIN hash-hash N.OBJEKTNR=G2.OBJEKTNR
    (JOIN hash-hash O.OBJEKTART=G2.OBJEKTART
      (SECONDARY_FILTER rel G1.GEO_COVERS_G2.GEO
        (JOIN rel-ind G1.GEO_COVERS_G2.GEO
          (BHOLZ_250_GEO AS G1)
          (INDEX (G2.GEO)
            (SECONDARY_FILTER rel G2.GEO_ANYINTERACT_GEOMETRY[POLYGON;7622.0;6568.0]
              (SELECTION ind G2.GEO_ANYINTERACT_GEOMETRY[POLYGON;7622.0;6568.0]
                (BHOLZ_25_GEO AS G2))))))
      (OBJEKTARTEN AS O))
    (BHOLZ_25_NAMEN AS N)))
```

Der dazugehörige Operatorbaum:



G1 := bholz_250_geo
G2 := bholz_25_geo
N := bholz_25_namen
O := objektarten

φ_1 := G1.geo COVERS G2.geo
φ_2 := O.objektart = G2.objektart
φ_3 := N.objektnr = G2.objektnr
φ_4 := G2.geo
ANYINTERACT GEOMETRY[POLYGON; 7622; 6568]

$\tau := \sigma^{\text{Rel}_s}_{\varphi_4}(\sigma^{\text{Index}_p}_{\varphi_4} \text{G2})$
A := geo

Abbildung D.9: optimierte räumliche Testanfrage 4

Wieder ist eine Zerlegung in vier Teilanfragen nötig, um die Reihenfolge der Operatoren in Oracle 10g umzusetzen:

```
create table tmp as
select *
from
  bholz_25_geo g2
where
  SDO_RELATE(
   geo,
   SDO_GEOMETRY(2003,NULL,NULL,
    SDO_ELEM_INFO_ARRAY(1,3,3),
    SDO_ORDINATE_ARRAY(
          3546445.2265625,
         5898233.66224649,
        3554067.266796875,
      5904801.7090483615)),
     'MASK=ANYINTERACT
        QUERYTYPE=WINDOW')
                    ='TRUE';
```

```
Elapsed: 00:00:00.26

Execution Plan
----------------------------------------------------------------------
  0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=3 Card=309 Bytes=8343)
  1    0   TABLE ACCESS (BY INDEX ROWID) OF 'BHOLZ_25_GEO'(TABLE)
                                          (Cost=3 Card=309 Bytes=8343)
  2    1     DOMAIN INDEX OF 'BHOLZ_25_GEOM_IDX'(INDEX (DOMAIN))

Statistics
----------------------------------------------------------------------
   536 recursive calls      48264 bytes sent via SQL*Net to client
     2 db block gets         2421 bytes received via SQL*Net from client
  3926 consistent gets        183 SQL*Net roundtrips to/from client
     0 physical reads           2 sorts (memory)
   116 redo size                0 sorts (disk)
                             2720 rows processed
```

```
create index tmp_idx on tmp(geo) indextype is MDSYS.SPATIAL_INDEX;
```

```
Elapsed: 00:00:01.05
```

```
create materialized view tmp2 as
  select g2.objektid as objektid2,
         g1.objektid as objektid1,
         g2.objektnr,
         g2.objektart
  from   tmp g2,
         bholz_250_geo g1
  where  SDO_RELATE(g2.geo,
                    g1.geo,
                  'MASK=COVEREDBY
       QUERYTYPE=WINDOW')='TRUE';
```

```
Elapsed: 00:00:10.52

Execution Plan
----------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS
                                  (Cost=1417 Card=19168 Bytes=74199328)
   1    0   NESTED LOOPS (Cost=1417 Card=19168 Bytes=74199328)
   2    1     TABLE ACCESS (FULL) OF 'BHOLZ_250_GEO' (TABLE)
                                       (Cost=13 Card=701 Bytes=19628)
   3    1     TABLE ACCESS (BY INDEX ROWID) OF 'TMP' (TABLE)
                                     (Cost=1417 Card=27 Bytes=103761)
   4    3       DOMAIN INDEX OF 'TMP_IDX' (INDEX (DOMAIN))
```

```
Statistics
----------------------------------------------------------------------------
      13660  recursive calls        603  bytes sent via SQL*Net to client
       1404  db block gets          430  bytes received via SQL*Net from client
      55821  consistent gets          2  SQL*Net roundtrips to/from client
          0  physical reads        1404  sorts (memory)
          0  redo size                0  sorts (disk)
                                      2  rows processed
```

```
select
  o.name,
  n.name,
  g2.objektid2
from
  tmp2 g2,
  objektarten o,
  bholz_25_namen n
where
  n.objektnr
      =g2.objektnr
and
  o.objektart
     =g2.objektart
order by
  n.name;
```

```
Elapsed: 00:00:00.02

Execution Plan
----------------------------------------------------------------------------------------
0     SELECT STATEMENT Optimizer=ALL_ROWS(Cost=11 Card=3 Bytes=171)
1   0  SORT (ORDER BY) (Cost=11 Card=3 Bytes=171)
2   1    TABLE ACCESS (BY INDEX ROWID) OF 'BHOLZ_25_NAMEN'(TABLE)
                                                          (Cost=3 Card=2 Bytes=26)
3   2      NESTED LOOPS (Cost=10 Card=3 Bytes=171)
4   3        NESTED LOOPS (Cost=4 Card=2 Bytes=88)
5   4          MAT_VIEW ACCESS (FULL) OF 'TMP2' (MAT_VIEW)(Cost=3 Card=2 Bytes=48)
6   4          TABLE ACCESS (BY INDEX ROWID) OF 'OBJEKTARTEN'(TABLE)
                                                          (Cost=1 Card=1 Bytes=20)
7   6            INDEX(UNIQUE SCAN) OF 'PK_OBJEKTARTEN'(INDEX(UNIQUE))(Cost=0 Card=1)
8   3        INDEX(RANGE SCAN) OF 'PK_BUCHHOLZ_25_OBJNAM'(INDEX(UNIQUE))
                                                               (Cost=1 Card=2)
```

```
Statistics
----------------------------------------------------------------------------
         32  recursive calls        531  bytes sent via SQL*Net to client
          0  db block gets          430  bytes received via SQL*Net from client
         16  consistent gets          2  SQL*Net roundtrips to/from client
          0  physical reads           1  sorts (memory)
          0  redo size                0  sorts (disk)
                                      2  rows processed
```

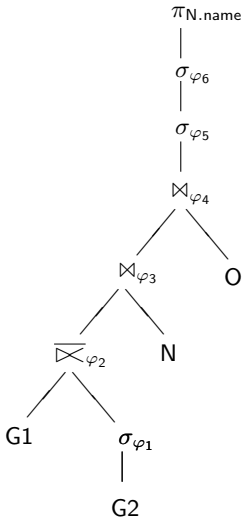
Die vierte räumliche Anfrage konnte 3, 19-mal schneller beantwortet werden.

D.2.5 oQuery 5

Im Folgenden die fünfte räumliche Testanfrage mit zwei relationalen Joins, einem räumlichen Antisemijoin sowie zwei relationalen und einer räumlichen Selektion:

$$
\begin{aligned}
&\pi_{\mathsf{N.name}}\\
&\quad(\sigma_{\mathsf{G1.geo\ INSIDE\ GEOMETRY[POLYGON;9262;4367]}}\\
&\qquad(\sigma_{\mathsf{O.objektart=2111}}\\
&\qquad\quad((\mathsf{bholz_25_objekte\ as\ O}) \bowtie_{\mathsf{O.objektnr=G1.objektnr}}\\
&\qquad\qquad((\mathsf{bholz_25_namen\ as\ N}) \bowtie_{\mathsf{G1.objektnr=N.objektnr}}\\
&\qquad\qquad\quad((\mathsf{bholz_25_geo_f\ as\ G1}) \overline{\ltimes}_{\mathsf{G1.geo\ ANYINTERACT\ G2.geo}}\\
&\qquad\qquad\qquad(\sigma_{\mathsf{G2.objektart=3541}}(\mathsf{bholz_25_geo\ as\ G2})))))))
\end{aligned}
$$

Der Operatorbaum zu dieser ist:



G1 := bholz_25_geo_f
G2 := bholz_25_geo
O := bholz_25_objekte
N := bholz_25_namen

φ_1 = G2.objektart = 3541
φ_2 = G1.geo ANYINTERACT G2.geo
φ_3 = G1.objektnr = N.objektnr
φ_4 = O.objektnr = G1.objektnr
φ_5 = O.objektart = 2111
φ_6 = G1.geo
INSIDE GEOMETRY[POLYGON; 9262; 4367]

Abbildung D.10: räumliche Testanfrage 5

Nachfolgend die Anfrage in SQL:

```
select n.name
from   bholz_25_geo_f g1,
       bholz_25_namen n,
       bholz_25_objekte o
where  o.objektart=2111
and    o.objektnr=g1.objektnr
and    n.objektnr=g1.objektnr
and
  SDO_GEOM.RELATE(
  g1.geo,'INSIDE',
  SDO_GEOMETRY(
            2003,NULL,NULL,
  SDO_ELEM_INFO_ARRAY(1,3,3),
  SDO_ORDINATE_ARRAY(
          3552328.00127551,
         5911548.630055703,
        3561590.7942784256,
        5915916.204286752))
            ,0.05)='INSIDE'
and not exists (
  select g2.objektid
  from   bholz_25_geo g2
  where SDO_RELATE(
          g2.geo,g1.geo,
          'MASK=ANYINTERACT
    QUERYTYPE=WINDOW')='TRUE'
   and   g2.objektart=3541)
order by n.name;
```

```
Elapsed: 00:00:09.27

Execution Plan
----------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=422 Card=13 Bytes=676)
   1    0   FILTER
   2    1     HASH JOIN (Cost=420 Card=251 Bytes=13052)
   3    2       HASH JOIN (Cost=401 Card=152 Bytes=5928)
   4    3         TABLE ACCESS (FULL)OF 'BHOLZ_25_GEO_F'(TABLE)
                                          (Cost=378 Card=152 Bytes=4104)
   5    3         TABLE ACCESS (FULL) OF 'BHOLZ_25_OBJEKTE'(TABLE)
                                         (Cost=22 Card=2998 Bytes=35976)
   6    2       TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN'
                    (TABLE) (Cost=18 Card=23793 Bytes=309309)
   7    1     TABLE ACCESS (BY INDEX ROWID)OF 'BHOLZ_25_GEO'(TABLE)
                                               (Cost=2 Card=1 Bytes=27)
   8    7       DOMAIN INDEX OF 'BHOLZ_25_GEOM_IDX'(INDEX (DOMAIN))

Statistics
----------------------------------------------------------------------------
   37889 recursive calls       3379 bytes sent via SQL*Net to client
     590 db block gets          639 bytes received via SQL*Net from client
  170380 consistent gets         21 SQL*Net roundtrips to/from client
       9 physical reads         615 sorts (memory)
       0 redo size                0 sorts (disk)
                                293 rows processed
```

Gleiche Anfrage, nur für `RELOpt` übersetzt:

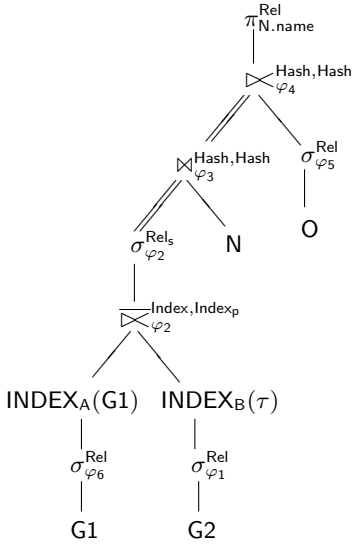
```
(PROJECTION N.name
 (SELECTION G1.geo_INSIDE_GEOMETRY[POLYGON;9262;4367]
  (SELECTION O.objektart=2111
   (JOIN O.objektnr=G1.objektnr (bholz_25_objekte as O)
    (JOIN G1.objektnr=N.objektnr (bholz_25_namen as N)
     (ANTISEMIJOIN G1.geo_ANYINTERACT_G2.geo (bholz_25_geo_f as G1)
      (SELECTION G2.objektart=3541 (bholz_25_geo as G2)))))))
```

Ergebnis der `RELOpt`-Optimierung:

```
(PROJECTION rel (N.NAME)
  (SEMIJOIN hash-hash O.OBJEKTNR=G1.OBJEKTNR
    (JOIN hash-hash G1.OBJEKTNR=N.OBJEKTNR
      (SECONDARY_FILTER rel G1.GEO_ANYINTERACT_G2.GEO
```

```
        (ANTISEMIJOIN ind-ind G1.GEO_ANYINTERACT_G2.GEO
          (INDEX (G1.GEO)
            (SELECTION rel G1.GEO_INSIDE_GEOMETRY[POLYGON;9262.0;4367.0]
              (BHOLZ_25_GEO_F AS G1)))
          (INDEX (G2.GEO)
            (SELECTION rel G2.OBJEKTART=3541
              (BHOLZ_25_GEO AS G2)))))
      (BHOLZ_25_NAMEN AS N))
    (SELECTION rel O.OBJEKTART=2111
      (BHOLZ_25_OBJEKTE AS O))))
```

mit dem dazugehörigen, leichter lesbaren Operatorbaum:



G1 := bholz_25_geo_f
G2 := bholz_25_geo
O := bholz_25_objekte
N := bholz_25_namen

φ_1 := G2.objektart = 3541
φ_2 := G1.geo ANYINTERACT G2.geo
φ_3 := G1.objektnr = N.objektnr
φ_4 := O.objektnr = G1.objektnr
φ_5 := O.objektart = 2111
φ_6 := G1.geo
INSIDE GEOMETRY[POLYGON; 9262; 4367]

$\tau := \sigma^{Rel}_{\varphi_1}(G2)$
A := geo
B := geo

Abbildung D.11: optimierte räumliche Testanfrage 5

Um die von `RELOpt` empfohlene Reihenfolge der Operatoren in Oracle 10g umsetzen zu können, wird die Anfrage in sechs Teilanfragen zerlegt.

```
create table
 tmp as
select
  objektid
from
  bholz_25_geo
where
  objektart=3541;
```

```
Elapsed: 00:00:00.08

Execution Plan
----------------------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=521 Card=236 Bytes=3776)
   1    0   TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO'(TABLE)(Cost=521 Card=236 Bytes=3776)

Statistics
----------------------------------------------------------------------------------------
          1  recursive calls       5738  bytes sent via SQL*Net to client
          0  db block gets          650  bytes received via SQL*Net from client
       2344  consistent gets         22  SQL*Net roundtrips to/from client
          0  physical reads           0  sorts (memory)
          0  redo size                0  sorts (disk)
                                    303  rows processed
```

```
create table tmp2 as select * from bholz_25_geo_f
   where SDO_GEOM.RELATE(geo,'INSIDE',
              SDO_GEOMETRY(2003, NULL, NULL,SDO_ELEM_INFO_ARRAY(1,3,3),
              SDO_ORDINATE_ARRAY(3552328.00127551,5911548.630055703,3561590.7942784256,
                  5915916.204286752)),0.05)='INSIDE';
```

```
Elapsed: 00:00:05.23

Execution Plan
----------------------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=378 Card=152 Bytes=4104)
   1    0   TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_F'(TABLE) (Cost=378 Card=152 Bytes=4104)
Statistics
----------------------------------------------------------------------------------------
     16555  recursive calls      18998  bytes sent via SQL*Net to client
         0  db block gets         1200  bytes received via SQL*Net from client
    126341  consistent gets          72  SQL*Net roundtrips to/from client
         0  physical reads           1  sorts (memory)
         0  redo size                0  sorts (disk)
                                  1058  rows processed
```

```
create index tmp_idx on tmp(geo) indextype is MDSYS.SPATIAL_INDEX;
```

```
Elapsed: 00:00:00.40
```

```
create index tmp2_idx on tmp2(geo) indextype is MDSYS.SPATIAL_INDEX;
```

```
Elapsed: 00:00:00.57
```

```
create table
 tmp3 as
select
 objektid
from
 TABLE
  (SDO_JOIN
   ('tmp','geo',
    'tmp2','geo',
    'MASK=ANYINTERACT',
    0)) g2,
    tmp2 x
where
 x.rowid=g2.rowid2;
```

```
Elapsed: 00:00:00.14

Execution Plan
----------------------------------------------------------------------------------
 0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=52 Card=88564 Bytes=1859844)
 1    0   HASH JOIN (Cost=52 Card=88564 Bytes=1859844)
 2    1     TABLE ACCESS (FULL) OF 'TMP2' (TABLE)(Cost=27 Card=1084 Bytes=20596)
 3    1     COLLECTION ITERATOR (PICKLER FETCH)OF 'SDO_JOIN' (PROCEDURE)

Statistics
----------------------------------------------------------------------------------
        579  recursive calls       1472  bytes sent via SQL*Net to client
          0  db block gets          474  bytes received via SQL*Net from client
        947  consistent gets          6  SQL*Net roundtrips to/from client
          0  physical reads           7  sorts (memory)
          0  redo size                0  sorts (disk)
                                     61  rows processed
```

```
select
  /*+ USE_HASH(g1,n)*/
  n.name
from
  tmp2 g1,
  bholz_25_objektnamen n
where
  n.objektnr=g1.objektnr
and exists(
  select
   /*+ USE_HASH(g1, o)*/
   o.objektnr
  from
   bholz_25_objekte o
  where
   o.objektnr=g1.objektnr
  and  o.objektart=2111)
and not exists(
 select
  x.objektid
 from
  tmp3 x
 where
  x.objektid=g1.objektid);
```

```
Elapsed: 00:00:00.03

Execution Plan
----------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=72 Card=1 Bytes=44)
   1    0   HASH JOIN (SEMI) (Cost=72 Card=1 Bytes=44)
   2    1     HASH JOIN (Cost=50 Card=1 Bytes=32)
   3    2       HASH JOIN (RIGHT ANTI)(Cost=31 Card=1 Bytes=19)
   4    3         TABLE ACCESS (FULL) OF 'TMP3' (TABLE)
                                              (Cost=3 Card=61 Bytes=427)
   5    3         TABLE ACCESS (FULL) OF 'TMP2' (TABLE)
                                           (Cost=27 Card=1196 Bytes=14352)
   6    2       TABLE ACCESS (FULL) OF 'BHOLZ_25_OBJEKTNAMEN'(TABLE)
                                         (Cost=18 Card=23793 Bytes=309309)
   7    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_OBJEKTE'(TABLE)
                                           (Cost=22 Card=2998 Bytes=35976)

Statistics
----------------------------------------------------------------------------
          0  recursive calls       3379  bytes sent via SQL*Net to client
          0  db block gets          639  bytes received via SQL*Net from client
        294  consistent gets         21  SQL*Net roundtrips to/from client
          0  physical reads           0  sorts (memory)
          0  redo size                0  sorts (disk)
                                    293  rows processed
```

Insgesamt konnte die fünfte räumliche Anfrage fast $1,29$-mal schneller beantwortet werden.

D.2.6 oQuery 6

Die sechste räumliche Testanfrage beinhaltet zwei relationale Joins, einen räumlichen Join, eine räumliche Selektion sowie eine relationale Aggregation. Bei dieser Anfrage wurde davon ausgegangen, dass kein Index auf bholz_50_geo existiert.

$$\Gamma_{\text{G2.objektid}\#\text{G2.objektid,count}(*)}$$
$$(\sigma_{\text{G1.geo EQUAL GEOMETRY[POLYGON;91;80]}}(\text{bholz_50_geo as G1}) \bowtie_{\text{G1.objektart}=\text{G3.objektart}}$$
$$((\text{bholz_25_geo as G3}) \bowtie_{\text{G2.geo CONTAINS G3.geo}}$$
$$((\text{objektarten as O}) \bowtie_{\text{O.objektart}=\text{G2.objektart}} (\text{bholz_250_geo as G2}))))$$

Der dazugehörige, leichter lesbare Operatorbaum:

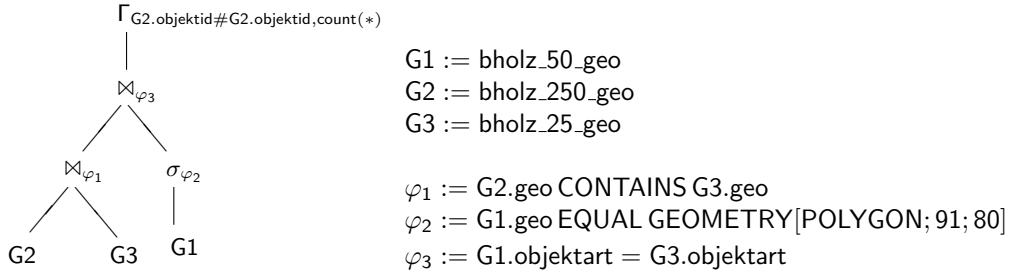


Abbildung D.12: räumliche Testanfrage 6

Nachfolgend die Anfrage in SQL:

```
select g2.objektid, count(*)
from bholz_25_geo g3, bholz_50_geo g1, bholz_250_geo g2
where SDO_RELATE(g1.geo, SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY (1, 1003, 1),
      SDO_ORDINATE_ARRAY(3560744.36, 5911605.86, 3560683.6, 5911595.48, 3560673.38, 5911592.62,3560672.23,
      5911585.61, 3560666.99, 5911536.1, 3560675.57, 5911535.3, 3560729.3, 5911529.01,3560757.56,
      5911525.86, 3560745.07, 5911599.93, 3560744.36, 5911605.86)), 'MASK=EQUAL QUERYTYPE=WINDOW')='TRUE'
and SDO_RELATE( g2.geo, g3.geo, 'MASK=CONTAINS QUERYTYPE=WINDOW') ='TRUE'
and g1.objektart=g3.objektart
group by g2.objektid;
```

```
Elapsed: 03:16:07.31

Execution Plan
----------------------------------------------------------------------------------------------
 0    SELECT STATEMENT Optimizer=ALL_ROWS(Cost=7422 Card=701 Bytes=59585)
 1   0  HASH (GROUP BY) (Cost=7422 Card=701 Bytes=59585)
 2   1    HASH JOIN (Cost=7004 Card=6459250 Bytes=549036250)
 3   2      TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO' (TABLE)(Cost=521 Card=30891 Bytes=834057)
 4   2      MERGE JOIN (CARTESIAN)(Cost=2392 Card=133309 Bytes=7731922)
 5   4        TABLE ACCESS (FULL) OF 'BHOLZ_50_GEO'(TABLE)(Cost=264 Card=190 Bytes=5700)
 6   4        BUFFER (SORT) (Cost=2128 Card=701 Bytes=19628)
 7   6          TABLE ACCESS (FULL) OF 'BHOLZ_250_GEO'(TABLE)(Cost=11 Card=701 Bytes=19628)

Statistics
----------------------------------------------------------------------------------------------
 6631489 recursive calls     72 redo size                          3304607 sorts (memory)
       6 db block gets     1262 bytes sent via SQL*Net to client         0 sorts (disk)
99446698 consistent gets    456 bytes received via SQL*Net from client      41 rows processed
     553 physical reads       4 SQL*Net roundtrips to/from client
```

Gleiche Anfrage, nur für `RELOpt` übersetzt:

```
(AGGREGATION (G2.objektid#G2.objektid,count(*) as X.partcount)
 (JOIN G1.objektart=G3.objektart
  (SELECTION G1.geo_EQUAL_GEOMETRY[POLYGON;91;80] (bholz_50_geo as G1))
   (JOIN G2.geo_CONTAINS_G3.geo (bholz_25_geo as G3) (bholz_250_geo as G2))))
```

Ergebnis der `RELOpt`-Optimierung:

```
(AGGREGATION rel (G2.OBJEKTID#G2.OBJEKTID,count(G3.OBJEKTID) AS X.PARTCOUNT)
  (SECONDARY_FILTER rel G2.GEO_CONTAINS_G3.GEO
    (JOIN rel-ind G2.GEO_CONTAINS_G3.GEO
      (JOIN hash-hash G1.OBJEKTART=G3.OBJEKTART
        (SELECTION rel G1.GEO_EQUAL_GEOMETRY[POLYGON;91.0;80.0]
          (SELECTION rel gtype[G1.GEO]=POLYGON (BHOLZ_50_GEO AS G1)))
        (BHOLZ_25_GEO AS G3))
      (BHOLZ_250_GEO AS G2))))
```

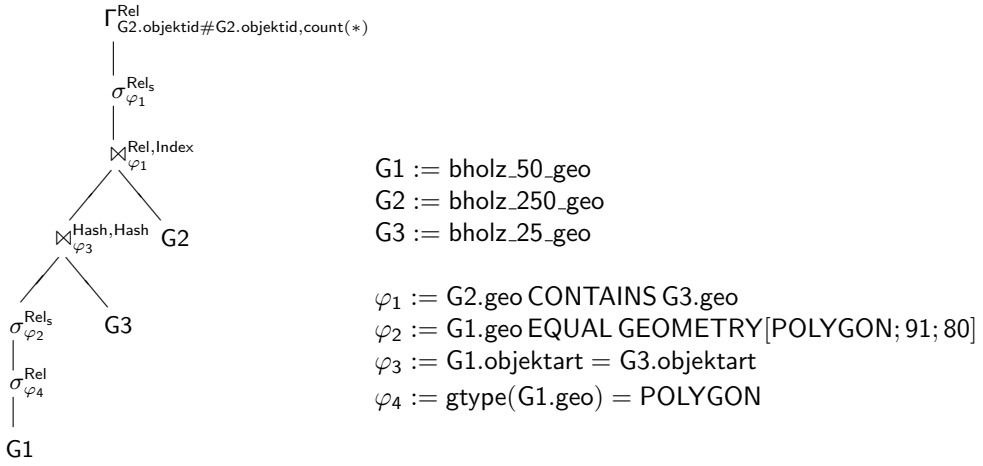


Abbildung D.13: optimierte räumliche Testanfrage 6

Diese Anfrage muss in zwei Teilanfragen zerlegt werden:

```
create materialized view tmp as
select  g1.objektid as objektid1, g1.objektart as objektart1,
        g1.geo as geo1, g1.objektnr as objektnr1,
        g3.objektid as objektid3, g3.objektart as objektart3,
        g3.geo as geo3, g3.objektnr as objektnr3
from    bholz_25_geo g3, bholz_50_geo g1
where   SDO_GEOMETRY.GET_GTYPE(g1.geo)=3
and     SDO_GEOM.RELATE(g1.geo,'EQUAL', SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1),
        SDO_ORDINATE_ARRAY(3560744.36, 5911605.86,3560683.6, 5911595.48,3560673.38, 5911592.62,
        3560672.23, 5911585.61,3560666.99, 5911536.1,3560675.57, 5911535.3, 3560729.3, 5911529.01,
        3560757.56, 5911525.86,3560745.07, 5911599.93, 3560744.36, 5911605.86)),0.05)='EQUAL'
and     g1.objektart=g3.objektart;
```

```
View: Elapsed: 00:00:03.33

Execution Plan
----------------------------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=785 Card=9214 Bytes=396202)
   1    0   HASH JOIN (Cost=785 Card=9214 Bytes=396202)
   2    1     TABLE ACCESS (FULL) OF 'BHOLZ_50_GEO'(TABLE) (Cost=264 Card=2 Bytes=60)
   3    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO'(TABLE) (Cost=521 Card=30891 Bytes=401583)

Statistics
----------------------------------------------------------------------------------------------
 5647  recursive calls        0  redo size 69225                                  0  sorts (memory)
    0  db block gets      69225  bytes sent via SQL*Net to client                 0  sorts (disk)
49839  consistent gets     3884  bytes received via SQL*Net from client       4714  rows processed
    0  physical reads       316  SQL*Net roundtrips to/from client
```

```
select
  /*+ ORDERED*/
  g2.objektid,
  count(*)
from
  tmp g13,
  bholz_250_geo g2
where
  SDO_RELATE(
    g2.geo,
    g13.geo3,
     'mask=contains
  querytype=window')
             ='TRUE'
group by objektid;
```

```
Elapsed: 00:00:46.93

Execution Plan
----------------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=19102 Card=701 Bytes=2693943)
   1    0   HASH (GROUP BY) (Cost=19102 Card=701 Bytes=2693943)
   2    1     NESTED LOOPS (Cost=9524 Card=32724 Bytes=125758332)
   3    2       MAT_VIEW ACCESS (FULL) OF 'TMP' (MAT_VIEW)
                                          (Cost=177 Card=4668 Bytes=17808420)
   4    2       TABLE ACCESS (BY INDEX ROWID) OF 'BHOLZ_250_GEO'
                                          (TABLE)(Cost=9524 Card=7 Bytes=196)
   5    4         DOMAIN INDEX OF 'BHOLZ_250_GEOM_IDX' (INDEX (DOMAIN))

Statistics
----------------------------------------------------------------------------------
     108304  recursive calls       1262  bytes sent via SQL*Net to client
       9428  db block gets          452  bytes received via SQL*Net from client
     469992  consistent gets           4  SQL*Net roundtrips to/from client
          0  physical reads         9428  sorts (memory)
          0  redo size                 0  sorts (disk)
                                      41  rows processed
```

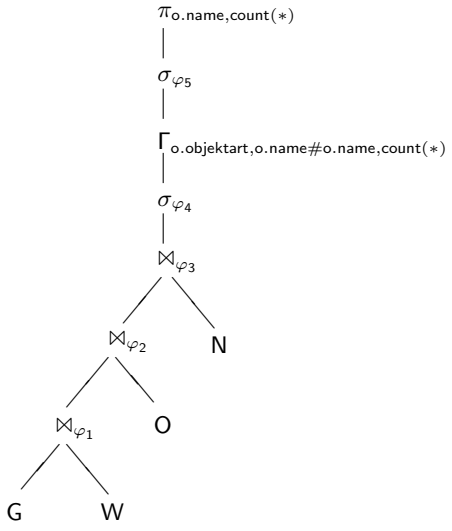
Insgesamt konnte die sechste räumliche Anfrage fast 234, 13-mal schneller beantwortet werden.

D.2.7 oQuery 7

Im Folgenden die siebte räumliche Testanfrage:

$$\pi_{\mathsf{N.name,count(W.objektid)}}$$
$$(\sigma_{\mathsf{count(W.objektid)>1}}$$
$$(\Gamma_{\mathsf{G.objektid,N.name\#N.name,count(W.objektid)}}$$
$$(\sigma_{\mathsf{G.objektart=2111 \wedge O.objektart=4107}}$$
$$((\mathsf{bholz_25_namen\ as\ N}) \bowtie_{\mathsf{N.objektnr=G.objektnr}}$$
$$((\mathsf{bholz_25_objekte\ as\ O}) \bowtie_{\mathsf{O.objektnr=W.objektnr}}$$
$$((\mathsf{bholz_25_geo\ as\ G}) \bowtie_{\mathsf{W.geo\ ANYINTERACT\ G.geo}}$$
$$(\mathsf{bholz_25_geo_f\ as\ W})))))))$$

mit dem dazugehörigen, leichter lesbaren Operatorbaum:



$W := \mathsf{bholz_25_geo_f}$
$G := \mathsf{bholz_25_geo}$
$N := \mathsf{bholz_25_namen}$
$O := \mathsf{bholz_25_objekte}$

$\varphi_1 := \mathsf{W.geo\ ANYINTERACT\ G.geo}$
$\varphi_2 := \mathsf{O.objektnr} = \mathsf{W.objektnr}$
$\varphi_3 := \mathsf{N.objektnr} = \mathsf{G.objektnr}$
$\varphi_4 := \mathsf{G.objektart} = 2111 \wedge \mathsf{O.objektart} = 4107$
$\varphi_5 := \mathsf{count(W.objektid)} > 1$

Abbildung D.14: räumliche Testanfrage 7

Nachfolgend die Anfrage in SQL:

```
select
 n.name,
 count(*)
from
 bholz_25_geo_f w,
 bholz_25_geo g,
 bholz_25_namen n,
 bholz_25_objekte o
where
 g.objektart=2111
and
 o.objektart=4107
and
 w.objektnr=o.objektnr
and
SDO_RELATE(
      g.geo,
      w.geo,
     'MASK=ANYINTERACT
     QUERYTYPE=WINDOW')
                 ='TRUE'
and
 n.objektnr=g.objektnr
group by
 g.objektid,n.name
having   count(*)>1;
```

```
Elapsed: 00:01:30.62

Execution Plan
----------------------------------------------------------------------------
 0  SELECT STATEMENT Optimizer=ALL_ROWS(Cost=11811 Card=12600 Bytes=995400)
 1  0 FILTER
 2  1   HASH (GROUP BY) (Cost=11811 Card=12600 Bytes=995400)
 3  2     HASH JOIN (Cost=10023 Card=251985 Bytes=19906815)
 4  3       TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN' (TABLE)
                                              (Cost=18 Card=23793 Bytes=309309)
 5  3       NESTED LOOPS (Cost=10002 Card=152814 Bytes=10085724)
 6  5         HASH JOIN (Cost=398 Card=4796 Bytes=187044)
 7  6           TABLE ACCESS (FULL) OF 'BHOLZ_25_OBJEKTE'(TABLE)
                                               (Cost=22 Card=4753 Bytes=57036)
 8  6           TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_F'(TABLE)
                                            (Cost=375 Card=15243 Bytes=411561)
 9  5         TABLE ACCESS (BY INDEX ROWID) OF 'BHOLZ_25_GEO'(TABLE)
                                              (Cost=10002 Card=32 Bytes=864)
10  9           DOMAIN INDEX OF 'BHOLZ_25_GEOM_IDX'(INDEX (DOMAIN))

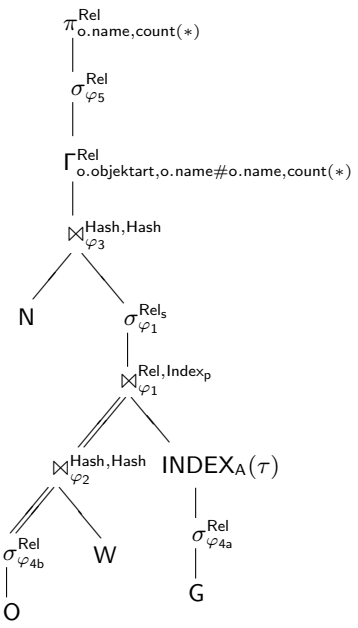
Statistics
----------------------------------------------------------------------------
    303537 recursive calls      18627 bytes sent via SQL*Net to client
      9430 db block gets         1497 bytes received via SQL*Net from client
    603220 consistent gets         99 SQL*Net roundtrips to/from client
         0 physical reads        9428 sorts (memory)
         0 redo size                0 sorts (disk)
                                 1458 rows processed
```

Gleiche Anfrage, nur für `RELOpt` übersetzt:

```
(PROJECTION (N.name, X.objectcount)
(SELECTION X.objectcount>1
 (AGGREGATION (G.objektid,N.name#N.name,(count(W.objektid)) as X.objectcount)
  (SELECTION (AND (G.objektart=2111,O.objektart=4107))
   (JOIN N.objektnr=G.objektnr (bholz_25_namen as N)
    (JOIN O.objektnr=W.objektnr (bholz_25_objekte as O)
     (JOIN W.geo_ANYINTERACT_G.geo
          (bholz_25_geo as G)(bholz_25_geo_f as W))))))))
```

Ergebnis der `RELOpt`-Optimierung:

```
(SELECTION rel X.OBJECTCOUNT>1
  (AGGREGATION rel (G.OBJEKTID,N.NAME#N.NAME,count(W.OBJEKTID) AS X.OBJECTCOUNT)
    (JOIN hash-hash N.OBJEKTNR=G.OBJEKTNR
      (BHOLZ_25_NAMEN AS N)
      (SECONDARY_FILTER rel W.GEO_ANYINTERACT_G.GEO
        (JOIN rel-ind W.GEO_ANYINTERACT_G.GEO
          (JOIN hash-hash O.OBJEKTNR=W.OBJEKTNR
            (SELECTION rel O.OBJEKTART=4107(BHOLZ_25_OBJEKTE AS O))
            (BHOLZ_25_GEO_F AS W))
          (INDEX (G.GEO)
            (SELECTION rel G.OBJEKTART=2111(BHOLZ_25_GEO AS G))))))))
```



W := bholz_25_geo_f
G := bholz_25_geo
N := bholz_25_namen
O := bholz_25_objekte

φ_1 := W.geo ANYINTERACT G.geo
φ_2 := O.objektnr = W.objektnr
φ_3 := N.objektnr = G.objektnr
φ_{4a} := G.objektart = 2111
φ_{4b} := O.objektart = 4107
φ_5 := count(W.objektid) > 1

$\tau := \sigma^{Rel}_{\varphi 4a}(G)$
A := geo

Abbildung D.15: optimierte räumliche Testanfrage 7

Um die von `RELOpt` empfohlene Reihenfolge der Operatoren in Oracle 10g umsetzen zu können, wird die Anfrage in drei Teilanfragen zerlegt.

```
create table tmp as select * from bholz_25_geo where objektart=2111;
```

```
Elapsed: 00:00:00.20

Execution Plan
----------------------------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=521 Card=3186 Bytes=50976)
   1    0   TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO' (TABLE) (Cost=521 Card=3186 Bytes=50976)

Statistics
----------------------------------------------------------------------------------------------
          1  recursive calls      55886  bytes sent via SQL*Net to client
          0  db block gets         2740  bytes received via SQL*Net from client
       2522  consistent gets        212  SQL*Net roundtrips to/from client
          0  physical reads           0  sorts (memory)
          0  redo size                0  sorts (disk)
                                   3152  rows processed
```

```
create index tmp_idx on tmp(geo) indextype is MDSYS.SPATIAL_INDEX;
```

```
Elapsed: 00:00:00.99
```

```
select
  n.name,
  count(*)
from
  bholz_25_geo_f w,
  tmp g,
  bholz_25_namen n,
  bholz_25_objekte o
where
  o.objektart=4107
and
  w.objektnr=o.objektnr
and
  SDO_RELATE(
   g.geo,
   w.geo,
   'MASK=ANYINTERACT
   QUERYTYPE=WINDOW')
              ='TRUE'
and
  n.objektnr=g.objektnr
group by
  g.objektid, n.name
having   count(*)>1;
```

```
Elapsed: 00:00:38.99

Execution Plan
----------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=10035 Card=50 Bytes=194550)
   1    0   FILTER
   2    1     HASH (GROUP BY) (Cost=10035 Card=50 Bytes=194550)
   3    2       HASH JOIN (Cost=10023 Card=218099 Bytes=848623209)
   4    3         TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN' (TABLE)
                                          (Cost=18 Card=23793 Bytes=309309)
   5    3         NESTED LOOPS (Cost=10002 Card=132264 Bytes=512919792)
   6    5           HASH JOIN (Cost=398 Card=4796 Bytes=187044)
   7    6             TABLE ACCESS (FULL) OF 'BHOLZ_25_OBJEKTE' (TABLE)
                                            (Cost=22 Card=4753 Bytes=57036)
   8    6             TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_F' (TABLE)
                                          (Cost=375 Card=15243 Bytes=411561)
   9    5           TABLE ACCESS (BY INDEX ROWID) OF 'TMP' (TABLE)
                                          (Cost=10002 Card=28 Bytes=107492)
  10    9             DOMAIN INDEX OF 'TMP_IDX' (INDEX (DOMAIN))

Statistics
----------------------------------------------------------------------------
      89541  recursive calls      18624  bytes sent via SQL*Net to client
       9428  db block gets         1497  bytes received via SQL*Net from client
     370702  consistent gets         99  SQL*Net roundtrips to/from client
          0  physical reads        9449  sorts (memory)
        116  redo size                0  sorts (disk)
                                    1458  rows processed
```

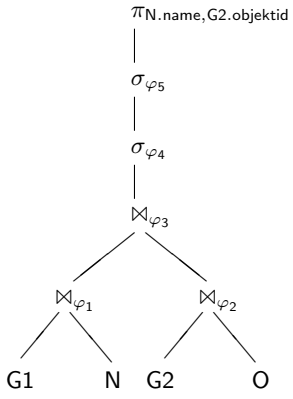
Insgesamt konnte die siebte räumliche Anfrage fast 2, 26-mal schneller beantwortet werden.

D.2.8 oQuery 8

Im Folgenden die achte räumliche Testanfrage bestehend aus zwei relationalen Joins, einem räumlichen Join, sowie zwei räumlichen und zwei relationalen Selektionen:

$$\pi_{\text{N.name,O.objektart}}$$
$$(\sigma_{\text{G1.geo ANYINTERACT GEOMETRY[POLYGON;7622;6568]} \wedge \text{G2.geo ANYINTERACT GEOMETRY[POLYGON;7622;6568]}}$$
$$(\sigma_{\text{O.objektart=3101} \wedge \text{N.name='NNNN'}}$$
$$(((\text{bholz_25_geo_f as G1}) \bowtie_{\text{G1.objektnr=N.Objektnr}} (\text{bholz_25_namen as N}))$$
$$\bowtie_{\text{G1.geo OVERLAPBDYDISJOINT G2.geo}}$$
$$((\text{bholz_25_geo_l as G2}) \bowtie_{\text{G2.objektnr=O.objektnr}} (\text{bholz_25_objekte as O})))))$$

Der dazugehörige, leichter lesbare Operatorbaum:



$\text{G1} := \text{bholz_25_geo_f}$
$\text{G2} := \text{bholz_25_geo_l}$
$\text{N} := \text{bholz_25_namen}$
$\text{O} := \text{bholz_25_objekte}$

$\varphi_1 := \text{G1.objektnr} = \text{N.Objektnr}$
$\varphi_2 := \text{G2.objektnr} = \text{O.objektnr}$
$\varphi_3 := \text{G1.geo OVERLAPBDYDISJOINT G2.geo}$
$\varphi_4 := \text{O.objektart} = 3101 \wedge \text{N.name} = \text{'NNNN'}$
$\varphi_5 := \text{G1.geo ANYINTERACT GEOMETRY[POLYGON; 7622; 6568]} \wedge$
$\text{G2.geo ANYINTERACT GEOMETRY[POLYGON; 7622; 6568]}$

Abbildung D.16: räumliche Testanfrage 8

sowie die Anfrage in SQL:

```
select g1.objektid, g2.objektid
from   bholz_25_geo_f g1,
       bholz_25_namen n,
       bholz_25_geo_l g2,
       bholz_25_objekte o
where  g1.objektnr=n.objektnr
and    SDO_GEOM.RELATE(g1.geo,
                'OVERLAPBDYDISJOINT',
                  g2.geo,0.05)
               ='OVERLAPBDYDISJOINT'
and    o.objektnr=g2.objektnr
and    o.objektart=3101
and    n.name='NNNN'
and    SDO_GEOM.RELATE(g1.geo,
                    'ANYINTERACT',
        SDO_GEOMETRY(2003,NULL,NULL,
           SDO_ELEM_INFO_ARRAY(1,3,3),
           SDO_ORDINATE_ARRAY(
                     3546445.2265625,
                    5898233.66224649,
                    3554067.266796875,
                  5904801.7090483615)),
                          0.05)='TRUE'
and    SDO_GEOM.RELATE(g2.geo,
                    'ANYINTERACT',
       SDO_GEOMETRY(2003,NULL,NULL,
         SDO_ELEM_INFO_ARRAY(1,3,3),
         SDO_ORDINATE_ARRAY(
                    3546445.2265625,
                   5898233.66224649,
                  3554067.266796875,
                5904801.7090483615)),
                         0.05)='TRUE';
```

```
Elapsed: 00:00:34.06

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS
                                (Cost=1054 Card=36 Bytes=2844)
   1    0   HASH JOIN (Cost=1054 Card=36 Bytes=2844)
   2    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_L'(TABLE)
                                (Cost=152 Card=153 Bytes=4131)
   3    1     MERGE JOIN (CARTESIAN)
                           (Cost=902 Card=74116 Bytes=3854032)
   4    3       HASH JOIN (Cost=397 Card=24 Bytes=960)
   5    4         TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN'
                           (TABLE) (Cost=18 Card=24 Bytes=312)
   6    4         TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_F'
                          (TABLE) (Cost=378 Card=152 Bytes=4104)
   7    3       BUFFER (SORT) (Cost=524 Card=3124 Bytes=37488)
   8    7         TABLE ACCESS (FULL) OF 'BHOLZ_25_OBJEKTE'
                         (TABLE) (Cost=21 Card=3124 Bytes=37488)

Statistics
----------------------------------------------------------
      62182  recursive calls
          0  db block gets
     786511  consistent gets
          0  physical reads
          0  redo size
        574  bytes sent via SQL*Net to client
        430  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          6  rows processed
```

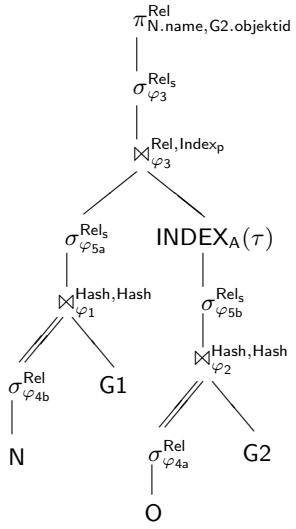
Die gleiche Anfrage nach der Übersetzung für `RELOpt`:

```
(PROJECTION (N.name,O.objektart)
 (SELECTION G1.geo_ANYINTERACT_GEOMETRY[POLYGON;7622;6568]
  (SELECTION G2.geo_ANYINTERACT_GEOMETRY[POLYGON;7622;6568]
   (SELECTION (AND (O.objektart=3101,N.name='NNNN'))
    (JOIN G1.geo_OVERLAPBDYDISJOINT_G2.geo
     (JOIN G1.objektnr=N.Objektnr(bholz_25_geo_f as G1)(bholz_25_namen as N))
     (JOIN G2.objektnr=O.objektnr(bholz_25_geo_l as G2)(bholz_25_objekte as O)))))))
```

Ergebnis der Optimierung durch `RELOpt`:

```
(PROJECTION rel (N.NAME,O.OBJEKTART)
  (SECONDARY_FILTER rel G1.GEO_OVERLAPBDYDISJOINT_G2.GEO
    (JOIN rel-ind G1.GEO_OVERLAPBDYDISJOINT_G2.GEO
      (SELECTION rel G1.GEO_ANYINTERACT_GEOMETRY[POLYGON;7622.0;6568.0]
        (JOIN hash-hash G1.OBJEKTNR=N.OBJEKTNR
          (SELECTION rel N.NAME='NNNN'
            (BHOLZ_25_NAMEN AS N))
          (BHOLZ_25_GEO_F AS G1)))
      (INDEX (G2.geo)
        (SELECTION rel G2.GEO_ANYINTERACT_GEOMETRY[POLYGON;7622.0;6568.0]
          (JOIN hash-hash G2.OBJEKTNR=O.OBJEKTNR
            (SELECTION rel O.OBJEKTART=3101
              (BHOLZ_25_OBJEKTE AS O))
            (BHOLZ_25_GEO_L AS G2)))))))
```

und der entsprechende Operatorbaum:



G1 := bholz_25_geo_f
G2 := bholz_25_geo_l
N := bholz_25_namen
O := bholz_25_objekte

φ_1 := G1.objektnr = N.Objektnr
φ_2 := G2.objektnr = O.objektnr
φ_3 := G1.geo OVERLAPBDYDISJOINT G2.geo
φ_{4a} := O.objektart = 3101
φ_{4b} := N.name = 'NNNN'
φ_{5a} := G1.geo ANYINTERACT GEOMETRY[POLYGON; 7622; 6568]
φ_{5b} := G2.geo ANYINTERACT GEOMETRY[POLYGON; 7622; 6568]

$\tau := \sigma^{Rel_s}_{\varphi 5b}\left(\sigma^{Rel}_{\varphi 4a}(O) \bowtie^{Hash,Hash}_{\varphi 2} G2\right)$
A := geo

Abbildung D.17: optimierte räumliche Testanfrage 8

Wir müssen diese Anfrage in fünf Teilanfragen zerlegen, um die von `RELOpt` empfohlene Reihenfolge der Operatoren in Oracle 10g umsetzen zu können.

```
create materialized view
   tmp2 as
select
  g2.objektid as id_l,
  g2.objektnr as nr_l,
  g2.geo as geo_l,
  g2.objectid as oid_l,
  o.objektart as art_o,
  o.objektnr as nr_o,
  o.ebene as ebene_o
from
  bholz_25_geo_l g2,
  bholz_25_objekte o
where
  o.objektnr=g2.objektnr
and o.objektart=3101;
```

```
Elapsed: 00:00:00.79

Execution Plan
----------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=172 Card=5142 Bytes=159402)
   1    0   HASH JOIN (Cost=172 Card=5142 Bytes=159402)
   2    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_OBJEKTE'(TABLE)
                                          (Cost=22 Card=3124 Bytes=37488)
   3    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_L'(TABLE)
                                       (Cost=148 Card=15285 Bytes=290415)

Statistics
----------------------------------------------------------------------------
         1  recursive calls     136010  bytes sent via SQL*Net to client
         0  db block gets         6073  bytes received via SQL*Net from client
      1240  consistent gets        515  SQL*Net roundtrips to/from client
         0  physical reads           0  sorts (memory)
         0  redo size                0  sorts (disk)
                                  7708  rows processed
```

```
create materialized view
   tmp1 as
select
   g1.objektid as id_f,
   g1.objektnr as nr_f,
   g1.geo as geo_f,
   g1.objectid as oid_f,
   n.objektnr as nr_n,
   n.typ,
   n.name
from
   bholz_25_geo_f g1,
   bholz_25_namen n
where
   g1.objektnr=n.objektnr
and n.name='NNNN';
```

```
Elapsed: 00:00:00.19

Execution Plan
----------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS(Cost=394 Card=24 Bytes=768)
   1    0   HASH JOIN (Cost=394 Card=24 Bytes=768)
   2    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_NAMEN'(TABLE)
                                          (Cost=18 Card=24 Bytes=312)
   3    1     TABLE ACCESS (FULL) OF 'BHOLZ_25_GEO_F'(TABLE)
                                       (Cost=375 Card=15243 Bytes=289617)

Statistics
----------------------------------------------------------------------------
         1 recursive calls      14510 bytes sent via SQL*Net to client
         0 db block gets         1013 bytes received via SQL*Net from client
      1791 consistent gets         55 SQL*Net roundtrips to/from client
         0 physical reads           0 sorts (memory)
         0 redo size                0 sorts (disk)
                                  803 rows processed
```

```
create table tmp4 as
select *
from   tmp2 g2
where
   SDO_GEOM_RELATE(
    g2.geo_l,
    'ANYINTERACT',
   SDO_GEOMETRY(2003,NULL,NULL,
   SDO_ELEM_INFO_ARRAY(1,3,3),
   SDO_ORDINATE_ARRAY(
               3546445.2265625,
              5898233.66224649,
             3554067.266796875,
          5904801.7090483615)),
                  0.05)='TRUE';
```

```
Execution Plan
----------------------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS
                                          (Cost=72 Card=84 Bytes=321048)
   1    0   MAT_VIEW ACCESS (FULL) OF 'TMP2' (MAT_VIEW)
                                          (Cost=72 Card=84 Bytes=321048)

Statistics
----------------------------------------------------------------------------
 7775  recursive calls    10624 bytes sent via SQL*Net to client
    0  db block gets        848 bytes received via SQL*Net from client
62236  consistent gets       40 SQL*Net roundtrips to/from client
  298  physical reads         0 sorts (memory)
    0  redo size              0 sorts (disk)
                            584 rows processed
```

```
create index tmp_idx on tmp4(geo_l) indextype is MDSYS.SPATIAL_INDEX;
```

```
Elapsed: 00:00:02.29
```

```
create materialized view
 tmp3 as
select *
from   tmp1 g1
where
  SDO_GEOM.RELATE(
   g1.geo_f,
   'ANYINTERACT',
   SDO_GEOMETRY(2003,NULL,NULL,
    SDO_ELEM_INFO_ARRAY(1,3,3),
    SDO_ORDINATE_ARRAY(
               3546445.2265625,
              5898233.66224649,
             3554067.266796875,
          5904801.7090483615))
                ,0.05)='TRUE';
```

```
Elapsed: 00:00:00.36

Execution Plan
----------------------------------------------------------------------------
 0     SELECT STATEMENT Optimizer=ALL_ROWS (Cost=25 Card=8 Bytes=30576)
 1   0   MAT_VIEW ACCESS (FULL) OF 'TMP1' (MAT_VIEW)
                                          (Cost=25 Card=8 Bytes=30576)

Statistics
----------------------------------------------------------------------------
  870 recursive calls     1324 bytes sent via SQL*Net to client
    0 db block gets        463 bytes received via SQL*Net from client
 6795 consistent gets        5 SQL*Net roundtrips to/from client
  110 physical reads         0 sorts (memory)
    0 redo size              0 sorts (disk)
                            54 rows processed
```

```
select
  g1.id_f,
  g2.id_l
from
  tmp3 G1,
  tmp4 G2
where
  SDO_RELATE(
   G2.geo_l,
   G1.geo_f,
   'MASK=
 OVERLAPBDYDISJOINT
 QUERYTYPE=WINDOW')
='TRUE';
```

```
Elapsed: 00:00:00.93

Execution Plan
----------------------------------------------------------------------------------------
0   SELECT STATEMENT Optimizer=ALL_ROWS(Cost=112 Card=315 Bytes=2411640)
1  0  NESTED LOOPS (Cost=112 Card=315 Bytes=2411640)
2  1    MAT_VIEW ACCESS (FULL) OF 'TMP3' (MAT_VIEW)(Cost=4 Card=54 Bytes=206388)
3  1    TABLE ACCESS (BY INDEX ROWID) OF 'TMP4'(TABLE)(Cost=112 Card=6 Bytes=23004)
4  3      DOMAIN INDEX OF 'TMP_IDX' (INDEX (DOMAIN))

Statistics
----------------------------------------------------------------------------------------
       1759  recursive calls        534  bytes sent via SQL*Net to client
        108  db block gets          430  bytes received via SQL*Net from client
       6590  consistent gets          2  SQL*Net roundtrips to/from client
         11  physical reads         122  sorts (memory)
          0  redo size                0  sorts (disk)
                                      6  rows processed
```

Insgesamt konnte die achte räumliche Anfrage fast 7, 47-mal schneller beantwortet werden.

D.2.9 Ergebnisse der räumlichen Anfragen

Tabelle D.4: Räumliche Anfragen und Testergebnisse

Anfrage	Gesamtkosten	$\mathsf{cost}_{\mathsf{I/O}}$	$\mathsf{cost}_{\mathsf{CPU}}$	Anz. Varianten	Iterationen	Grenzen dyn./stat.	Optimierungsdauer
oQ1	129539	109815	986222	1	-	0.01/1	-
	129539	109815	986222	2	7	1/5	198ms
	129539	109815	986222	2	7	1/10	198ms
	129539	109815	986222	3	8	1.1/5	241ms
	129539	109815	986222	3	8	1.1/10	241ms
	129539	109815	986222	31	11	$\log_{10}/5$	1255 ms
	129539	109815	986222	34	11	$\log_{10}/10$	1536ms
	129539	109815	986222	33	11	$\log_e/5$	1451ms
	129539	109815	986222	37	12	$\log_e/10$	1813ms
oQ2	5130632283	4518204430	30621392558	1	-	0.01/1	-
	62836990	58695289	207085046	23	14	1/5	1156ms
	62836990	58695289	207085046	23	14	1/10	1156ms
	62836990	58695289	207085046	232	21	1.1/5	15764ms
	62836990	58695289	207085046	359	21	1.1/10	36215ms
	62836990	58695289	207085046	247	35	$\log_{10}/1$	72763ms
	62836990	58695289	207085046	372	35	$\log_e/1$	123440ms
oQ3	93398371	79421360	698850549	1	-	0.01/1	-
	1836984	1789176	2390476	24	13	1/5	1217ms
	1836984	1789176	2390476	24	13	1/10	1340ms
	1828507	1782618	2294542	445	27	1.1/5	60319ms
	1828507	1782618	2294542	755	27	1.1/10	90579ms
	1830030	1789009	2051017	359	38	$\log_{10}/2$	446122ms
	1830030	1789009	2051017	465	38	$\log_e/2$	446575ms
oQ4	88870358	70501291	918453395	1	-	0.01/1	-
	100550	81776	938708	18	14	1/5	293ms
	100550	81776	938708	18	14	1/10	293ms
	100550	81776	938708	94	18	1.1/5	7862ms
	100550	81776	938708	151	19	1.1/10	13474ms
	100550	81776	938708	195	21	$\log_{10}/5$	15611ms
	100550	81776	938708	446	24	$\log_{10}/10$	63297ms
	100550	81776	938708	263	24	$\log_e/5$	134226ms
	100550	81776	938708	463	26	$\log_e/9$	577184ms

Anfrage	Gesamtkosten	$\texttt{cost}_{\texttt{I/O}}$	$\texttt{cost}_{\texttt{CPU}}$	Anz. Varianten	Iterationen	Grenzen dyn./stat.	Optimierungsdauer
oQ5	97571780	69514369	1402870498	1	-	0.01/1	-
	452152	376571	3779019	11	11	1/5	898ms
	452152	376571	3779019	11	11	1/10	898ms
	152189	126869	1265966	166	19	1.1/5	88178ms
	152189	126869	1265966	216	21	1.1/8	211007ms
	152189	126869	1265966	353	26	$\log_{10}/5$	99833ms
	152189	126869	1265966	488	28	$\log_{10}/7$	218107ms
	152189	126869	1265966	276	23	$\log_e/3$	446116ms
oQ6	97943979	77749829	1009707559	1	-	0.01/1	-
	301451	289453	599873	6	9	1/5	255ms
	301451	289453	599873	6	9	1/10	255ms
	301451	289453	599873	17	11	1.1/5	529ms
	301451	289453	599873	17	11	1.1/10	420ms
	301451	289453	599873	75	14	$\log_{10}/5$	1921ms
	301451	289453	599873	90	14	$\log_{10}/10$	26234ms
	301451	289453	599873	92	14	$\log_e/5$	1575ms
	301451	289453	599873	123	14	$\log_e/10$	2082ms
Q7	2607657650	2067192346	27023265169	1	-	0.01/1	-
	16464	12370	204629	20	13	1/5	813ms
	16464	12370	204629	20	13	1/10	813ms
	16464	12370	204629	164	19	1.1/5	23059ms
	16464	12370	204629	243	18	1.1/10	53031ms
	16464	12370	204629	435	24	$\log_{10}/5$	25115ms
	16464	12370	204629	729	24	$\log_{10}/10$	78915ms
	16464	12370	204629	435	24	$\log_e/5$	26836ms
	16464	12370	204629	729	24	$\log_e/10$	88491ms
oQ8	2997419078	2552817293	22230089221	1	-	0.01/1	-
	5935	3511	121179	31	15	1/5	2645ms
	5935	3511	121179	31	15	1/10	3311ms
	5935	3511	121179	81	17	1.1/5	11470ms
	5935	3511	121179	102	17	1.1/10	13710ms
	5935	3511	121179	381	19	$\log_{10}/5$	717633ms
	5935	3511	121179	331	22	$\log_e/2$	101309ms

Anfrage	Ausführungszeit		Teilanfrage						Verhältnis der
	Oracle 10g	RELOpt	I	II	III	IV	V	VI	Ausführungszeiten
oQ1	**00:00:16.40**	**00:00:09.16**	00:00:03.97	00:00:05.19	-	-	-	-	$5.5853659 \cdot 10^{-2}$
oQ2	**>53:08:19.49**	**00:02:11.21**	00:00:04.28	00:00:00.67	00:02:06.24	00:00:00.02	-	-	$< 6.8588787 \cdot 10^{-4}$
oQ3	**02:27:01.58**	**00:00:04.64**	00:00:00.12	00:00:03.18	00:00:01.31	00:00:00.03	-	-	$4.6703652 \cdot 10^{-4}$
oQ4	**00:00:37.76**	**00:00:11.85**	00:00:00.26	00:00:01.05	00:00:10.52	00:00:00.02	-	-	$3.1382415 \cdot 10^{-1}$
oQ5	**00:00:9.27**	**00:00:07.17**	00:00:00.08	00:00:05.23	00:00:00.40	00:00:00.57	00:00:00.14	00:00:00.03	$7.7346278 \cdot 10^{-1}$
oQ6	**03:16:07.31**	**00:00:50.26**	00:00:03.33	00:00:46.93		-	-	-	$4.2711546 \cdot 10^{-3}$
oQ7	**00:01:30.62**	**00:00:40.18**	00:00:00.20	00:00:00.99	00:00:38.99	-	-	-	$4.4338998 \cdot 10^{-1}$
oQ8	**00:00:34.06**	**00:00:04.56**	00:00:00.79	00:00:00.19	00:00:02.29	00:00:00.36	00:00:00.93	-	$1.3388139 \cdot 10^{-1}$

Tabelle D.5: Vergleich von Laufzeiten von optimierten räumlichen Anfragen zwischen RELOpt und Oracle 10g

Abbildungsverzeichnis

1.1 Ablauf der Anfragebearbeitung . . . 1
1.2 Dreiphasenoptimierung . . . 3
1.3 Einphasenoptimierung . . . 4
1.4 Wachstum der Variantenanzahl . . . 4
1.5 Mittel zur Erreichung der Dissertationsziele . . . 6

2.1 Funktionsweise eines Kostenmodells . . . 11
2.2 Beispiel für einen Anfragebaum . . . 13
2.3 Beispiel für einen Ausführungsplan . . . 13
2.4 intel® CPU-MIPS-Leistung . . . 16
2.5 Kombinatorische Formel (†) zur Abschätzung von Attributwertanzahlen . . . 19
2.6 Beispiel für einen Anfragebaum mit Pipelining . . . 27
2.7 Beispiel für einen generalisierten Baum . . . 32
2.8 TPC-H Query 4 Ausführungsplan . . . 33

3.1 Beispiel für einen stark selektierenden Verbund . . . 46

4.1 Räumliches Datenmodell . . . 60
4.2 Beispiel für einen objektrelationalen Anfragebaum . . . 61
4.3 Beispiel für eine Fensteranfrage . . . 62
4.4 Geometrienmodell . . . 64
4.5 Topologisches Beziehungsbeispiel zweier Geometrien . . . 65
4.6 Anfragemodell von räumlichen Datenbanken . . . 68
4.7 Visuelle Darstellung der räumlichen Beispielanfrage mit Hilfe von GisVisual 7.0[5] 76

5.1 Deaktivierte (□) bzw. aktivierte (◯) Varianten . . . 79
5.2 Variantenerzeugung . . . 81
5.3 Mehrmalige Nutzung einer Sortierung vs. einmaliger Indexnutzung . . . 82
5.4 Dynamische Kostenentwicklung . . . 82
5.5 Mögliche statische Variantengrenze . . . 83
5.6 Mögliche dynamische Variantengrenze . . . 83
5.7 Module des Termersetzungsoptimierungssimulators . . . 84
5.8 Der Suchraum des Termersetzers . . . 86

6.1 Beispiel eines Entstehungsgraphen - Hill-Climbing Strategie . . . 89
6.2 Beispiel eines Entstehungsgraphen – Simulated-Annealing Strategie . . . 90
6.3 Optimierungspotential der TPC-H Version 2.3.0 Anfragen . . . 93
6.4 Optimierungspotential räumlicher Anfragen . . . 96
6.5 Dauer der Testanfragen jeweils mit Oracle 10g und `RELOpt` optimiert . . . 97
6.6 Kommutativitätsregel F) mit erweiterten Bedingungen . . . 98
6.7 Regel H) mit Selektivitätsbedingungen . . . 99
6.8 Regel B) . . . 100

7.1 Paketstruktur von `RELOpt` . . . 104
7.2 Package *dbs.unihannover.sopt.struc.general* . . . 107
7.3 Package *dbs.unihannover.sopt.struc.general.tree* . . . 111
7.4 Sequenzablaufdiagramm der Einphasenoptimierung . . . 113
7.5 Komponentendiagramm der Einphasenoptimierung . . . 114

A.1 IO-Meter Festplatten . . . 117
A.2 CPU-MIPS Tabelle I . . . 117
A.3 CPU-MIPS Tabelle II . . . 118

D.1 TPC-H Version 2.3.0 Query 5 . . . 128
D.2 räumliche Testanfrage 1 . . . 139
D.3 optimierte räumliche Testanfrage 1 . . . 140
D.4 räumliche Testanfrage 2 . . . 141
D.5 optimierte räumliche Testanfrage 2 . . . 142
D.6 räumliche Testanfrage 3 . . . 144
D.7 optimierte räumliche Testanfrage 3 . . . 145
D.8 räumliche Testanfrage 4 . . . 147
D.9 optimierte räumliche Testanfrage 4 . . . 148
D.10 räumliche Testanfrage 5 . . . 150
D.11 optimierte räumliche Testanfrage 5 . . . 151
D.12 räumliche Testanfrage 6 . . . 153
D.13 optimierte räumliche Testanfrage 6 . . . 154
D.14 räumliche Testanfrage 7 . . . 155
D.15 optimierte räumliche Testanfrage 7 . . . 157
D.16 räumliche Testanfrage 8 . . . 158
D.17 optimierte räumliche Testanfrage 8 . . . 160

Tabellenverzeichnis

1.1 Vergleich von `RELOpt` mit bisherigen Optimierungsansätzen. 9

2.1 Darstellungen für Mengenoperatoren 17
2.2 Selektivität von Prädikaten 18
2.3 Kardinalitäten 18
2.4 Schätzungen für Attributwertanzahlen einer Selektion 19
2.5 Schätzungen für Attributwertanzahlen 20
2.6 Abschätzungen von Tupellängen 21
2.7 Selektionskosten 22
2.8 Projektionskosten 22
2.9 Kosten von Joins 23
2.10 Kosten eines Antisemi- bzw. Semijoins 25
2.11 Kosten einer Aggregation ohne Gruppierung 25
2.12 Kosten einer Aggregation mit Gruppierung 26
2.13 Kosten der Erzeugung von Indexen und Sortierungen 26
2.14 Pipelining bei Projektionen und Selektionen 27
2.15 Pipelining bei verschiedenen Joinoperationen 28
2.16 Pipelining bei einer Aggregation 28
2.17 Pipelining bei Indexen und Sortierungen 29
2.18 Vererbung von Sortierungen bei Selektion und Projektion 29
2.19 Vererbung von Sortierungen beim Join 30
2.20 Vererbung von Sortierungen beim Semijoin 30
2.21 Vererbung von Sortierungen beim Left-Outer-Join 31
2.22 Vererbung von Sortierungen bei Aggregationen 31

4.1 Statistiken zur räumlichen Beispielanfrage 62
4.2 Geometrietypen 64
4.3 Topologische Beziehungsoperatoren 66
4.4 Relationale Repräsentanten räumlicher Operatoren 69
4.5 Attributwertverteilungen bei geometrischen Attributen 70
4.6 Standardwerte für die topologische Funktion **EQUAL** 72
4.7 Topologische Beziehungsoperatoren 72
4.8 Selektionskosten 73
4.9 Kosten von Joins 73

6.1 Optimierungsstrategien 91
6.2 Anzahl der vorkommenden Joins mit den dabei verwendeten Basisrelationen in den TPC-H Version 2.3.0 Anfragen 92
6.3 Anzahl der vorkommenden Joins in den TPC-H Version 2.3.0 Anfragen 92
6.4 Werte der Variablen 94
6.5 Verteilung der Operatoren in den Anfragebäumen 95
6.6 Verteilung der physischen Operatoren in den Ausführungsplänen I 95

6.7 Verteilung der physischen Operatoren in den Ausführungsplänen II 95
6.8 Vergleich der Ergebnisse auf räumlichen Daten 96
6.9 heuristische algebraische Regelanordnung 99

B.2 Vergleich der Join-Ordering-Algorithmen 122

C.1 Standardwerte für die topologische Funktion **OVERLAPBDYDISJOINT** . . 123
C.2 Standardwerte für die topologische Funktion **OVERLAPBDYINTERSECT** . 123
C.3 Standardwerte für die topologische Funktion **EQUAL** 124
C.4 Standardwerte für die topologische Funktion **INSIDE** 124
C.5 Standardwerte für die topologische Funktion **TOUCH** 124
C.6 Standardwerte für die topologische Funktion **DISJOINT** 124
C.7 Standardwerte für die topologische Funktion **CONTAINS** 124
C.8 Standardwerte für die topologische Funktion **ON** 125
C.9 Standardwerte für die topologische Funktion **COVERS** 125
C.10 Standardwerte für die topologische Funktion **COVEREDBY** 125
C.11 Standardwerte für die topologische Funktion **DETERMINE** 125
C.12 Standardwerte für die topologische Funktion **ANYINTERACT** 125

D.1 Spezifikation des Datenbankrechners 127
D.2 Relationale Anfragen und Testergebnisse 130
D.3 TPC-H Version 2.3.0 Benchmarkergebnisse 136
D.4 Räumliche Anfragen und Testergebnisse 162
D.5 Vergleich von Laufzeiten von optimierten räumlichen Anfragen zwischen `RELOpt` und Oracle 10g 164

Literaturverzeichnis

[1] Ashraf Aboulnaga and Jeffrey F. Naughton. Accurate Estimation of the Cost of Spatial Selections. In *ICDE Conference*, pages 123–134, San Diego, California, USA, 2000.

[2] Swarup Acharya, Viswanath Poosala, and Sridhar Ramaswamy. Selectivity Estimation in Spatial Databases. In *SIGMOD Conference*, pages 13–24, Philadelphia, Pennsylvania, USA, 1999. ACM Press.

[3] Nabil R. Adam and Aryya Gangopadhyay. *Database Issues in Geographic Information Systems*. Kluwer Academic Publishers, 1997.

[4] Walid G. Aref and Hanan Samet. Optimization for Spatial Query Processing. In *VLDB Conference*, pages 81–90. Morgan Kaufmann Publishers Inc., 1991.

[5] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD Conference*, pages 261–272, Dallas, Texas, USA, 2000.

[6] Ludger Becker and Ralf Hartmut Güting. Rule-based optimization and query processing in an extensible geometric database system. *ACM Transaction Database System*, 17(2):247–303, 1992.

[7] Gerhard Belina, Manfred Mittlböck, and Bernhard Zagel. Anbindung von ArcGIS und GeoMedia an Oracle Spatial. In *UniGIS Update Konferenz*, Salzburg, Austria, 2005.

[8] Arthur J. Bernstein, Michael Kifer, and Philip M. Lewis. *Database Systems An Application-Oriented Approach*. Addison Wesley, 2005.

[9] José A. Blakeley, William J. McKenna, and Goetz Graefe. Experiences Building the Open OODB Query Optimizer. In *SIGMOD Conference*, pages 287–296, Washington, D.C., USA, 1993.

[10] Reinhard Bündgen. *Termersetzungssysteme. Theorie, Implementierung, Anwendung*. Viweg Verlag, 1998.

[11] Stefano Ceri and Giuseppe Pelagatti. *Distributed Databases: Principles and Systems*. McGraw-Hill Book Company, 1984.

[12] Surajit Chaudhuri and Kyuseok Shim. Including Group-by in Query Optimization. In *VLDB Conference*, pages 354–366, Santiago, Chile, 1994.

[13] Mitch Cherniack and Stanley B. Zdonik. Changing the Rules: Transformations for Rule-Based Optimizers. In *SIGMOD Conference*, pages 61–72, Seattle, Washington, USA, 1998.

[14] Richard L. Cole and Goetz Graefe. Optimization of Dynamic Query Evaluation Plans. In *SIGMOD Conference*, pages 150–160, Minneapolis, Minnesota, USA, 1994.

[15] Thomas M. Connolly and Carolyn E. Begg. *Database Systems*. Addison Wesley, 2005.

[16] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Algorithmen - Eine Einführung*. Oldenbourg Verlag, 2004.

[17] Nachum Dershowitz. *Rewriting Techniques and Applications*. Springer, Berlin, 1998.

[18] Max J. Egenhofer and Jayant Sharma. Topological Relations Between Regions in R and Z. In *SSD Conference*, pages 316–336, Singapore, China, 1993.

[19] Ramiz Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 2003.

[20] Alexander Fabig. Entwicklung eines Parsers für relationale Anfragen und einer Verwaltung von Ausführungsplänen für den Anfragesimulator `RELOpt`. Bachelor's thesis, University of Hannover, Hannover, Germany, 2005.

[21] Miguel Rodrigues Fornari and Cirano Iochpe. A spatial hash join algorithm suited for small buffer size. In *GIS Conference*, pages 118–126, Washington DC, USA, 2004. ACM Press.

[22] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[23] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer D. Widom. *Database Systems. The Complete Book*. Prentice Hall, 2003.

[24] Tim Gorman. The search for intelligent life in the cost-based optimizer. *SageLogix Incoporation*, 2001.

[25] Goetz Graefe. Software Modularization with the EXODUS Optimizer Generator. *IEEE Database Engineering Bulletin*, 9(4):37–45, 1986.

[26] Goetz Graefe. *Rule-Based Query Optimization in Extensible Database Systems*. PhD thesis, University of Wisconsin, USA, 1987.

[27] Goetz Graefe and David J. DeWitt. The EXODUS Optimizer Generator. In *SIGMOD Conference*, pages 160–172, San Francisco, CA, USA, 1987.

[28] Goetz Graefe and David Maier. Query Optimization in Object-Oriented Database Systems: A prospectus. In *OODBS Conference*, pages 358–363, Bad Mnster am Stein-Ebernburg, Germany, 1988.

[29] Goetz Graefe and William J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *ICDE Conference*, pages 209–218, Vienna, Austria, 1993.

[30] Torsten Grust, Maurice van Keulen, and Jens Teubner. Staircase join: Teach a relational dbms to watch its (axis) steps. In *VLDB Conference*, pages 524–525, Berlin, Germany, 2003.

[31] Peter Gulutzan and Trudy Pelzer. *SQL Performance Tuning*. Addison-Wesley Professional, 2002.

[32] Laura M. Haas, Johann Christoph Freytag, Guy M. Lohman, and Hamid Pirahesh. Extensible Query Processing in Starburst. In *SIGMOD Conference*, pages 377–388, Portland, Oregon, USA, 1989.

[33] Theo Härder and Erhard Rahm. *Datenbanksysteme. Konzepte und Techniken der Implementierung*. Springer, Berlin, 2001.

[34] Andreas Heuer and Joachim Kröger. Query Optimization in the CROQUE Project. In *DEXA Conference*, pages 489–499, Zürich, Switzerland, 1996. Springer-Verlag.

[35] Matthias Jarke and Jürgen Koch. Query optimization in database systems. *ACM Computer Survey*, 16(2):111–152, 1984.

[36] Navin Kabra. *Query optimization for object-relational database systems.* PhD thesis, University of Wisconsin, USA, 1999.

[37] Navin Kabra and David J. DeWitt. Opt++: An object-oriented implementation for extensible database query optimization. *VLDB Journal*, 8(1):55–78, 1999.

[38] Alfons Kemper and André Eickler. *Datenbanksysteme - Eine Einführung.* Oldenbourg, 2004.

[39] Alfons Kemper and Guido Moerkotte. Advanced Query Processing in Object Bases Using Access Support Relations. In *VLDB Conference*, pages 290–301, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[40] Werner Kießling. Foundations of preferences in database systems. In *VLDB Conference*, pages 311–322, Hong Kong, China, 2002.

[41] Won Kim. On Optimizing an SQL-like Nested Query. *ACM Transactions on Database Systems*, 7(3):443–469, 1982.

[42] Carsten Kleiner. *Modelling Spatial, Temporal and Spatio-Temporal Data in Object-Relational Database Systems.* PhD thesis, University of Hannover, Hannover, Germany, 2003.

[43] Ravi Kanth V. Kothuri, Albert Godfrind, and Euro Beinat. *Expert Oracle Spatial.* Hanser Fachbuchverlag, 2005.

[44] Ravi Kanth V. Kothuri, Siva Ravada, and Daniel Abugov. Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. In *SIGMOD Conference*, pages 546–557, Madison, Wisconsin, 2002. ACM Press.

[45] Hans-Peter Kriegel, Thomas Brinkhoff, and Ralf Schneider. Efficient spatial query processing in geographic database systems. *Data Engineering Bulletin*, 16(3):10–15, 1993.

[46] Hans-Peter Kriegel, Martin Pfeifle, Marco Pötke, and Thomas Seidl. A Cost Model for Spatial Intersection Queries on RI-Trees. In *DASFAA Conference*, pages 331–338, Jeju Island, Korea, 2004.

[47] Joachim Kröger, Regina Illner, Steffen Rost, and Andreas Heuer. Query Rewriting and Search in CROQUE. In *ADBIS Conference*, pages 288–302, London, UK, 1999. Springer-Verlag.

[48] Joachim Kröger, Stefan Paul, and Andreas Heuer. On the Ordering of Rewrite Rules (Extended Abstract). In *ADBIS Conference*, pages 157–159, London, UK, 1998. Springer-Verlag.

[49] Christopher Lawson. *The Art and Science of Oracle Performance Tuning.* Apress, 2003.

[50] Mavis K. Lee, Johann Christoph Freytag, and Guy M. Lohman. Implementing an Interpreter for Functional Rules in a Query Optimizer. In *VLDB Conference*, pages 218–229, Los Angeles, California, USA, 1988.

[51] Udo W. Lipeck. Räumliche Datenstrukturen Vorlesung. Hannover, Germany, 2004.

[52] Udo W. Lipeck. Grundlagen der Datenbanksysteme Vorlesung. Hannover, Germany, 2006.

[53] Udo W. Lipeck. Multidimensionale Datenbanken Vorlesung. Hannover, Germany, 2006.

[54] Udo W. Lipeck and Daniela Mantel. Matching Cartographic Objects in Spatial Databases. In *ISPRS Conference*, pages 172–176, Istanbul, Turkey, 2004.

[55] Ming-Ling Lo and Chinya V. Ravishankar. Spatial Hash-Joins. In *SIGMOD Conference*, pages 247–258, Montreal, Quebec, Canada, 1996. ACM Press.

[56] Guy M. Lohman. Grammar-like Functional Rules for Representing Query Optimization alternatives. In *SIGMOD Conference*, pages 18–27, Chicago, Illinois, USA, 1988.

[57] Mazeyar E. Makoui. Heuristische Anfrageoptimierungen in Relationalen Datenbanken. Master's thesis, University of Hannover, Hannover, Germany, 2003.

[58] Mazeyar E. Makoui. Configurable and Extensible Query Optimization by Controlled Term Rewriting. In *Grundlagen von Datenbanken Workshop*, pages 105–109, Wittenberg Lutherstadt, Germany, 2006.

[59] Mazeyar E. Makoui and Udo W. Lipeck. Anfrageoptimierung in objektrelationalen Datenbanken mit bedingten Termersetzungen. In *Grundlagen von Datenbanken Workshop*, pages 83–88, Wörlitz, Germany, 2005.

[60] Nikos Mamoulis and Dimitris Papadias. Complex spatial queries. pages 311–346, Amsterdam, Netherland, 2004. Kluwer Academic Publishers.

[61] William J. McKenna. *Volcano Query Optimizer Generator Manual*. PhD thesis, University of Colorado, Colorado, USA, 1993.

[62] Bernhard Mitschang. *Anfrageverarbeitung in Datenbanksystemen. Entwurfs- und Implementierungskonzepte*. Vieweg Verlag, 1995.

[63] Guido Moerkotte. *Konstruktion von Anfrageoptimierern für Objektbanken*. Shaker-Verlag, Aachen, 1995.

[64] Guido Moerkotte and Thomas Neumann. Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. In *VLDB Conference*, Seoul, Korea, 2006. VLDB Endowment.

[65] Kiyoshi Ono and Guy M. Lohman. Measuring the complexity of join enumeration in query optimization. In *VLDB Conference*, pages 314–325, Brisbane, Australia, 1990. Morgan Kaufmann Publishers Inc.

[66] Dimitris Papadias, Nikos Mamoulis, and Vasilis Delis. Approximate spatio-temporal retrieval. *ACM Transaction Database System*, 19(1):53–96, 2001.

[67] Jignesh M. Patel and David J. DeWitt. Partition based spatial-merge join. In *SIGMOD Conference*, pages 259–270, Montreal, Quebec, Canada, 1996. ACM Press.

[68] Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible Rule Based Query Rewrite Optimization in Starburst. In *SIGMOD Conference*, pages 39–48, San Diego, CA, USA, 1992.

[69] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw Hill Higher Education, 2002.

[70] Naveen Reddy and Jayant R. Haritsa. Analyzing plan diagrams of database query optimizers. In *VLDB Conference*, pages 1228–1239, Trondheim, Norway, 2005. VLDB Endowment.

[71] Philippe Rigaux, Michel Scholl, Luc Segoufin, and Stephane Grumbach. Building a constraint-based spatial database system: model, languages, and implementation. *Information Systems*, 28(6):563–595, 2003.

[72] Philippe Rigaux, Michel O. Scholl, and Agnes Voisard. *Spatial Databases. With Application to GIS.* Morgan Kaufmann Publishers Inc., 2001.

[73] Radoslaw Rudnicki. Entwicklung eines bedingten Termersetzungssystems und Umsetzung von Optimierungsregeln im Anfragesimulator `RELOpt`. Bachelor's thesis, University of Hannover, Hannover, Germany, 2005.

[74] Gunter Saake, Andreas Heuer, and Kai-Uwe Sattler. *Datenbanken: Implementierungstechniken.* Mitp-Verlag, 2005.

[75] Hanan Samet and Walid G. Aref. Spatial data models and query processing. In *Modern Database Systems*, pages 338–360. 1995.

[76] Wolfgang Scheufele and Guido Moerkotte. On the complexity of generating optimal plans with cross products (extended abstract). In *PODS Conference*, pages 238–248, Tucson, Arizona, USA, 1997. ACM Press.

[77] Wolfgang Scheufele and Guido Moerkotte. Efficient Dynamic Programming Algorithms for Ordering Expensive Joins and Selections. In *EDBT Conference*, pages 201–215, Valencia, Spain, 1998. Springer-Verlag.

[78] Markus Schneider. *Implementierungskonzepte für Datenbanksysteme.* Springer, Berlin, 2004.

[79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access Path Selection in a Relational Database Management system. In *SIGMOD Conference*, pages 23–34, Boston, Massachusetts, USA, 1979.

[80] Shashi Shekhar and Sanjay Chawla. *Spatial Databases. A Tour.* Prentice Hall, 2002.

[81] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts.* McGraw-Hill Publishing Co., 2005.

[82] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and Randomized Optimization for the Join Ordering Problem. *VLDB Journal*, 6(3):191–208, 1997.

[83] Yannis Theodoridis, Emmanuel Stefanakis, and Timos K. Sellis. Efficient Cost Models for Spatial Queries Using R-Trees. *Knowledge and Data Engineering*, 12(1):19–32, 2000.

[84] Michael Tiedge. Entwicklung und Implementierung einer topologischen Erweiterung für objektbasierte räumliche Datenbanken. Master's thesis, University of Hannover, Hannover, Germany, 2003.

[85] Aris Tsois and Timos K. Sellis. The Generalized Pre-Grouping Transformation: Aggregate-Query Optimization in the Presence of Dependencies. In *VLDB Conference*, pages 644–655, Berlin, Germany, 2003.

[86] Bennet Vance and David Maier. Rapid bushy join-order optimization with Cartesian products. In *SIGMOD Conference*, pages 35–46, Montreal, Quebec, Canada, 1996. ACM Press.

[87] Gottfried Vossen. *Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme.* Addison-Wesley, 2000.

[88] Hendrik Warneke. Erweiterung eines Simulators für relationale Anfrageoptimierungen. Bachelor's thesis, University of Hannover, Hannover, Germany, 2003.

[89] Richard H. Wolniewicz and Goetz Graefe. Algebraic Optimization of Computations over Scientific Databases. In *VLDB Conference*, pages 13–24, Dublin, Ireland, 1993.

[90] Weipeng P. Yan. *Rewrite optimization of SQL queries containing group-by.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1995.

[91] Weipeng P. Yan and Per-Ake Larson. Performing group-by before join. In *IEEE Conference*, pages 89–100. IEEE Computer Society Press, 1994.

[92] Weipeng P. Yan and Per-Ake Larson. Eager Aggregation and Lazy Aggregation. In *VLDB Journal*, pages 345–357, 1995.

Erklärungen

Hiermit versichere ich, Mazeyar E. Makoui, die vorliegende Dissertation ohne fremde Hilfe und nur unter Verwendung der von mir aufgeführten Quellen und Hilfsmittel angefertigt zu haben (gemäß 6(1) b) GPO).

Weiterhin erkläre ich, dass die vorliegende Dissertation nicht schon als Diplomarbeit oder ähnliche Prüfungsarbeit verwendet worden ist (gemäß 6(1) c) GPO).

Hannover, 4. Juli 2006.

(Mazeyar E. Makoui)