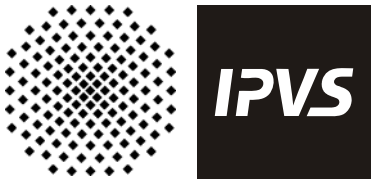
Universität Stuttgart
Fakultät Informatik, Elektrotechnik
und Informationstechnik



Diplomarbeit Nr. 2406

Entwicklung einer Kosten- und Statistik - API für Datenbankmanagementsysteme

Maxim Marchassin

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer:	Dipl. Inf. Tobias Kraft
begonnen am:	14. Oktober 2005
beendet am:	13. April 2006
CR-Klassifikation:	H.2.4, H.2.7

Institut für Parallele und
Verteilte Systeme
Abteilung
Anwendungssoftware
Universitätsstr. 38
D–70569 Stuttgart

Kurzfassung

Das Ceops-Projekt befasst sich u.a. mit der Optimierung von Anfragesequenzen. Dabei wird die Optimierung ausgehend von einer auf Heuristiken basierten Kontrollstrategie gesteuert. Diese Strategie soll durch eine kostenbasierte Strategie ersetzt werden, die die Kostenschätzwerte des darunterliegenden Datenbanksystems nutzt. Die vorliegende Diplomarbeit spezifiziert und entwirft eine generische Schnittstelle (API, Application Programming Interface) für den Zugriff auf verschiedene Datenbankmanagementsysteme. Die Schnittstelle bietet umfangreiche Funktionalität zum Auslesen und Modifizieren von Statistiken einzelner Tabellen und für die Kostenbestimmung einzelner SQL-Anweisungen.

Inhaltsverzeichnis

1 Einführung

Moderne Datenbankmanagementsysteme (DBMS) speichern und verwalten immer größere Datenmengen. Dabei gewinnt in der heutigen Zeit auch die Schnelligkeit der Informationsbereitstellung immer mehr an Bedeutung. Um Anfragen an riesige Datenbanken mit mehreren Gigabyte oder sogar Terabyte Daten in möglichst kurzer Zeit ausführen zu können, sind effiziente Optimierungstechniken auf allen Ebenen des Zugriffsprozesses gefragt.

Neben ständigen Verbesserungen und Neuentwicklungen im Hardware-Bereich, spielt die softwareseitige Optimierung der Anfrage- und Zugriffsabläufe eine immer wichtigere Rolle. Die meisten der auf dem Markt erhältlichen Datenbankmanagementsysteme (DBMS) verfügen über durchaus ausgereifte Zugriffs- und Verwaltungskomponenten für die Arbeit mit solch riesigen Datenbeständen. Darüber hinaus kommen die so genannten Anfrageoptimierer (Query Optimizer) zum Einsatz, um einen bestmöglichen Ausführungsplan für jede an das System gerichtete Anfrage zu ermitteln. Dieser Ausführungsplan beschreibt die Reihenfolge für die auszuführenden Operatoren, wobei jeder Ausführungsplan mit bestimmten Ausführungskosten verbunden ist. Die Ausführungskosten sowie in manchen Fällen die Anzahl der betroffenen Tupel in der Datenbank (Anfragenkardinalität) können wiederum anhand von bereits vorliegenden Systemstatistiken geschätzt werden.

Eine weitere Möglichkeit für die Verbesserung liegt in der Optimierung von Anfragesequenzen. Eine Anfragesequenz ist eine Folge von Anfragen, wobei jede dieser Anfragen auf die temporär gespeicherten Ergebnisse vorheriger Anfragen der Folge zugreifen kann.

1.1 Motivation

Bereits eine einfache auf Heuristiken basierte Kontrollstrategie für die Optimierung von Anfragesequenzen mit Umformungsregeln führt zu einer deutlichen Performanzsteigerung. Allerdings gilt diese Aussage meistens für einfache Anfragen. Denn sollte nach der Umformung einer Sequenz eine sehr komplexe Anfrage entstehen, führt diese entsprechend zu einer Komplexitätssteigerung bei der Berechnung des besten Ausführungsplans für diese Sequenz. Des Weiteren sind komplexe Anfragen anfälliger für falsche Schätzungen innerhalb eines Ausführungsplans.

Ziel dieser Diplomarbeit ist es deshalb, eine generische Kosten- und Statistikschnittstelle (API) für Datenbankmanagementsysteme zu entwickeln. Diese Schnittstelle ermöglicht die Einführung einer verbesserten kostenbasierten Kontrollstrategie für die Auswahl der Restrukturierungsregeln im Optimierungsvorgang, welche mit statistischen Kostenschätzwerten des zugrunde liegenden Datenbanksystems arbeitet.

1.2 Gliederung

Die vorliegende Diplomarbeit gliedert sich in 6 Kapitel. Nach einer kurzen Einleitung im ersten Kapitel werden die genaue Vorgehensweise und die während der Arbeit erzielten Resultate in einzelnen Kapiteln Schritt für Schritt beschrieben. Zum Schluss folgt eine Zusammenfassung der erzielten Ergebnisse mit einem Ausblick für den zukünftigen Einsatz im Rahmen des Ceops-Projekts und eventuell in weiteren Projekten auf dem Gebiet der DBMS-Optimierung.

Kapitel 2 widmet sich den theoretischen Grundlagen und den bereits vorliegenden Erkenntnissen für die Optimierung von Anfragesequenzen, welche u.a. im Ceops-Proket gewonnen wurden. Dieses Wissen gilt als unentbehrlich für alle nachfolgenden Arbeiten und wird deshalb ausführlich besprochen.

Das dritte Kapitel befasst sich mit der Analyse der drei kommerziellen Datenbankmanagementsysteme IBM DB2 V8.2, Oracle 10g und Microsoft SQL Server 2005 (Yukon). Der Schwerpunkt dieser Untersuchung liegt in der Identifikation der von dem jeweiligen DBMS bereitgestellten Statistikwerte und in der vollständigen Beschreibung der angebotenen Zugriffsoperationen oder allgemeiner Schnittstellen.

Nach der Analysephase folgt im vierten Kapitel die genaue Spezifikation einer generische Kosten- und Statistik-Schnittstelle (API). Zuerst werden die im dritten Kapitel (Analysephase) gewonnenen Statistikwerte für alle drei Datenbanksysteme zusammengefasst und mit geeigneten Datenstrukturen für den Datenaustausch in Verbindung gebracht. Diese Datenstrukturen dienen der Vereinheitlichung der Datenstrukturen unterschiedlicher DBMS.

Als „Proof-Of-Concept“ stellt das fünfte Kapitel eine prototypische Implementierung der entwickelten Schnittstelle in der Programmiersprache Java vor. Besonderes Augenmerk gilt hier vor allem der Abbildung zwischen den zum Teil proprietären Datenstrukturen der betrachten Datenbankmanagementsysteme und den in Kapitel 4 erarbeiteten Schnittstellen-Datenstrukturen.

Das sechste und letzte Kapitel enthält eine Zusammenfassung der gesamten Diplomarbeit. Darüber hinaus wird zum Schluss ein kleiner Ausblick auf weiterführende Arbeiten gegeben.

1.3 Ablauf

Die Durchführung der Diplomarbeit wurde von Anfang an nach einem im Voraus erstellten Projektplan vollzogen. Um handfeste Ergebnisse zu bestimmten Kalenderdaten kontrollieren zu können, wurden außerdem 5 Meilensteine definiert. Diese Meilensteine unterteilten die komplette Arbeitszeit in 5 große Arbeitspakete oder Phasen mit einer anfänglichen Einarbeitungs-, 3 großen Ausarbeitungs- und einer Abschlussphase:

Meilenstein	**Bezeichnung**	**Datum**
Meilenstein 1	Projektplanfertigstellung, Abschluss Einarbeitung	28.10.2005
Meilenstein 2	Abschluss Phase 1 (Analyse)	17.12.2005
Meilenstein 3	Abschluss Phase 2 (Spezifikation und Entwurf)	23.01.2006
Meilenstein 4	Abschluss Phase 3 (Implementierung)	17.03.2006
Meilenstein 5	Fertigstellung der Diplomarbeit, Abgabe	13.04.2006

2 Grundlagen und Definitionen

Dieses Kapitel widmet sich den für diese Diplomarbeit wichtigen Grundlagen und speziellen Begriffen, welche den Hintergrund der Arbeit ausleuchten. Diese sind für den gesamten Inhalt der Ausarbeitung von großer Bedeutung und tragen zum allgemeinen Verständnis des Themas bei.

Um die eigentliche Bedeutung der zu entwickelnden Kosten- und Statistik-API für den gesamten Prozess der Anfrageoptimierung zu verstehen, wird zu Beginn des Kapitels auf regel- und kostenbasierte Optimierung der Datenbankanfragen eingegangen. Die für diese Arbeit interessante kostenbasierte Optimierungsstrategie spielt außerdem eine wichtige Rolle bei der Optimierung von Anfragesequenzen mit der Caurse-Grained-Optimization Methode (CGO). Da sowohl die kostenbasierte Optimierung als auch CGO die Existenz von zahlreichen Kostenschätzwerten und Statistiken in DBMSs voraussetzt, werden zum Abschluss des 2. Kapitels verschiedene existierende Statistikarten als Grundlage für die Analysephase erörtert.

2.1 Regel- und kostenbasierte Optimierung

In jedem ausgereiften Datenbankmanagementsystem existieren meistens mehrere Möglichkeiten, eine an das System gerichtete Anfrage auszuführen. Der Ausführungsplan mit der besten Gewichtung im System bzw. mit den niedrigsten Ausführungskosten wird dann entsprechend am schnellsten die gewünschten Ergebnisse liefern oder die geringste Ressourcenauslastung gewährleisten. Dieser beste Ausführungsplan bezogen auf eine bestimmte Anfrage wird vom so genannten Optimierer (Optimizer) ermittelt.

Definition 2.1: „*Optimierer (Optimizer, Optimierungsprogramm) ist die Komponente des SQL-Compilers, die einen Zugriffsplan für eine SQL-Anweisung in der Datenbearbeitungssprache (z.B. DML) auswählt. Dies geschieht, indem der Ausführungsaufwand mehrerer alternativer Zugriffspläne modelliert wird und der Zugriffsplan mit dem geringsten geschätzten Aufwand ausgewählt wird“* [IBM04a].

In anderen Worten - der Optimierer ist eine Funktion der Datenbank, mit welcher eine Reihe von unter unterschiedlichen Bedingungen abgeleiteten Zugriffspfaden erzeugt wird, um die als am effizientesten geltende Art der Ausführung einer SQL-Anweisung bzw. den effizientesten Ausführungsplan auszuwählen [PAD03]. Unter beanspruchten Ressourcen werden in diesem Zusammenhang solche Werte wie Prozessorauslastung, Anzahl der Eingabe- bzw. Ausgabeoperationen, Speicherbedarf, usw. zusammengefasst.

Die Berechnung des besten Zugriffsplans für eine Anfrage kann mit zwei verschiedenen Methoden erfolgen, welche auch als Kombination auftreten können:

- Regelbasierte Optimierung (Rule Based Optimizer, RBO) benutzt eine Reihe einfacher gewichteter Regeln, um für jeden möglichen Zugriffsplan eine optimale Entscheidung zu treffen. Statistische Information wie z, B. Datenverteilung in Tabellen werden bei dieser Methode nicht berücksichtigt. Diese Art der

Optimierung wurde früher in Datenbankmanagementsystemen mit überwiegend statischen Komponenten und starren Strukturen eingesetzt.

- Kostenbasierte Optimierung (Cost Based Optimizer, CBO) orientiert sich an dem für die auszuführende Anweisung benötigten Aufwand. D.h. jedem Ausführungsplan werden bestimmte Ausführungskosten zugeordnet. Die Pläne mit höheren Kosten beanspruchen mehr Systemressourcen, die mit niedrigeren Ausführungskosten werden entsprechend effizienter ausgeführt. Zum Schluss wird ein Plan mit den niedrigsten Kosten als Ausführungsplan für die Anfragebearbeitung ausgewählt. Dieser sollte dem Ziel der kostenbasierten Optimierung entsprechend entweder den besten Durchsatz oder die geringste Ressourcennutzung während der Berechnung der Ergebnismenge gewährleisten. Um dieses Ziel zu erreichen, werden beim kostenbasierten Verfahren zahlreiche Statistiken zur Schätzung der Kosten für jeden möglichen Ausführungsplan hinzugezogen. Mit Hilfe von Verteilungsstatistiken (Histogrammen) und verschiedenen Eigenschaften von Tabellen, Spalten, Indizes und Partitionen generiert der Optimizer alle möglichen Permutationen der Zugriffspfade und sucht aus dieser Menge den Zugriffsplan mit den besten Kosten heraus.

Die folgende Abbildung zeigt die Stellung des Optimierers im Gesamtkonzept der Anfragenauswertung eines Datenbankmanagementsystems von Oracle:

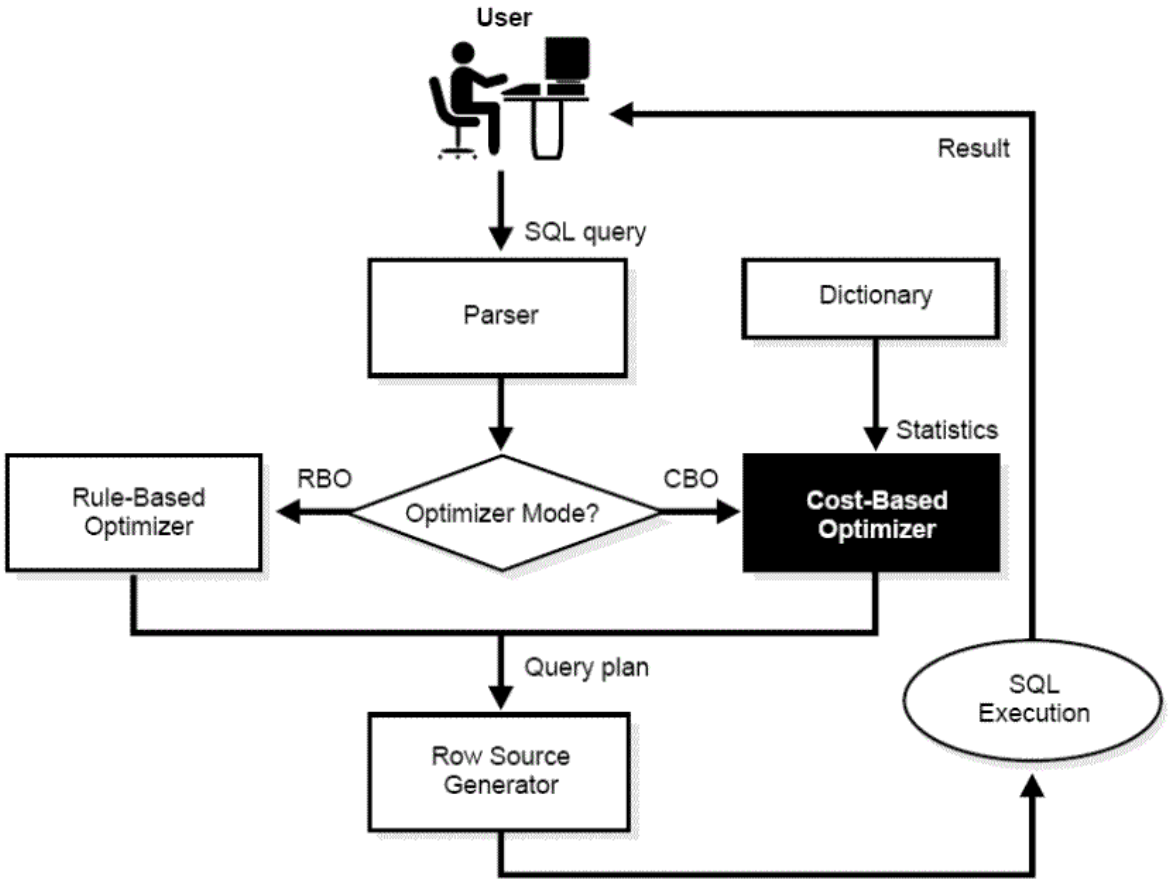


Abbildung 2.1: Optimierer und Auswertung von SQL-Anfragen in Oracle [GRE02]

2.2 Coarse-Grained Optimization (CGO)

Eine weitere Möglichkeit der Optimierung in Datenbankmanagementsystemen ergibt sich, wenn anstatt einer einfachen Anfrage eine ganze Sequenz von Anfragen im System vorliegt. Zusammenhängende Anfragesequenzen werden zum Beispiel oft in OLAP- und Data-Mining-Anwendungen während der Interaktion mit dem Benutzer generiert (s. Abbildung 2.2).

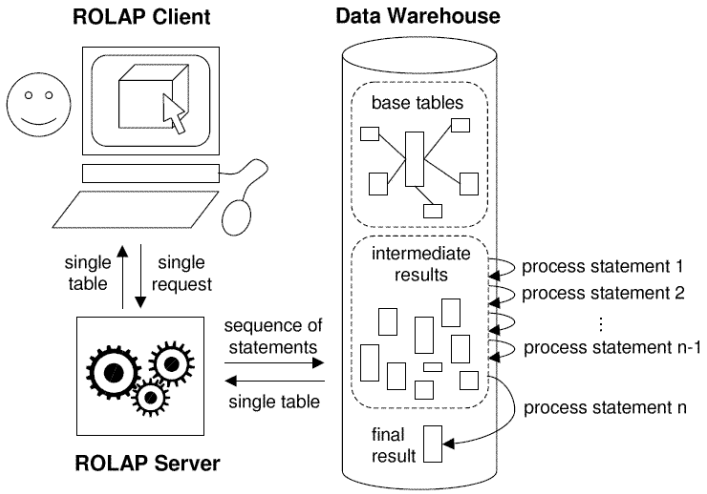


Abbildung 2.2: Typisches ROLAP Szenario, [KMRSB03]

In den heutigen DBMSs werden solche Anfragesequenzen meistens als eine Sequenz von isolierten Einzelanweisungen behandelt, welche dann unabhängig voneinander optimiert werden. Eine Lösung für dieses Problem wird mit der so genannten Coarse-Grained Optimization - Methode angeboten.

Definition 2.2: *Coarse-Grained Optimization (CGO, Grobkörnige Optimierung) ist ein Ansatz zur Restrukturierung einer Anfragesequenz in eine semantisch äquivalente Sequenz, welche weniger Systemressourcen als die Originalsequenz beansprucht. Die Restrukturierung erfolgt dabei nach so genannten Rewrite-Regeln* [KMRSB03].

Der Äquivalenzbegriff bezieht sich in diesem Kontext ausschließlich auf die Endergebnisse der allerletzten Anfragen in beiden Sequenzen, d.h. einige oder sogar alle Zwischenergebnisse der Anfragen können sich durchaus unterscheiden.

Ein CGO-Optimierer lässt sich durch einfache Heuristiken steuern, welche die Anwendung der Restrukturierungsregeln beeinflussen. Diese Heuristiken können wie folgt unterteilt werden:

- Eigenschaften der Regelmenge / Menge der Rewrite-Regeln
- Eigenschaften der zu betrachtenden Anfragesequenzen
- Eigenschaften der Datenbank bzw. des DBMS

Ein im Rahmen des Ceops-Projekts (**C**ost-based, database-**E**xternal **OP**timization of query generating information **S**ystems) bereits entwickelter Prototyp verwendet eine auf den Eigenschaften der Regelmenge basierte Kontrollstrategie für den CGO-Optimierer. Die durch den Restrukturierungsprozess entstandene Verzögerung wird

bei diesem Ansatz jedoch nicht beachtet und kann deshalb in einigen Fällen zu einer Verschlechterung der Ausführungszeit führen.

Diese Diplomarbeit beschäftigt sich im weiteren Verlauf mit den Eigenschaften von Anfragesequenzen und Datenbanken. Durch den Zugriff auf Kostenschätzwerte und Datenbank-Statistiken wird die Einführung einer kostenbasierten Kontrollstrategie und Weiterentwicklung des o.g. Prototyps möglich.

2.3 Statistikarten und Kostenschätzwerte in DBMSs

Die modernen Datenbankmanagementsysteme bieten eine umfangreiche Sammlung an statistischen Informationen (Statistiken) und Schätzwerten, welche meistens im Data Dictionary abgelegt werden. Diese werden vom CBO-Optimierer für die Findung des bestmöglichen Ausführungsplans verwendet und spielen deshalb eine sehr wichtige Rolle für den gesamten Optimierungsprozess in heutigen DBMS. Wie bereits geschildert können diese Informationen auch im Rahmen einer kostenbasierte Kontrollstrategie beim CGO-Optimierer zum Einsatz.

Bei allen Statistikarten wird zwischen genauen und geschätzten Statistiken unterschieden. Die exakte Berechnung von statistischen Informationen nimmt mehr Zeit in Anspruch und ist deshalb nicht immer vorteilhaft (z.B. bei großen Datenmengen). Allerdings liefert die kostenbasierte Optimierung bessere Ergebnisse bei der Arbeit mit exakten Statistiken. Soll der gesamte Datenbestand einer Schätzung zwecks Statistikerstellung unterzogen werden, kommt die so genannte Sampling-Methode zum Einsatz. Dabei werden aus der gesamten Datenmenge stichprobenartig Proben entnommen, deren Eigenschaften für den ganzen Datenbestand stellvertretend sind [HOT88].

Im weiteren Verlauf werden die wichtigsten Statistikarten diskutiert, welche in den meisten DBMSs vorkommen. Für die gesamte Diplomarbeit sollen außerdem folgende Voraussetzungen gelten:

- Die Statistiken befinden sich im logisch konsistenten Zustand
- Alle Statistiken sind vorhanden und immer aktuell
- Alle für die Speicherung der Ausführungspläne notwendigen Tabellen sind bereits angelegt
- Es wird nach Möglichkeit mit exakten Statistikdaten ohne Schätzungen gearbeitet, um damit bestmögliche Ergebnisse bei der Optimierung zu erzielen.

2.3.1 Statistikerstellung

In heutigen Datenbanksystemen existieren meistens mehrere Möglichkeiten für die Erstellung bzw. Sammlung statistischer Informationen über Datenbank-Objekte. Nachfolgend sind einige dieser Möglichkeiten aufgeführt:

- Manuelle Statistikerstellung – wird durch die Eingabe eines speziellen Befehls angestoßen.

- Manuelle Statistikerstellung mit Profilen – Befehle zur Statistikerstellung mit Parametern für bestimmte Statistiken (z.B. Angabe bestimmter Statistikarten) können vom Benutzer als Profile gespeichert werden, um spätere Aufrufe dieser Befehle zu vereinfachen.

- Statistikerstellung mit automatisch erzeugten Profilen – Statistische Profile werden basierend auf Datenbankaktivitäten automatisch durch das DBMS angelegt. Diese Profile können dann für die automatische Statistikerstellung (s.u.) oder für die durch den Benutzer initiierte manuelle Statistikerstellung genutzt werden.

- Automatische Statistikerstellung – Statistiken werden vom DBMS automatisch im Hintergrund erstellt. Das DBMS entscheidet dann, welche Statistiken überhaupt notwendig sind und wann diese aktualisiert werden müssen.

2.3.2 Einfache Statistiken

In allen 3 betrachteten Datenbankmanagementsystemen wird stets zwischen Tabellen-, Spalten-, Index- und Verteilungsstatistiken unterschieden. Bis auf Verteilungsstatistiken können alle Statistikarten in einfachen (flachen) Datenstrukturen gespeichert werden. Sie werden deshalb auch als „einfache Statistiken“ bezeichnet. Im Gegensatz dazu sind zur Speicherung der Verteilungsstatistiken komplexe Datenstrukturen notwendig.

Definition 2.3: *Kardinalität einer Tabelle bzw. Spalte ist die Anzahl in der Tabelle / Spalte gespeicherten Tupel / Werte. D.h. die Kardinalität einer Tabelle / Spalte entspricht immer der Anzahl der Zeilen / Wert in dieser Tabelle / Spalte*.

Die wichtigsten einfachen Statistiken sind in der folgenden Aufstellung zu finden:

- Tabellenstatistiken
  - Tabellenkardinalität / Anzahl Zeilen
  - Informationen über Speicherverbrauch durch die Tabelle

- Spaltenstatistiken
  - Anzahl unterschiedlicher Spaltenwerte
  - Minimalwert (Minimum)
  - Maximalwert (Maximum)
  - Anzahl der NULL-Werte
  - Mittlere Spaltenlänge

- Indexstatistiken
  - Anzahl unterschiedlicher Schlüsselwerte
  - Kardinalität

2.3.3 Histogramme (Verteilungsstatistiken)

Definition 2.4: *Ein Histogramm ist die graphische oder zahlenmäßig in Form von einer Tabelle erfasste Darstellung der Häufigkeitsverteilung von Messwerten. Man geht dabei von den nach Größe geordneten Daten aus und teilt den gesamten Bereich der Stichprobe in k Klassen auf. Diese müssen nicht notwendig gleich breit sein. Über jeder Klasse wird ein Rechteck errichtet, dessen Fläche proportional zur klassenspezifischen Häufigkeit ist. Ist die Fläche des Rechtecks gleich der absoluten Häufigkeit, wird das Histogramm absolut genannt, wenn die relativen Häufigkeiten verwendet werden, wird es entsprechend als relativ oder normiert bezeichnet*. [WIK].

Beispiel: Grafische Darstellung der Zahl der PKWs auf 1000 Personen in 32 ausgewählten Ländern:

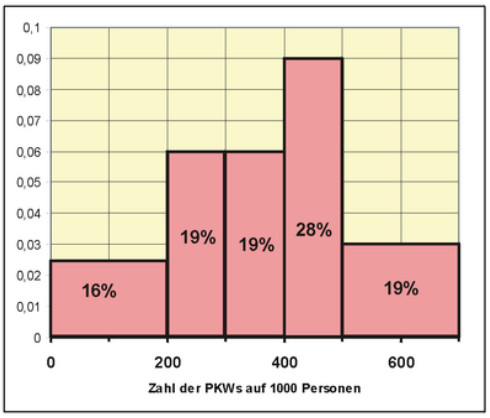


Abbildung 2.3: Herkömmliches Histogramm [WIK]

Definition 2.5: *Im Datenbankbereich werden die Histogramme erzeugt, indem die Datenverteilung des zu bearbeitenden Attributs in mehrere disjunkte Buckets (Eimer) unterteilt wird. Die Buckets können als aneinander gereihte Eimer veranschaulicht werden, in die jeweils ein bestimmtes Intervall der Datenverteilung hineinfällt. Für jedes dieser Buckets werden nicht mehr die einzelnen Attributwerte, sondern nur seine Ober- und Untergrenze, sowie eine Approximation der Attributwerte und deren Häufigkeiten angegeben*. [MÜL05]

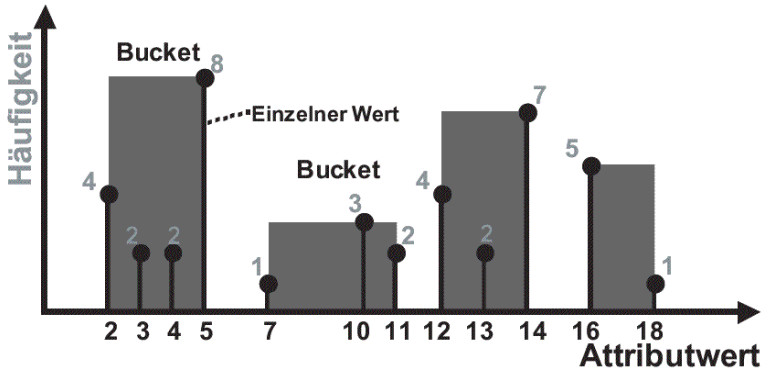


Abbildung 2.4: Häufigkeitsverteilung = Histogramm [MÜL05]

Histogramme werden u.a. auch als Verteilungsstatistiken bezeichnet. In DBMSs werden diese dazu verwendet, um Datenwertsverteilungen in Tabellenform approximiert und komprimiert abzubilden. Sie werden während der Optimierungsphase eingesetzt, da sie die Verteilung der Daten in einer Tabellenspalte beschreiben. In einigen DBMSs unterscheidet man außerdem zwischen Frequenz- und Quantilhistogrammen.

Definition 2.6: *In einem Frequenzhistogramm wird jeder in der Spalte vorkommende Wert einem separaten Histogramm-Bucket zugeordnet. Jeder Bucket enthält dann die Anzahl aller Vorkommen für diesen bestimmten Wert*. [ORA06a]

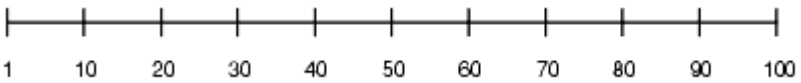
Durch diese Vorgehensweise können dann Informationen zur Anzahl der Duplikate in einer Spalte, die am häufigsten auftretenden Werten, usw. gesammelt werden.

Beispiel 2.1: Frequenzhistogramm mit 5 Werten

Wert	Anzahl der Vorkommen
1	254
2	5
3	123
4	5644
5	55

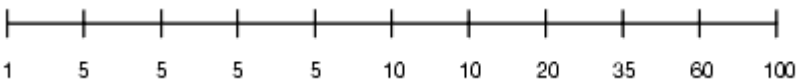
Definition 2.7: *In Quantilhistogrammen werden alle Spaltenwerte in disjunkte Bereiche (Buckets) aufgeteilt. Bei höhenbalancierten Histogrammen dieser Art werden Bereiche mit ungefähr gleich vielen Elementen gewählt. Die Quantilstatistiken liefern Informationen darüber, wie Datenwerte einer Spalte verteilt sind. Die Statistiken der so genannten K-Quantile stellen den Wert V dar, bei dem oder unter dem mindestens K Werte liegen. Ein K-Quantil lässt sich durch Sortieren der Werte in aufsteigender Reihenfolge ermitteln. Der K-Quantilwert ist der Wert an der K-ten Position von der Untergrenze des Wertebereichs betrachtet.* [ORA06a, IBM04b].

Beispiel 2.2: Angenommen eine Spalte C mit Werten zwischen 1 und 100 und ein Histogramm mit 10 Buckets sind vorhanden. Falls die Daten in C gleichverteilt sind, dann könnte das Histogramm zu Spalte C wie folgt aussehen. Die Zahlen stellen dabei die Obergrenzen der Buckets dar:



Die Anzahl der Elemente in jedem Bucket entspricht einem Zehntel der Gesamtzahl aller Datensätze in der Spalte. In diesem Beispiel haben dann vier Zehntel aller Datensätze Werte zwischen 60 und 100, falls eine Gleichverteilung der Werte gegeben ist.

Sollten die Werte andererseits nicht gleichverteilt sein, dann kann dieses Histogramm auch die folgende Form annehmen:



Die meisten Zeilen haben in diesem Fall den Wert „5“. Lediglich 1/10 der Zeilen hat Werte zwischen 60 und 100 [ORA06a].

2.3.4 Kostenschätzwerte und Anfragenkardinalitäten

Kostenschätzwerte werden wie der Name schon sagt geschätzt. D.h. sie stellen zu keinem Zeitpunkt exakte Werte dar, und werden unter Berücksichtigung möglicher Fehler verwendet. Diese Arbeit beschränkt sich im weiteren Verlauf auf zwei folgende anfragenbezogene Schätzwerte:

- Ausführungskosten:
  Geschätzte Kosten des Ausführungsplanes für die Anfrage

- Kardinalität:
  Geschätzte Anzahl der Tupel im Anfrageergebnis

2.3.5 Externe Zugriffsoperationen und Integritätserhaltung

Unabhängig davon, ob ein DBMS spezielle externe Zugriffsoperationen für die Abfrage bzw. Aktualisierung von Katalogstatistiken bereitstellt oder eine Eigenentwicklung diese Arbeiten übernehmen muss, sollte die Erhaltung der Datenbankintegrität immer im Vordergrund stehen. Diese Bedingung mag vielleicht bei Leseoperationen keine Rolle spielen, sie gewinnt jedoch umso mehr an Bedeutung, wenn neue bzw. geänderte statistische Informationen in einem Schreibzugriff in den Statistiktabellen gespeichert werden. Um die logische Datenkonsistenz in der Datenbank gespeicherten Statistiken zu erhalten, sind bei einer Aktualisierung der Katalogstatistiken Wertebereiche und Formate der zu speichernden Statistiken zu beachten. Darüber hinaus muss die Konsistenz der Beziehungen zwischen verschiedenen Statistiken erhalten bleiben.

In allen betrachteten DBMS sind bereits vom Hersteller zahlreiche Einschränkungen (CONSTRAINS) für die Wahrung der Integrität definiert. Sollte ein Konflikt zwischen einem neuen oder aktualisierten Statistikwert und einer anderen Statistik festgestellt werden, wird vom DBMS eine Fehlermeldung ausgegeben und der Schreibvorgang nicht zugelassen. Allerdings können sich im Arbeitsablauf einige Konstellationen ergeben, in denen Wert-, Format- oder Beziehungskonflikte nicht erkannt werden und somit keine Fehlermeldung an den Benutzer erfolgt. Dies ist beispielsweise der Fall, wenn zwei zusammengehörigen Statistiken in verschiedenen Katalogtabellen gespeichert sind. Als Beitrag zur Vermeidung solcher Konflikte und zur Erhaltung der Datenintegrität werden in Kapitel 3 entsprechende herstellerspezifische Einschränkungen für Zugriffsoperationen diskutiert.

3 Analyse der Datenbankmanagementsysteme

Ausgehend von der Aufgabenstellung für diese Arbeit und unter Zuhilfenahme der im vorherigen Kapitel dargestellten Grundlagen sollen in diesem Kapitel drei am Markt sehr verbreitete kommerzielle Datenbankmanagementsysteme IBM DB2 Version 8.2, Oracle Version 10g und Microsoft SQL Server 2005 (Entwicklungsname: Yukon) analysiert werden. Das Ziel dieses Kapitels ist eine Aufstellung für die analysierten DBMSs mit den verfügbaren Statistikwerten, möglichen Zugriffsmethoden und datenbankspezifischen Einschränkungen für diese Zugriffsoperationen.

3.1 IBM DB2 Version 8.2

Die folgenden Informationen wurden mit Hilfe von [IBM04a, IBM04b, IBM04c] gesammelt. Zwecks Übersichtlichkeit sind meistens nur für diese Diplomarbeit relevanten Ausschnitte betrachtet worden.

3.1.1 Möglichkeiten der Statistikerstellung

Statistische Informationen für spezifizierte Tabellen und Indexe werden vom DB2 Server in den mit identischen Daten befüllten Systemkatalog-Sichten der Schemata SYSCAT und SYSSTATS abgelegt. Dabei kann auf die Inhalte der in SYSCAT abgelegten Sichten nur lesend zugegriffen werden. Im Gegensatz dazu könnem die Werte der in SYSSTATS abgelegten Sichten sowohl gelesen als auch mit UPDATE Anweisungen modifiziert werden. Diese Sichten sind deshalb für diese Arbeit besonders interessant.

Zur Gewährleistung der Effizienz sowohl des Dienstprogramms RUNSTATS als auch der nachfolgenden Zugriffsplananalyse ist es sinnvoll, Verteilungsstatistikdaten nur für solche Tabellenspalten zu erfassen, die in Klauseln wie WHERE, GROUP BY, usw. verwendet werden. Darüber hinaus kann es auch nützlich sein, Statistikdaten für kombinierte Gruppen von Spalten zu erfassen. Das Optimierungsprogramm verwendet solche Informationen zur Erkennung von Spaltenkorrelationen bei der Abschätzung der Selektivität von Anfragen, die auf die Spalten in der Gruppe verweisen.

Dem Benutzer eines IBM DB2 V8.2 Systems stehen folgende Möglichkeiten der Statistikerstellung zur Verfügung:

- Manuelle Statistikerstellung mit dem RUNSTATS-Befehl.
- Automatische Statistikerstellung mit dem RUNSTATS-Befehl als Teil von „DB2 Automated Table Maintenance“. Folgende Parameter müssen auf “ON” gesetzt werden:
    1. AUTO_MAINT
    2. AUTO_TBL_MAINT
    3. AUTO_RUNSTATS

- (Auto-)Sampling mit der TABLESAMPLE-Option des RUNSTATS-Befehls

3.1.2 Tabellenstatistiken – SYSSTAT.TABLES, SYSCAT.TABLES

SYSSTAT.TABLES und SYSCAT.TABLES enthalten statistische Informationen zu einigen beziehengsweise allen Tabellen aus dem Systemkatalog.

Tabelle 3.1 : SYSSTAT.TABLES, SYSCAT.TABLES

Statistik	Datentyp	Beschreibung
TABSCHEMA	VARCHAR(128)	Qualifizierte Schemabezeichnung
TABNAME	VARCHAR(128)	Qualifizierte Tabellenbezeichnung
CARD	BIGINT	Anzahl der Zeilen in der Tabelle
NPAGES	INTEGER	Anzahl nicht leerer mit Tabellenzeilen belegter Seiten
FPAGES	INTEGER	Anzahl von der Tabelle belegter Seiten
OVERFLOW	INTEGER	Anzahl der Überlaufzeilen in der Tabelle
ACTIVE_BLOCKS	INTEGER	Gesamtzahl belegter Blöcke (MDC-Tabellen)

Sollten für eine Tabelle, für die keine Indizes vorliegen, die Indexstatistiken angefordert werden (s.u.), wird die CARD-Statistik nicht mit einem neuen Wert aktualisiert, d.h. die vorige CARD-Statistik wird beibehalten.

3.1.3 Spaltenstatistiken SYSCAT.COLUMNS, SYSSTAT.COLUMNS

SYSSTAT.COLUMNS und SYSCAT.COLUMNS enthalten Spaltenstatistiken. Die nachfolgenden Spaltenstatistiken bis auf SUB_COUNT und SUB_DELIM_LENGTH werden immer für die erste Spalte im Indexschlüssel erfasst.

Tabelle 3.2 : SYSSTAT.COLUMNS, SYSCAT. COLUMNS

Statistik	Datentyp	Beschreibung
TABSCHEMA	VARCHAR(128)	Qualifizierte Schemabezeichnung
TABNAME	VARCHAR(128)	Qualifizierte Tabellenbezeichnung
COLNAME	VARCHAR(128)	Qualifizierte Spaltenbezeichnung
COLCARD	BIGINT	Anzahl unterschiedlicher Werte in der Spalte
AVGCOLLEN	INTEGER	Mittlere Spaltenlänge
HIGH2KEY	INTEGER	2. Maximum / Zweithöchster Wert der Spalte
LOW2KEY	INTEGER	2. Minimum / Zweitniedrigster Wert der Spalte
NUMNULLS	INTEGER	Anzahl der Nullwerte in der Spalte
SUB_COUNT	SMALLINT	Durchschnittliche Anzahl von Unterelementen
SUB_DELIM_LENGTH	SMALLINT	Durchschnittslänge jedes Begrenzers, der ein Unterelement trennt

Die Werte SUB_COUNT und SUB_DELIM_LENGTH werden nur für Werte vom Typ CHAR, VARCHAR, GRAPHIC und VARGRAPHIC erfasst. Aus STATS_TIME aus SYSCAT.TABLES können das Datum und die Uhrzeit der letzten Statistikerstellung entnommen werden.

3.1.4 Indexstatistiken SYSCAT.INDEXES, SYSSTAT.INDEXES

SYSSTAT.INDEXES und SYSCAT. INDEXES enthalten Indexstatistiken für einige bzw. alle Tabellen aus dem Systemkatalog. Detaillierte Indexstatistiken werden erst nach der Spezifikation des „DETAILED“ Schlüssels beim RUNSTATS Aufruf angelegt.

Tabelle 3.3 : SYSSTAT.INDEXES, SYSCAT. INDEXES

Statistik	Datentyp	Beschreibung
TABSCHEMA	VARCHAR(128)	Qualifizierte Schemabezeichnung
TABNAME	VARCHAR(128)	Qualifizierte Tabellenbezeichnung
INDNAME	VARCHAR(128)	Qualifizierte Indexbezeichnung
NLEAF	INTEGER	Anzahl der Blattseiten (Blöcke) im Index
NLEVELS	SMALLINT	Anzahl der Indexstufen
CLUSTERRATIO	INTEGER	Grad der Clusterbildung der Tabellendaten
CLUSTERFACTOR	SMALLINT	Feinerer Grad der Clusterbildung
DENSITY	INTEGER	Dichte = SEQUENTIAL_ PAGES / Anzahl der Index-Enthaltenen Seiten (in Prozent)
FIRSTKEYCARD	BIGINT	Anzahl unterschiedlicher Werte in der 1. Spalten des Indexes
FIRST2KEYCARD	BIGINT	Anzahl unterschiedlicher Werte in den ersten 2 Spalten des Indexes
FIRST3KEYCARD	BIGINT	Anzahl unterschiedlicher Werte in den ersten 3 Spalten des Indexes
FIRST4KEYCARD	BIGINT	Anzahl unterschiedlicher Werte in den ersten 4 Spalten des Indexes
FULLKEYCARD	BIGINT	Anzahl unterschiedl. Werte / Schlüssel in allen Spalten des Indexes
PAGE_FETCH_PAIRS	VARCHAR(254)	Geschätzte Anzahl der Seitenabrufe für verschiedene Puffergrößen
SEQUENTIAL_PAGES	INTEGER	Anzahl der Blattseiten, die auf der Platte in der durch den Indexschlüssel definierten Reihenfolge mit wenigen oder gar keinen großen dazwischen liegenden Lücken gespeichert sind
NUMRIDS	BIGINT	Anzahl der RIDs (DatensatzID = Record Identifiers = RID) im Index einschließlich gelöschter RIDs in Indizes des Typs 2
NUMRIDS_DELETED	BIGINT	Gesamtanzahl der RIDs, die im Index als gelöscht markiert sind, außer den RIDs in Blattseiten, in denen alle RIDs als gelöscht markiert sind
NUM_EMPTY_LEAFS	BIGINT	Anzahl der Blattseiten, in denen alle RIDs als gelöscht markiert sind
AVERAGE_SEQUENCE_PAGES	DOUBLE	Durchschnittliche Anzahl der Indexseiten, auf die sequentiell zugegriffen werden kann.
AVERAGE_RANDOM_PAGES	DOUBLE	Durchschnittliche Anzahl zufälliger Indexseiten zwischen sequentiellen Seitenzugriffen
AVERAGE_SEQUENCE_GAP	DOUBLE	Lücke zwischen den Sequenzen
AVERAGE_SEQUENCE_FETCH_PAGES	DOUBLE	Durchschnittliche Anzahl der Tabellenseiten, auf die sequentiell zugegriffen werden kann.
AVERAGE_RANDOM_FETCH_PAGES	DOUBLE	Durchschnittliche Anzahl zufälliger Tabellenseiten zwischen sequentiellen Seitenzugriffen Abrufen von Tabellenzeilen über den Index
AVERAGE_SEQUENCE_FETCH_GAP	DOUBLE	Lücke zwischen sequenziellen Zugriffen beim Abrufen von Tabellenzeilen über den Index

3.1.5 Histogramme – SYSCAT.COLDIST, SYSSTAT.COLDIST

In IBM DB2 Version 8.2 können beide im Grundlagenkapitel beschriebene Histogrammtypen (Frequenz- und Quantilhistogramme) erstellt werden. SYSSTAT.COLDIST und SYSCAT.COLDIST enthalten Verteilungsstatistiken bzw, Histogramme zu einer durch ein Schema, einen Tabellen- und einen Spaltennamen definierten Tabellenspalte. Die Verteilungsstatistiken werden nach der Angabe der WITH DISTRIBUTION Klausel beim Aufruf von RUNSTATS erstellt. Mit dem Parameter NUM_FREQVALUES wird die Anzahl der häufigsten Werte spezifiziert, die für die Histogrammerstellung hinzugezogen werden sollen. So würde zum Beispiel die Angabe NUM_FREQVALUES=3 bedeuten, dass ein Histogramm für die drei am häufigsten vorkommende Werte angelegt wird. Mit NUM_QUANTILES wird die Anzahl der Quantile im Histogramm spezifiziert.

Tabelle 3.4 : SYSSTAT.COLDIST, SYSCAT.COLDIST

Statistik	Datentyp	Beschreibung
TABSCHEMA	VARCHAR(128)	Qualifizierte Schemabezeichnung
TABNAME	VARCHAR(128)	Qualifizierte Tabellenbezeichnung
COLNAME	VARCHAR(128)	Qualifizierte Spaltenbezeichnung
TYPE	CHARACTER	Histogrammtyp (F=Frequenzhistogramm, Q=Quantilhistogramm)
SEQNO	SMALLINT	Bucket-Nummer
COLVALUE	VARCHAR(254)	Wert, für den Frequenzstatistiken erstellt werden bzw. k-Quantil, für den Quantilstatistiken erstellt werden (Bucket-Obergrenze) NULL wenn keine Spaltenwerte in einem Bucket vorhanden.
VALCOUNT	BIGINT	Anzahl der Vorkommen (Frequenz) des im COLVALUE gespeicherten Wertes in der Spalte bzw. Anzahl der Spaltenwerte die kleiner als im COLVALUE gespeicherter Wert sind
DISTCOUNT	INTEGER	Anzahl unterschiedlicher Werte im Bucket

3.1.6 Kostenschätzwerte und Anfragenkardinalitäten

Mit der von IBM DB2 Version 8.2 angebotenen so genannten “Explain” - Funktionalität erhält man Zugriff auf die vom Optimierer gewählten Zugriffspläne sowie auf einige für die Optimierung hilfreiche Schätzwerte. Alle Daten werden dabei als Explain-Instanzen für jeweils eine oder mehrere SQL-Anfrage repräsentiert. Jede Explain-Instanz enthält Information über die SQL Kompilierungsumgebung und den für die Anfrage gewählten Zugriffsplan. Im weiteren Verlauf wird die Syntax vom Explain-Befehl anhand von Beispielen erklärt.

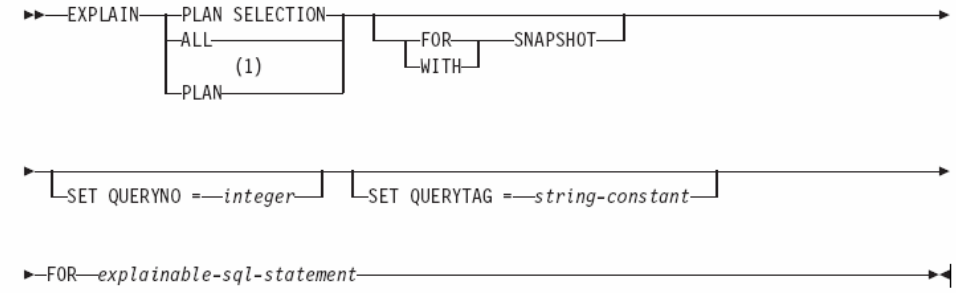


Abbildung 3.1: Explain-Syntax [IBM04b]

Beispiel 1:

```
EXPLAIN PLAN SET QUERYNO = 13  FOR SELECT C1  FROM T1;
```

Beispiel 2:

```
EXPLAIN PLAN SELECTION SET QUERYTAG ='TEST13' FOR SELECT C1 FROM T1;
```

Beispiel 3:

```
EXPLAIN PLAN SELECTION SET QUERYNO = 1
                       SET QUERYTAG ='APITEST'
FOR SELECT C1 FROM T1;
```

Nach einem EXPLAIN Aufruf werden alle Explain-Informationen (z.B. CPU_COST, oder IO_COST, usw.) in den nachfolgenden Tabellen gespeichert. Außer diesen Tabellen sind noch weitere Explain-Tabellen vorhanden, die jedoch für diese Arbeit nicht von Interesse sind.

Tabelle 3.5 : Tabellen mit Explain-Daten

Tabellenname	**Beschreibung**
EXPLAIN_ARGUMENT	Enthält individuelle Charakteristika für jeden verfügbaren Operator
EXPLAIN_INSTANCE	Die Haupttabelle für alle Explain-Informationen. Alle anderen Tabellen enthalten explizite Links auf eine der Zeilen in dieser Tabelle. Sowohl der Anfragenquelltext als auch die Kompilierungsumgebung sind in dieser Tabelle gespeichert.
EXPLAIN_OBJECT	Enthält für die Generierung des Zugriffplans notwendigen Datenobjekte
EXPLAIN_OPERATOR	Enthält alle für die Anfragekompilierung notwendigen SQL Operatoren.
EXPLAIN_PREDICATE	Enthält von einem Operator anwendbare Prädikate
EXPLAIN_STATEMENT	Enthält Texte der auf verschiedenen Explain-Ebenen existierende SQL-Anfragen angefangen mit der Benutzer-Anfrage und während der Zugriffplanerstellung vom Optimierer generierten Zusatzinformationen (Snapshots)
EXPLAIN_STREAM	Enthält Eingangs- und Ausgangsdatenströme zwischen einzelnen Operatoren und Datenobjekten.

Mit diesem Wissen wird es nun möglich, die für diese Arbeit interessanten Kostenschätzwerte und Anfragenkardinalitäten aus den Explain-Tabellen abzufragen. Die nachfolgende Query ermittelt die Ausführungskosten TOTAL_COST und die Kardinalität der Anfrage STREAM_COUNT aus Tabellen EXPLAIN_STATEMENT, EXPLAIN_OPERATOR und EXPLAIN_STREAM. Die Anfrage wird anhand eines Labels (z.B. querytag = ’APITEST’) und einer Ordnungsnummer (z.B. queryno = 1) gekennzeichnet, um später mit Hilfe dieser Kennzeichnung auf die EXPLAIN Informationen zu dieser Anfrage zugreifen zu können.

```
SELECT stmt.total_cost, strm.stream_count
FROM explain_statement stmt, explain_operator oper, explain_stream strm
WHERE  stmt.querytag                = 'APITEST'
     AND stmt.queryno               = 1
     AND stmt.explain_level         = 'P'
     AND stmt.EXPLAIN_REQUESTER     = oper.EXPLAIN_REQUESTER
     AND stmt.EXPLAIN_TIME          = oper.EXPLAIN_TIME
     AND stmt.SOURCE_NAME           = oper.SOURCE_NAME
     AND stmt.SOURCE_SCHEMA         = oper.SOURCE_SCHEMA
     AND stmt.EXPLAIN_LEVEL         = oper.EXPLAIN_LEVEL
     AND stmt.STMTNO                = oper.STMTNO
     AND stmt.SECTNO                = oper.SECTNO
     AND oper.operator_type         = 'RETURN'
     AND oper.EXPLAIN_REQUESTER     = strm.EXPLAIN_REQUESTER
```

```
AND oper.EXPLAIN_TIME          = strm.EXPLAIN_TIME
AND oper.SOURCE_NAME           = strm.SOURCE_NAME
AND oper.SOURCE_SCHEMA         = strm.SOURCE_SCHEMA
AND oper.EXPLAIN_LEVEL         = strm.EXPLAIN_LEVEL
AND oper.STMTNO                = strm.STMTNO
AND oper.SECTNO                = strm.SECTNO
AND oper.operator_id           = strm.target_id
```

Der gesuchte Kostenwert ist explizit in der EXPLAIN_STATEMENT Tabelle gespeichert. Für die Kardinalität muss die Kante ermittelt werden, die in den Wurzelknoten (RETURN) des zugehörigen Ausführungsplans führt. Diese Länge dieser Kante STREAM_COUNT entspricht zahlenmäßig der Kardinalität der Anfrage.

3.1.7 Externe Zugriffsoperation und Integritätserhaltung

Wie bereits beschrieben erstellt und verwaltet der DB2 Datenbankmanager zwei Sätze an Sichten auf die Systemkatalog-Tabellen. Sichten aus dem SYSCAT Schema sind ausschließlich lesbar. SELECT-Zugriffsrecht an diesen Sichten ist standardmäßig auf PUBLIC festgelegt, d.h. für alle Benutzer erlaubt. Alle SYSSTAT Sichten dürfen Benutzer und ausschließlich mittels UPDATE Anweisungen modifiziert werden.

Diese Sichten enthalten vom Optimierer verwendete statistische Informationen. Die Datenwerte in einigen Spalten dürfen zwecks Performanzoptimierung mit gewissen Einschränkungen mit UPDATE modifiziert werden. Vor jeder Änderung der Statistikwerte ist es ratsam, RUNSTATS einzusetzen, um den neusten Stand aller Statistiken zu gewährleisten.

Die folgenden allgemeinen Prüfungen sollten vor der Aktualisierung einer Katalogstatistik in IBM DB2 V8.2 durchgeführt werden:

1. Numerische Statistikdaten müssen -1 bzw. größer oder gleich 0 sein, d.h. der Wertebereich dieser Statistiken lässt sich durch {-1} U [0, m] definieren, wobei m den für den jeweiligen Datentyp größtmöglichen Wert darstellt. Wert -1 liegt vor, wenn Statistiken nicht vorhanden oder noch nicht erstellt sind.

2. Numerische Statistikdaten, die Prozentwerte darstellen (z. B. CLUSTERRATIO in der Katalogsicht SYSSTAT.INDEXES), müssen zwischen 0 und 100 liegen. Der Wertebereich dieser Statistiken lässt sich durch {-1} U [0,100] für Gleitkommazahlen definieren.

Regeln für die manuelle Aktualisierung von Tabellenstatistiken:

1. $\forall$ Tabellen t $\forall$ Tabellenspalten s € t: $CARD_t \geq COLCARD_s$,
   (Anzahl der Tabellenzeilen ≥ Anzahl unterschiedlicher Spaltenwerte)

2. $\forall$ Tabellen t: $CARD_t > NPAGES_t$

3. $\forall$ Tabellen t: $FPAGES_t > NPAGES_t$

4. $\forall$ Tabellen t: $FPAGES_t - NPAGES_t$ = Anzahl nicht belegter Seiten

5. $\forall$ Tabellenindexe i: $NPAGES_t \leq PAGE\_FETCH\_PAIRS_i$

6. $\forall$ Tabellenindexe i, $CARD_t > PAGE\_FETCH\_PAIRS_i$

Regeln für die manuelle Aktualisierung von Spaltenstatistiken:

1. $\forall$ Spalten s: HIGH2KEY, LOW2KEY liegen im Wertebereich des Datentyps der Spalte

2. $\forall$ Spalten s:
   $length(HIGH2KEY_s) \leq MIN(33, length(Datentyp))$
   $length(LOW2KEY_s) \leq MIN(33, length(Datentyp))$
   (length() = Wertlänge, zusätzliche Anführungszeichen werden nicht mitgezählt)

3. $\forall$ x € {VARCHAR} $\vee$ x € {CHAR} $\Rightarrow$ Speicherung der HIGH2KEY bzw. LOW2KEY-Werte mit zusätzlichen Anführungszeichen
   (z.B. abc => 'abc', ab'c => 'ab''c')

4. $\forall$ Spalten s: $HIGH2KEY_s > LOW2KEY_s$, distinct(s) > 3
   (distinct() = Anzahl untersch. Spaltenwerte)

5. $\forall$ Tabellen t $\forall$ Tabellenspalten s € t: $COLCARD_s \geq CARD_t$,

6. $\forall$ Tabellen t $\forall$ Tabellenspalten s € t: $NUMNULLS_s \leq CARD_t$

7. $AVGCOLLEN_{alt} \geq AVGCOLLEN_{neu}$

8. Keine Statistiken für LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB

Regeln für die manuelle Aktualisierung von Indexstatistiken:

1. FIRSTKEYCARD = FULLKEYCARD (für einspaltigen Index)

2. FIRSTKEYCARD = COLCARD

3. FIRSTKEYCARD ≤ FIRST2KEYCARD ≤ FIRST3KEYCARD ≤ FIRST4KEYCARD ≤ FULLKEYCARD ≤ CARD (für mehrspaltige Indizes)

4. FIRST2KEYCARD = FIRST3KEYCARD = FIRST4KEYCARD = -1, falls Statistik nicht benötigt (z.B. für dreispaltigen Index FIRST4KEYCARD = -1)

5. SEQUENTIAL_PAGES € {-1} U [0, NLEAF]

6. DENSITY € {-1} U [0, 100], DENSITY € $\Re$ ($\Re$ Menge der reellen Zahlen)

7. Trennung der Einzelwerte durch Leerzeichen in PAGE_FETCH_PAIRS

8. ∀ Einzelwert aus PAGE_FETCH_PAIRS x: length(x) ≤ 10 Λ
   x ≤ MAXINT, MAXINT = 2147483647
   (length() = Wertlänge)

9. CLUSTERFACTOR > 0 ⇒ ∃PAGE_FETCH_PAIRS

10. Es müssen sich genau 11 Wertepaare in einer einzelnen Statistik PAGE_FETCH_PAIR befinden.

11. Die Einträge für die Puffergrößen in PAGE_FETCH_PAIRS müssen eine aufsteigende Wertefolge bilden.

12. ∀ x € PAGE_FETCH_PAIRS ≤ MIN(NPAGES, 524287), 32Bit Betriebssys.
    ∀ x € PAGE_FETCH_PAIRS ≤ MIN(NPAGES, 2147483647), 64Bit BS

13. ∀ Fetches f € PAGE_FETCH_PAIRS, Alle f sind absteigend sortiert

14. ∀ Tabellen ∀ Fetches f, f ≥ $NPAGES_t$

15. ∀ Tabellen t ∀ Fetch Sizes u € PAGE_FETCH_PAIRS, u ≤ $CARD_t$

16. Wenn der Wert für „buffer size“ in zwei aufeinander folgenden Paaren übereinstimmt, muss der Wert für „page fetch“ in beiden Paaren ebenfalls übereinstimmen.

17. CLUSTERRATIO € {-1} U [0, 100], CLUSTERRATIO € ℜ

18. CLUSTERFACTOR € {-1} U [0, 1], CLUSTERFACTOR € ℜ

19. Es muss immer mindestens einer der Werte für CLUSTERRATIO und CLUSTERFACTOR gleich -1 sein.

20. Wenn für CLUSTERFACTOR ein positiver Wert angegeben ist, müssen gültige Statistikdaten in der Spalte PAGE_FETCH_PAIR enthalten sein.

Regeln für die manuelle Aktualisierung von Verteilungsstatistiken:

Für Frequenzhistogramme:

1. Die Werte in der Spalte VALCOUNT müssen bei steigenden Werten für SEQNO unverändert bleiben oder abnehmen.

2. COLVALUE ≤ COLCARD € SYSSTAT.COLUMNS

3. sum(VALCOUNT) ≤ CARD € SYSSTAT.TABLES (sum() = Summe aller Werte)

4. LOW2KEY ≤ COLVALUE ≤ HIGH2KEY
   x, y € COLVALUE: ∃! x ≤ LOW2KEY Λ ∃! y ≥ HIGH2KEY

Für Quantilhistogramme:

1. Die Werte in der Spalte COLVALUE müssen bei steigenden Werten für SEQNO unverändert bleiben oder abnehmen.

2. Die Werte in der Spalte VALCOUNT müssen bei steigenden Werten für SEQNO steigen

3. $\exists v, v = COLCARD \wedge v = MAX(COLVALUE), v \in VALCOUNT$
   Der größte Wert in der Spalte COLVALUE muss einen entsprechenden Eintrag in der Spalte VALCOUNT haben, der gleich der Anzahl von Zeilen in der Spalte ist.

4. $LOW2KEY \leq COLVALUE \leq HIGH2KEY$
   $\exists! x, y \in COLVALUE : x \leq LOW2KEY \wedge y \geq HIGH2KEY,$

5. length(COLVALUE) ≤ 32 (length() = Anzahl Zeichen)

3.2 Oracle Version 10g

Die nachfolgenden Informationen wurden mit Hilfe von [ORA06a, ORA06b, ORA06c, ORA06d, ORA06e, PAD03] gesammelt. Bei der Analyse des Oracle V10G DBMSs wurde mit der gleichen Vorgehensweise wie bei IBM DB2 (s.o.) gearbeitet. D.h. zuerst wurde festgestellt, welche Statistikwerte im Oracle DBMS vorhanden sind. Daraufhin folgt eine Untersuchung der für diese Statistiken verfügbaren Zugriffsmethoden und existierende Einschränkungen untersucht.

3.2.1 Möglichkeiten der Statistikerstellung

Alle (Optimierer-)Statistiken eines Oracle 10g Datenbankmanagementsystems werden im Data Dictionary abgelegt. Der Zugriff auf diese Statistiken erfolgt über entsprechende Sichten.

Dem Benutzer eines Oracle 10g Systems stehen folgende Möglichkeiten der Statistikerstellung zur Verfügung:

- Manuelle Statistikerstellung mit Hilfe von DBMS_STATS. Dieses PL/SQL Paket bietet neben Prozeduren für die Erstellung von zahlreichen Katalogstatistiken (s. Tabelle 3.6) auch umfangreiche Funktionalität für den Lese- und Schreibzugriff auf diese Statistiken an.

  Beispiel:
  ```
  DBMS_STATS.GATHER_TABLE_STATS(OWNNAME=>'SYSMAN',
          TABNAME=>'APITEST',METHOD_OPT => 'FOR ALL COLUMNS SIZE 5');
  ```

Tabelle 3.6 : Statistikerstellung mit Prozeduren aus dem DBMS_STATS Paket

Prozedurbezeichnung	Was wird erstellt
GATHER_INDEX_STATS	Indexstatistiken
GATHER_TABLE_STATS	Tabellen-, Spalten und Indexstatistiken
GATHER_SCHEMA_STATS	Statistiken für alle Objekte in einem Schema
GATHER_DICTIONARY_STATS	Statistiken für alle Dictionary Objekte
GATHER_DATABASE_STATS	Statistiken für alle Objekte in der Datenbank

- Manuelle Statistikerstellung mit dem Analyze-Befehl. Alle Statistiken, die über die Prozedur GATHER_TABLE_STATS erstellt werden können, können alternativ mit dem Analyze-Befehl erstellt werden. Daneben werden einige zusätzliche Statistiken erzeugt (z.B. ENDPOINT_ACTUAL_VALUE s.u.), wobei man dadurch mit langsameren Ausführungszeiten dieses Befehls im Vergleich zu der GATHER_TABLE_STATS Prozedur rechnen muss.

  ```
  analyze table APITEST compute statistics for all columns size 5;
  ```

- Automatische Statistiksammlung mit GATHER_STATS_JOB. Dieser Oracle Job wird zusammen mit der Datenbank erstellt und vom Oracle Scheduler verwaltet. Die automatische Statistiksammlung mit GATHER_STATS_JOB lässt sich wie folgt aktivieren bzw. deaktivieren:

  Aktivierung: `DBMS_SCHEDULER.ENABLE('GATHER_STATS_JOB');`
  Deaktivierung: `DBMS_SCHEDULER.DISABLE('GATHER_STATS_JOB');`

- Der aktuelle Status von GATHER_STATS_JOB lässt sich mit der folgenden SQL-Anfrage überprüfen:

  ```
  SELECT * FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME = 'GATHER_STATS_JOB';
  ```

- (Auto-)Sampling mit ESTIMATE_PERCENT für DBMS_STATS Prozeduren

  ```
  DBMS_STATS.GATHER_SCHEMA_STATS('OE',DBMS_STATS.AUTO_SAMPLE_SIZE);
  ```

3.2.2 Tabellenstatistiken – ALL_TAB_STATISTICS

Alle bereits vorliegenden Tabellenstatistiken sind über die ALL_TAB_STATISTICS Sicht für alle Benutzer mit entsprechenden Rechten lesbar.

- USER_TAB_STATISTICS enthält ausschließlich Statistiken für Tabellen aus dem Benutzer-Schema und bildet eine Untermenge von ALL_TAB_STATISTICS

- DBA_TAB_STATISTICS enthält ausschließlich Statistiken für Tabellen aus dem DBA-Schema (Datenbankadministrator) und bildet eine Untermenge von ALL_TAB_STATISTICS

Tabelle 3.7: ALL_TAB_STATISTICS

Statistik	Datentyp	Beschreibung
OWNER	VARCHAR2(30)	Qualifizierte Schemabezeichnung
TABLE_NAME	VARCHAR2(30)	Qualifizierte Tabellenbezeichnung
PARTITION_NAME	VARCHAR2(30)	Qualifizierte Partitionbezeichnung
PARTITION_POSITION	NUMBER	Partitionposition
SUBPARTITION_NAME	VARCHAR2(30)	Subpartitionbezeichnung
SUBPARTITION_POSITION	NUMBER	Subpartitionposition
OBJECT_TYPE	VARCHAR2(12)	Objekttyp € {Tabelle, Partition, Subpartition}
NUM_ROWS	NUMBER	Anzahl der Zeilen in der Tabelle
BLOCKS	NUMBER	Anzahl von der Tabelle belegter Blöcke (Seiten)
EMPTY_BLOCKS	NUMBER	Anzahl leerer Blöcke (Seiten) ohne Tabellenzeilen
AVG_SPACE	NUMBER	Verfügbarer freier Speicher (Durchschnittswert) im Objekt
CHAIN_CNT	NUMBER	Anzahl verketteter Zeitelen im Objekt
AVG_ROW_LEN	NUMBER	Mittlere Zeilenlänge
AVG_SPACE_FREELIST_BLOCK	NUMBER	Verfügbarer freier Speicher (Durchschnittswert) für alle Blöcke
NUM_FREELIST_BLOCKS	NUMBER	Anzahl der freien (in der Freispeicherliste eingetragener) Blöcke
AVG_CACHED_BLOCKS	NUMBER	Durchschnittliche Anzahl der Blöcke im (Buffer)Cache
AVG_CACHE_HIT_RATIO	NUMBER	Durchschnittlicher Prozentsatz der Cachezugriff
SAMPLE_SIZE	NUMBER	Sample-Größe
LAST_ANALYZED	DATE	Datum und Uhrzeit der letzten Statistikerstellung
GLOBAL_STATS	VARCHAR2(3)	Statistikerstellung erfolgte für alle Partitionen als Ganzes (Yes/No)
USER_STATS	VARCHAR2(3)	Statistikerstellung durch Benutzer initiiert (Yes/No)
STATTYPE_LOCKED	VARCHAR2(5)	Typ der Statistiksperre € {DATA, CACHE, ALL}

Anmerkung: Für die Erstellung einer prototypischen Implementierung wird zur Vereinfachung und zusätzlich zu den in 2.3 genannten Voraussetzungen angenommen, dass jede Tabelle in einer Partition abgelegt wird. D.h. PARTITION_NAME, PARTITION_POSITION, SUBPARTITION_NAME und SUBPARTITION_POSITION werden nicht berücksichtigt.

3.2.3 Spaltenstatistiken – ALL_TAB_COL_STATISTICS

ALL_TAB_COL_STATISTICS Sicht enthält Spaltenstatistiken und Informationen zu den Verteilungsstatistiken der entsprechenden Spalte. Diese Sicht ist für alle Benutzer mit entsprechenden Rechten lesbar.

- USER_TAB_COL_STATISTICS enthält ausschließlich Statistiken für Tabellenspalten aus dem Benutzer-Schema und bildet eine Untermenge von ALL_TAB_COL_STATISTICS

- DBA_TAB_COL_STATISTICS enthält ausschließlich Statistiken für Tabellenspalten aus dem DBA-Schema (Datenbankadministrator) und bildet eine Untermenge von ALL_TAB_ COL_STATISTICS

Tabelle 3.8: ALL_TAB_COL_STATISTICS

Statistik	Datentyp	Beschreibung
OWNER	VARCHAR2(30)	Qualifizierte Schemabezeichnung / Besitzerinformationen
TABLE_NAME	VARCHAR2(30)	Qualifizierte Tabellenbezeichnung

COLUMN_NAME	VARCHAR2(30)	Qualifizierte Spaltenbezeichnung
NUM_DISTINCT	NUMBER	Anzahl unterschiedlicher Werte in der Spalte
LOW_VALUE	RAW(32)	2. Minimum / Zweitniedrigster Wert der Spalte
HIGH_VALUE	RAW(32)	2. Maximum / Zweithöchster Wert der Spalte
DENSITY	NUMBER	Informationen zur Spaltendichte
NUM_NULLS	NUMBER	Anzahl der Nullwerte in der Spalte
NUM_BUCKETS	NUMBER	Anzahl der Buckets in der Spaltenverteilungsstatistik
LAST_ANALYZED	DATE	Datum und Uhrzeit der letzten Statistikerstellung
SAMPLE_SIZE	NUMBER	Sample-Größe
GLOBAL_STATS	VARCHAR2(3)	Statistikerstellung erfolgte für alle Partitionen als Ganzes (Yes/No)
USER_STATS	VARCHAR2(3)	Statistikerstellung durch Benutzer initiiert (Yes/No)
AVG_COL_LEN	NUMBER	Mittlere Spaltenlänge
HISTOGRAM	VARCHAR2(15)	Histogrammtyp € {NONE, FREQUENCY, HEIGHT BALANCED}

3.2.4 Indexstatistiken – ALL_IND_STATISTICS

ALL_IND_STATISTICS Sicht enthält Indexstatistiken. Diese Sicht ist für alle Benutzer mit entsprechenden Rechten lesbar.

- USER_IND_STATISTICS enthält ausschließlich Statistiken für Indexe aus dem Benutz-Schema und bildet eine Untermenge von ALL_IND_STATISTICS
- DBA_IND_STATISTICS enthält ausschließlich Indexstatistiken aus dem DBA-Schema (Datenbankadministrator) und bildet eine Untermenge von ALL_IND_STATISTICS

Tabelle 3.9: ALL_IND_STATISTICS

Statistik	**Datentyp**	**Beschreibung**
OWNER	VARCHAR2(30)	Qualifizierte Schemabezeichnung / Besitzerinformationen
INDEX_NAME	VARCHAR2(30)	Qualifizierte Indexbezeichnung
PARTITION_NAME	VARCHAR2(30)	Qualifizierte Partitionbezeichnung
PARTITION_POSITION	NUMBER	Partitionposition
SUBPARTITION_NAME	VARCHAR2(30)	Subpartitionbezeichnung
SUBPARTITION_POSITION	NUMBER	Subpartitionposition
OBJECT_TYPE	VARCHAR2(12)	Objekttyp € {Index, Partition, Subpartition}
BLEVEL	NUMBER	Ebene im B-Baum
LEAF_BLOCKS	NUMBER	Anzahl der Blattseiten (Blöcke) im Index
DISTINCT_KEYS	NUMBER	Anzahl unterschiedl. Werte / Schlüssel im Index
AVG_LEAF_BLOCKS_PER_KEY	NUMBER	Durchschnittliche Anzahl der Blattblöcke pro Schlüssel
AVG_DATA_BLOCKS_PER_KEY	NUMBER	Durchschnittliche Anzahl der Datenblöcke pro Schlüssel
CLUSTERING_FACTOR	NUMBER	Gibt an, wie gut/schlecht die Tabellenzeilen geordnet sind
NUM_ROWS	NUMBER	Anzahl der Zeilen im Index
AVG_CACHED_BLOCKS	NUMBER	Durchschnittliche Anzahl der Blöcke im (Buffer)Cache
AVG_CACHE_HIT_RATIO	NUMBER	Durchschnittlicher Prozentsatz der Cachezugriffe
SAMPLE_SIZE	NUMBER	Sample-Größe
LAST_ANALYZED	DATE	Datum und Uhrzeit der letzten Statistikerstellung
GLOBAL_STATS	VARCHAR2(3)	Statistikerstellung erfolgte für alle Partitionen als Ganzes (Yes/No)

USER_STATS	VARCHAR2(3)	Statistikerstellung durch Benutzer initiiert (Yes/No)

3.2.5 Verteilungsstatistiken (Histogramme) – ALL_TAB_HISTOGRAMS

Verteilungsstatistiken bzw. Histogramme für alle dem aktuellen Benutzer zugängliche Tabellen und Sichten sind in ALL_TAB_HISTOGRAMS gespeichert. Oracle unterscheidet zwischen Frequenz- und höhenbalancierten Histogrammen (Quantilhistogrammen). Der Histogrammtyp wird über die HISTOGRAM Spalte in *_TAB_ COL_STATISTICS festgelegt. Sollte eine Tabelle aus mehreren Partitionen bestehen, wird für jede dieser Partition ein eigenes Histogramm erstellt (s. 3.2.3)

- USER_ TAB_HISTOGRAMS enthält Histogramme aus dem Benutzereigenen Schema und bildet eine Untermenge von ALL_TAB_HISTOGRAMS
- DBA_TAB_HISTOGRAMS enthält Histogramme aus dem DBA-Schema und bildet eine Untermenge von ALL_ TAB_HISTOGRAMS

Tabelle 3.10 : ALL_TAB_HISTOGRAMS

Statistik	Datentyp	Beschreibung
OWNER	VARCHAR(128)	Qualifizierte Schemabezeichnung
TABLE_NAME	VARCHAR(128)	Qualifizierte Tabellenbezeichnung
COLUMN_NAME	VARCHAR(128)	Qualifizierte Spaltenbezeichnung
ENDPOINT_NUMBER	CHARACTER	Bucket-Nummer
ENDPOINT_VALUE	SMALLINT	Normalisierter k-Quantil der Quantilstatistiken (Bucket-Obergrenze)
ENDPOINT_ACTUAL_VALUE	VARCHAR(254)	Nicht normalisierter k-Quantil (Bucketobergrenze), falls Zeichenkette

Anmerkung: ENDPOINT_VALUE enthält normalisierte Werte (Bucket-Obergrenzen) aus analysierten Spalten. Bei Zeichenketten werden nur die ersten 5 Bytes der Zeichenkette in normalisierter Darstellung gespeichert. ENDPOINT_ACTUAL_VALUE enthält für solche Werte die Originalzeichenketten, falls die Verteilungsstatistiken mit dem analyze Befehl erstellt worden sind. In einigen Fällen wie z.B. bei der Benutzung von Get-Prozeduren aus DBMS_STATS werden keine Originalwerte in ENDPOINT_ACTUAL_VALUE bereitgestellt. Eine inoffizielle Lösung und von Oracle nicht bestätigte und leider nicht vollständige Lösung dieses Problems stammt aus einem Internet-Forum für Oracle Nutzer [OAT]. Dabei muss jeder normalisierte Datenwert aus der Spalte ENDPOINT_VALUE zuerst in Hexadezimal-Darstellung umgewandelt werden. Die auf diese Weise gewonnene Zahl wird als Zeichenkette von links nach rechts in einer Schleife gelesen. Je zwei Hex-Zeichenwerte entsprechen dabei dem ASCII-Code eines Zeichens aus der Originalzeichenkette. Mit diesem Algorithmus lassen sich bis zu fünf erste Zeichen der Zeichenkette ermitteln. Eine Implementierung dieser Methode wurde bei der Prototyperstellung erarbeitet (s. Kapitel 5).

3.2.6 Kostenschätzwerte und Anfragenkardinalitäten

Mit dem EXPLAIN PLAN Befehl verfügt das DBMS von Oracle über eine ausgereifte Möglichkeit, auf die vom Optimierer gewählten Ausführungspläne zuzugreifen. Diese Funktionalität steht der von DB2 Explain sehr nahe. Im weiteren Verlauf wird die Syntax vom Explain-Befehl anhand von Beispielen erklärt. Row Source Tree bildet den Kern jedes Ausführungsplans und enthält detaillierte Informationen über Anordnung der angesprochenen Tabellen, Zugriffs- und Join-Methoden für die von der Anfrage betroffenen Tabellen und verschiedene Datenoperationen wie z.B. Sortieren. Diese Informationen werden in der Plantabelle (PLAN_TABLE) abgelegt, welche außerdem für jede Operation die entsprechenden Kostenwerte und Kardinalität beinhaltet.

Um die Explain-Informationen auf einfache Art und Weise auszugeben, kann man entweder direkt auf die Plantabelle zugreifen oder das DBMS_XPLAN Paket nutzen. Die Anwendung des EXPLAIN PLAN Befehls und des DBMS_XPLAN Pakets wird an Beispielen erläutert:

```
EXPLAIN PLAN
  [ SET STATEMENT_ID = 'text' ]
  [ INTO [ schema. ]table [ @ dblink ] ]
  FOR statement ;
```

Abbildung 3.2: EXPLAIN PLAN - Syntax [ORA06d]

Beispiel 1:
```
EXPLAIN PLAN statement_id='API' FOR SELECT last_name FROM employees;
```

Beispiel 2: `SELECT COST FROM PLAN_TABLE WHERE statement_id='API'`
Ermittlung des Kostenschätzwertes für die durch statement_id='API' gekennzeichnete Anfrage.

Beispiel 3:
```
SELECT * FROM table(DBMS_XPLAN.DISPLAY('PLAN_TABLE','APITEST','ALL'));
```
Auslesen aller Daten aus PLAN_TABLE für Tabelle APITEST.

Mit der Angabe einer freiwählbaren STATEMENT_ID kann die Kennzeichnung der jeweiligen Anfrage vorgenommen. Die in Beispiel 3 verwendete PLAN_TABLE wird standardmäßig als Ausgabe-Tabelle für EXPLAN PLAN eingesetzt und automatisch als globale Tabelle angelegt, um alle Ergebnisse (z.B. Ausführungspläne) der EXPLAIN PLAN Anfragen für alle Benutzer zu speichern. Ein Auszug aus dieser Tabelle wird nachfolgend gezeigt:

Tabelle 3.11 : Auszug aus PLAN_TABLE

Statistik	Datentyp	Beschreibung
STATEMENT_ID	VARCHAR2(30)	In EXPLAIN PLAN angegebene STATEMENT_ID
PLAN_ID	NUMBER	Eindeutige Planbezeichnung in der Datenbank
TIMESTAMP	DATE	Datum und Uhrzeit der EXPLAIN PLAN Generierung
OPERATION	VARCHAR2(30)	Name der ausgeführten Operation

COST	NUMERIC	Vom Optimierer geschätzte Kosten der Anfrage. Dieser Wert hat keine Maßeinheit und errechnet sich als Funktion von CPU_COST und IO_COST Spalten.
CARDINALITY	NUMERIC	Geschätzte Kardinalität der Anfrage
CPU_COST	NUMERIC	Geschätzte von der Anfrage beanspruchte CPU-Kosten
IO_COST	NUMERIC	Geschätzte von der Anfrage beanspruchte I/O-Kosten

Die für diese Arbeit interessanten COST und CARDINALITY Werte können mit einfachen SELECT-Zugriffen ausgelesen werden.

3.2.7 Externe Zugriffsoperation und Integritätserhaltung

Alle bereits beschriebenen Statistiksichten erlauben ausschließlich Lesezugriff mittels einer SELECT-Anfrage. Des Weiteren können gleiche Statistiken mit Hilfe von GET-Prozeduren aus dem DBMS_STATS Paket gelesen werden:

- GET_TABLE_STATS Lesezugriff auf Tabellenstatistiken

```
DBMS_STATS.GET_TABLE_STATS (
   ownname     VARCHAR2,                -- Schema
   tabname     VARCHAR2,                -- Tabellenname
   partname    VARCHAR2 DEFAULT NULL,   -- Partitionname
   stattab     VARCHAR2 DEFAULT NULL,   -- Name externer Tabelle
   statid      VARCHAR2 DEFAULT NULL, -- StistikID in der ext.Tabelle
   numrows OUT NUMBER,                  -- Anzahl Zeilen
   numblks OUT NUMBER,                  -- Anzahl Blöcke
   avgrlen OUT NUMBER,                  -- Durchschn. Länge
   statown     VARCHAR2 DEFAULT NULL);  -- Schema der ext. Tabelle
```

- GET_COLUMN_STATS Lesezugriff auf Spaltenstatistiken und Histogramme

```
DBMS_STATS.GET_COLUMN_STATS (
   ownname     VARCHAR2,
   tabname     VARCHAR2,
   colname     VARCHAR2,                -- Spaltenname
   partname    VARCHAR2 DEFAULT NULL,
   stattab     VARCHAR2 DEFAULT NULL,
   statid      VARCHAR2 DEFAULT NULL,
   distcnt OUT NUMBER,                  -- Anzahl untersch. Werte
   density OUT NUMBER,                  -- Spaltendichte
   nullcnt OUT NUMBER,                  -- Anzahl NULL-Werte
   srec    OUT StatRec,                 -- StatRec-Struktur
   avgclen OUT NUMBER,                  -- Durchschn. Länge
   statown     VARCHAR2 DEFAULT NULL);
```

Neben einfachen Spaltenstatistiken werden mit Hilfe dieser Prozedur auch Histogrammdaten ausgelesen und in einer StatRec-Struktur abgelegt. StatRec lässt sich dabei für Histogramme mit Zahlenwerten wie folgt definieren:

```
type StatRec is record (
  epc    NUMBER,         -- Anzahl Buckets
  minval RAW(2000),      -- Minimum
  maxval RAW(2000),      -- Maximum
  bkvals NUMARRAY,       -- Bucket-Nummern
  novals NUMARRAY);      -- Bucket-Obergrenzen

TYPE numarray  IS VARRAY(256) OF NUMBER;
```

oder für Histogramme mit Zeichen(-ketten)werten

```
type StatRec is record (
  epc    NUMBER,
  minval RAW(2000),
  maxval RAW(2000),
  bkvals NUMARRAY,
  novals CHARARRAY);

TYPE chararray IS VARRAY(256) OF VARCHAR2(4000);
```

- GET_INDEX_STATS Lesezugriff auf Indexstatistiken

```
DBMS_STATS.GET_INDEX_STATS (
   ownname      VARCHAR2,
   indname      VARCHAR2,-- Indexbezeichnunug
   partname     VARCHAR2 DEFAULT NULL,
   stattab      VARCHAR2 DEFAULT NULL,
   statid       VARCHAR2 DEFAULT NULL,
   numrows  OUT NUMBER,  -- Anzahl Blöcke
   numlblks OUT NUMBER,  -- Anzahl Zeilen
   numdist  OUT NUMBER,  -- Anzahl untersch. Schlüssel
   avglblk  OUT NUMBER,  -- Durchsch. Anzahl der Blattblöcke pro Schl.
   avgdblk  OUT NUMBER,  -- Durchsch. Anzahl der Datenblöcke pro Schl.
   clstfct  OUT NUMBER,  -- Clustering-Faktor
   indlevel OUT NUMBER,  -- Höhe vom Index (Partition)
   statown      VARCHAR2 DEFAULT NULL);
```

Beispiel: Es werden Spaltenstatistiken (Anzahl Nullwerte l_nullcnt, Anzahl unterschiedlicher Werte in der Spalte l_distcnt, Spaltendichte l_density und durchschnittliche Spaltenlänge l_avgclen für die Spalte PUNKTE (Tabelle APITEST) abgerufen. Histogrammdaten sind in l_rec zu finden.

Beispiel eines Lesezugriffs auf Spaltenstatistiken in Oracle DBMS

```
declare
  l_distcnt number;
  l_density number;
  l_nullcnt number;
  l_srec    dbms_stats.statrec;
  l_avgclen number;

  begin
    dbms_stats.get_column_stats(ownname => 'SYSMAN', tabname => 'APITEST',
                                colname => 'PUNKTE', distcnt => l_distcnt,
                                density => l_density, nullcnt => l_nullcnt,
                                srec => l_srec, avgclen => l_avgclen);
end;
```

In DBMS_STATS Paket sind außerdem weitere Prozeduren zum Ausführen folgender Operationen zu finden:

- Modifizieren/Aktualisieren (SET_*)
- Löschen (DELETE_*),
- Exportieren (EXPORT_*)

- Importieren (IMPORT_*),
- Sperren (LOCK_*)
- Entsperren (UNLOCK_*),
- Wiederherstellen (RESTORE_*)

Die SET-Prozeduren versetzen den Benutzer in die Lage, schreibend auf statistische Informationen zuzugreifen. Die SET-Funktionalität umfasst folgende Prozeduren:

- SET_TABLE_STATS Schreibzugriff auf Tabellenstatistiken

```
DBMS_STATS.SET_TABLE_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2 DEFAULT NULL,
   statid   VARCHAR2 DEFAULT NULL,
   numrows  NUMBER   DEFAULT NULL,
   numblks  NUMBER   DEFAULT NULL,
   avgrlen  NUMBER   DEFAULT NULL,
   flags    NUMBER   DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

- SET_COLUMN_STATS Schreibzugriff auf Spaltenstatistiken. Diese Prozedur ist Oracle die einzige Möglichkeit, Histogrammdaten und Maximum- bzw. Minimum-Werte einer Spalte zu aktualisieren [BR04].

```
DBMS_STATS.SET_COLUMN_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   colname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2 DEFAULT NULL,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL,
   distcnt  NUMBER   DEFAULT NULL,
   density  NUMBER   DEFAULT NULL,
   nullcnt  NUMBER   DEFAULT NULL,
   srec     StatRec  DEFAULT NULL,
   avgclen  NUMBER   DEFAULT NULL,
   flags    NUMBER   DEFAULT NULL, -- nur interne Oracle-Zwecke
   statown  VARCHAR2 DEFAULT NULL);
```

- SET_INDEX_STATS Schreibzugriff auf Indexstatistiken

```
DBMS_STATS.SET_INDEX_STATS (
   ownname  VARCHAR2,
   indname  VARCHAR2,
   partname VARCHAR2  DEFAULT NULL,
   stattab  VARCHAR2  DEFAULT NULL,
   statid   VARCHAR2  DEFAULT NULL,
   numrows  NUMBER    DEFAULT NULL,
   numlblks NUMBER    DEFAULT NULL,
   numdist  NUMBER    DEFAULT NULL,
   avglblk  NUMBER    DEFAULT NULL,
   avgdblk  NUMBER    DEFAULT NULL,
   clstfct  NUMBER    DEFAULT NULL,
   indlevel NUMBER    DEFAULT NULL,
```

```
    flags     NUMBER    DEFAULT NULL, -- nur interne Oracle-Zwecke
    statown   VARCHAR2  DEFAULT NULL);
```

Der Benutzer kann unter Angabe der Parameter statown, stattab und statid auf Statistiken außerhalb vom Data Dictionary zugreifen. Statown spezifiziert ein Schema und stattab eine externe Tabelle für Schreib- bzw. Leseoperationen. Mit statid können dann mehrere Sätze mit Statistikdaten verwaltet werden. Die auf diese Weise außerhalb vom Data Dictionary gespeicherten oder modifizierten Statistiken werden vom Optimierer nicht beachtet. Wenn diese Parameter fehlen, werden alle Zugriffe direkt im Data Dictionary durchgeführt.

Beispiel: In Abbildung 3.4 wird ein Schreibzugriff auf Spalten- und Verteilungsstatistiken demonstriert. Vor dem Einsatz der SET_COLUMN_STATS Prozedur müssen alle Felder der StatRec-Struktur mit entsprechenden Werten belegt werden. Mittels DBMS_STATS.PREPARE_COLUMN_VALUES erfolgt dann die Umwandlung dieser Werte in die Oracle interne RAW-Darstellung.

```
select n2, count(0)
from tp1 group by n2;
```

N2	COUNT(0)
0	99943
998	33
999	25

```
DECLARE
   SREC DBMS_STATS.STATREC;
   NOVALS DBMS_STATS.NUMARRAY;
BEGIN
   SREC.EAVS := 0;
   SREC.CHVALS := NULL;
   SREC.EPC := 3;
   NOVALS := DBMS_STATS.NUMARRAY(0,998,999);
   SREC.BKVALS := DBMS_STATS.NUMARRAY(99943,33,25);
   DBMS_STATS.PREPARE_COLUMN_VALUES (SREC,NOVALS);
   DBMS_STATS.SET_COLUMN_STATS(NULL, 'TP1', 'N2', NULL,
      NULL, NULL, 3, .000049373, 0, SREC, 2, 2);
   END;
```

global column statistics

table	column	NDV	density	nulls	lo	hi	av lg	bkts	G	U
TP1	N1	10,000	1.0000E-04	0	0	9999	3	1	N	N
	N2	3	4.9373E-05	0	0	999	2	2	Y	N
	N3	1,000	1.0000E-03	0	0	999	3	1	N	N

global frequency histogram

table	column	EP	value
TP1	N2	99943	0
TP1	N2	99976	998
TP1	N2	100001	999

Abbildung 3.4: Schreibzugriff mit SET_COLUMNS_STATS - Syntax [BR04].

Wenn der Minimal- bzw. Maximalwert einer Spalte ohne Änderung der Verteilungsstatistiken aktualisiert werden soll, dann wird dieser in MINVAL bzw. MAXVAL der StatRec-Struktur geschrieben. Sie werden dann zusammen mit den zuvor ausgelesenen Verteilungsstatistiken der betroffenen Spalte und eventuell weiteren Spaltenstatistiken an SET_COLUMNS_STATS übergeben. Das folgende Beispiel soll diese Vorgehensweise verdeutlichen:

Beispiel eines Schreibzugriffs auf Min-/Max-Werte einer Spalte

```
declare
  DISTCNT number;
```

```
  DENSITY number;
  NULLCNT number;
  AVGCLEN number;
  SREC    DBMS_STATS.statrec;

  begin
    -- Zuerst vorhandene Daten auslesen
    DBMS_STATS.get_column_stats(ownname => 'SYSMAN', tabname => 'APITEST',
                                colname => 'NAME', distcnt => DISTCNT,
                                density => DENSITY, nullcnt => NULLCNT,
                                   SREC => SREC, avgclen => AVGCLEN);

   -- Und jetzt min/max setzen
   SREC.MINVAL := UTL_RAW.CAST_TO_RAW('uuu');
   SREC.MAXVAL := UTL_RAW.CAST_TO_RAW('xxx');

   -- Schreiben der Struktur zurück in die DB
   DBMS_STATS.SET_COLUMN_STATS('SYSMAN','APITEST','NAME',NULL,NULL,
                               NULL,DISTCNT,DENSITY,NULLCNT,SREC);
end;
```

Werden Statistikwerte mittels der oben genannten SET-Prozeduren aktualisiert oder zum ersten Mal erstellt, sollten unbedingt folgende Bedingungen beachtet werden:

1. Numerische Statistikdaten müssen größer oder gleich 0 sein, d.h. der Wertebereich dieser Statistiken lässt sich durch [0, MAX] definieren, wobei MAX den maximalmöglichen Wert vom Typ NUMBER darstellt

2. Numerische Statistikdaten, die Prozentwerte darstellen (z. B. AVG_CACHE_HIT_RATIO in der Katalogsicht ALL_TAB_STATISTICS), müssen zwischen 0 und 100 liegen. Der Wertebereich dieser Statistiken lässt sich durch [0,100] für Gleitkommazahlen definieren.

Regeln für die manuelle Aktualisierung von Tabellenstatistiken:

1. ∀Tabellen t, ∀Tabellenspalten s € t: $NUM\_ROWS_t \geq NUM\_DISTINCT_s$, NUM_DISTINCT € *_TAB_COL_STATISTICS
   Anzahl der Tabellenzeilen ≥ Anzahl unterschiedlicher Spaltenwerte

2. ∀ Tabellen t: $NUM\_ROWS_t > BLOCKS_t$

3. ∀ Tabellen t: $BLOCKS_t + EMPTY\_BLOCKS_t$ = Gesamtzahl der Blöke

4. AVG_CACHE_HIT_RATIO € [0, 100], AVG_CACHE_HIT_RATIO € ℜ

Regeln für die manuelle Aktualisierung von Spaltenstatistiken:

1. ∀ Spalten s: $HIGH\_VALUE_s$, LOW_VALUE_s liegen im Wertebereich des Datentyps der Spalte

2. ∀ Spalten s: length($HIGH\_VALUE_s$) ≤ 32, length(LOW_VALUE_s) ≤ 32
   (length() = Wertlänge)

3. ∀ Spalten s: $HIGH\_VALUE_s > LOW\_VALUE_s$, distinct(s) > 3
   (distinct() = Anzahl untersch. Spaltenwerte)

4. ∀Tabellen t, ∀Tabellenspalten s € t: $NUM\_DISTINCT_s \leq NUM\_ROWS_t$

5. ∀Tabellen t, ∀Tabellenspalten s € t: $NUM\_NULLS_s \leq NUM\_ROWS_t$

6. AVG_CACHE_HIT_RATIO € [0, 100], AVG_CACHE_HIT_RATIO € ℜ

7. DENSITY € [0, 100], DENSITY € ℜ

8. $AVG\_COL\_LEN_{alt} \geq AVG\_COL\_LEN_{neu}$

Regeln für die manuelle Aktualisierung von Indexstatistiken:

1. Anzahl unterschiedl. Werte bzw. Schlüssel € NUMBER

2. CLUSTERING_FACTOR € [0, 1], CLUSTERING_FACTOR € ℜ

Regeln für die manuelle Aktualisierung von Verteilungsstatistiken:

1. Histogrammwerte liegen im Wertebereich des Spaltendatentyps

2. ∀ Histogrammwerte h: length(h) ≤ 32 (length() = Wertlänge)

3. LOW_VALUE ≤ ENDPOINT_VALUE ≤ HIGH_VALUE

3.3 Microsoft SQL Server 2005 (Yukon)

Im diesem Abschnitt wird das Microsoft SQL Server 2005 Datenbankmanagementsystem unter Verwendung von [HK05, MSDN, RJB02] analysiert. Einzelne Ergebnisse der Analyse werden im Folgenden dargestellt.

3.3.1 Möglichkeiten der Statistikerstellung

Alle statistischen Informationen im Microsoft SQL Server 2005 Datenbankmanagementsystem werden in Form von Objekten gespeichert. Diese Objekte tragen Bezeichnung „statblobs“ (Statistics Binary Large Objects) und sind in der internen Katalogsicht sys.sysobjvalues zu finden. Alle Statistiktabellen (wie z.B. sys.stats) verweisen mittels einer ObjectID auf diese Statistikobjekte.

Dem Datenbankbenutzer mit entsprechenden Rechten stehen folgende Möglichkeiten der Statistikerstellung zur Verfügung:

- Manuelle Statistikerstellung und Modifikation

    - CREATE STATISTICS
    - UPDATE STATISTICS
    - DROP STATISTICS
    - CREATE INDEX
    - DROP INDEX

  Beispiel: Exakte Statistiken für alle Spalten der Tabelle APITEST

```
STATISTICS stats
ON APITEST(ID,NAME,[ALTER],PUNKTE)
WITH FULLSCAN, NORECOMPUTE
```

- Manuelle Statistikerstellung und Aktualisierung für alle Spalten in allen Tabellen mit sp_createstats und sp_updatestats

- Automatische Statistikerstellung wird standardmäßig durchgeführt. Die Steuerung erfolgt über den ALTER DATABASE Befehl mit verschiedenen Optionen:

    - AUTO_CREATE_STATISTICS und AUTO_UPDATE_STATISTICS
      Beispiel: ALTER DATABASE DIPLOM SET AUTO_UPDATE_STATISTICS OFF

    - sp_autostats-Methode ist veraltet und sollte nicht mehr eingesetzt werden

- (Auto-)Sampling mit einer standardmäßig vorgegebenen oder gewünschten Sampling-Rate. Die Sample-Größe wird mindestens mit 8MB angenommen.

3.3.2 Tabellenstatistiken – SYS.STATS

SYS.STATS beinhaltet alle im System angelegten Tabellenstatistiken. Diese sind ausschließlich für den Lesezugriff freigegeben und werden in statblob-Objekten in internen Katalogtabellen verwaltet.

Tabelle 3.12 : SYS.STATS

Statistik	Datentyp	Beschreibung
OBJECT_ID	int	Objekt ID des zugehörigen Statistikobjekts
NAME	sysname	Qualifizierte Statistikbezeichnung, eindeutig innerhalb Objekts
STATS_ID	int	Statistik ID, eindeutig innerhalb Objekts
AUTO_CREATED	bit	Statistikerstellung automatisch durch den Query Prozessor veranlasst
USER_CREATED	bit	Statistikerstellung durch Benutzer initiiert (yes/no)
NO_RECOMPUTE	bit	Statistikerstellung erfolgte mit NORECOMPUTE-Option

Zusätzlich zu diesen Tabellenstatistiken werden auf Tabellenebene noch weitere Statistiken erstellt, die kein Bestandteil eines Statistikobjekts sind. Der Optimierer nutzt diese Statistiken in einigen Fällen für die Kostenbestimmung von Anfragen:

- rows € sys.sysindexes, Anzahl der Zeilen in der Tabelle oder im Index
- dpages € sys.sysindexes**,** Anzahl von der Tabelle / vom Index belegter Seiten

3.3.3 Spaltenstatistiken – SYS.STATS_COLUMNS

SYS**.**STATS_COLUMNS beinhaltet alle im System angelegten Spaltenstatistiken. Diese sind ausschließlich für den Lesezugriff freigegeben.

Tabelle 3.13 : SYS.STATS_COLUMNS

Statistik	Datentyp	Beschreibung
OBJECT_ID	int	Objekt ID vom Objekt, welches diese Spalte enthält
STATS_ID	int	ID des Statistik Objekts, welches diese Spalte beinhaltet
STATS_COLUMN_ID	int	Ordnungsnummer (ID) innerhalb der Spaltenstatistikmenge
COLUMN_ID	int	Spalten ID aus sys.columns

Spaltenbezogene Statistiken werden in einem statblob-Objekt gespeichert. Dabei werden folgende Statistikwerte erfasst:

Tabelle 3.14 : Spalten- und Verteilungsstatistiken

Statistik	Datentyp	Beschreibung
NAME	varchar	Statistikbezeichnung
UPDATED	datetime	Datum und Uhrzeit der Statistikerstellung
ROWS	int	Anzahl für die Histogrammerstellung verwendeter Zeilen
ROWS SAMPLED	int	Anzahl für die Histogrammerstellung verwendeter Stellvertreterzeilen
STEPS	int	Anzahl der Histogrammschritte (Anzahl der Buckets)
DENSITY	float	Informationen zur Spaltendichte = 1/distinct()
AVERAGE KEY LENGTH	float	Durchschnittliche Schlüssellänge
STRING INDEX	varchar	Yes für Spalten mit Zeichendaten, sonst No

Werden bei der Statistikerstellung mehrere Spalten gleichzeitig analysiert, dann spricht man von mehrspaltigen Statistiken. Diese sind eine Besonderheit vom Microsoft SQL Servers und bestehen aus einem Histogramm für die erste zu analysierende Spalte, einem Dichtewerte für die 1. Spalte und zusätzlichen Dichtewerten für jede Spaltenkombination (Präfix). Alle diese Informationen werden zusammen mit den bereits beschriebenen Statistiken aus Tabelle 3.14 in einem statblob-Objekt gespeichert.

Tabelle 3.15 : Statistiken Spaltenstatistiken

Statistik	Datentyp	Beschreibung
ALL DENSITY	float	Spaltendichte für jede Spaltenkombination = 1/distinct()
AVERAGE LENGTH	float	Durchschnittliche Schlüssellänge für jede Spaltenkombination in Byte
COLUMNS	varchar	Spaltenkombinationen getrennt durch ein Komma

3.3.4 Indexstatistiken – SYS.STATS

Indexstatistiken werden zusammen mit Tabellenstatistiken (s. 3.3.2) in SYS.STATS abgelegt. Diese sind ausschließlich für den Lesezugriff freigegeben.

Tabelle 3.16 : SYS.STATS

Statistik	Datentyp	Beschreibung
OBJECT_ID	int	Objekt ID des für die Statistikerstellung verwendeten Objekts
NAME	sysname	Qualifizierte Statistikbezeichnung, eindeutig innerhalb Objekts
STATS_ID	int	Statistik ID, eindeutig innerhalb Objekts
AUTO_CREATED	bit	Statistikerstellung automatisch durch den Query Prozessor veranlasst
USER_CREATED	bit	Statistikerstellung durch Benutzer initiiert
NO_RECOMPUTE	bit	Statistikerstellung erfolgte mit NORECOMPUTE-Option

3.3.5 Histogramme – SYSCAT.COLDIST

Im Gegensatz zu den DB2 und Oracle Systemen verarbeitet Microsoft SQL Server 2005 DBMS nur Quantilhistogramme. Ein Histogramm (Verteilungsstatistik) in diesem DBMS kann bis zu 200 Datenwerte einer gegebenen Spalte enthalten. Alle Werte dieser Spalte bzw. nur stellvertretende Werte werden sortiert abgelegt. Diese sortierte Wertesequenz wird dann auf bis zu 199 meist unterschiedlich lange Intervalle (nicht höhenbalanciert) verteilt. Falls eine Tabellenspalte mehr als 200 Datenwerte enthält, werden stellvertretend Datenwerte als Bucket-Obergrenzen ausgewählt und im Histogramm gespeichert. Die folgende Tabelle beschreibt alle Attribute eines Histogramms:

Tabelle 3.17 : Elemente eines Histogramms

Statistik	Datentyp	Beschreibung
RANGE_HI_KEY	varchar	Bucket-Obergrenze
RANGE_ROWS	int	Anzahl der Werte im Bucket (Bucket-Nummer)
EQ_ROWS	int	Anzahl der Werte im Bucket, die der Bucket-Obergrenze gleichen
AVG_RANGE_ROWS	int	Durchschnittliche Anzahl der Zeilen pro distinct-Wert
DISTINCT_RANGE_ROWS	int	Anzahl unterschiedlicher Werte im Bucket (Bucket-Kardinalität), die von der Bucket-Obergrenze abweichen.

3.3.6 Kostenschätzwerte und Anfragenkardinalitäten

Die vom Optimierer ermittelten Ausführungspläne mit entsprechenden Ausführungskosten (TotalSubtreeCost) und Kardinalitäten (EstimateRows) von Anfragen werden vom Microsoft DBMS automatisch nach jeder Anfrage zurückgegeben, falls die SHOWPLAN_ALL Option aktiviert ist. Die Aktivierung sollte in einem separaten SQL Server Batch (Durchlauf) veranlasst werden. Die Ausführung der eigentlichen Anfragen wird nach der Aktivierung der SHOWPLAN_ALL Option unterbunden.

Tabelle 3.18 : Rückgaben nach Aktivierung der SHOWPLAN_ALL Option

Statistik	Datentyp	Beschreibung
STMTTEXT	VARCHAR2(30)	Darstellung der Anfrage in SQL-Notation
STMTID	NUMBER	Eindeutige Anfragen ID
ESTIMATEROWS	VARCHAR2(30)	Geschätzte Kardinalität der Anfrage
ESTIMATEIO	NUMERIC	Geschätzte von der Anfrage beanspruchte I/O-Kosten.
ESTIMATECPU	NUMERIC	Geschätzte von der Anfrage beanspruchte CPU-Kosten
AVGROWSIZE	NUMERIC	Durchschnittliche Datensatzgröße
TOTALSUBTREECOST	NUMERIC	Vom Optimierer geschätzte Kosten der Anfrage

Beispiel 1: Aktivierung/Deaktivierung der SHOWPLAN-Option
SET SHOWPLAN_ALL ON oder SET SHOWPLAN_ALL OFF

Beispiel 2: Bestimmung der geschätzten Kosten und der Anfragekardinalität
1) Aktivierung der SHOWPLAN_ALL Option in Batch 1: SET SHOWPLAN_ALL ON;
*2) Anfrage an das DBMS in Batch 2: SELECT * FROM APITEST;*
1) Deaktivierung der SHOWPLAN_ALL Option in Batch 3; SET SHOWPLAN_ALL OFF;

	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp
1	SELECT * FROM APITEST;	1	1	0	NULL	NULL
2	|–Table Scan(OBJECT:([DIPLOM].[dbo].[APITEST]))	1	2	1	Table Scan	Table Scan

DefinedValues	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost
NULL	17	NULL	NULL	NULL	0,0033007
[DIPLOM].[dbo].[APITEST].[ID], [DIPLOM].[dbo].[A...	17	0,003125	0,0001757	52	0,0033007

Abbildung 3.5: Ausgabe mit der aktivierten SHOWPLAN_ALL Option

3.3.7 Externe Zugriffsoperation und Integritätserhaltung

Der SQL Server 2005 erlaubt manuellen Lesezugriff auf alle bereits analysierten Statistiken. Um die Statistiken zwecks Auswertung abzufragen, wird eine Microsoft-Transact-SQL Prozedur namens DBCC SHOW_STATISTICS verwendet. Spalten rows und dpages aus sys.sysindexes können durch eine SELECT-Anfrage ausgelesen werden.

```
DBCC SHOW_STATISTICS ( 'table_name' | 'view_name' , target )
[ WITH [ NO_INFOMSGS ] < option > [ , n ] ]
< option > :: =     STAT_HEADER | DENSITY_VECTOR | HISTOGRAM
```

Abbildung 3.6: DBCC SHOW_STATISTICS - Syntax [MSDN]

Mit Hilfe von DBCC SHOW_STATISTICS werden Statistiken aus Tabellen 3.14, 3.15, 3.17 ausgelesen und in Abbildung 3.6 zu sehen als „Multidimensional RowSet“ zurückgegeben.

Beispiel 1: Zugriff auf Statistiken “apistats” für Tabelle APITEST (s. Abbildung 3.7)
USE DIPLOM; DBCC SHOW_STATISTICS ('APITEST', apistats);

Beispiel 2: Zugriff auf Statistiken für den Index “IDIND” (Index auf Spalte ID)
USE DIPLOM; DBCC SHOW_STATISTICS ('APITEST', IDIND);

Beispiel 3: Zugriff auf dpages-Statistik
SELECT name, dpages FROM sys.sysindexes WHERE name='APIID'

Anmerkung zum Schreibzugriff: Als einziges System der drei analysierten Systeme stellt Microsoft SQL Server 2005 keine Funktionalität zur Verfügung, mit deren Hilfe bestimmte Statistiken modifiziert werden können. Alle Statistikwerte lassen sich ausschließlich automatisch oder manuell über CREATE STATISTICS, UPDATE STATISTICS, usw. erstellen und aktualisieren.

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index
1	apistats	Mar 21 2006 11:15AM	17	17	17	0	21,82353	NO

	All density	Average Length	Columns
1	0,05882353	4	ID
2	0,05882353	10,52941	ID, NAME
3	0,05882353	18,05882	ID, NAME, ALTER
4	0,05882353	21,82353	ID, NAME, ALTER, PUNKTE

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	1	0	1	0	1
2	2	0	1	0	1
3	3	0	1	0	1
4	4	0	1	0	1
5	5	0	1	0	1
6	6	0	1	0	1
7	7	0	1	0	1
8	8	0	1	0	1
9	9	0	1	0	1
10	10	0	1	0	1
11	11	0	1	0	1
12	12	0	1	0	1
13	13	0	1	0	1
14	14	0	1	0	1
15	15	0	1	0	1
16	16	0	1	0	1
17	17	0	1	0	1

Abbildung 3.7: Ergebnis einer "DBCC SHOW_STATISTICS"-Ausführung

3.4 Zusammenfassung der gewonnenen Daten

Die im bisherigen Verlauf dieser Arbeit aus verschiedenen DBMSs gewonnenen Informationen über Statistiken und Kostenschätzwerte werden an dieser Stelle zusammengefasst. Daraufhin sollte aus der Zusammenfassung dieser Werte eine einheitliche Struktur für die zu entwickelnde Schnittstelle entstehen.

3.4.1 Tabellenstatistiken

Die folgende Tabelle enthält eine Gegenüberstellung aller Tabellenstatistiken aus den drei untersuchten DBMS. Semantisch gleiche oder ähnliche Statistikwerte werden für jedes System mit gleichen Ordnungszahlen nummeriert. Die mit * gekenn-

zeichnete Statistikwerte werden bei der Berechnung einiger fehlender Statistikwerte hinzugezogen. Anschließend werden die gekennzeichneten und berechneten Statistiken zusammen mit deren semantischen Bedeutung in Tabelle 3.20 gezeigt.

Tabelle 3.19 : Gegenüberstellung der Tabellenstatistiken

IBM DB2 V8.2	Oracle 10g	Microsoft Server 2005
TABSCHEMA	***OWNER***	***OBJECT_ID***
TABNAME	***TABLE_NAME***	***NAME***
CARD (1)	***PARTITION_NAME***	***STATS_ID***
NPAGES (2)	***PARTITION_POSITION***	AUTO_CREATED
FPAGES (*)	***SUBPARTITION_NAME***	USER_CREATED
OVERFLOW	***SUBPARTITION_POSITION***	NO_RECOMPUTE
ACTIVE_BLOCKS	OBJECT_TYPE	ROWS (1) aus sys.sysindexes
	NUM_ROWS (1)	DPAGES (2) aus sys.sysindexes
	BLOCKS (2)	
	EMPTY_BLOCKS (*)	
	AVG_SPACE	
	CHAIN_CNT	
	AVG_ROW_LEN	
	AVG_SPACE_FREELIST_BLOCKS	
	NUM_FREELIST_BLOCKS	
	AVG_CACHED_BLOCKS	
	AVG_CACHE_HIT_RATIO	
	SAMPLE_SIZE	
	LAST_ANALYZED	
	GLOBAL_STATS	
	USER_STATS	
	STATTYPE_LOCKED	

Anmerkung: Mit Fettschrift und Kursiv hervorgehobene Werte in dieser und folgenden Tabellen (z.B. TABSCHEMA, TABNAME, usw.) sind als Schlüssel zu verstehen. Die Blockgröße bzw. Seitengröße in einzelnen DBMS kann außerdem unterschiedlich sein.

Tabelle 3.20 : Semantische Bedeutung ausgewählter Tabellenstatistiken

Semantik	IBM DB2 V8.2	Oracle 10g	SQL Server
Anzahl der Zeilen in der Tabelle	CARD	NUM_ROWS	ROWS
Anzahl nicht leerer Blöcke (Seiten)	NPAGES	BLOCKS	DPAGES
Anzahl von der Tabelle belegter Blöcke	FPAGES	BLOCKS + EMPTY_BLOCKS	
Anzahl leerer Blöcke (Seiten)	FPAGES - NPAGES	EMPTY_BLOCKS	

3.4.2 Spaltenstatistiken

Tabelle 3.21 enthält eine Gegenüberstellung aller Spaltenstatistiken aus den analysierten DBMS. Semantisch gleiche oder ähnliche Statistikwerte werden für jedes

System mit gleichen Zahlen nummeriert. Die mit * gekennzeichnete Statistikwerte werden bei der Berechnung einiger fehlender Statistikwerte hinzugezogen. Für Microsoft SQL Server 2005 wurden Statistiken aus Tabellen 3.13, 3.14, 3.15 zusammengefasst. Anschließend werden die gekennzeichneten und berechneten Statistiken zusammen mit deren semantischen Bedeutung in Tabelle 3.22 gezeigt.

Tabelle 3.21 : Gegenüberstellung der Spaltenstatistiken

IBM DB2 V8.2	Oracle 10g	Microsoft Server 2005
TABSCHEMA	***OWNER***	***OBJECT_ID***
TABNAME	***TABLE_NAME***	***STATS_ID***
COLNAME	***COLUMN_NAME***	***STATS_COLUMN_ID***
COLCARD (1, 6*)	NUM_DISTINCT (1)	***COLUMN_ID***
AVGCOLLEN (2)	LOW_VALUE (4)	***NAME***
HIGH2KEY (3)	HIGH_VALUE (3)	UPDATED (7)
LOW2KEY (4)	DENSITY (6)	ROWS (1)
NUMNULLS (5)	NUM_NULLS (5)	ROWS SAMPLED
SUB_COUNT	NUM_BUCKETS	STEPS
SUB_DELIM_LENGTH	LAST_ANALYZED(7)	DENSITY (6)
	SAMPLE_SIZE	AVERAGE KEY LENGTH
	GLOBAL_STATS	STRING INDEX
	USER_STATS	ALL DENSITY (6)
	AVG_COL_LEN (2)	AVERAGE LENGTH
	HISTOGRAM	COLUMNS

Tabelle 3.22 : Semantische Bedeutung ausgewählter Spaltenstatistiken

Semantik	IBM DB2 V8.2	Oracle 10g	SQL Server
Anzahl unterschiedlicher Werte	COLCARD	NUM_DISTINCT	ROWS
Mittlere Spaltenlänge	AVGCOLLEN	AVG_COL_LEN	
(2.) Maximum	HIGH2KEY	HIGH_VALUE	
(2.) Minimum	LOW2KEY	LOW_VALUE	
Anzahl der Nullwerte in der Spalte	NUMNULLS	NUM_NULLS	
Informationen zur Spaltendichte	CARD/COLCARD	DENSITY	(ALL) DENSITY
Datum der letzten Statistikerstellung	STATS_TIME	LAST_ANALYZED	UPDATED

3.4.3 Indexstatistiken

Tabelle 3.23 enthält eine Gegenüberstellung aller Indexstatistiken. Semantisch gleiche oder ähnliche Statistikwerte werden für jedes System mit gleichen Zahlen nummeriert. Die mit * gekennzeichnete Statistikwerte werden bei der Berechnung einiger fehlender Statistikwerte hinzugezogen. Für Microsoft SQL Server 2005 wurden Statistiken aus Tabellen 3.15, 3.16 zusammengefasst. Anschließend werden die gekennzeichneten und berechneten Statistiken zusammen mit deren semantischen Bedeutung in Tabelle 3.24 gezeigt.

Tabelle 3.23 : Gegenüberstellung der Indexstatistiken

IBM DB2 V8.2	Oracle 10g	Microsoft Server 2005

TABSCHEMA	***OWNER***	***OBJECT_ID***
TABNAME	***INDEX_NAME***	***NAME***
INDNAME	***PARTITION_NAME***	***STATS_ID***
NLEAF	***PARTITION_POSITION***	AUTO_CREATED
NLEVELS	***SUBPARTITION_NAME***	USER_CREATED
CLUSTERRATIO	***SUBPARTITION_POSITION***	NO_RECOMPUTE
CLUSTERFACTOR	OBJECT_TYPE	ALL DENSITY
DENSITY	BLEVEL	AVERAGE LENGTH (1)
FIRSTKEYCARD	LEAF_BLOCKS	COLUMNS
FIRST2KEYCARD	DISTINCT_KEYS (1)	
FIRST3KEYCARD	AVG_LEAF_BLOCKS_PER_KEY	
FIRST4KEYCARD	AVG_DATA_BLOCKS_PER_KEY	
FULLKEYCARD (1)	CLUSTERING_FACTOR	
PAGE_FETCH_PAIRS	NUM_ROWS	
SEQUENTIAL_PAGES	AVG_CACHED_BLOCKS	
NUMRIDS	AVG_CACHE_HIT_RATIO	
NUMRIDS_DELETED	SAMPLE_SIZE	
NUM_EMPTY_LEAFS	LAST_ANALYZED	
AVERAGE_SEQUENCE_PAGES	GLOBAL_STATS	
AVERAGE_RANDOM_PAGES	USER_STATS	
AVERAGE_SEQUENCE_GAP		
AVERAGE_SEQUENCE_FETCH		
AVERAGE_RANDOM_FETCH		
AVERAGE_SEQUENCE_FETCH		

Tabelle 3.24 : Semantische Bedeutung ausgewählter Indexstatistiken

Semantik	IBM DB2 V8.2	Oracle 10g	SQL Server
Anzahl unterschiedlicher Schlüssel	FULLKEYCARD	DISTINCT_KEYS	1 / ALL DENSITY

3.4.4 Verteilungsstatistiken (Histogramme)

Tabelle 3.25 enthält eine Gegenüberstellung aller Spaltenstatistiken. Semantisch gleiche oder ähnliche Statistikwerte werden für jedes System mit gleichen Zahlen nummeriert. Die mit * gekennzeichnete Statistikwerte werden bei der Berechnung einiger fehlender Statistikwerte hinzugezogen. Anschließend werden die gekennzeichneten und berechneten Statistiken zusammen mit deren semantischen Bedeutung in Tabelle 3.26 gezeigt.

Tabelle 3.25 : Gegenüberstellung der Verteilungsstatistiken

IBM DB2 V8.2	Oracle 10g	Microsoft Server 2005
TABSCHEMA	OWNER	RANGE_HI_KEY (2)
TABNAME	TABLE_NAME	RANGE_ROWS (1)
COLNAME	COLUMN_NAME	EQ_ROWS (3)
TYPE	ENDPOINT_NUMBER (1)	AVG_RANGE_ROWS
SEQNO (1)	ENDPOINT_VALUE (2)	DISTINCT_RANGE_ROWS (3)
COLVALUE (2)	ENDPOINT_ACTUAL_VALUE (2)	
VALCOUNT (4)		
DISTCOUNT (3)		

Während der Beschreibung der Semantik dieser Statistikart sollte unbedingt bedacht werden, dass einige Werte in Abhängigkeit vom jeweiligen Typ der Verteilungsstatistik unterschiedliche Bedeutungen haben können (zum Beispiel COLVALUE in DB2).

Tabelle 3.26 : Semantische Bedeutung ausgewählter Verteilungsstatistiken

Semantik	IBM DB2 V8.2	Oracle 10g	SQL Server 2005
Bucket-Nummer	SEQNO	ENDPOINT_NUMBER	RANGE_ROWS
Wert, für den Frequenzstatistiken erstellt werden	COLVALUE	ENDPOINT_VALUE, ENDPOINT_ACTUAL_VALUE	
Bucket-Obergrenze (nur Quantilhistogramme)	COLVALUE	ENDPOINT_VALUE, ENDPOINT_ACTUAL_VALUE	RANGE_HI_KEY
Anzahl der Vorkommen des in COLVALUE gespeicherten Wertes	VALCOUNT		
Anzahl der Spaltenwerte die kleiner als im COLVALUE gespeicherter Wert	VALCOUNT		
Anzahl unterschiedlicher Bucket-Werte	DISTCOUNT		DISTINCT_RANGE_ROWS + MIN(1, EQ_ROWS)

3.4.5 Kostenschätzwerte und Kardinalitäten

Tabelle 3.27 zeigt eine Gegenüberstellung einiger wichtiger Zugriffsplaninformationen aus den drei untersuchten DBMS, welche unter anderem Kostenschätzwerte und Kardinalitäten für Anfragen enthalten. Semantisch gleiche oder ähnliche Statistikwerte werden für jedes System mit gleichen Zahlen nummeriert. Die mit * gekennzeichnete Statistikwerte werden bei der Berechnung einiger fehlender Statistikwerte hinzugezogen. Anschließend werden die gekennzeichneten und berechneten Statistiken zusammen mit deren semantischen Bedeutung in Tabelle 3.28 gezeigt.

Tabelle 3.27 : Gegenüberstellung der Zugriffsplaninformationen

IBM DB2 V8.2	Oracle 10g	Microsoft Server 2005
TOTAL_COST (1)	STATEMENT_ID	STMTTEXT
STREAM_COUNT (2)	PLAN_ID	STMTID
CPU_COST (3)	TIMESTAMP	ESTIMATEROWS (2)
IO_COST (4)	OPERATION	ESTIMATEIO (4)
	COST (1)	ESTIMATECPU (3)
	CARDINALITY (2)	AVGROWSIZE
	CPU_COST (3)	TOTALSUBTREECOST (1)
	IO_COST (4)	

Tabelle 3.28 : Semantische Bedeutung ausgewählter Zugriffsplaninformationen

Semantik	IBM DB2 V8.2	Oracle 10g	SQL Server
Geschätzte Kosten einer Anfrage (ms)	TOTAL_COST	COST	TOTALSUBTREECOST
Geschätzte Kardinalität einer Anfrage	STREAM_COUNT	CARDINALITY	ESTIMATEROWS
Geschätzte CPU Kosten	CPU_COST	CPU_COST	ESTIMATECPU
Geschätzte Ein-/Ausgabekosten	IO_COST	IO_COST	ESTIMATEIO

4 Schnittstellenspezifikation und Architekturentwurf

Zwei wichtige Ziele dieser Diplomarbeit sind die Spezifikation und der Entwurf einer generi-schen Kosten- und Statistikschnittstelle für Datenbankmanagementsysteme. Zwei wichtige Anforderungen waren bei dieser Arbeit vielseitige Einsetzbarkeit und einfache Erweiterbarkeit dieser Software. Trotz der umfangreichen Funktionalität sollte diese Schnittstelle außerdem vollständige Entkopplung von dem darunterliegenden Datenbanksystem gewährleisten.

4.1 Spezifikation der Kosten- und Statistik-API

Im Folgenden wird die Kosten- und Statistik-API exakt spezifiziert. Die Spezifikation der Schnittstelle macht Aussagen über deren Syntax und Semantik. Die Syntax besteht aus den Signaturen aller Methoden der Schnittstelle, formuliert in der Programmiersprache Java. Alle Signaturen werden mit entsprechenden Methodennamen, Parametertypen und Rückgabetypen angegeben. Die Spezifikation der Schnittstellensemantik erfolgt in natürlicher Sprache unter Zuhilfenahme der Java-Notation.

4.1.1 Funktionsumfang der Schnittstelle

Die Schnittstelle bietet folgende Funktionalität:

- Lesezugriff auf Tabellenstatistiken
- Schreibzugriff auf Tabellenstatistiken
- Lesezugriff auf Spaltenstatistiken
- Schreibzugriff auf Spaltenstatistiken
- Lesezugriff auf Indexstatistiken
- Schreibzugriff auf Indexstatistiken
- Lesezugriff auf Verteilungsstatistiken (Histogramme)
- Schreibzugriff auf Verteilungsstatistiken (Histogramme)
- Lesezugriff auf Kostenschätzwerte von Anfragen
- Lesezugriff auf Kardinalitäten von Anfragen

4.1.2 Abstraktes Datenmodell der Umgebung

Um das Gesamtbild der für die Schnittstelle relevanten Umgebung besser zu verstehen, werden alle für die Spezifikation wichtigen Datenbankobjekte, Statistiken und Relationen in Abbildung 4.1 dargestellt. Dieses als Entity-Relationship-Diagramm aufgefasstes Datenmodell der Umgebung wird absichtlich sehr abstrakt ohne Angabe der Entity-Attribute gehalten, um jede Bindung an ein bestimmtes DBMS zu vermeiden.

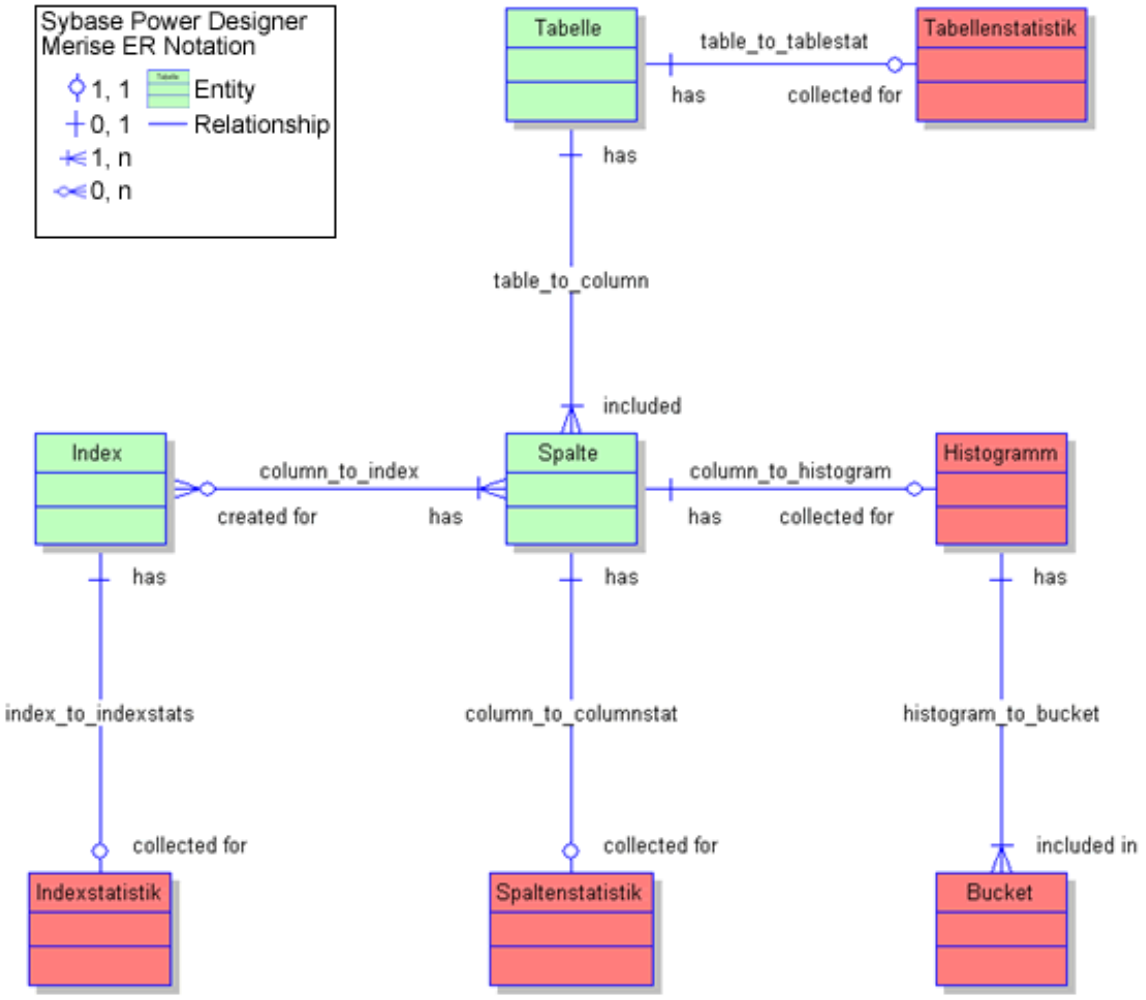


Abbildung 4.1: DB-Objekte, Statistiken und Relationen

4.1.3 Spezifikation der Datenstrukturen und Typisierung

Ausgehend von den im vorigen Kapitel gewonnenen Erkenntnissen werden nachfolgende Datenstrukturen bzw. Datentypen zum Austausch von Statistikwerten spezifiziert. Die Spezifikation erfolgt dabei in Java-Notation, um die spätere Implementierung zu vereinfachen.

Typisierung der tabellenbezogenen Statistiken:

Bei allen Tabellenstatistiken handelt sich um einfache (keine zusammengesetzte) Datentypen. Alle Werte dieser Kategorie werden als Long-Elemente spezifiziert (s. Tabelle 4.1).

Datentyp: Long
Beschreibung: Ganze Zahlen
Wertebereich: $[-2^{63}; 2^{63}-1]$

Tabelle 4.1 : Tabellenstatistiken und Ihre Datentypen

Beschreibung	Bezeichnung	Datentyp
Anzahl der Zeilen in der Tabelle	numRows	Long
Anzahl nicht leerer Blöcke (Seiten)	numUsedBlocks	Long
Anzahl leerer Blöcke (Seiten)	numEmptyBlocks	Long
Gesamtzahl von der Tabelle belegter Blöcke (Seiten)	numBlocks	Long

Typisierung der spaltenbezogenen Statistiken:

Bei den Spaltenstatistiken handelt es sich bis auf Maximum/Minimum-Werte einer Spalte ebenfalls um einfache Datentypen. Die Maximum- bzw. Minimum-Werte werden als Object spezifiziert. Diese Statistiken sind immer vom Spaltentyp abhängig und können sowohl als ganze und Gleitkommazahlen als auch in Form von Zeichen(-ketten) auftreten. Beim Zugriff auf diese Statistiken über die API wird deshalb immer mit Objekten gearbeitet, die Werte aller Datentypen aufnehmen können (s. Tabelle 4.2).

Datentyp: Long
Typsemantik: Ganze Zahlen
Wertebereich: $[-2^{63}; 2^{63}-1]$

Datentyp: Double
Typsemantik: Gleitkommazahlen
Wertebereich: $[-2^{-1074}; (2-2^{-52})\cdot 2^{1023}]$

Datentyp: Object
Typsemantik: Abstrakter Datentyp
Wertebereich: [Ganze Zahlen, Gleitkommazahlen, Einzelzeichen, Zeichenketten]

Tabelle 4.2 : Spaltenstatistiken mit spezifizierten Datentypen

Beschreibung	Bezeichnung	Datentyp
Spaltenkardinalität	colCard	Long
Mittlere Spaltenlänge	colAvgLength	Long
(2.) Maximum	colHighValue	Object
(2.) Minimum	colLowValue	Object
Anzahl der Nullwerte in der Spalte	colNumNulls	Long
Informationen zur Spaltendichte	colDensity	Double

Typisierung der indexbezogenen Statistiken:

Indexstatistiken sind in allen untersuchten DBMSs sehr spezifisch. Ein Wert lässt sich jedoch für alle 3 DBMSs ermitteln und über die Schnittstelle generisch bereitstellen (s. Tabelle 4.3).

Datentyp: Long
Typsemantik: Ganze Zahlen
Wertebereich: $[-2^{63}; 2^{63}-1]$

Tabelle 4.3 : Indexstatistiken mit spezifizierten Datentypen

Beschreibung	Bezeichnung	Datentyp
Durchschnittliche Anzahl der Schlüssel	numDistinctKeys	Long

Typisierung der Verteilungsstatistiken:

Für den Austausch von Verteilungstatistiken bzw. Histogrammen über die Kosten- und Statistikschnittstelle werden neben einfachen Datentypen auch komplexe Datenstrukturen für die Speicherung von Histogrammen eingesetzt (s. Tabelle 4.4). Denn ein Histogramm besteht meistens aus mehreren Buckets, und diese haben wiederum eigene Charakteristika (z.B. numDv, rangeHigh, ...). Die in dieser Arbeit eingesetzte Histogramm-Klasse [KRA05] wird im späteren Verlauf diskutiert.

Datentyp: Double
Typsemantik: Gleitkommazahlen
Wertebereich: $[-2^{-1074}; (2-2^{-52})\cdot 2^{1023}]$

Datentyp: Object
Typsemantik: Abstrakter Datentyp
Wertebereich: {Ganze Zahlen, Gleitkommazahlen, Einzelzeichen, Zeichenketten}

Datentyp: Histogramm
Typsemantik: Histogramm nach Definition 2.7 und [KRA05]
Wertebereich: {Histogramm, IntegerHistogram, DecimalHistogram, CharHistogram}

Tabelle 4.4: Histogramm-Buckets

Beschreibung	Bezeichnung	Datentyp
Bucket-Nummer	bucketNum	Long
Bucket-Obergrenze	rangeHigh	Object
Anzahl unterschiedlicher Bucket-Werte	numDv	double
Anzahl der Null-Buckets (ohne Elemente)	numNulls	double
Bucket-Kardinalität	bucketCard	double

Tabelle 4.5: Histogramme (Verteilungsstatistiken)

Beschreibung	Bezeichnung	Datentyp
Abstraktes Histogramm ohne Werttypisierung	histogram	Histogram

Histogramm mit Zechen(-ketten)werten	charHistogram	CharHistogram
Histogramm mit ganzzahligen Werten	intHistogram	IntegerHistogram
Histogramm mit Gleitkomma-Werten	decHistogram	DecimalHistogram

Typisierung der anfragenbezogenen Statistiken:

Kostenschätzwerte und Kardinalitäten von Anfragen lassen sich als Long- und Double erfassen (s. Tabelle 4.5).

Datentyp: Long
Typsemantik: Ganze Zahlen
Wertebereich: $[-2^{63}; 2^{63}-1]$

Datentyp: Double
Typsemantik: Gleitkommazahlen
Wertebereich: $[-2^{-1074}; (2-2^{-52})\cdot 2^{1023}]$

Tabelle 4.6 : Kostenschätzwerte und Kardinalitäten von Anfragen

Beschreibung	Bezeichnung	Datentyp
Geschätzte Kosten einer Anfrage	queryCost	Double
Geschätzte Kardinalität einer Anfrage	queryCard	Long

Spezifikation von API-Hilfselementen:

Um eine Verbindung mit der Datenbank aufzubauen oder zum Beispiel den Namen einer Tabellenspalte für die Statistikabfrage anzugeben, werden neben echten Statistikdaten auch Service- bzw. Hilfsinformation über die Kosten- und Statistikschnittstelle übergeben.

Datentyp: String
Typsemantik: Zeichen und Zeichenketten
Wertebereich: ASCII / UTF8

Tabelle 4.7 : Hilfsstrukturen

Beschreibung	Bezeichnung	Datentyp
Datenbankbezeichnung	dbName	String
Name des Datenbankbenutzers	dbUser	String
Passwort des Datenbankbenutzers	dbPass	String
Schemabezeichnung	schema	String
Tabellenbezeichnung	tableName	String
Spaltenbezeichnung	colName	String
Indexbezeichnung	indName	String
SQL-Anfrage	query	String

4.1.4 Spezifikation der Zugriffsmethoden

Nach der Spezifikation der Datenstrukturen und -typen für den Austausch der Informationen zwischen der Schnittstelle und der Außenwelt folgt im weiteren Verlauf die vollständige Spezifikation der über die API verfügbaren Zugriffsmethoden. Jede Methodenspezifikation besteht wie oben beschrieben aus einem Methodennamen und optionalen Übergabeparametern und Rückgabewerten. Dabei werden ausschließlich bereits spezifizierte Datentypen in Spezifikationsbestandteilen verwendet. Sollten an bestimmten Stellen Ausnahmebehandlungen notwendig sein, werden sie ebenfalls in der Spezifikation aufgeführt. Die globalen Vorbedienungen wurden bereist in 2.3 diskutiert.

Spezifikation der Zugriffsmethoden für Tabellenstatistiken:

1. **Spezifikation:** long getNumBlocks()

   - **Beschreibung:** Liefert die Gesamtzahl von der Tabelle belegter Blöcke (Seiten)

   - **Rückgabewerte:** numBlocks Anzahl der Blöcke

2. **Spezifikation:** setNumBlocks(long numBlocks)

   - **Beschreibung:** Setzt bzw. schreibt die Gesamtzahl von der Tabelle belegter Blöcke (Seiten).

   - **Übergabeparameter:** numBlocks - Anzahl der Blöcke

   - **Ausnahmenbehandlung:** ValidationException

3. **Spezifikation:** setNumBlocks(long numBlocks,
   long numUsedBlocks,
   long numEmptyBlocks)

   - **Beschreibung:** Setzt bzw. schreibt die Gesamtzahl der der Tabelle belegter Blöcke.

   - **Übergabeparameter:** numBlocks - Gesamtzahl der belegten Blöcke
     numUsedBlocks - Anzahl benutzter Blöcke
     numEmptyBlocks - Anzahl nicht benutzter Blöcke

   - **Ausnahmenbehandlung:** ValidationException

4. **Spezifikation:** long getNumUsedBlocks()

   - **Beschreibung:** Liefert die Anzahl von der Tabelle benutzter Blöcke (Seiten).

- **Rückgabewerte:** numUsedBlocks Anzahl benutzter Blöcke (Seiten)

5. **Spezifikation:** setNumUsedBlocks(long numUsedBlocks)

   - **Beschreibung:** Setzt bzw. schreibt die Anzahl von der Tabelle (Objekt) benutzter Blöcke (Seiten).
   - **Übergabeparameter:** numUsedBlocks - Anzahl benutzter Blöcke (Seiten)
   - **Ausnahmenbehandlung:** ValidationException

6. **Spezifikation:** long getNumEmptyBlocks()

   - **Beschreibung:** Liefert die Anzahl nicht benutzter Blöcke (Seiten).
   - **Rückgabewerte:** numEmptyBlocks Anzahl nicht benutzter Blöcke (Seiten)

7. **Spezifikation:** setNumEmptyBlocks(long numEmptyBlocks)

   - **Beschreibung:** Setzt bzw. schreibt die Anzahl nicht benutzter Blöcke (Seiten).
   - **Übergabeparameter:** numEmptyBlocks - Anzahl nicht benutzter Blöcke (Seiten)
   - **Ausnahmenbehandlung:** ValidationException

8. **Spezifikation:** long getNumRows()

   - **Beschreibung:** Liefert die Anzahl Zeilen in der Tabelle (Kardinalität).
   - **Rückgabewerte:** numRows Anzahl Zeiten in der Tabelle

9. **Spezifikation:** setNumRows(long numRows)

   - **Beschreibung:** Setzt bzw. schreibt die Anzahl Zeilen in der Tabelle.
   - **Übergabeparameter:** numRows - Anzahl der Tabellenzeilen
   - **Ausnahmenbehandlung:** ValidationException

Spezifikation der Zugriffsmethoden für Spaltenstatistiken:

1. **Spezifikation:** long getColNumNulls(String colName)
   - **Beschreibung:** Liefert die Anzahl der Nullwerte in der Spalte
   - **Rückgabewerte:** numNulls Anzahl der Nullwerte in der Spalte

2. **Spezifikation:** setColNumNulls(long colNumNulls, String colName)
   - **Beschreibung:** Setzt bzw. schreibt die Anzahl der Nullwerte in der Spalte.
   - **Übergabeparameter:** colNumNulls - Anzahl der Nullwerte in der Spalte
     colName - Spaltenbezeichnung
   - **Ausnahmenbehandlung:** ValidationException

3. **Spezifikation:** Object getColLowValue(String colName)
   - **Beschreibung:** Liefert den (2.) Minimalwert in der Spalte / (2.) Minimum)
   - **Übergabeparameter:** colName - Spaltenbezeichnung
   - **Rückgabewerte:** colLowValue (2.)Spaltenminimum

4. **Spezifikation:** setColLowValue(Object colLowValue, String colName)
   - **Beschreibung:** Setzt bzw. schreibt den (2.)Minimalwert / (2.)Minimum der Spalte.
   - **Übergabeparameter:** colLowValue - (2.)Spaltenminimum
     colName - Spaltenbezeichnung
   - **Ausnahmenbehandlung:** ValidationException

5. **Spezifikation:** Object getColHighValue(String colName)
   - **Beschreibung:** Liefert den (2.)Maximalwert der Spalte / (2.)Maximum
   - **Übergabeparameter:** colName - Spaltenbezeichnung
   - **Rückgabewerte:** colHighValue (2.)Spaltenmaximum

6. **Spezifikation:** setColHighValue(Object colLowValue, String colName)

   - **Beschreibung:** Setzt bzw. schreibt den (2.)Maximalwert der Spalte / (2.)Maximum.

   - **Übergabeparameter:** colHighValue - (2.)Spaltenmaximum
     colName - Spaltenbezeichnung

   - **Ausnahmenbehandlung:** ValidationException

7. **Spezifikation:** setColLowHighValue (Object colLowValue,
   Object colHighValue,
   String colName)

   - **Beschreibung:** Setzt bzw. schreibt den (2.)Minimal-/(2.)Maximalwert der Spalte.

   - **Übergabeparameter:** colLowValue - (2.)Spaltenminimum
     colHighValue - (2.)Spaltenmaximum
     colName - Spaltenbezeichnung

   - **Ausnahmenbehandlung:** ValidationException

8. **Spezifikation:** Object getColAvgLength(String colName)

   - **Beschreibung:** Liefert die mittlere Spaltenlänge

   - **Übergabeparameter:** colName - Spaltenbezeichnung

   - **Rückgabewerte:** colAvgLength Mittlere Spaltenlänge

9. **Spezifikation:** setColAvgLength(long colAvgLength, String colName)

   - **Beschreibung:** Setzt bzw. schreibt die mittlere Spaltenlänge.

   - **Übergabeparameter:** colAvgLength - Mittlere Spaltenlänge
     colName - Spaltenbezeichnung

   - **Ausnahmenbehandlung:** ValidationException

10. **Spezifikation:** long getColCard(String colName)

    - **Beschreibung:** Liefert die Anzahl unterschiedlicher Werte in der Spalte

    - **Übergabeparameter:** colName - Spaltenbezeichnung

- **Rückgabewerte:** colCard - Anzahl unterschiedlicher Werte in der Spalte

11. **Spezifikation:** setColCard(long colCard, String colName)

    - **Beschreibung:** Setzt bzw. schreibt die Anzahl unterschiedl. Werte in der Spalte.

    - **Übergabeparameter:** colCard - Anzahl unterschiedlicher Werte in der Spalte
      colName - Spaltenbezeichnung

    - **Ausnahmenbehandlung:** ValidationException

12. **Spezifikation:** int getColType(String colName)

    - **Beschreibung:** Liefert den in der Datenbank festgelegten Datentyp einer Tabellenspalte

    - **Übergabeparameter:** colName - Spaltenbezeichnung

    - **Rückgabewerte:** colType Spaltentyp

13. **Spezifikation:** double getColDensity(String colName)

    - **Beschreibung:** Liefert die Dichte einer Spalte aus der DB

    - **Übergabeparameter:** colName - Spaltenbezeichnung

    - **Rückgabewerte:** colDensity - Spaltendichte

14. **Spezifikation:** setColDensity(double colDensity, String colName)

    - **Beschreibung:** Setzt bzw. schreibt die Dichte einer Spalte.

    - **Übergabeparameter:** colDensity - Spaltendichte
      colName - Spaltenbezeichnung

    - **Ausnahmenbehandlung:** ValidationException

Spezifikation der Zugriffsmethoden für Indexstatistiken:

1. **Spezifikation:** long getNumDistinctKeys(String indName)

    - **Beschreibung:** Liefert die Anzahl unterschiedlicher Schlüssel im Index

- **Übergabeparameter:** indName - Indexbezeichnunug
- **Rückgabewerte:** numDistinctKeys - Anzahl unterschiedlicher Schlüssel im Index

2. **Spezifikation:** setNumDistinctKeys(long numDistinctKeys, String indName)
   - **Beschreibung:** Setzt bzw. schreibt die Anzahl unterschiedl. Schlüssel im Index.
   - **Übergabeparameter:** numDistinctKeys - Anzahl unterschiedlicher Schlüssel
     indName - Indexbezeichnung
   - **Ausnahmenbehandlung:** ValidationException

Spezifikation der Zugriffsmethoden für Verteilungsstatistiken:

1. **Spezifikation:** Histogram getHistogram(String colName)
   - **Beschreibung:** Liefert das komplette Histogramm für eine Spalte
   - **Übergabeparameter:** colName - Spaltenbezeichnung
   - **Rückgabewerte:** histogram - Komplettes Histogramm für eine Spalte
   - **Ausnahmenbehandlung:** ValidationException

2. **Spezifikation:** setHistogram(Histogram histogram, String colName)
   - **Beschreibung:** Setzt bzw. schreibt das komplette Histogramm für eine Spalte.
   - **Übergabeparameter:** histogram - Komplettes Histogramm
     colName - Spaltenbezeichnung
   - **Ausnahmenbehandlung:** ValidationException

3. **Spezifikation:** int getNumBuckets(String colName)
   - **Beschreibung:** Liefert die Anzahl der Buckets im Histogramm
   - **Übergabeparameter:** colName - Spaltenbezeichnung
   - **Rückgabewerte:** numBuckets Anzahl der Histogramm-Buckets

Spezifikation der Zugriffsmethoden für Kostenschätzwerte und Kardinalitäten:

1. **Spezifikation:** double getQueryCost(String query)
   - **Vorbedingung:** Notwendige (Explain-)Tabellen sind bereits angelegt
   - **Beschreibung:** Liefert den Kostenschätzwert einer Query
   - **Übergabeparameter:** query - SQL-konforme Anfrage
   - **Rückgabewerte:** queryCost Kostenschätzwert einer Query

2. **Spezifikation:** int getQueryCard(String query)
   - **Vorbedingung:** Notwendige (Explain-)Tabellen sind bereits angelegt
   - **Beschreibung:** Liefert die Ergebniskardinalität einer Query
   - **Übergabeparameter:** query - SQL-konforme Anfrage
   - **Rückgabewerte:** queryCard Kardinalität der Query

4.2 Architekturentwurf

Nachdem die Schnittstellenspezifikation vollständig abgeschlossen ist, wird in diesem Abschnitt auf den Architekturentwurf der Schnittstelle eingegangen. Zu diesem Zweck wird nachfolgend die bisher als Ganzes geführte Schnittstellenkomponente in mehrere Schichten eingeteilt.

4.2.1 Mehrschichtiger Aufbau und Konsistenzsicherung

Der Konsistenzerhaltung während der Arbeit mit der Kosten- und Statistik-API wird eine sehr wichtige Rolle im Gesamtkonzept zugesprochen. Besonders bei Schreibzugriffen auf zahlreiche Statistikwerte mittels angebotener set-Methoden sollten die Datenbank und das gesamte Datenbanksystem im konsistenten Zustand bleiben. Die Schnittstelle muss darüber hinaus einem generischen Aufbau folgen, um von einem bestimmten Datenbankmanagementsystem unabhängig zu sein.

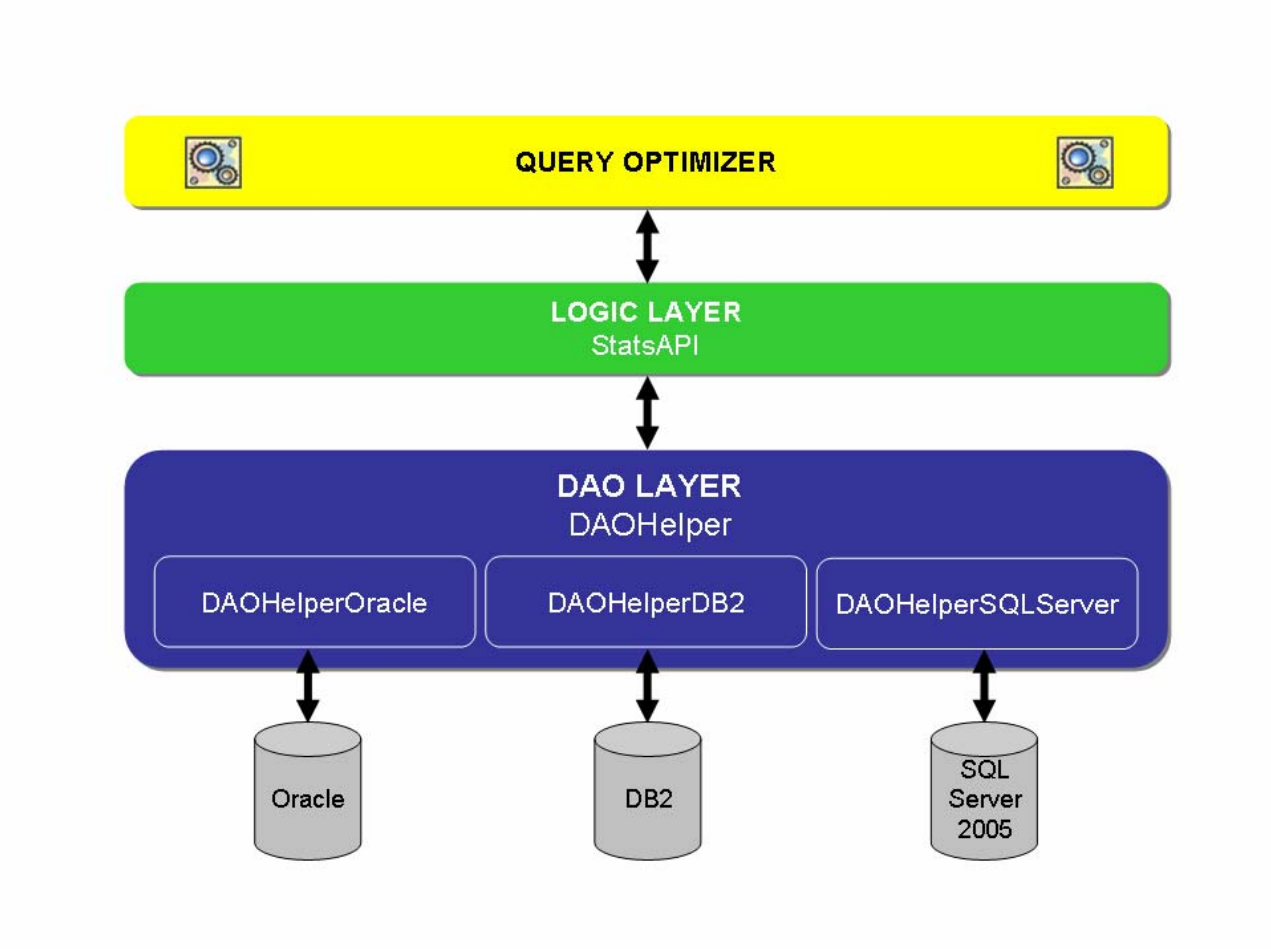


Abbildung 4.2: Mehrschichtige Architektur beim Schnittstellenentwurf

4.2.2 Logic Layer

Logic Layer (Logikschicht) entspricht der Außenschnittstelle im System und hat direkten Kontakt mit dem Benutzer bzw. mit der „Wirt“-Software. Sie wird nach der Schnittstellenspezifikation aufgebaut und ist durch eine DAO-Schicht (s.u.) von dem eingesetzten DBMS getrennt. Der Zugriff auf die gewünschten Statistikdaten erfolgt dann über die DAO-Schicht, wobei die bei Schreibzugriffen notwendige Validierung der zu speichernden Daten bereits in der Logik Schicht erfolgt. Darüber hinaus können in dieser Schicht gespeicherten Statistiken ohne einen erneuten Datenbankzugriff zwecks Performanzsteigerung zugänglich gemacht werden. Sollte eine Erweiterung der Schnittstelle in Betracht gezogen werden, bleibt diese Schicht von allen Änderungen unberührt.

4.2.3 Database Access Object (DAO) Layer

Die DAO Schicht ermöglicht den tatsächlichen Zugriff auf das darunterliegende Datenbankmanagementsystem und enthält alle datenbankspezifische Zugriffsmethoden, welche über eine definierte Schnittstelle (DAOHelper) zugänglich sind. Pro DBMS muss dann lediglich eine Art Treiber (DAOHelper) mit entsprechenden SQL

Anweisungen implementiert werden, die Logikschicht bleibt von dieser Implementierung aber unberührt. In Abbildung 4.2 wird die komplette DAO Schicht in blau dargestellt.

4.2.4 UML Modellierung der Schnittstellenkomponenten

Der mehrschichtige Schnittstellenaufbau sollte nun, um eine prototypische Implementierung zu ermöglichen, bis ins Detail durchmodelliert werden. Zu diesem Zweck werden weitere Modelle erstellt, welche in Form von UML-Paket- und Klassendiagrammen umgesetzt werden.

Das auf der nächsten Seite dargestellte Paketdiagramm (Abbildung 4.3) besteht aus 4 Elementen:

1. dbstatsapi Paket – Prototypische Implementierung der in dieser Diplomarbeit entwickelten Kosten- und Statistik-API

2. histogramPropagator Paket – Enthält Histogrammklassen für verschiedene Datentypen. Diese Implementierung stammt aus [KRA05].

3. drivers – Beinhaltet alle JDBC Treiber (IBM DB2 V8.2, Oracle 10g und Microsoft SQL Server 2005). Diese müssen für den Betrieb der Schnittstelle immer vorhanden sein.

4. DBStatsApiTest – Beispiel- und Testkomponente für die Kosten- und Statistik-API. Mit einfachen Test Cases werden alle Methoden der Schnittstelle auf Ihre Funktionsfähigkeit getestet:

    - testStatsAPIDB2 – Testfälle für IBM DB2 V8.2
    - testStatsAPIOracle – Testfälle für Oracle 10g
    - testStatsAPISQLServer – Testfälle für Microsoft SQL Server

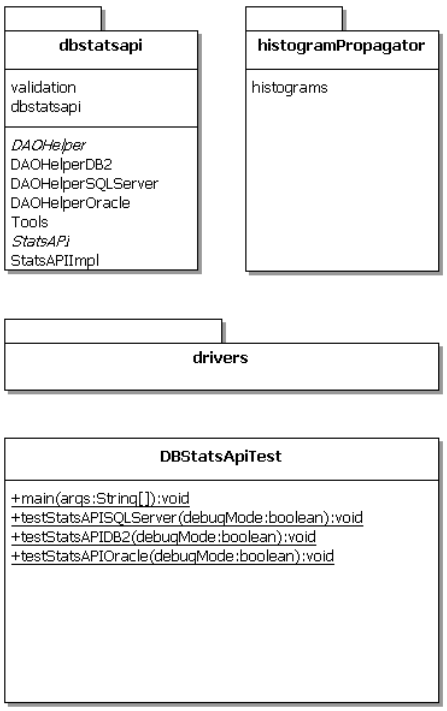


Abbildung 4.3: Schnittstellenpaket dbstatsapi,
Fremdpaket histogramPropagator, JDBC Treiber und Testanwendung

Abbildungen 4.4, 4.5 zeigen UML Klassendiagramme für Klassen Bucket und Histogram. Von einem abstrakten Bucket kann entweder ein CharBacket für die Erfassung von Zeichenketten, ein DecimalBucket für die Erfassung von Gleitkommazahlen oder ein IntegerBucket für die Erfassung von ganzen Zahlen abgeleitet werden.

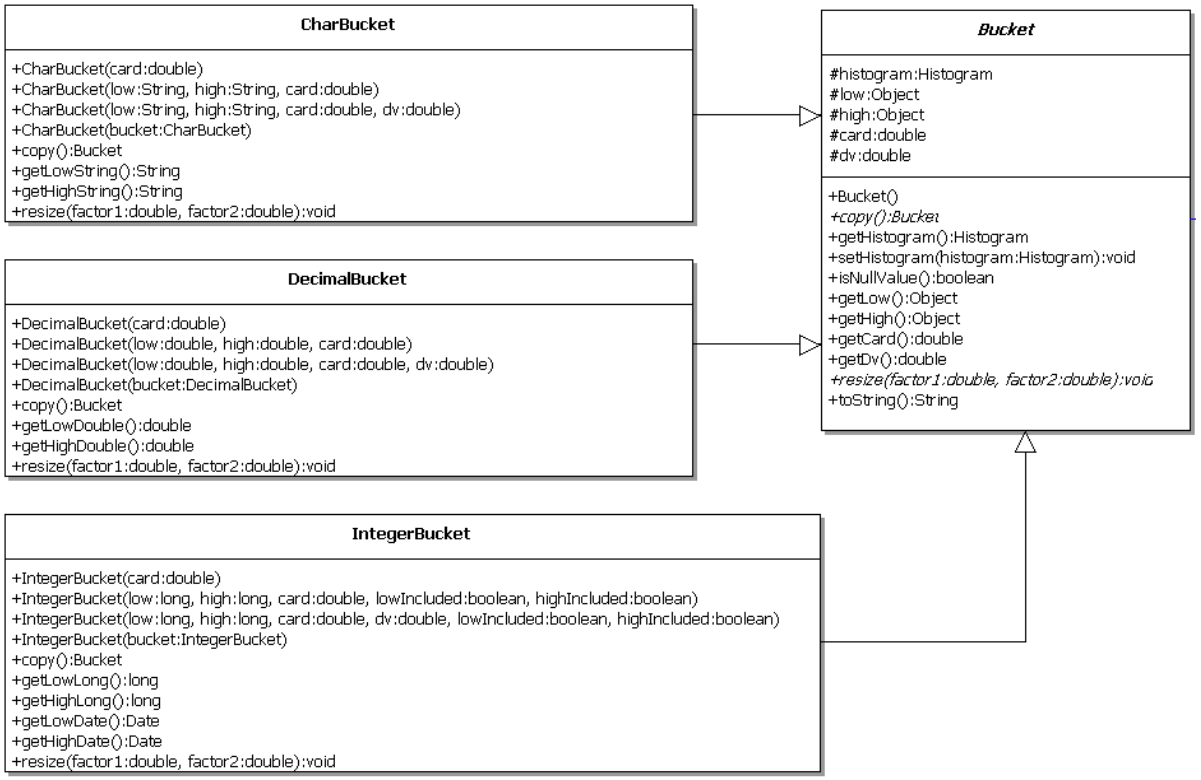


Abbildung 4.4: UML - Klassendiagramm „Bucket“

Basierend auf den Bucket-Klassen folgt eine ähnlich aufgebaute Definition von Histogrammen. Abbildung 4.5 zeigt ein Klassendiagramm mit einer abstrakten Histogramm-Definition und entsprechenden typgebundenen CharHistogram (für Zeichenketten), DecimalHistogram (für Gleitkommazahlen) und IntegerHistogram (für Ganze Zahlen) Definitionen.

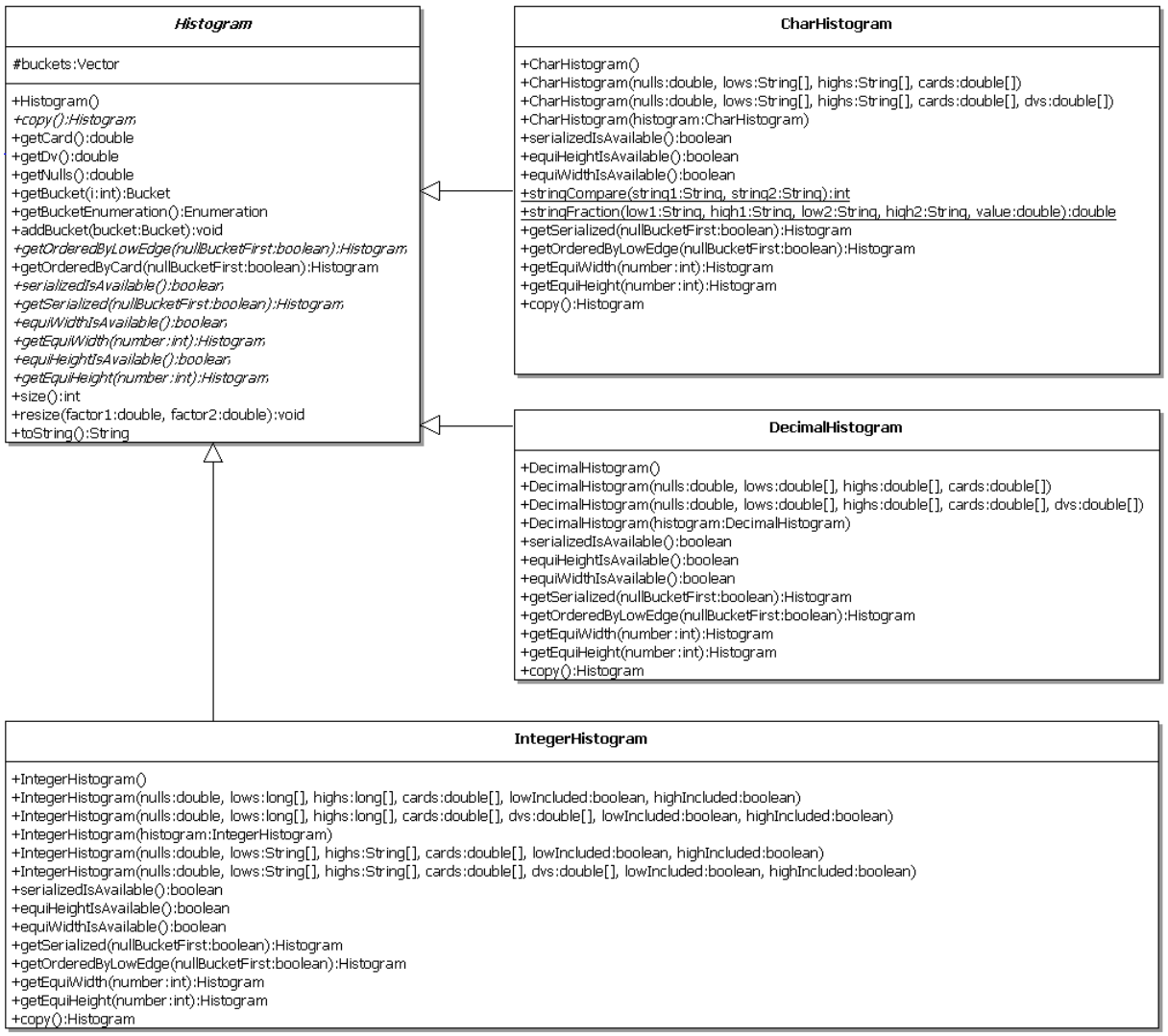


Abbildung 4.5: UML - Klassendiagramm „Histogramm“

Bei der Erstellung von Histogrammen in DBMSs wird immer der Datentyp der jeweiligen Spalte berücksichtigt. Beispielsweise werden die als Bucket-Obergrenzen ausgewählten Spaltenwerte auch typgebunden im Histogramm erfasst. Die in Abbildungen 4.4 und 4.5 gezeigten Klassen ermöglichen den Austausch von kompletten Histo-grammen unabhängig vom Datentyp der Histogrammelemente.

Das UML Klassendiagramm aus Abbildung 4.6 noch mal zeigt die in zwei Schichten aufgeteilte Kosten- und Statistik-API. Die nach der Spezifikation aus 4.1.4 deklarierte Java-Schnittstelle StatsAPI bildet zusammen mit der Schnittstellenimplementierung StatsAPIIml die in 4.2.2 beschriebene Logikschicht (Logic Layer). Die StatsAPIImpl Klasse steht in einer Assoziationsbeziehungen zu Klassen Tools, Validation und zur DAOHelper Schnittstelle (Teil von DAO Layer).

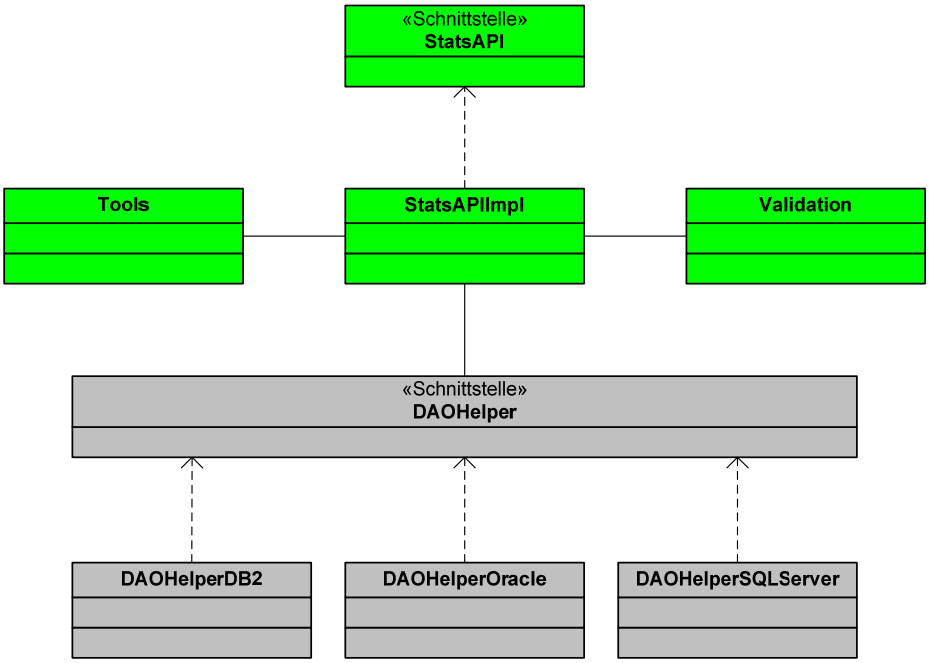


Abbildung 4.6: UML Klassendiagramm der Kosten- und Statistik-API

Abbildung 4.7 zeigt eine Detailansicht der Statistik- und Kostenschnittstelle StatsAPI mit allen für den Datenaustausch definierten Methoden, Parametern und Datentypen.

Abbildung 4.7: UML Klassendiagramm StatsAPI (Detailansicht)

Die einzelnen DAOHelper Klassen DAOHelperDB2 für IBM DB2, DAOHelperOracle für Oracle und DAOHelperSQLServer für Microsoft SQL Server bilden die DAO Schicht zur Unterstützung der Kosten- und Statistikschnittstelle bei allen Datenbankzugriffen. Sie stellt für jede Get-Methode aus der Logikschicht eine entsprechende datenbankspezifische Get-Methode zur Verfügung, die mit Hilfe von einer oder mehreren SQL-Anfragen auf Statistikdaten zugreift. Die auf diese Weise gelesenen Statistikwerte werden anschließend an die Logikschicht weitergereicht. Wenn eine Aktualisierung von Statistiken über die Schnittstelle veranlasst wird, gehen alle zu speichernden Daten über die Set-Methoden der Logikschicht erst nach einer Validierung zur DAO-Schicht weiter. Eine passende und bis auf das Präfix mit der aufrufenden Methode namengleiche Save-Methode schreibt dann die Daten mittels SQL in die Datenbank.

Abbildung 4.7: UML Klassendiagramm DAOHelper (Detailansicht)

Die Validierungskomonente (Abbildung 4.9) sorgt für die Konsistenzerhaltung statistischer Daten in der Datenbank, indem sie die Einhaltung der in Kapitel 3 ermittelten DBMSs spezifischen Einschränkungen bei Schreibzugriffen kontrolliert. D.h. sollen bestimmte Statistiken in der Datenbank aktualisiert werden, dann werden alle über die Statistik- und Kostenschnittstelle an die Logikschicht übergebenen Daten mit Hilfe der Validation Klasse auf Korrektheit geprüft. Es werden sowohl Wertebereiche und Datentypen als auch semantische Einschränkungen für entsprechende Werte in

dieser Prüfung berücksichtigt. Zum Beispiel wird validateNatural() zur Feststellung eingesetzt, ob eine Statistik als natürliche Zahl übergeben wurde. validateNumBlocks() prüft zuerst die Datentypen der übergebenen Blocks-Statistiken (numBlocks, numUsedBlocks und numEmptyBlocks). Danach wird eine semantische Prüfung angestoßen. Diese kontrolliert dann, ob die Gleichung numBlocks = numUsedBlocks + numEmptyBlocks erfüllt ist. Eine Ausnahmenbehandlung wird ebenfalls angestoßen, sollten unzulässige Werte festgestellt werden.

Abbildung 4.9: StatsAPIValidation und ValidationException

Eine weitere unterstützende Komponente der Schnittstellenimplementierung trägt den Namen „Tools“ (s. Abbildung 4.8). Sie fasst alle für die Typ- und Formatumwandlung zuständigen Methoden zusammen. Diese Methoden werden zur Vereinheitlichung der meist unterschiedlichen Datentypen und Formate von Datenbanksystemen eingesetzt und garantieren eine konstante Außenschnittstelle (API).

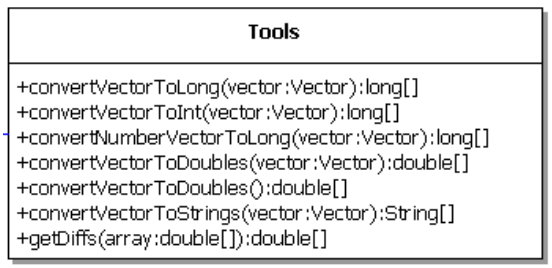


Abbildung 4.9: Tools-Komponente für die Formatumwandlung

In der folgenden Tabelle werden alle Tools-Methoden kurz beschrieben:

Tabelle 4.1 : Tools-Methoden

Methode	**Beschreibung**
convertVectorToLong	Wandelt ein Vektor-Objekt mit Double-Werten in ein Array mit Elementen vom Typ long um.
convertVectorToInt	Wandelt ein Vektor-Objekt mit Double-Werten in ein Array mit Elementen vom Typ long um.
convertNumberVectorToLong	Wandelt ein Vektor-Objekt mit Werten vom Oracle-Typ „NUMBER“ in ein Array mit Elementen vom Typ long um. Sonderfall speziell für Oracle, da die Unterscheidung zwischen int, double, usw. fehlt.
convertVectorToDoubles	Wandelt ein Vektor-Objekt mit Integer-Werten in ein Array mit Elementen vom Typ double um (Einsatz z.B. für card, numDv, numNulls in Histogramm-Objekten)
convertVectorToDoubles	Wandelt ein Vektor-Objekt mit Integer-Werten in ein Array mit Elementen vom Typ double um (Einsatz z.B. für card, numDv, numNulls in Histogramm-Objekten)
convertVectorToStrings	Wandelt ein Vektor-Objekt mit String-Elementen in ein Array mit Elementen vom Typ String um
getDiffs	Wandelt ein Array mit Double-Werten in ein Array mit Differenz-Elementen (Differenz = Array[i] – Array[i-1]) um.

5 Implementierung

In diesem Kapitel erfolgt die Vorstellung eines in Java implementierten Prototyps der Kosten- und Statistik-API für die betrachteten DBMSs. Neben der Realisierung des Zugriffs auf die benötigten Daten sind hier insbesondere die Abbildungen zwischen den Datenstrukturen in den DBMSs und den Datenstrukturen der Schnittstellendefinition bzw. zwischen den Datenstrukturen unterschiedlicher DBMSs interessant. Die Implementierung des Prototyps basiert auf der in Kapitel 4 ausgearbeiteten Schnittstellenspezifikation. Um den Rahmen dieser Arbeit nicht zu sprengen, werden im Folgenden nur einige besonders interessante Code-Ausschnitte vorgestellt und beschrieben. Der interessierte Leser wird an dieser Stelle an den vollständigen Quellcode der API verwiesen.

5.1 Logic Layer – StatsAPI und StatsAPIImpl

Die Logikschicht der Kosten und Statistik-API wurde in der Klasse StatsAPIImpl umgesetzt. Diese Klassen implementiert die Java-Schnittstelle StatsAPI und bildet den Kernpunkt der gesamten Anwendung.

5.1.1 Verwendete Fremdpakete und –klassen

Während der Prototypimplementierung wurden einige bereits existierende Pakete und Klassen verwendet. Der komplette Import-Block der Schnittstellenimplementierung wird nachfolgend gezeigt:

Quellcode 5.1

```
import java.sql.Types;
import java.util.Vector;

import histogramPropagator.histograms.Histogram;
import histogramPropagator.histograms.CharHistogram;
import histogramPropagator.histograms.DecimalHistogram;
import histogramPropagator.histograms.IntegerHistogram;

import dbstatsapi.validation.StatsAPIValidation;
import dbstatsapi.validation.exceptions.ValidationException;

import dbstatsapi.Tools;
```

So definiert java.sql.Types alle JDBC Datentypen. Diese wurden in der Anwendung eingesetzt, um eine Trennung von jeder datenbankspezifischen Typisierung zu erreichen. Die von dem jeweiligen JDBC Treiber in dieser Form übermittelte Datentyp-Informationen wurden u.a. für spätere Casting- und Konvertierungsoperationen zwecks Vereinheitlichung eingesetzt.

Vector stellt dynamisch wachsende Array-Konstrukte zur Verfügung. Diese Arrays kamen zum Beispiel bei der Implementierung von Zugriffsmethoden für Histogramme zum Einsatz.

5.1.2 Klassenkonstruktor der Schnittstelle

Die Initialisierung der Schnittstelle erfolgt über den Klassenkonstruktor. Das gewünschte DBMS wird durch die Angabe einer der folgenden Klassenkonstanten festgelegt:

- DBMS_DB2 = 1;
- DBMS_ORACLE = 2;
- DBMS_SQLSERVER = 3;

Neben dem DBMS-Typ sind Zugangsdaten für die Datenbank, ein Schema und ein Tabellenname anzugeben. Diese Informationen werden dann den Verbindungsaufbau zu der Datenbank und für den Zugriff auf Statistiken verwendet. Als letzter Parameter folgt ein boolean-Wert zur Einschaltung bzw. Unterdrückung der Ausgaben in Zugriffsmethoden verwendeter SQL Queries. Wenn die Daten vollständig sind, wird nun ein neues datenbankspezifisches DAO-Objekt erzeugt und die Validierungs- und Tools-Komponenten initialisiert.

Quellcode 5.2

```
public StatsAPIImpl(byte type, String dbName, String dbUser, String dbPass,
                String schema, String tableName, boolean debug) {

        this.dbmsType = type;

        switch (this.dbmsType) {
        case DBMS_DB2:
                DAOHelperDB2 DAOHelperDB2 = new DAOHelperDB2(dbName, dbUser,
                                dbPass, schema, tableName, debug);
                this.DAOHelper = DAOHelperDB2;
                break;

        case DBMS_ORACLE:
                DAOHelperOracle DAOHelperOracle = new DAOHelperOracle(dbName,
                                dbUser, dbPass, schema, tableName, debug);
                this.DAOHelper = DAOHelperOracle;
                break;

        case DBMS_SQLSERVER:
                DAOHelperSQLServer DAOHelperSQLServer = new DAOHelperSQLServer(
                                dbName, dbUser, dbPass, schema, tableName, debug);
                this.DAOHelper = DAOHelperSQLServer;
                break;
        }

        // Validierungs- und Tools-Komponenten erzeugen
        validation = new StatsAPIValidation();
        tools = new Tools();
}
```

5.1.3 Lesezugriff - Methode getHistogram()

getHistogram() soll den Lesezugriff auf Statistiken veranschaulichen. Diese Methode liefert das komplette Verteilungsstatistiken für eine bestimmte Tabellenspalte. Der Lesezugriff auf die Datenbank erfolgt über die DAO-Methode getHistogramData(). Nach einem erfolgreichen Zugriff folgt die Umwandlung der ausgelesenen Histogrammdaten, um sie danach in einem Histogramm-Objekt abzulegen. Welcher Histogramm-Typ (Char, Integer oder Decimal) erstellt wird, hängt allein von dem über

getColType() ermittelten Spaltentyp ab. Da sowohl Bucket-Kardinalitäten als auch die Anzahl unterschiedlicher Werte pro Bucket im IBM DB2 DBMS nur für Quantile vorliegen, müssen diese Werte mit getDiff()für einzelne Buckets berechnet werden. Wenn alle von einem tools-Objekt bereitgestellten convert()-Methoden ausgeführt sind, kann am Ende ein CharHistogram, IntegerHistogram oder DecimalHistogram erstellt und als Ergebnis von getHistogram() zurückgegeben werden.

Quellcode 5.3

```
public Histogram getHistogram(String colName) throws ValidationException {
        // Spaltentyp aus der Datenbank holen
        int colType = DAOHelper.getColType(colName);

        // Histogrammdaten zu der Spalte aus der Datenbank holen
        Vector histogramData = DAOHelper.getHistogramData(colName);
        validation.validateStats(histogramData, "Verteilungstatistiken");

        // Vector-Objekte für die Aufnahme der Histogrammdaten
        Vector objLows = (Vector) histogramData.get(0);
        Vector objHighs = (Vector) histogramData.get(1);
        Vector objCards = (Vector) histogramData.get(2);
        Vector objDvs = (Vector) histogramData.get(3);

        double[] cards = tools.convertVectorToDoubles(objCards);
        double[] dvs = tools.convertVectorToDoubles(objDvs);


        if (dbmsType == DBMS_DB2) {
                cards = tools.getDiffs(cards); // Differenzen ausrechnen
                dvs = tools.getDiffs(dvs);     // Differenzen ausrechnen
        }

        // Anzahl der NULL-Werte in der Spalte ermitteln
        double colNumNulls = getColNumNulls(colName);

        switch (colType) {
        case Types.NUMERIC:
                // Sonderfall speziell für Oracle,
                // da die Unterscheidung zwischen int, double, usw. fehlt
                double[] numLows = tools.convertNumberVectorToDouble(objLows);
                double[] numHighs = tools.convertNumberVectorToDouble(objHighs);
                this.histogram = new IntegerHistogram(colNumNulls, numLows,
                        numHighs, cards, dvs, true, true);
                break;

        case Types.INTEGER:
                long[] lngLows = tools.convertVectorToInt(objLows);
                long[] lngHighs = tools.convertVectorToInt(objHighs);
                this.histogram = new IntegerHistogram(colNumNulls, lngLows,
                        lngHighs, cards, dvs, true, true);
                break;

        case Types.VARCHAR:
                String[] strLows = tools.convertVectorToStrings(objLows);
                String[] strHighs = tools.convertVectorToStrings(objHighs);
                this.histogram = new CharHistogram(colNumNulls, strLows, strHighs,
                        cards);
                break;

        case Types.DOUBLE:
                double[] dblLows = tools.convertVectorToDoubles(objLows);
                double[] dblHighs = tools.convertVectorToDoubles(objHighs);
                this.histogram = new DecimalHistogram(colNumNulls, dblLows,
                        dblHighs, cards, dvs);
                break;
        }
        return this.histogram;
}
```

5.1.4 Schreibzugriff - Methode setHistogram()

Sollen Verteilungstatistiken für eine Spalte in der Datenbank abgelegt werden, um zum Beispiel den Optimierer in seiner Entscheidung zu beeinflussen, kommt die setHistogramm() Methode zum Einsatz. Nach der Validierung der in Form eines Objekts der Klasse „Histogram“ übergebenen Daten, werden diesen an in der Logikschicht aktualisiert und an die DAO Schicht durch den Aufruf von saveHistogram() weitergegeben. Sollten während des Validierungsvorganges Unstimmigkeiten festgestellt werden, wird die Ausführung unterbrochen und mit der entsprechenden Meldung in der Ausnahmenbehandlung (ValidationException) auf diese(-n) Fehler hingewiesen.

Quellcode 5.4

```
public void setHistogram(Histogram histogram, String colName)
throws ValidationException {
        this.validation.validateHistogram(histogram, getNumBuckets(colName),
                        this);
        this.histogram = histogram;
        boolean saveFlag = DAOHelper.saveHistogram(colName, this.histogram);
}
```

5.2 DAO Layer – DAOHelper und Implementierungen

Nach Fertigstellung der Logic Layer musste noch die DAO Schicht der Anwendung programmiert werden. Im Rahmen der Prototyperstellung wurden entsprechende Implementierungen der DAOHelper Schnittstelle für die drei analysierten Datenbankmanagementsysteme IBM DB2 V8.2, Oracle 10g und Microsoft Server 2005 erstellt. Die Funktionsweise dieser Implementierungsklassen wird wieder anhand von Quellcode-Ausschnitten dargestellt.

5.2.1 Verbindungsaufbau zur Datenbank

Wenn in Java eine Datenbankverbindung aufgebaut werden soll, dann werden DBMS spezifische Datenbanktreiber benötigt [HOR, SUN]. Diese Treiber müssen außerdem wie in Quellcodeausschnitt 5.5 zu sehen im System registriert werden. Dies geschieht mittels Class.forName(" … ”).newInstance() Methode. Für den API-Prototyp werden folgende Treiber benötigt:

- db2java.zip für IBM DB2 V8.2
- classes12.zip für Oracle 10g
- sqljdbc.jar für Microsoft SQL Server 2005

Der eigentliche Verbindungsaufbau wird dann mit DriverManager.getConnection vollzogen.

Quellcode 5.5 : connectDB aus DAOHelperDB2

```
public Connection connectDB() {
        try {
                // Datenbank Treiber laden
```

```
                Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
                // Verbindung herstellen
                this.connection = DriverManager.getConnection("jdbc:db2:" + dbName,
                                dbUser, dbPass);
                this.statement = this.connection.createStatement();
        } catch (SQLException ex) {
                System.err.println("SQL Exception: " + ex.getMessage());
        } catch (Exception e) {
                System.err.println("DB2-JDBC Treiber konnte nicht geladen" +
                        " werden!\n\n");
                e.printStackTrace();
        }
        return this.connection;
}
```

5.2.2 DAOHelperDB2 - Lesezugriff mit get-Methoden

Als Beispiel eines für IBM DB2 implementierten Lesezugriffs folgt die Methode getColCard() zur Bestimmung der Spaltenkardinalität. Hierbei wird der Statistikwert über eine SQL-Anfrage aus der Sicht SYSSTATS.COLUMNS ausgelesen und als Long-Wert an die Logic Layer zurückgegeben. Wenn Statistiken noch nicht erstellt oder nicht verfügbar sind, wird bei einem Lesezugriff eine vordefinierte Klassenkonstante (CONST_NO_STATS = -1) oder bei komplexen Statistiken ein NULL-Wert zurückgegeben. Die jeweilige Spalte für die Statistikabfrage wird über das Schema, Tabellennamen und Spaltennamen in der WHERE-Klausel festgelegt.

Quellcode 5.6 : DAOHelperDB2 - Lesezugriff mit einer get-Methode

```
public long getColCard(String colName) {
        long stats = CONST_NO_STATS;
        try {
                query = "SELECT COLCARD " +
                "FROM SYSSTAT.COLUMNS " +
                "WHERE TABSCHEMA = '" + this.schema + "' " +
                "  AND TABNAME = '" + this.tableName + "'" +
                "  AND COLNAME = '" + colName + "'";

                if(this.debugMode)
                        System.out.println("getColCard: " + query);

                ResultSet resultSet = this.statement.executeQuery(query);

                // wenn Datensatz vorhanden, dann Daten auslesen
                if(resultSet.next())
                        stats = resultSet.getLong(1);
        } catch (SQLException ex) {
                System.err.println("SQL Exception: " + ex.getMessage());
        }
        return stats;
}
```

5.2.3 DAOHelperDB2 - Schreibzugriff mit save-Methoden

Bei einem Schreibzugriff auf statistische Informationen eines DB2 Systems wird eine Update-Anweisung eingesetzt. Alle eventuell bestehende Einschränkungen für diesen Schreibzugriff wurden bereits in der Logic Layer kontrolliert. Wenn die Aktualisierung erfolgreich durchgeführt worden ist, wird eine positive Meldung (true) an die Logic Layer geschickt. SQL Ausnahmen (SQLExceptions) werden in dieser Methode ebenfalls abgefangen.

Quellcode 5.7 : DAOHelperDB2 – Einfacher Schreibzugriff mit einer save-Methode

```
public boolean saveColCard(long colCard, String colName) {
       try {
              query = "UPDATE SYSSTAT.COLUMNS " +
              "SET COLCARD = " + colCard +  " " +
              "WHERE TABSCHEMA = '" + this.schema + "' " +
              "  AND TABNAME = '" + this.tableName + "'" +
              "  AND COLNAME = '" + colName + "'";

              if(this.debugMode)
                     System.out.println("saveColCard: " + query);

              this.statement.executeUpdate(query);
       } catch (SQLException ex) {
              System.err.println("SQL Exception: " + ex.getMessage());
       }
       return true;
}
```

5.2.4 DAOHelperDB2 – Kostenschätzwerte und Kardinalitäten

Die getQueryCost() Methode enthält im Gegensatz zu bisher aufgeführten Code-Ausschnitten zwei separate SQL Anweisung und Zugriffe auf die Datenbank. Im ersten Teil der Methode wird mit Hilfe des EXPLAIN PLAN Befehls die übergebene SQL Query ausgewertet und mit einer Kennzeichnung (queryno = 1, querytag='API') versehen. Beim zweiten Zugriff auf die Datenbank wird ein Kostenschätzwert total_cost für die Anfrage anhand der Kennzeichnung ermittelt. Die Kardinalität für das Anfrageergebnis wird nach dem gleichen Prinzip in getQueryCard() ausgelesen.

Quellcode 5.8 : DAOHelperDB2 – Zugriff auf den Kostenschätzwert für eine Anfrage

```
    public double getQueryCost(String query) {
        double stats = CONST_NO_STATS;
       try {
            // Zuerst muss der Explain-Befehl auf die Query angesetzt werden
            String explainQuery = "EXPLAIN PLAN SET queryno = 1 ";
            explainQuery += "SET querytag = 'API' FOR " + query;

            if(this.debugMode)
                System.out.println("getQueryCost-Explain: " +  explainQuery);

            this.statement.execute(explainQuery);

            // Nach der Ausführung des Explain-Befehl werden
            // nun die Kosten ausgelesen
            query = "SELECT stmt.total_cost " +
            "FROM explain_statement stmt, explain_operator oper, " +
            "    explain_stream strm " +
            "WHERE stmt.queryno             = 1" +
            "  AND stmt.querytag            = 'API'" +
            "  AND stmt.explain_level       = 'P'" +
            "  AND stmt.EXPLAIN_REQUESTER   = oper.EXPLAIN_REQUESTER" +
            "  AND stmt.EXPLAIN_TIME        = oper.EXPLAIN_TIME" +
            "  AND stmt.SOURCE_NAME         = oper.SOURCE_NAME" +
            "  AND stmt.SOURCE_SCHEMA       = oper.SOURCE_SCHEMA" +
            "  AND stmt.EXPLAIN_LEVEL       = oper.EXPLAIN_LEVEL" +
            "  AND stmt.STMTNO              = oper.STMTNO" +
            "  AND stmt.SECTNO              = oper.SECTNO" +
            "  AND oper.operator_type       = 'RETURN'" +
            "  AND oper.EXPLAIN_REQUESTER   = strm.EXPLAIN_REQUESTER" +
            "  AND oper.EXPLAIN_TIME        = strm.EXPLAIN_TIME" +
            "  AND oper.SOURCE_NAME         = strm.SOURCE_NAME" +
            "  AND oper.SOURCE_SCHEMA       = strm.SOURCE_SCHEMA" +
            "  AND oper.EXPLAIN_LEVEL       = strm.EXPLAIN_LEVEL" +
            "  AND oper.STMTNO              = strm.STMTNO" +
```

```
            "  AND oper.SECTNO              = strm.SECTNO" +
            "  AND oper.operator_id         = strm.target_id " +
            "ORDER BY stmt.explain_time DESC";

            if(this.debugMode)
                System.out.println("getQueryCost-Kosten: " +  query);

            ResultSet resultSet = this.statement.executeQuery(query);

            // wenn Datensatz vorhanden, dann Daten auslesen
            if(resultSet.next())
                stats = resultSet.getFloat(1);
        } catch (SQLException ex) {
            System.err.println("SQL Exception: " + ex.getMessage());
        }
        return stats;
    }
```

5.2.5 DAOHelperDB2 – Zugriffsmethoden für Histogramme

IBM DB2 DBMS stellt alle Verteilungsstatistiken über die Sicht SYSSTAT.COLDIST bereit. Die für IBM DB2 V8.2 implementierte getHistogramData() Methode greift auf diese Sicht mit einer SELECT Anweisung zu, um ausschließlich Quantilstatistiken (TYPE='Q') für die als Parameter festgelegte Spalte auszulesen. Der Cursor wird im Resultset (Ergebnismenge) solange weitergesetzt, bis alle Quantile vollständig ausgelesen sind. Alle Elemente jedes Datensatzes (Bucket-Obergrenzen, Bucket-Kardinalitäten, usw.) werden dann entsprechend dem Spaltentyp colType in Vector-Objekten gespeichert. Zu beachten sind die Abfrage der möglichen NULL-Werte in Buckets mit colValue != null und die übereinstimmenden Unter- und Obergrenzen beim ersten Bucket im Histogramm. Des Weiteren müssen bei Zeichenwerten zusätzliche Anführungsstriche mit der Java Methode substring() (s. Klasse String) entfernt werden.

Gleitkommazahlen müssen als Spezialfall mit der getDouble() und nicht wie bei anderen Werten mit getObject() Methode aus dem Resultset ausgelesen werden, um die spätere Umwandlungsoperation von Object nach Double zu vermeiden. Dieser Spezialfall wurde nicht nur für DB2 sondern für alle 3 DBMS implementiert. Nachdem alle Histogrammdaten vollständig als Einzel-Vektoren vorliegen, werden diese als Elemente (Dimensionen) vom multidimensionalen histogramData-Vektor an die Logikschicht zurückgegeben, um anschließend ein Objekt der Histogramm-Klassen zu erzeugen und als Endergebnis zurückzugeben.

Quellcode 5.9 : DAOHelperDB2 – Lesezugriff mit getHistogramData()

```
public Vector getHistogramData(String colName) {
        Vector histogramData = new Vector();
        int bucketNumber = 0; // Bucketnummer
        Object colValue = null; // Obergrenze
        Object valCount = null; // Anzahl Wert <= Obergrenze
        Object numDv = null;
        double nulls = 0;

        Object low = null;
        Object high = null;
        Double dblLow = null;
        Double dblHigh = null;

        Vector lows = new Vector();
        Vector highs = new Vector();
        Vector cards = new Vector();
```

```
        Vector dvs = new Vector();

        try {
                query = "SELECT * " + "FROM SYSSTAT.COLDIST "
                + "WHERE TABSCHEMA = '" + this.schema + "' "
                + "  AND TABNAME = '" + this.tableName + "'"
                + "  AND COLNAME='" + colName + "'" + "  AND TYPE='Q' "
                + "ORDER BY SEQNO";

                if (this.debugMode)
                        System.out.println("getHistogramData: " + query);

                ResultSet resultSet = this.statement.executeQuery(query);

                // Spaltentyp aus der Datenbank holen
                int colType = getColType(colName);

                while (resultSet.next()) {
                        // wenn Datensatz vorhanden, dann Daten auslesen
                        bucketNumber = resultSet.getInt(5); // SEQNO
                        colValue = resultSet.getObject(6); // COLVALUE = rangeHigh

                        if (colValue != null) {
                                valCount = resultSet.getObject(7); // VALCOUNT
                                numDv = resultSet.getObject(8); // DISTCOUNT

                                cards.add(valCount);
                                dvs.add(numDv);

                                switch (colType) {

                                default: // Default-Case für ganze Zahlen
                                        high = colValue;
                                        // 1. Untergrenze im Histogramm=Obergrenze
                                        if (low == null)
                                                low = high;
                                        lows.add(low);
                                        highs.add(high);
                                        // Utergrenze(i)=Obergrenze(i-1)
                                        low = high;
                                        break;

                                case Types.DOUBLE:
                                        dblHigh = Double.
                                                valueOf(resultSet.getDouble(6));
                                        // 1. Untergrenze im Histogramm=Obergrenze
                                        if (dblLow == null)
                                                dblLow = dblHigh;
                                        lows.add(dblLow);
                                        highs.add(dblHigh);
                                        dblLow = dblHigh;
                                        break;

                                case Types.VARCHAR:
                                        // Bei VarChar Werten wird das Ergebnis in ''
                                        // eingeschlossen, deshalb werden die ''
                                        // entfernt
                                        high = colValue.toString().substring(1,
                                                colValue.toString().length() - 1);
                                        // 1. Untergrenze im Histogramm=Obergrenze
                                        if (low == null)
                                                low = high;
                                        lows.add(low);
                                        highs.add(high);
                                        // Utergrenze(i)=Obergrenze(i-1)
                                        low = high;
                                        break;
                                }
                        }
                }

                if (lows.size() > 0) {
                        // falls Verteilungsstatistik vorhanden
                        histogramData.add(lows);
                        histogramData.add(highs);
                        histogramData.add(cards);
                        histogramData.add(dvs);
                        return histogramData;
```

```
                }
        } catch (SQLException ex) {
                System.err.println("SQL Exception: " + ex.getMessage());
        }
        return null;
}
```

Um einzelne Werte eines Histogramms mit saveHistogram() in der Datenbank abzuspeichern, muss das an die Methode übergebene Histogramm-Objekt in einer for-Schleife Bucket für Bucket gelesen werden, um die numDv, numNulls und rangeHigh Attribute jedes einzelnen Buckets und einzeln in der Datenbank zu sichern.

Quellcode 5.10 : DAOHelperDB2 – Schreibzugriff mit saveHistogram()

```
public boolean saveHistogram(String colName, Histogram histogram) {
        try {
                Object highValue = null;
                double card = 0;
                double totalCard = 0;
                double dv = 0;

                for(int i=1; i <= histogram.size(); i++) {
                        highValue = histogram.getBucket(i-1).getHigh();
                        card = histogram.getBucket(i-1).getCard();
                        totalCard += card;
                        dv = histogram.getBucket(i-1).getDv();

                        query = "UPDATE SYSSTAT.COLDIST " +
                        "SET COLVALUE = '" + highValue.toString() + "', " +
                        "    VALCOUNT = " + totalCard + ", " +
                        "    DISTCOUNT = " + dv + " " +
                        " WHERE TABSCHEMA = '" + this.schema + "' " +
                        "  AND TABNAME = '" + this.tableName + "' " +
                        "  AND COLNAME = '"+ colName + "' " +
                        "  AND SEQNO = " + i;

                        if(this.debugMode)
                                System.out.println("saveHistogram: " + query);

                        this.statement.executeUpdate(query);
                }

        } catch (SQLException ex) {
                System.err.println("SQL Exception: " + ex.getMessage());
        }
        return true;
}
```

5.3 Oracle Version 10g

5.3.1 DAOHelperOracle - Lesezugriff mit get-Methoden

Eine get-Methode für das Oracle 10g DBMS unterscheidet sich lediglich in der SQL-Anweisung von seinem DB2 Pedant. Da Oracle keine spezielle Markierung für nicht vorhandene Statistikwerte verwendet, werden alle 0-Werte als nicht vorhandene Statistik eingestuft.

Quellcode 5.11 : DAOHelperOracle – Einfacher Lesezugriff

```
public int getColNumNulls(String colName) {
	int stats = CONST_NO_STATS;
	try {
		query = "SELECT NUM_NULLS " +
		"FROM SYS.USER_TAB_COL_STATISTICS " +
		"WHERE TABLE_NAME = '" + this.tableName + "' " +
		"AND COLUMN_NAME = '" + colName + "'";

		if(this.debugMode)
			System.out.println("getColNumNulls: " + query);

		ResultSet resultSet = this.statement.executeQuery(query);

		if(resultSet.next())
			// wenn Datensatz vorhanden, dann Daten auslesen
			if(resultSet.getInt(1) != 0)
				// nur wenn Statistik vorhanden
				stats = resultSet.getInt(1);
	} catch (SQLException ex) {
		System.err.println("SQL Exception: " + ex.getMessage());
	}
	return stats;
}
```

5.3.2 DAOHelperOracle - Schreibzugriff mit save-Methoden

Die Speicherung der Statistikdaten erfolgt wie 3.2.7 beschrieben ausschließlich über Prozeduren aus dem Oracle-Paket DBMS_STATS (s. Quellcode 5.12).

Quellcode 5.12 : DAOHelperOracle – Einfacher Schreibzugriff

```
public boolean saveColNumNulls(long colNumNulls, String colName) {
	try {
		// Schreibzugriff nur über DBMS_STATS
		query = "BEGIN " +
		"DBMS_STATS.SET_COLUMN_STATS(OWNNAME=>'"+ schema + "', " +
					"TABNAME=>'" + tableName + "', " +
					"COLNAME=>'"+ colName + "', " +
					"NULLCNT=>"+ colNumNulls +"); " +
		"END  ;";
		if(this.debugMode)
			System.out.println("saveColNumNulls: " + query);
		// Neue Statistik durchschreiben
		this.statement.executeUpdate(query);
	} catch (SQLException ex) {
		System.err.println("SQL Exception: " + ex.getMessage());
	}
	return true;
}
```

5.3.3 DAOHelperOracle - Kostenschätzwerte und Kardinalitäten

Das Lesen von Kostenschätzwerten und Kardinalitäten von Anfragen erweist sich auch in Oracle als sehr einfach. Der Zugriff erfolgt wieder in zwei Schritten. Zuerst wird die zu analysierende SQL Query in einer EXPLAIN PLAN Anweisung eingesetzt. Die durch diesen Schritt erzeugten Daten werden dann explizit über die zuvor definierte statement_id aus der PLAN_TABLE ausgelesen.

Quellcode 5.13 : DAOHelperOracle – Zugriff auf die Kardinalität einer Anfrage

```
public int getQueryCard(String query) {
	int stats = CONST_NO_STATS;
	try {
		// Zuerst muss der Explain-Befehl auf die Query angesetzt werden
		String explainQuery = "EXPLAIN PLAN SET statement_id='API' FOR ";
		      explainQuery += query;

		if(this.debugMode)
			System.out.println("getCard-Explain: " +  explainQuery);

		this.statement.execute(explainQuery);

		// Nach der Ausführung des Explain-Befehls wird
		// nun Kardinalität ausgelesen
		query = "SELECT cardinality " +
		"FROM PLAN_TABLE " +
		"WHERE statement_id='API'";

		if(this.debugMode)
			System.out.println("getCard-Kardinalität: " +  query);

		ResultSet resultSet = this.statement.executeQuery(query);

		if(resultSet.next())
			// wenn Datensatz vorhanden, dann Daten auslesen
			stats = resultSet.getInt(1);
	} catch (SQLException ex) {
		System.err.println("SQL Exception: " + ex.getMessage());
	}
	return stats;
}
```

5.3.4 DAOHelperOracle – Zugriffsmethoden für Histogramme

Vorbedingung: Alle Tabellen bestehen zwecks Vereinfachung nur aus einer Partition. Damit wird es sichergestellt, dass für jede Spalte maximal ein Histogramm vorliegt. Diese Vorbedingung ist nur für die im Rahmen dieser Arbeit entstehende DAO-Schicht Implementierung relevant und kann ansonsten vernachlässigt werden.

Die Implementierung der getHistogramData() Methode erforderte sehr viel Vorarbeit und die Erstellung einer zusätzlichen Konvertierungsmethode (Quellcode 5.14) für Verteilungsstatistiken. Der Hintergrund und die Funktionsweise dieser Methode wurden in 3.2.5 erläutert. Die Berechnung der Hex-Zeichenkette mit ASCII erfolgt bereits in der SQL Anfrage über substr(to_char(endpoint_value,'fm'||rpad('x',30,'x')),1,13), wobei to_char die Umwandlung von der normalisierten Darstellung in die Hexadezimaldarstellung durchführt.

Quellcode 5.14 : DAOHelperOracle – Lesezugriff mit getHistogramData()

```
private String getStrColValue(Object endpoint_number) {
	String numColValue = endpoint_number.toString();
	String strColValue = "";
	String hexCode="";
	char character;
	int code; // ASCII Code

	while(numColValue.length()>1) {
		hexCode = numColValue.substring(0,2);

		// ASCII Code in Hex zu Int
```

```
                code = Integer.parseInt(hexCode,16);
                // Zeichen über ASCII Code erhalten
                character = (char)code;

                // 1.Zeichen der Zeichenkette (=Hex-Werte) entfernen
                numColValue = numColValue.substring(2, numColValue.length());

                // Wenn Zeichen nicht 0, dann zum String hinzufügen
                if(code != 0) {
                        strColValue += character;
                }
        }

        return strColValue;
}
```

Man beachte jedoch, dass diese Vorgehensweise nicht immer zuverlässig funktioniert. Bei Spaltenwerten vom Typ VARCHAR2 wird deshalb falls vorhanden ENDPOINT_ACTUAL_VALUE als Bucket-Obergrenze verwendet(s. Quellcode 5.15). Des Weiteren erfasst Oracle keine Statistiken für die Anzahl unterschiedlicher Werte pro Bucket. Dieser Wert wird jedoch bei der Erstellung von Verteilungstatistiken in Oracle (s.o.) als SIZE Parameter an das DBMS übergeben oder kann aus anderen Statistikwerten (NUM_BUCKETS, NUM_DISTINCT) berechnet werden. Die Anzahl der Null-Werte pro Bucket sind ebenfalls nicht vorhanden. Der bereits für die DB2 Implementierung der getHistogramData()-Methode besprochene Spezialfall beim Zugriff auf Gleitkommazahlen muss auch für Oracle beachtet werden.

Quellcode 5.14 : DAOHelperOracle – Lesezugriff mit getHistogramData()

```
public Vector getHistogramData(String colName) {
        Vector histogramData = new Vector();
        int bucketNumber = 0; // Bucketnummer
        Object colValue = null; // Obergrenze
        Object valCount = null; // Anzahl Wert <= Obergrenze
        Object numDv = null;

        Object low = null;
        Object high = null;
        Double dblLow = null;
        Double dblHigh = null;

        Vector lows = new Vector();
        Vector highs = new Vector();
        Vector cards = new Vector();
        Vector dvs = new Vector();

        try {
                query = "SELECT ENDPOINT_NUMBER, "
                     + "substr(to_char(endpoint_value,'fm'||rpad('x',30,'x')),1,13),"
                        + "ENDPOINT_ACTUAL_VALUE "
                        +   "FROM SYS.ALL_TAB_HISTOGRAMS "
                        + "WHERE TABLE_NAME='" + this.tableName + "'"
                        + "  AND COLUMN_NAME='" + colName + "' "
                        + "ORDER BY ENDPOINT_NUMBER";

                if (this.debugMode)
                        System.out.println("getHistogramData: " + query);

                ResultSet resultSet = this.statement.executeQuery(query);

                // Spaltentyp aus der Datenbank holen
                int colType = getColType(colName);

                while (resultSet.next()) {
                        // wenn Datensatz vorhanden, dann Daten auslesen
                        bucketNumber = resultSet.getInt(1);
                        colValue = resultSet.getObject(2);

                        if (colValue != null) {
                                valCount = "5"; // s. SIZE, Keine Statistik vorhanden
                                numDv = "0";   // Keine Statistikdaten vorhanden
```

```
                                cards.add(card + "");
                                dvs.add(numDv);

                                switch (colType) {
                                default:
                                        high = colValue;
                                        if (low == null
                                                low = high;
                                        lows.add(low);
                                        highs.add(high);
                                        // Utergrenze(i)=Obergrenze(i-1)
                                        low = high;
                                        break;

                                case Types.VARCHAR:
                                        high = getStrColValue(colValue);
                                        high.toString();

                                        // 1. Untergrenze im Histogramm=Obergrenze
                                        if (low == null)
                                                low = high;
                                        lows.add(low);
                                        highs.add(high);
                                        // Utergrenze(i)=Obergrenze(i-1)
                                        low = high;
                                        break;

                                case Types.DOUBLE:
                                        dblHigh = Double.
                                                  valueOf(resultSet.getDouble(6));
                                        if (dblLow == null
                                               dblLow = dblHigh;
                                        lows.add(dblLow);
                                        highs.add(dblHigh);
                                        dblLow = dblHigh;
                                break;
                                }
                        }
                }

                if (lows.size() > 0) { // falls Verteilungsstatistik vorhandne
                        histogramData.add(lows);
                        histogramData.add(highs);
                        histogramData.add(cards);
                        histogramData.add(dvs);
                        return histogramData;
                }
        } catch (SQLException ex) {
                System.err.println("SQL Exception: " + ex.getMessage());
        }
        return null;
}
```

Die in 3.2.7 besprochene Vorgehensweise zum Speichern von Histogrammdaten in Oracle Systemen wurde in saveHistogram() umgesetzt. Die zu speichernde Verteilungsstatistiken werden in einem Histogramm-Objekt übergeben. Darüber hinaus ist die Angabe einer Spalte (colName Parameter) für die im Konstruktor angegebene Tabelle für die Statistikaktualisierung mittels SET_COLUMN_STATS notwendig. Einige Histogrammdaten (z.B. Bucket-Obergrenzen) werden in Abhängigkeit von Ihrem Datentyp entweder in ein Array vom Typ CHARARRAY oder vom Typ NUMARRAY geschrieben. Die auf diese weise erzeugte StatRec Struktur wird zum Schluss über SET_COLUMN_STATS in die Datenbank geschrieben.

Quellcode 5.15 : DAOHelperOracle –Schreibzugriff mit saveHistogram()

```
public boolean saveHistogram(String colName, Histogram histogram) {
        try {
                Object highValue = null;
```

```
                String values = "";
                int type;

                int numBuckets = histogram.size();
                double nullcnt = histogram.getNulls();
                double distcnt = histogram.getDv();
                double density = histogram.getDv() / distcnt;
                String histClass = histogram.getClass().toString();
                if (histClass.
                    equals("class histogramPropagator.histograms.CharHistogram")) {
                       type = Types.VARCHAR;
                } else
                       type = Types.NUMERIC;

                for (int i = 1; i <= numBuckets; i++) {
                       switch (type) {
                       case Types.VARCHAR:
                              highValue = histogram.getBucket(i - 1).getHigh();
                              values = values + "'" + highValue + "'";
                              break;

                       case Types.NUMERIC:
                              highValue = histogram.getBucket(i - 1).getHigh();
                              values = values + highValue;
                              break;
                       }

                       if (i != numBuckets) {
                              values = values + ", ";
                       }
                }
                query = "DECLARE " + "SREC DBMS_STATS.STATREC;"
                + "NOVALS DBMS_STATS.NUMARRAY;"
                + "NOVALSCHAR DBMS_STATS.CHARARRAY;" + "BEGIN "
                + "SREC.EAVS := 0; " + "SREC.CHVALS := NULL; "
                + "SREC.EPC := " + numBuckets + "; "
                + "SREC.BKVALS := NULL;";

                // Array in Abhängigkeit vom Datentyp der Elemente anlegen
                switch (type) {
                case Types.VARCHAR:
                  query += "NOVALSCHAR := DBMS_STATS.CHARARRAY(" + values + "); "
                     + "DBMS_STATS.PREPARE_COLUMN_VALUES(SREC,NOVALSCHAR); ";
                       break;

                case Types.NUMERIC:
                  query += "NOVALS := DBMS_STATS.NUMARRAY(" + values + "); "
                     + "DBMS_STATS.PREPARE_COLUMN_VALUES(SREC,NOVALS);";
                       break;
                }
                query += "DBMS_STATS.SET_COLUMN_STATS('" + this.schema + "'," + "'"
                + this.tableName + "'," + "'" + colName + "', "
                + "NULL, NULL, NULL, " + distcnt + ", " + density + ", "
                + nullcnt + ", " + "srec); " + "END;";

                if (this.debugMode)
                       System.out.println("saveHistogram: " + query);

                this.statement.executeUpdate(query);
        } catch (SQLException ex) {
                System.err.println("SQL Exception in saveHistogram: "
                               + ex.getMessage());
        } return true;
 }
```

5.4 Microsoft SQL Server 2005 (Yukon)

Wie in Kapitel 3 festgestellt sind in SQL Server 2005 keine Schreibzugriffe möglich. Deshalb werden im Weiteren nur get-Methoden für den Lesezugriff auf Statistiken vorgestellt.

5.4.1 Lesezugriff mit get-Methoden

Statistische Informationen eines SQL Server Systems lassen sich mit dem Befehl DBCC SHOW_STATISTICS auslesen (s. 3.3.7). Die Ergebnismenge dieser Anfrage beinhaltet unterschiedliche Statistikarten und wird als „Multidimensional Resultset“ zurückgegeben. Die gewünschten Daten können dann gegebenenfalls nach der Anwendung von getMoreResults() aus dem entsprechenden Resultset ausgelesen werden.

Quellcode 5.16 : DAOHelperSQLServer – Einfacher Lesezugriff auf Spaltenstatistiken

```
public int getColAvgLength(String colName) {
	int stats = CONST_NO_STATS;

	try {
		query = "USE DIPLOM; " +
		"DBCC SHOW_STATISTICS ('APITEST', '" + colName + "'); ";

		if(this.debugMode)
			System.out.println("getColAvgLength: " + query);

		this.statement.execute(query);

		// Zum 2. Resultset wechseln
		this.statement.getMoreResults();
		this.statement.getMoreResults();
		resultSet = this.statement.getResultSet();

		// wenn Datensatz vorhanden, dann Daten auslesen
		if(resultSet.next())
			stats = resultSet.getInt(2);
		resultSet.close();
	} catch (SQLException ex) {
		System.err.println("SQL Exception: " + ex.getMessage());
	}
	return stats;
}
```

5.4.2 DAOHelperSQLServer - Kostenschätzwerte und Kardinalitäten

Die Methode getQueryCost() macht von der SHOWPLAN_ALL Option Gebrauch (s. 3.3.6). Nach der Aktivierung dieser Option kann sowohl der Kostenschätzwert als auch die Kardinalität einer Anfrage (getQueryCard() Methode) aus dem ersten Datensatz der Ergebnismenge ermittelt werden.

Quellcode 5.17 : DAOHelperSQLServer – Zugriff auf den Kostenschätzwert für eine Anfrage

```
public double getQueryCost(String query) {
	double stats = CONST_NO_STATS;
	try {
		// SHOWPLAN_ALL muss eingeschaltet werden
		String spQuery = "SET SHOWPLAN_ALL ON;";
		statement.execute(spQuery);
```

```
                if(this.debugMode)
                        System.out.println("getQueryCost-Kosten: " +  query);

                this.statement.executeQuery(query);
                ResultSet resultSet = this.statement.getResultSet();

                // nur den 1. Datensatz lesen
                if(resultSet.next())
                        // wenn Datensatz vorhanden, dann Daten auslesen
                        stats = resultSet.getFloat(13); // TotalSubtreeCost

                // SHOWPLAN_ALL muss jetzt ausgeschaltet werden
                spQuery = "SET SHOWPLAN_ALL OFF;";
                statement.execute(spQuery);

        } catch (SQLException ex) {
                System.err.println("SQL Exception: " + ex.getMessage());
        }
        return stats;
}
```

5.4.3 DAOHelperSQLServer – Zugriffsmethoden für Histogramme

Der letzte Ausschnitt aus dem Quellecode des Prototyps zeigt die getHistogramData() Methode für Microsoft SQL Server 2005. Sie ähnelt sehr stark der gleichnamigen Implementierung für das IBM DB2 System. Der Unterschied liegt in der Verwendung von DBCC SHOW_STATISTICS zum Auslesen von Verteilungsstatistiken. Die im dritten Resultset gespeicherten Daten haben außerdem ein wenig abweichende Semantik. So wird die Anzahl der unterschiedlichen Werte in einem Histogramm-Bucket aus der Summe von DISTINCT_RANGE_ROWS und einer 1 gebildet, falls es EQ_ROWS>0 ist. Dieser Fall tritt ein, wenn es in einem Bucket Werte gibt, die mit dem Wert der jeweiligen Bucket-Obergrenze übereinstimmen.

Quellcode 5.21 : DAOHelperSQLServer – Lesezugriff mit getHistogramData()

```
public Vector getHistogramData(String colName) {
        Vector histogramData = new Vector();
        int bucketNumber = 0;   // Bucketnummer
        Object colValue = null; // Obergrenze
        Object valCount = null; // Anzahl Wert <= Obergrenze
        int numDv = 0;
        int numEq = 0;
        double nulls = 0;

        Object low =  null;
        Object high = null;
        Double dblLow = null;
        Double dblHigh = null;

        Vector lows = new Vector();
        Vector highs = new Vector();
        Vector cards = new Vector();
        Vector dvs = new Vector();


        try {
                query = "USE DIPLOM; " +
                "DBCC SHOW_STATISTICS ('APITEST', apistats); ";

                if(this.debugMode)
                        System.out.println("getHistogramData: " + query);

                this.statement.execute(query);
                this.statement.getMoreResults(); // Grundinfo
                this.statement.getMoreResults(); // Density/Avg Resultset
                this.statement.getMoreResults();
                this.statement.getMoreResults(); // Histogramm Resultset
```

```
                ResultSet resultSet = this.statement.getResultSet();

                // Spaltentyp aus der Datenbank holen
                int colType = getColType(colName);

                // wenn Datensatz vorhanden, dann Daten auslesen
                while(resultSet.next()) {
                        colValue =  resultSet.getObject(1);

                        if(colValue != null) {
                                valCount = resultSet.getObject(2); // RANGE_ROWS
                                numEq = resultSet.getInt(3);  // EQ_ROWS
                                numDv = resultSet.getInt(4);  // DISTINCT_RANGE_ROWS
                                if(numEq > 0)
                                        numDv = numDv + 1;
                                cards.add(valCount);
                                dvs.add(numDv+""); // Als Objekt speichern


                                switch(colType) {
                                default:
                                        high = colValue;
                                       // 1. Untergrenze im Histogramm=Obergrenze
                                       if(low==null)
                                          low=high;
                                        lows.add(low);
                                        highs.add(high);
                                        // Untergrenze(i)=Obergrenze(i-1)
                                        low = high;
                                        break;

                                case Types.DOUBLE:
                                        dblHigh = Double.
                                                  valueOf(resultSet.getDouble(6));
                                        if(dblLow==null)
                                               dblLow=dblHigh;
                                        lows.add(dblLow);
                                        highs.add(dblHigh);
                                        dblLow = dblHigh;
                                        break;

                                case Types.VARCHAR:
                                        // 1. Untergrenze im Histogramm=Obergrenze
                                        if(low==null)
                                                low=high;
                                        lows.add(low);
                                        highs.add(high);
                                        // Utergrenze(i)=Obergrenze(i-1)
                                        low = high;
                                        break;
                                }
                        }
                }

                if(lows.size()>0) { // falls Verteilungsstatistik vorhandne
                        histogramData.add(lows);
                        histogramData.add(highs);
                        histogramData.add(cards);
                        histogramData.add(dvs);
                        return histogramData;
                }
        } catch (SQLException ex) {
                System.err.println("SQL Exception: " + ex.getMessage());
        }
        return null;
}
```

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung der Arbeit

Optimierungsprozesse spielen eine sehr wichtige Rolle für den gesamten Betreib von Datenbankmanagementsystemen. Sowohl bei der herkömmlichen und in modernen DBMSs eingesetzten kostenbasierten Optimierung (Cost Based Optimization) als auch bei der Optimierung von zusammenhängenden Anfragesequenzen (z.B. Caurse-Grained Optimization im Ceops-Projekt) werden statistische Informationen und Kostenschätzwerte aus DBMSs hinzugezogen. Sei es die Berechnung des besten Ausführungsplanes aufgrund von geschätzten Ausführungskosten einer Anfrage oder die Restrukturierung von Anfragesequenzen aufgrund einer kostenbasierte Kontrollstrategie.

Um die Optimierungsprozesse kennen zu lernen und die kompletten Zusammenhänge dieser Prozesse mit verschiedenen Statistiken zu verstehen, wurde zu Beginn der Arbeit umfangreiches Grundlagenwissen sowohl über die Regel- und Kostenbasierte Optimierung als auch über die so genannte Caurse-Grained Optimization Methode vorgestellt und entsprechende Definitionen eingeführt. Außerdem wurden im Grundlagenkapitel neben unterschiedlichen Möglichkeiten für die Statistikerstellung in modernen DBMSs auch die wichtigsten Statistikarten und Zugriffsoperation für diese Statistiken besprochen.

In der Analysephase (Kapitel 3) wurden die drei bekannten kommerziellen DBMSs IBM DB2 V8.2, Oracle 10g und Microsoft SQL Server 2005 (Yukon) untersucht. Als Ergebnis dieses Abschnitts ließen sich für alle 3 DBMSs ausführliche Listen mit Statistikwerten für Tabellen (Tabellenstatistiken), Spalten (Spalten- und Verteilungsstatistiken) und Indexe (Indexstatistiken) mit entsprechenden datenbankspezifischen Datentypen, Beschreibungen und erlaubten Zugriffsoperationen erstellen.

Anschließend erfolgte die Spezifikation einer generischen von dem darunterliegenden Datenbanksystem unabhängigen Kosten- und Statistikschnittstelle (API). Nach dem an die Spezifikation angelehnten Entwurf wurde außerdem eine prototypische Implementierung dieser Schnittstelle für die o.g. DBMS in der Programmiersprache Java erstellt. Die Beachtung aller datenbankspezifischen Einschränkungen (Constrains) versetzt jeden Benutzer dieser Schnittstelle in die Lage, statistische Informationen aus dem jeweiligen Datenbanksystem auf einfache Art und Weise auszulesen und zu aktualisieren. Das Datenbanksystem und die Statistikdaten bleiben dabei auch nach Schreibzugriffen in einem logisch konsistenten Zustand.

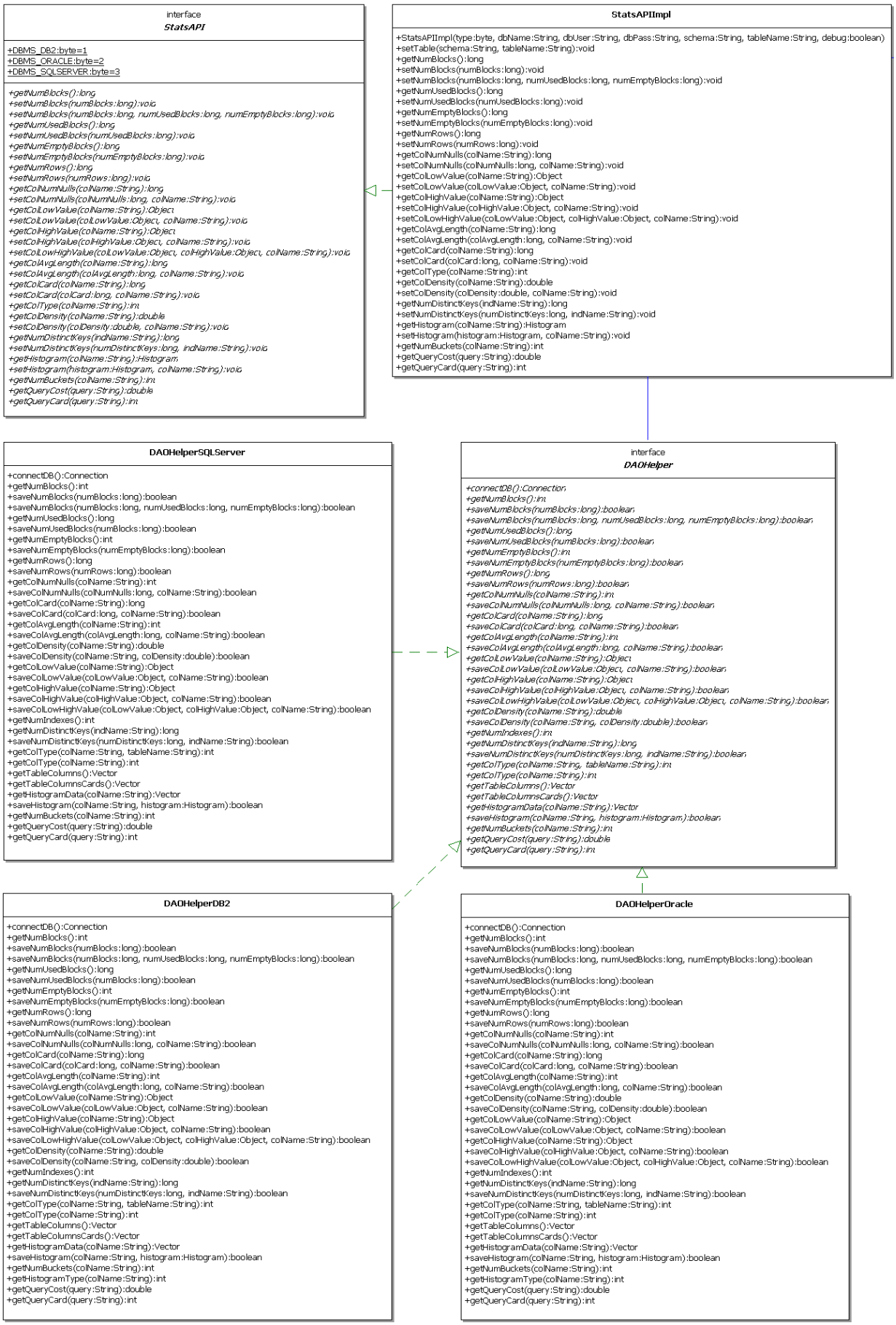
6.2 Ausblick

Die im Rahmen dieser Arbeit entwickelte generische Kosten- und Statistik-API für Datenbankmanagementsysteme stellt zahlreiche Funktionen zur Verfügung, um die Statistiken einzelner Datenbankobjekte (z.B. Tabellen, Spalten, Indexe) auszulesen, für die Kostenbestimmung zu modifizieren und zwecks Performanzoptimierung im Data Dictionary des jeweiligen Datenbanksystems wieder abzuspeichern.

Wie bereits mehrmals geschildert, sind vor allem schreibende Zugriffe auf DBMS außerordentlich kritisch, weil die Datenbankintegrität dadurch zerstört werden kann. Die besondere Aufmerksamkeit sollte deshalb dem Ausbau der systemeigenen Validierungskomponente geschenkt werden. So wäre es zum Beispiel vorstellbar, die im Prototyp einheitlich für alle drei DBMS implementierte Validierungsklasse aufzuteilen und für jedes Datenbanksystem unter Beachtung besonderer Gegebenheiten mit einer einheitlichen Schnittstelle (Interface) zu implementieren.

Sollte die Schnittstelle im Rahmen des Ceops-Projekts zum Einsatz kommen, wäre eine nahtlose Integration in den vorhanden CGO-Optimierer mit der Anpassung der verwendeten Datenstrukturen sehr vorteilhaft. Durch diese Maßnahme wäre die Verkürzung der Ausführungszeiten für die über die Schnittstelle laufenden Anfrage möglich. Eine zukünftige Erweiterung der Schnittstelle für andere DBMSs würde sich dank dem generischen Aufbau als ziemlich einfach erweisen und könnte deswegen sehr schnell durchgeführt werden.

I. Anhang A – Detailliertes UML Klassendiagram



II. Glossar

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CBO	Cost Based Optimization
CGO	Coarse-Grained Optimization
DAO	Data Access Object
DAOHelper	Schnittstellendefinition (Interface) für die DAO Schicht
DAOHelperDB2	Implementierung der DAOHelper-Schnittstelle für IBM DB2 V8.2
DAOHelperOracle	Implementierung der DAOHelper-Schnittstelle für Oracle 10g
DAOHelperSQLServer	Implementierung der DAOHelper-Schnittstelle für Microsoft SQL Server 2005
DAO Layer	Schicht für den Zugriff auf Datenbank
DBMS	Datenbankmanagementsystem, Database Management System
ER	Entity Relationship
Interface	Schnittstelle
Logic Layer	Logikschicht der Kosten- und Statistik-API, welche Statistiken zwischenspeichert und vor bei Schreibzugriffen notwendige Vali dierungsoperationen durchführt.
RBO	Rule Based Optimization
Resultset	Ergebnismenge
StatsAPI	Interface der Kosten- und Statistik-API
StatsAPIImpl	Implementierungsklasse der Kosten- und Statistik-API
UML	Unified Modeling Language
UTF8 Unicode	Transformation Format mit 8-Bit

III. Literaturverzeichnis

[BR04] Breitling, W.; Using DBMS_STATS in access path optimization, 2004

[DAT] http://www.datenbank-plsql.de; Oracle PL-SQL Tutorial - Die Programmiersprache von Oracle

[IBM04a] IBM DB2 Universal Database, Handbuch auf den Seiten http://publib.boulder.ibm.com/infocenter/db2luw/v8/topic/com.ibm.db2.udb.doc/tutr/db2tv/veid_cncp_aplan.htm, 2004

[IBM04b] IBM DB2 Universal Database, Administration Guide: Performance, 2004

[IBM04c] IBM DB2 Universal Database, SQL Reference Volume 1, Version 8.2

[DEIN92] Deininger, M., Lichter, H., Ludewig, J., Schneider, K.; Studien-Arbeiten. Ein Leitfaden zur Vorbereitung, Durchführung und Betreuung von Studien-, Diplom- und Doktorarbeiten am Beispiel Informatik. vdf, Zürich und Teubner, Stuttgart, 1992

[GRE02] Green, C.D.; Oracle 9i Database Performance Tuning Guide and Reference, Release 2, 2002

[HER] Hertel, P.; Projekt Diplomarbeit

[HFLP] Haas, M. L., Freytag, J.C., Lohman, G.M., Pirahesh, H.; Extensible Query Processing in Starburst, IBM Almaden Research Center, San Jose, CA 95120

[HOR] Horn, Torsten; http://www.torsten-horn.de/techdocs/java-sql.htm

[HOT88] Hou, W.C.; Ozsoyoglu,G.; Taneja, B.K.: Statistical estimators for relational algebra expressions. In Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1988

[IOA03] Ioannidis, Y.; The History of Histograms (abridged), University of Athens, Panepistimioupolis, 2003

[KMRSB03] Kraft, T., Mitschang, B., Rantzau, R., Schwarz, H.; Coarse-Grained Optimization: Techniques for Rewriting SQL Statement Sequences, 2003

[MSDN] Microsoft Developers Network, http://msdn2.microsoft.com, Stand 5.12.2005

[MÜL05] Müller, T.; Statistikpropagation und Kostenschätzung für Anfragesequenzen, Diplomarbeit Nr. 2277, 2005

[HK05] Hanson, E.N., Kollar, L.; Statistics Used by the Query Optimizer in Microsoft SQL Server, 2005

[OAT] Oracle AskTom Forum; auf Seiten http://asktom.oracle.com/pls/ask/f?p=4950:8:::::F4950_P8_DISPLAYID:707586567563

[ORA06a] Oracle Database Performance Tuning Guide 10g Release 1 (10.1), Part Number B10752-01

[ORA06b] Oracle Database Administrator's Guide 10g Release 1 (10.1), Part Number B10739-01

[ORA06c] Oracle Database Reference 10g Release 1 (10.1), Part Number B10755-01

[ORA06d] Oracle Database SQL Quick Reference 10g Release 1 (10.1), Part Number B10758-01

[ORA06e] PL/SQL Packages and Types Reference 10g Release 1 (10.1), Part Number B10802-01

[PAD03] Padhi, K.P.; Oracle Optimizer: Moving to and working with CBO, Database Journal, 2003

[RJB02] Rankings, R., Jensen, P., Bertucci, P.; Microsoft SQL Server 2000 Unleashed, SAMS Publishing, USA, 2002

[SAC+] Selinger, P.G., Astrahan, M.M., Chamberlin, D. D., Lorie, A.A., Price, T.G.; IBM Research Division, San Jose, California 95193; Access Path Selection in a Relational Database Management System

[SUN] SUN Corporation; Creating Complete JDBC Applications; http://java.sun.com/docs/books/tutorial/jdbc/basics/complete.html

[KRA05] Kraft, T.; Internal Technical Report, Universität Stuttgart, 2005

[UMA] Universität Magdeburg; UML Tutorial http://www-ivs.cs.uni-magdeburg.de/~dumke/UML/inhalt.htm

[WA03] Watt, S.: Oracle SQL*Plus User's Guide and Reference Release 10.1 Part No. B12170-01, Stand 12.2003

[WIK] Wikipedia; http://de.wikipedia.org

[OMG] Object Management Group, Unified Modeling Language, http://www.uml.org

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.

(Maxim Marchassin)