Studiengang:	**Informatik**
Betreuer:	**Dipl.-Inf. Tobias Kraft**
Prüfer:	**Prof. Dr.-Ing. habil. Bernhard Mitschang**
Beginn am:	**09.07.2003**
Beendet am:	**08.01.2004**
CR-Klassifikation:	**H.2.4, H.3.4, H.5.2**

Studienarbeit Nr. 1914

Erstellung einer Testumgebung für Anfragesequenz-Rewrite-Regeln

Tobias Flaig

Fakultät Elektrotechnik, Informatik,
Informationstechnik
Universität Stuttgart
Institut für Parallele und Verteilte Systeme
Abteilung Anwendersoftware
Universitätsstr. 38
70569 Stuttgart

Inhaltsverzeichnis

1. Einleitung und Grundlagen....1
1.1 Anfrage....2
1.2 Anfragesequenz....2
1.3 Rewrite-Regeln....3
1.3.1 Merge Select....3
1.3.2 Merge Where....5
1.3.3 Merge Having....7
1.3.4 Where To Group....9
1.3.5 Join To Table....11
2. Software....14
2.1 Anforderungen an das Softwareumfeld....16
2.2 Interner Aufbau der Software....17
2.3 Konfiguration der Testumgebung....20
2.4 Bedienung der Software und ihre Randbedingungen....21
2.4.1 Basistabellen erzeugen....21
2.4.2 Indizes....25
2.4.3 Automatische Anfragengenerierung....25
2.4.3.1 Merge Select....26
2.4.3.2 Merge Where....27
2.4.3.3 Merge Having....28
2.4.3.4 Where To Group....29
2.4.3.5 Join To Table....30
2.4.4 Teststart und Protokollfenster....31
3. Testreihen....33
3.1 Theoretische Überlegungen und Erwartungen....34
3.1.1 Prozentuale Gleichverteilung....34
3.1.2 Verteilung im Verhältnis x%=100%-y%....36
3.2 Testreihen und ihre Auswertung....38
3.2.1 Prozentuale Gleichverteilung....38
3.2.2 Verteilung im Verhältnis x%=100%-y%....41
4. Zusammenfassung und Ausblick....44
5. Quellennachweis....45

Anhang: CD-ROM mit Software, Quelltext und Dateien für Stored Procedure

© IPVR, Universität Stuttgart, 2004

1. Einleitung und Grundlagen

Im Rahmen dieser Studienarbeit sollte eine Testumgebung für Anfragesequenzen erstellt werden.
Eine Anfrage stellt eine Anweisung an eine Datenbank dar, um gewünschte Informationen auszuwählen. Eine Anfragesequenz ist eine Folge von Anfragen, wobei jede dieser Anfragen auf die Ergebnisse voriger Anfragen der Folge zugreifen kann. Dies wird dadurch realisiert, dass jede Anfrage ihr Ergebnis temporär in einer Tabelle abspeichert, die nach Abarbeitung der Anfragesequenz wieder gelöscht wird.

Zur Optimierung dieser Anfragesequenzen werden Rewrite-Regeln eingesetzt. Diese Rewrite-Regeln sind jedoch nicht allgemeingültig, d.h. je nach Konstellation der Anfragesequenz kann deren Anwendung zu einer Verbesserung, aber auch zu einer Verschlechterung der Ausführungszeiten führen. Innerhalb dieser Studienarbeit soll nun eine Testumgebung erstellt werden, die für die gegebenen Rewrite-Regeln jeweils automatisch verschiedene Varianten von Anfragesequenzen generiert und für diese und die zugehörigen transformierten Sequenzen die Ausführungszeiten erfasst. Die Messergebnisse sollen darüber Aufschluss geben, in welchen Fällen eine Verbesserung bzw. Verschlechterung eintritt und welche Faktoren hierfür relevant sind.

© IPVR, Universität Stuttgart, 2004

1.1 Anfrage

Eine Anfrage ist eine Folge von SQL-Anweisungen, die eine Tabelle oder Sicht erzeugen. Diese Tabelle oder Sicht wird mit Tupeln an Informationen gefüllt.
Die Auswahl der Informationen findet über die SQL-Anweisung statt.

Im Rahmen dieser Studienarbeit wird nur eine Auswahl von SQL-Anweisungen aus der Arbeit „Rewrite-Strategien für generierte Anfragesequenzen im Online Analytical Processing“ [1] betrachtet. Hierbei handelt es sich um die Regeln:

- SELECT ... FROM F…(Merge Select)
- SELECT … FROM F… WHERE … (Merge Where)
- SELECT … FROM F…. GROUP BY ... HAVING ... (Merge Having)
- SELECT … FROM F... WHERE ... GROUP BY .... (Where to Group)
- SELECT … FROM F,G… WHERE (Join to Table)

1.2 Anfragesequenz

Eine Anfragesequenz ist die Folge von Anfragen, welche in Abhängigkeit stehen.
Eine Anfrage A erzeugt eine Tabelle und füllt diese mit Informationen. Wenn eine Anfrage B aus dieser erzeugten Tabelle wieder Informationen gewinnt, dann ist B von A abhängig.
Eine Anfrage kann auch in mehreren Abhängigkeiten stehen.
Möglich ist auch, dass mehr als eine Anfrage abhängig von einer Anderen ist.

© IPVR, Universität Stuttgart, 2004

1.3 Rewrite-Regeln

Im folgenden Abschnitt werden die betrachteten Rewrite-Regeln aus [1] kurz erläutert.

Aus mehreren Anfragen, welche sich nur geringfügig unterscheiden, werden die gemeinsamen Anteile gewonnen. Aus diesen Anteilen wird eine neue Anfrage gebildet.
Das Ergebnis der neuen Anfrage enthält mindestens die Informationen, welche in allen Ergebnissen der gegebenen Anfragen vorkommt. Das heißt die neue Anfrage liefert als Ergebnis mindestens die kleinste gemeinsame Ergebnismenge.

Die gegebenen Anfragen können umformuliert werden und unter zur Hilfenahme dieser Ergebnismenge, ihr gesamtes Ergebnis errechnen.
Die kleinste gemeinsame Ergebnismenge stellt ein Zwischenergebnis dar, welches nicht bei allen Anfragen wieder berechnet werden muss. Dies kann zu Zeiteinsparungen bei der Berechnung der Ergebnismengen führen.

1.3.1 Merge Select

Bei Merge Select werden gleiche Teile mehrerer Anfragen der Form „SELECT x FROM y“, wobei x für die Attribute und y für Tabellen oder Sichten steht, zusammen gefasst und in einer Tabelle gespeichert.
Aus dieser Tabelle können die Informationen für die eigentlichen Anfragen wieder ausgelesen werden. Dies kann zu verbesserten Anfragezeiten führen, da nicht alle Informationen neu gewonnen und berechnet werden müssen.

Vorraussetzung für diese Möglichkeit ist, dass alle Anfragen die folgenden Bedingungen erfüllen:

- Alle Anfragen stimmen bis auf die SELECT-Klausel überein.
- Keine der Anfragen enthält ein DISTINCT.

Aus den gegebenen Anfragen wird die Anfrage für die gemeinsame Ergebnismenge folgendermaßen erzeugt:

1. Alle Klauseln, bis auf die SELECT Klausel, werden aus den Anfragen übernommen.
2. Aus den SELECT Klauseln alle Attribute und ihre Definitionen übernehmen und Duplikate entfernen.
3. Bezeichnerkollisionen in den vereinigten Attributdefinitionen durch Umbenennung beheben.

Aus der neuen Tabelle können die Informationen der eigentlichen Anfragen folgendermaßen wieder gewonnen werden:

- „SELECT x FROM Y“ gewinnt die Informationen aus der neuen Tabelle wieder, wobei x für die zu gewinnenden Attribute steht. Hierbei ist auf etwaige Umbenennungen durch Bezeichnerkollisionen zu achten. Y steht für die neue Tabelle der gemeinsamen Anfrage.

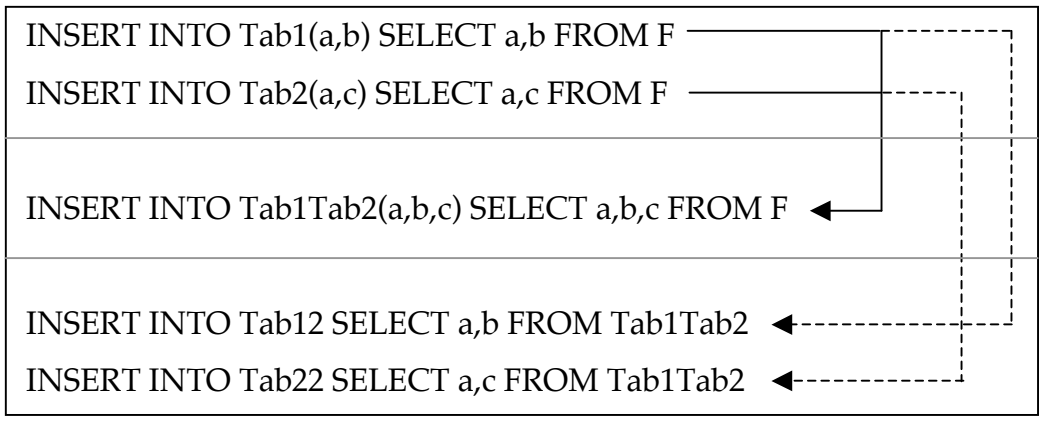

Abb. 1: Beispiel zu Merge Select

1.3.2 Merge Where

Bei Merge Where werden gleiche Teile mehrerer Anfragen der Form „SELECT x FROM y WHERE z“, wobei x für die Attribute, y für Tabellen oder Sichten und z für Vergleichsoperationen steht, zusammen gefasst. Das Ergebnis wird in eine Tabelle gespeichert.
Aus dieser Tabelle können die Informationen für die eigentlichen Anfragen wieder ausgelesen werden. Dies kann zu verbesserten Anfragezeiten führen, da nicht alle Informationen neu gewonnen und berechnet werden müssen. Das bedeutet, dass in der neuen Tabelle nur Informationen gespeichert sind, welche die Bedingungen z erfüllen.
Im schlechtesten Fall repräsentiert die neue Tabelle die gesamte Basistabelle, wenn die Bedingungen z alle Informationen aus der Basistabelle abdecken.

Vorraussetzung für die Möglichkeit der Zusammenfassung mehrerer Anfragen ist, dass alle die folgende Bedingungen erfüllen:

- Alle Anfragen stimmen bis auf die SELECT und WHERE Klausel überein.
- Keine der Anfragen enthält eine DISTINCT Klausel.

Aus den gegebenen Anfragen wird die Anfrage für die gemeinsame Ergebnismenge folgendermaßen erzeugt:

1. Alle Klauseln bis auf die SELECT und die WHERE Klauseln werden übernommen.
2. Aus den SELECT Klauseln werden alle Attribute und ihre Definitionen übernommen und Duplikate entfernt.
3. Die Attribute der WHERE Klauseln werden übernommen und mit OR, nach der Entfernung von Duplikaten, verknüpft.
4. Alle in den WHERE Klauseln vorkommenden Attribute werden in die vereinigte SELECT Klausel übernommen, sofern es sich nicht um Duplikate handelt.

5. Bezeichnerkollisionen in der vereinigten SELECT Klausel werden durch umbenennen behoben.

Aus der neuen Tabelle können die Informationen der eigentlichen Anfragen folgendermaßen wieder gewonnen werden:

- „SELECT x FROM Y WHERE z“ gewinnt die Informationen aus der neuen Tabelle wieder. Wobei x für die zu gewinnenden Attribute steht. Hierbei ist auf etwaige Umbenennungen durch Bezeichnerkollisionen zu achten. Y steht für die neue Tabelle der gemeinsamen Anfrage und z für die Bedingungen der WHERE Klausel. Auch bei den Bedingungen der WHERE Klausel ist auf Umbenennungen durch Bezeichnerkollisionen zu achten.

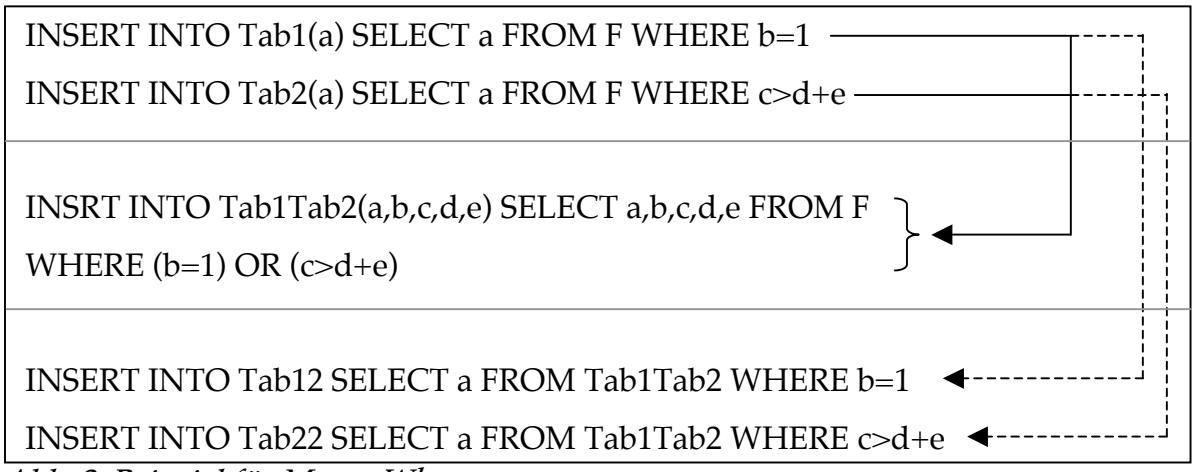

Abb. 2: Beispiel für Merge Where

1.3.3 Merge Having

Bei Merge Having werden gleiche Teile mehrere Anfragen der Form „SELECT w FROM x GROUP BY y HAVING z“, wobei w für die Attribute, x für die Tabellen oder Sichten, y für die Attribute der GROUP BY Klausel und z für die Attribute der HAVING Klausel steht, zusammen gefasst. Das Ergebnis wird in einer Tabelle gespeichert.
Aus dieser Tabelle können die Informationen für die eigentlichen Anfragen wieder ausgelesen werde. Dies kann zu verbesserten Anfragezeiten führen, da nicht alle Informationen neu gewonnen werden müssen. Das heißt, es müssen nicht alle Informationen der Basistabelle darauf überprüft werden, ob sie den Bedingungen der HAVING Klausel gerecht werden. In der neuen Tabelle sind nur Informationen gespeichert, welche mindestens einer Bedingung der HAVING Klauseln entsprechen.
Im schlechtesten Fall sind wieder alle Informationen der Basistabelle in der neuen Tabelle enthalten, wenn die Bedingungen der HAVING Klauseln die gesamte Basistabelle abdecken.

Vorraussetzung für die Möglichkeit mehrere Anfragen zusammen zu fassen ist, dass sie folgende Bedingung erfüllen:

- Alle Anfragen unterscheiden sich nur in den SELECT und HAVING Klauseln

Aus den gegebenen Anfragen wird die gemeinsame Ergebnisanfrage folgendermaßen erzeugt:

1. Alle übereinstimmenden Klauseln werden in die neue Anfrage übernommen.
2. Die SELECT Klauseln werden vereinigt und, nach dem Entfernen von Duplikaten, in die neue Anfrage übernommen.
3. HAVING Klauseln der Anfragen werden mit OR verknüpft, Duplikate entfernt, und in die neue Anfrage übernommen.

4. Alle Attribute und Aggregate der HAVING Klauseln werden der vereinigten SELECT Klausel hinzugefügt, sofern es sich nicht um Duplikate handelt.
5. Bezeichnerkollisionen der Attributdefinitionen werden durch umbenennen behoben.

Aus der neuen Tabelle können die Informationen der eigentlichen Anfragen folgendermaßen wieder gewonnen werden:

- „SELECT x FROM Y GROUP WHERE z“ gewinnt die Informationen aus der neuen Tabelle wieder, wobei x für die zu gewinnenden Attribute steht, Y steht für die neue Tabelle der gemeinsamen Anfrage. Für die Bedingungen der HAVING Klausel steht z. Bei den Attributen aus x und z sind Umbenennungen durch Bezeichnerkollisionen zu beachten.

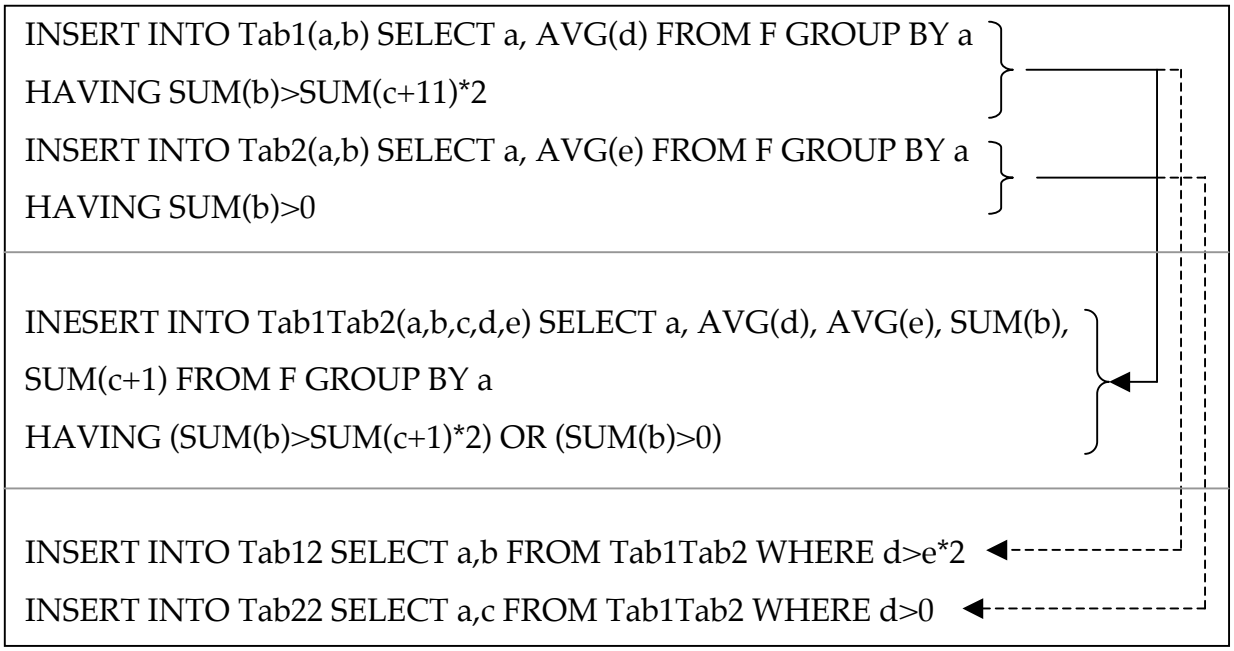

Abb. 3: Beispiel für Merge Having

1.3.4 Where To Group

Bei Where To Group werden gleiche Teile mehrerer Anfragen der Form „SELECT w FROM x WHERE y GROUP BY z“, wobei w für die Attribute, x für die Tabellen oder Sichten, y für die Bedingungen der WHERE Klausel und z für die Attribute der GROUP BY Klausel steht, zusammen gefasst. Das Ergebnis wird in einer Tabelle gespeichert.
Aus dieser Tabelle können die Informationen für die eigentlichen Anfragen wieder ausgelesen werden. Dies kann zu verbesserten Anfragezeiten führen, da nicht alle Informationen neu gewonnen werden müssen. Das heißt, es müssen nicht alle Informationen der Basistabelle darauf überprüft werden, ob sie den Bedingungen der WHERE Klausel gerecht werden. In der neuen Tabelle sind nur Informationen gespeichert, welche mindestens einer Bedingung der WHERE Klauseln entsprechen. Im schlechtesten Fall sind wieder alle Informationen der Basistabelle in der neuen Tabelle enthalten, wenn die Bedingungen der WHERE Klauseln die gesamte Basistabelle abdecken.

Vorraussetzung für die Möglichkeit mehrere Anfragen zusammen zu fassen ist, dass sie folgende Bedingung erfüllen:

- Alle Anfragen unterscheiden sich nur in den SELECT, WHERE und GROUP BY Klauseln.
- Die WHERE Klauseln stimmen jeweils bis auf ein Prädikat pro Anfrage überein.
- Das Prädikat in dem sich die Anfragen unterscheiden hat entweder die Form a=k oder k=a oder a IN K, wobei a ein Attribut, k eine Konstante und K eine Menge von Konstanten ist.
- Die GROUP BY Klauseln stimmen bei allen Anfrage der Menge, die sich in einem Prädikat der Form a=k oder k=a unterscheiden, überein.

- Die GROUP BY Klauseln stimmen bei allen Anfragen der Menge, die sich in einem Prädikat der Form a IN K unterscheiden, überein. Zusätzlich zu den GROUP BY Klauselelementen der Anfrage, die sich in einem Prädikat der Form a=k oder k=a unterscheiden, enthält sie noch das Attribut a dieses Prädikats.

Aus den gegebenen Anfragen wird die gemeinsame Ergebnisanfrage folgendermaßen erzeugt:

1. Alle übereinstimmenden Klauseln werden in die neue Anfrage übernommen
2. Die SELECT Klauseln werden vereinigt und, nach dem Entfernen von Duplikaten, in die neue Anfrage übernommen. Bezeichnerkollisionen werden durch umbenennen behoben.
3. WHERE Klauseln der Anfragen werden mit OR verknüpft, Duplikate entfernt, und in die neue Anfrage übernommen.
4. Alle übereinstimmenden Elemente der GROUP BY Klauseln der vereinigten SELECT Klausel werden hinzugefügt, sofern es sich nicht um Duplikate handelt.

Aus der neuen Tabelle können die Informationen der eigentlichen Anfragen folgendermaßen wieder gewonnen werden:

- „SELECT w FROM X WHERE y GROUP BY z“ gewinnt die Informationen aus der neuen Tabelle wieder. Wobei w für die zu gewinnenden Attribute steht. X steht für die neue Tabelle der gemeinsamen Anfrage. Es steht y für die WHERE Bedingungen und z für GROUP BY Attribute der eigentlichen Anfrage. Bei w,y und z ist auf Umbenennungen durch Bezeichnerkollisionen zu achten.

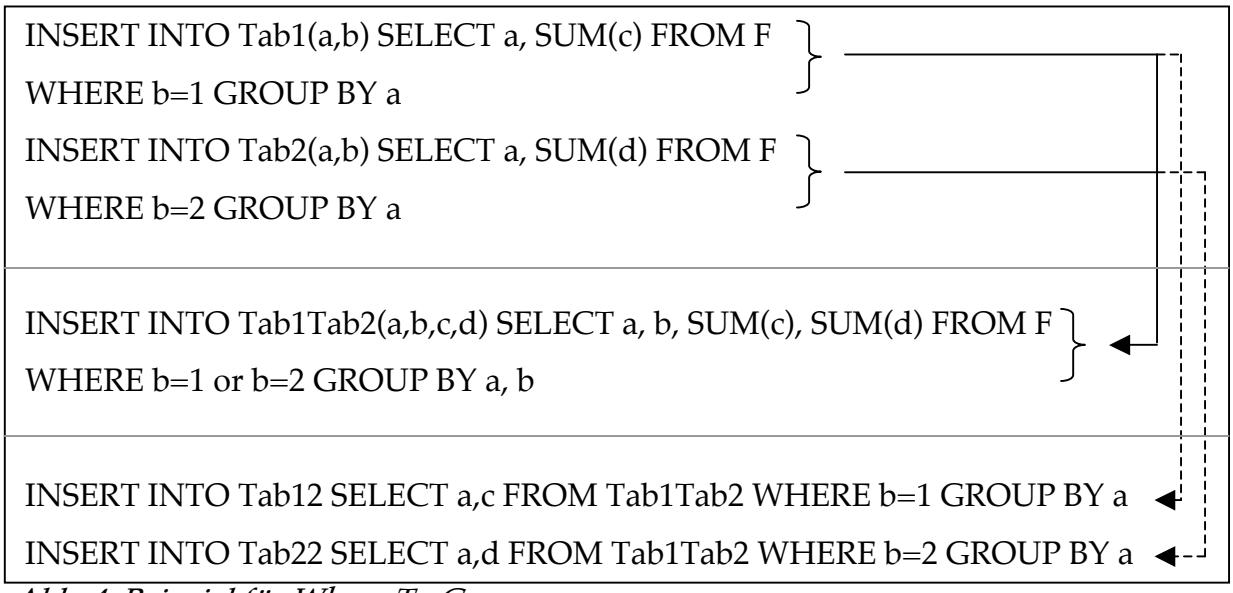

Abb. 4: Beispiel für Where To Group

1.3.5 Join To Table

Bei Join To Table werden gleiche Teile mehrerer Anfragen der Form „SELECT x FROM y WHERE z“, wobei x für die Attribute, y für die Tabellen oder Sichten und z für die Bedingungen der WHERE Klausel steht, zusammen gefasst. Das Ergebnis wird in einer Tabelle gespeichert.
Aus dieser Tabelle können die Informationen für die eigentlichen Anfragen wieder ausgelesen werde. Dies kann zu verbesserten Anfragezeiten führen, da nicht alle Informationen neu gewonnen werden müssen. Das heißt es müssen nicht alle Informationen der Basistabelle darauf überprüft werden, ob sie den Bedingungen der WHERE Klausel gerecht werden. In der neuen Tabelle sind nur Informationen gespeichert, welche mindestens einer Bedingung der WHERE Klauseln entsprechen. Im schlechtesten Fall enthält die neue Tabelle alle Informationen, welche durch Join’s der Tabelle entstehen, also das kartesische Produkt aller Basistabellen. Dieser Fall tritt ein, wenn die Bedingungen der WHERE Klauseln alle Informationen in allen Basistabellen abdeckt.

Vorraussetzung für die Möglichkeit mehrere Anfragen zusammen zu fassen ist, das sie folgende Bedingung erfüllen:

- Es existiert eine Menge von Join Prädikaten, in denen alle Anfragen der Menge übereinstimmen.
- In den FROM Klauseln existieren mehrere Elemente, in denen die Anfragen übereinstimmen und die durch direkte oder transitive Vergleiche der Primärschlüssel miteinander verbunden sind.

Aus den gegebenen Anfragen wir die gemeinsame Ergebnisanfrage folgendermaßen erzeugt:

1. Tabellen oder Sichten welche in den FROM Klauseln übereinstimmen, werden in die neue Anfrage übernommen.
2. Join Prädikate der WHERE Klauseln, welche übereinstimmen, werden in die neue WHERE Klausel übernommen.
3. Join Prädikate der WHERE Klauseln, welche durch Transitivitäten in den Klauseln, übereinstimmen, werden in die neue WHERE Klausel übernommen.
4. Alle Attribute aus den WHERE Klauseln, welche sich auf die gemeinsamen Tabellen oder Sichten beziehen, werden der neuen SELECT Klausel hinzugefügt.
5. Duplikate in der neuen SELECT Klausel werden entfernt.

Aus der neuen Tabelle können die Informationen der eigentlichen Anfragen folgendermaßen wieder gewonnen werden:

- „SELECT x FROM Y WHERE z“ gewinnt die Informationen aus der neuen Tabelle wieder. Wobei x für die zu gewinnenden Attribute steht. Hierbei ist auf etwaige Umbenennungen durch Bezeichnerkollisionen zu achten. Y steht für die neue Tabelle der gemeinsamen Anfrage und Tabellen oder Sichten, welche nicht in der neuen gemeinsamen Anfrage enthalten sind. Für die Bedingungen der eigentlichen Anfrage, welche nicht in der gemeinsamen Anfrage vertreten sind, steht z.

Bei z ist auf Umbenennungen durch Bezeichnerkollisionen oder Tabellennamensänderungen, durch die neue gemeinsame Tabelle, zu achten.

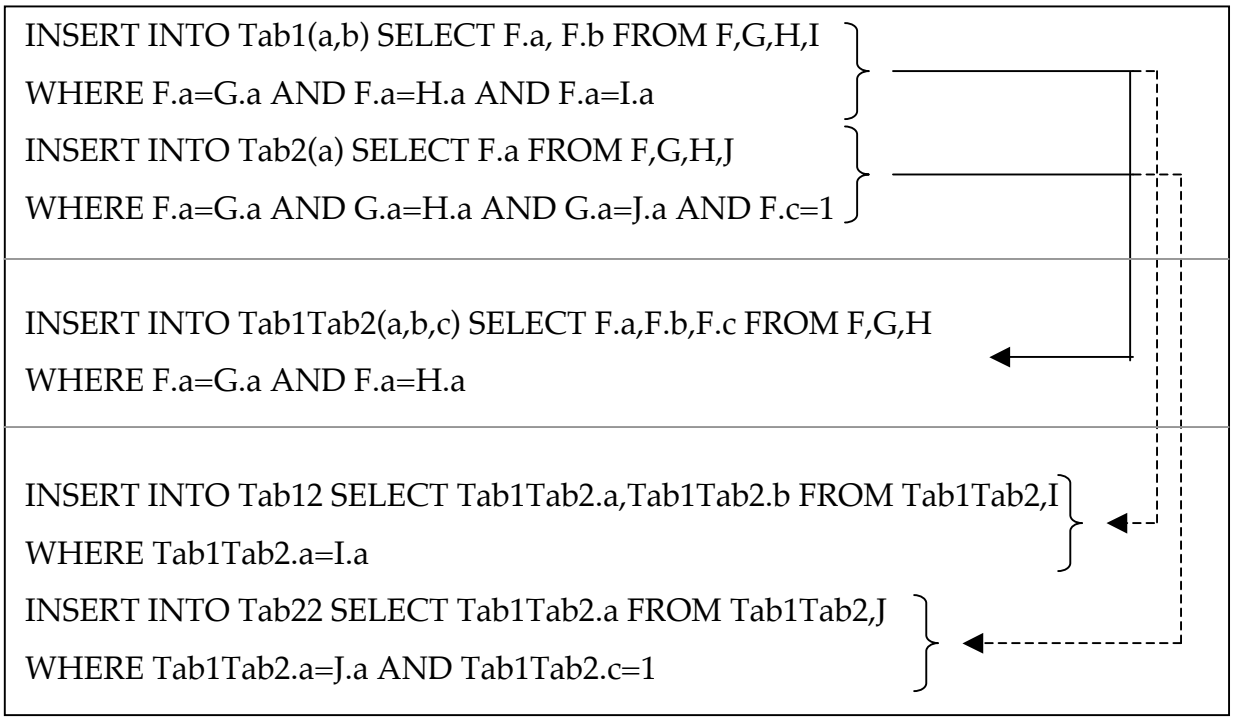

Abb. 5: Beispiel für Join To Table

2. Software

Im Rahmen dieser Studienarbeit sollte eine Testumgebung für Rewrite-Regeln erstellt werden. Diese Software wurde in Java implementiert. Als Datenbanksystem zum Ausführen der Anfragen wird IBM DB2 verwendet.

Die Software ist in der Lage aufgrund von Parametern zwei zu testende Anfragen zu erzeugen. Diese Anfragen stellen nur einen Vorschlag dar. Diese Anfragen können manuell verändert werden.

Es besteht auch die Möglichkeit Anfragen manuell, ohne Verwendung der automatischen Generierung, zu erzeugen. Hierfür muss jede Anfrage separat in eines der vorhanden Textfelder eingegeben werden.
Für die manuelle Eingabe sind folgende Konventionen zu beachten:

- Die Basistabelle trägt immer den Namen „F“.
- Die Tabelle für die erste Anfrage trägt den Namen „Tab1“.
- Die Tabelle für die zweite Anfrage trägt den Namen „Tab2“.
- Attribute aller Tabellen sind mit „attrib#“ zu bezeichnen, wobei „#“ für eine Zahl >=1 steht.
- Alle Anfragen sind in der Form „INSERT INTO X(y) Z“, wobei X für den Tabellenname, y für die Attribute der Tabelle und Z für eine entsprechende SELECT Anfrage steht.
  Die Attribute y müssen immer angegeben werden, da sie für die automatische Tabellengenerierung benötigt werden.

Diese Konventionen werden von der Software nicht explizit geprüft. Das bedeutet es ist jegliche Eingabe möglich. Vor dem Starten des Test werden aber alle Anfragen auf der Datenbank getestet. Sollten bei der Eingabe Fehler unterlaufen sein, so werden diese Unstimmigkeiten von der Datenbank gemeldet und im Protokollfenster angezeigt.

Nach dem Starten eines Tests werden folgende Schritte von der Software ausgeführt:

1. Überprüfen der gegebenen Anfragen auf Ausführbarkeit, in Anlehnung an [3], durch Senden der Anfragen an die Datenbank.
   Sollte hierbei ein Fehler auftreten, wird dieser im Protokollfenster beschrieben und der Testlauf bricht automatisch ab.
2. Erzeugen der Basistabelle „F“.
   Aufgrund der eingestellten Parameter wird die Basistabelle erzeugt und mit zufälligen Integer Werten gefüllt.
3. Etwaig geforderte Indizes werden auf der Basistabelle „F“ erzeugt.
4. Die erste gegebene Anfrage, „INSERT INTO Tab1 ... SELECT ...“, wird auf der Datenbank ausgeführt und die Laufzeit gemessen, welche im Protokollfenster nach Beendigung der Anfrage ausgegeben wird.
5. Die zweite gegebene Anfrage, „INSERT INTO Tab2 ... SELECT ...“, wird auf der Datenbank ausgeführt und die Laufzeit gemessen, welche im Protokollfenster nach Beendigung der Anfrage ausgegeben wird.
6. Es wird „INSERT INTO Tab12 SELECT * FROM Tab1“ auf der Datenbank ausgeführt um das Gewinnen der Informationen aus der Tabelle der ersten Anfrage zu simulieren. Die Verarbeitungszeit der Anfrage wird ebenfalls gemessen und protokolliert.
7. Es wird „INSERT INTO Tab22 SELECT * FROM Tab2“ auf der Datenbank ausgeführt um das Gewinnen der Informationen aus der Tabelle der zweiten Anfrage zu simulieren. Die Verarbeitungszeit der Anfrage wird ebenfalls gemessen und protokolliert.
8. Die gemeinsame Anfrage, „INSERT INTO Tab1Tab2 ... SELECT ...“, wird nach den Rewrite-Regeln erzeugt und auf der Datenbank zur Ausführung gebracht. Die neue Tabelle trägt den Namen Tab1Tab2. Es erfolgt ebenfalls die Zeitname und ihre Protokollierung.
9. Etwaig geforderte Indizes werden auf der neuen Tabelle „Tab1Tab2“ erzeugt.
10. Auf Basis der Rewrite-Regeln wird „INSERT INTO Tab1Tab2Tab1 SELECT X FROM Tab1Tab2 Y“ für die Gewinnung der Informationen der erste Anfrage, wobei X die benötigten Attribute und Y für weitere benötigte SQL-Phrasen

steht, zur Ausführung gebracht. Auch hier erfolgt eine protokollierte Zeitnahme.

11. Auf Basis der Rewrite-Regeln wird „INSERT INTO Tab1Tab2Tab2 SELECT X FROM Tab1Tab2 Y“ für die Gewinnung der Informationen der zweiten Anfrage, wobei X die benötigten Attribute und Y für weitere benötigte SQL-Phrasen steht, zur Ausführung gebracht. Es erfolgt auch hier eine protokollierte Zeitnahme.
12. Es werden alle erzeugten Tabellen und Indizes gelöscht.

Die Zeitnahme der einzelnen Anfragen startet nach dem Erzeugen der zugehörigen Tabelle und endet nach der Rückmeldung der Datenbank. Beim Auswerten der Zeitnahmen ist darauf zu achten, dass, auch wenn ein Fehler, welcher im Protokoll zu erkennen ist, während der Ausführung einer Anfrage auftritt, eine Zeitnahme stattfindet und diese protokolliert wird.

Im Folgenden wird genauer auf die Anforderungen der Software und ihre Implementierung eingegangen. Es werden auch ihre Bedingungen an das Softwareumfeld und Randbedingungen ihrer Funktionalität erläutert.

2.1 Anforderungen an das Softwareumfeld

Anforderungen:

- Java Runtime Enviroment 1.4.1 oder höher
- IBM DB2 Datenbank
- CLASSPATH auf db2java.zip
- Stored Procedure

Die Software wurde in Java 1.4.1 implementiert, deshalb muss für ihre Funktionstüchtigkeit auf dem ausführenden System mindestens das Java Runtime Enviroment 1.4.1 installiert sein.

Die Software arbeitet auf einer IBM DB2 Datenbank, welche auf dem System selbst verfügbar, oder durch das ausführende System, erreichbar sein muss.
Auf der zu benutzenden Datenbank muss eine Stored Procedure zum Erzeugen von Indizes zur Verfügung stehen.
Zum Installieren dieser Stored Procedure werden folgende Dateien, welche sich auf der CD im Anhang befinden, benötigt:

- compile
- sp_runstats.db2
- sp_runstats.sqc

Innerhalb dieser Dateien müssen noch der Datenbankname und nötigenfalls Pfade angepasst werden.

Die Testumgebung steht als JAR-File zur Verfügung. Dieses JAR beinhaltet alle zur Software gehörigen Klassen. Als zusätzliche Klasse wird „COM.ibm.db2.jdbc.app.DB2Driver“ benötigt, welche in dem Paket db2java.zip enthalten ist. Dieses Paket gehört zu IBM's DB2 Datenbank und wird durch das JAR-File über den Pfad „/opt/IBMdb2/V7.1/java/db2java.zip“ referenziert.
Sollte dieser Pfad nicht mit dem des Systems, auf welchem die Datenbank installiert ist, übereinstimmen, so muss der CLASSPATH manuell vor dem Start der Testumgebung gesetzt werden.

2.2 Interner Aufbau der Software

Die Testumgebung wurde in Klassen untergliedert. Diese Klassen können im Rahmen von Erweiterungen, Verbesserungen und Fehlerbehebungen einfach ersetzt oder verändert werden. Abb. 6 auf Seite 19 zeigt eine Übersicht der Zusammenarbeit der Klassen.

Im Folgenden werden die verwendeten Klassen aufgezählt und ihre Aufgaben kurz erläutert.

Klassen welche für die GUI verantwortlich sind:

- JDBench
  Implementiert die GUI und ist für die zentrale Steuerung aller anderen Klassen zuständig.
- indexWindow
  Implementierung der GUI für das Fenster zur Indizeauswahl.
- logWindow
  Implementierung der GUI für das Protokollfenster.
- msgBox
  Implementiert die GUI zum anzeigen des Hinweisfensters.

Klassen welche die Rewrite-Regeln implementieren und für den Testablauf zuständig sind:

- joinToTable
  Implementierung des Testablaufes für die Rewrite-Regel Join To Table.
- mergeHaving
  Implementierung des Testablaufes für die Rewrite-Regel Merge Having.
- mergeSelect
  Implementierung des Testablaufes für die Rewrite-Regel Merge Select.
- mergeWhere
  Implementierung des Testablaufes für die Rewrite-Regel Merge Where.
- whereToGroup
  Implementierung des Testablaufes für die Rewrite-Regel Where To Group.

Für die Datenbankinteraktion benötigte Klassen:

- createIndex
  Implementiert die Erzeugung von Indizes.

- createTable
  Implementiert das Erstellen und Füllen der Tabellen. Es wird ein Füllen der Tabellen durch INSERT Statements so wie das Füllen der Tabellen durch rekursives SQL unterstütz.
- dbConnect
  Hier findet der Verbindungsaufbau zur Datenbank statt.
- destroyTable
  Es wird ein DROP TABLE Statement an die Datenbank gesendet.

Klassen welche intern zum Realisieren der Rewrite-Regeln benutzt werden:

- findSameAttrib
  Ein übergebener Vector von Strings wird auf Duplikate überprüft. Nach dem Entfernen der Duplikate wird der Vector von Strings bereinigt zurückgegeben.
- makeTransitiv
  Die WHERE Klausel eines Join To Table Tests wird auf Transitivitäten überprüft und die daraus resultierenden Beziehungen werden zur weiteren Verarbeitung hinzugefügt.
  Bsp.: WHERE F.a=G.a AND F.a=H.a ➔ WHERE F.a=G.a AND F.a=H.a AND G.a=H.a
- sqlParse
  Parsed ein SQL-Statement und gibt es in einem Vector der Form:
  Tabellenname (String), Namen der Spalten der Tabelle (Vector[String]), Auftreten von DISTINCT (Boolean), Attribute der SELECT Klausel (Vector[String]), Tabellen der FROM Klausel (Vector[String]), WHERE Klausel (Vector[Klausel (String), Attribute der Klausel (Vector[String])]), GROUP BY Klausel (Vector[Attribute der Klausel (Vector[String]), HAVING Klausel (String), Attribute der HAVING Klausel (Vector[String])])
  zurück.

- stringReplace
  Implementierung einer Funktion zum Ersetzen eines Teilstrings in einem String.
  Bsp.: stringReplace(„ATTRIB“, „attrib1“, „WHERE attrib1=2 AND attrib1=3“) wird folgendermaßen verarbeitet:
  WHERE attrib1=2 AND attrib1=3 ➔ WHERE ATTRIB =2 AND ATTRIB=3

Abbildung 6 zeigt eine Übersicht der Zusammenarbeit der Klassen.

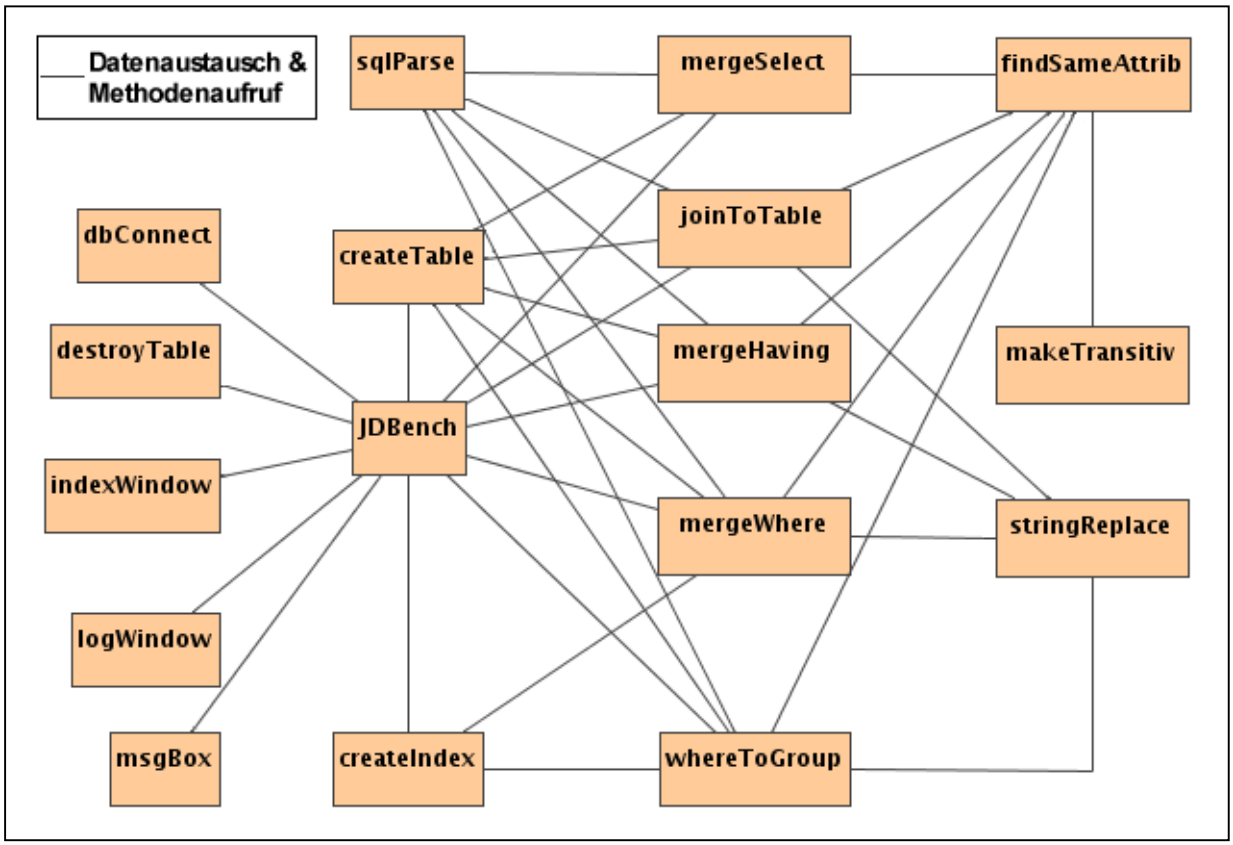

Abb. 6: Zusammenarbeit der Klassen der Software

2.3 Konfiguration der Testumgebung

Vor der Ausführung der ersten Testreihen müssen die Informationen für die Verbindung zur Datenbank eingegeben werden. Diese Informationen werden nach dem Beenden der Software aus Sicherheitsgründen nicht gespeichert.

Unter dem Karteikartenreiter KONFIGURATION müssen Datenbankname, Username und Password eingeben werden.

2.4 Bedienung der Software und ihre Randbedingungen

In diesem Kapitel wird die Bedienung der Testumgebung erläutert. Des weiteren wird auf etwaige Randbedingungen der verschiedenen Tests eingegangen.

2.4.1 Basistabellen erzeugen

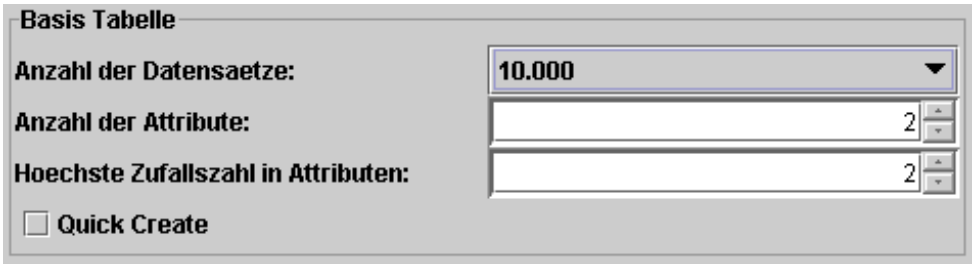

Abb. 7: Basistabellen erzeugen

Die Testumgebung übernimmt bei den Testläufen die automatische Generierung der Basistabellen, auf welchen die Anfragen ausgeführt werden. Die Parameter zum Füllen der Basistabellen mit Informationen können auf jeder Seite für das jeweilige Testszenario eingestellt werden.

Hierbei kann die Anzahl der Datensätze, also der Tuppel in der Tabelle, gewählt werden. Der Wertebereich erstreckt sich über 10k/100k/1M und 10M Tuppel.

Die Anzahl der Attribute in der Basistabelle kann frei gewählt werden. Die Software berücksichtigt automatisch Randbedingungen, welche durch die automatisch generierten Anfragen gestellt werden. Das bedeutet, wenn eine Anfrage auf 5

Attribute der Basistabelle zugreift, dann kann für die Anzahl der Attribute keine Zahl kleiner 5 gewählt werden.

Alle Attribute der Basistabellen werden automatisch mit Integer Zufallszahlen gefüllt. Der Wertebereich der Zufallszahlen kann ebenfalls eingestellt werden.

Beim Füllen der Tabellen wird bei Merge Where, Merge Having und Where To Group auf spezielle Wertebereiche der Attribute attrib1 und attrib2 in der Basistabelle geachtet.

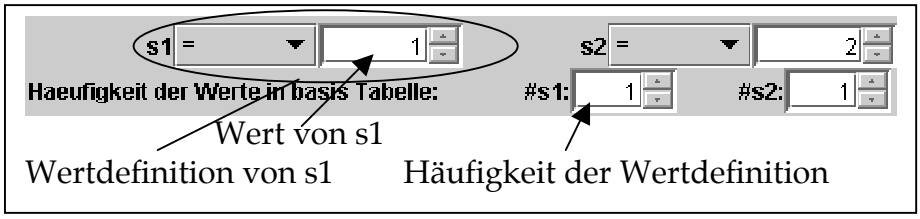

Abb. 8: Spezielle Wertebereiche

Die Darstellung in Abb. 8 zeigt die Einstellmöglichkeiten für die speziellen Wertebereiche.
S1 und s2 stehen hierbei für Werte zum Beispiel in WHERE Klauseln. Diese Klauseln haben bei der automatischen Erzeugung der Anfragen dann zum Beispiel die Form „WHERE attrib1=1“.
Des weiteren kann gewählt werden, wie häufig der Wert von s1 oder s2 in der Basistabelle auftritt. Beim Erzeugung der Basistabellen wird die jeweilig Wertedefinition für die INSERT Statement verwendet. Bezogen auf Abb.8 bedeutet dies, dass s1 den Wert 1 hat, also eine Klausel der Form „attrib1=1“ und eine Klausel der Form „attrib2=2“ erzeugt wird. Beim füllen der Basistabelle wird genau in einem Tuppel attrib1 die Wertdefinition „s1=1“ erfüllen.
Es wird auch genau ein Tuppel geben, in welchem attrib2=2 ist. Dieses kann, durch das zufällige Füllen der Tabelle, das selbe Tuppel wie für s1 sein aber auch jedes beliebig andere.

Das Füllen der Tabellen ist über INSERT Statements realisiert. Dieses Füllen kann aufgrund der großen Zahl von Tuppeln sehr viel Zeit in Anspruch nehmen. Aus diesem Grund wurde zusätzlich „Quick Create“ implementiert. „Quick Create“ basiert auf rekursivem SQL, wie in [2] beschrieben, und die Fülloperation wird von der Datenbank übernommen.

```
int AnzahlS1, AnzahlS2
int s1, s2
For (int x=0; x<10000; x++){
        generiere Zufallszahl1
        generiere Zufallszahl2
        If AnzahlS1>0 und Zufallszahl1=s1{
                AnzahlS1=AnzahlS1-1
        } else {
                repeat until Zufallszahl1!=s1 {
                        generiere Zufallszahl1
                }
        }
        If AnzahlS1>0 und Zufallszahl2=s2{
                AnzahlS2=AnzahlS2-1
        } else {
                repeat until Zufallszahl2!=s2 {
                        generiere Zufallszahl2
                }
        }
        If AnzahlS1=10000-x{
                Zufallszahl1=s1
                AnzahlS1=AnzahlS1-1
        }
        If AnzahlS2=10000-x{
        Zufallszahl2=s2
        AnzahlS2=AnzahlS2-1
        }

        „INSERT INTO F VALUES(Zufallszahl1, Zufallszahl)“
}
```

Abb. 9: Beispiel Attributwertgenerierung

Das in Abb.9 dargestellte Füllen der Basistabelle über INSERT Statements wurde als Schleife implementiert welche, begrenzt durch die ausgewählte Anzahl von Datensätzen, pro Schleifendurchlauf ein INSERT Statement an die Datenbank sendet. Dieses Statement enthält für jedes Attribut der Tabelle eine zufällige Integer Zahl, welche sich im Rahmen der Benutzereinstellungen bewegt. Im Beispiel Abb.9 wird die Tabelle F mit 2 Attributen, mit 10000 Tuppeln von Zufallszahlen befüllt. Die Parameter AnzahlS1 und AnzahlS2 repräsentieren die bei Merge Where, Merge Having und Where To Group einzustellende Häufigkeit des Vorkommens einer bestimmten Zufallszahl.
S1 und s2 sind in diesem Zusammenhang die zugehörigen Zahlen.
Soll statt s1 und s2 ein Wertebereich gewählt werden, so müssen die „=“ Zeichen im Beispiel durch entsprechende Operatoren ersetzt werden.
Beim Füllen der Tuppel mit „Quick Create“ wird bei Merge Where, Merge Having und Where To Group **keine** Rücksicht auf die speziellen Wertebereiche genommen. Alle anderen Vorgaben wie Anzahl der Datensätze, Attribute und höchste Zufallszahl werden eingehalten.
Das Beispiel in Abb. 10 zeigt die Realisierung von „Quick Create“, wie in [2] beschrieben, zum Füllen der Tabelle F, welche zwei Attribute beinhaltet. Hierbei fungiert das Attribut attrib1 als Zähler und wird nicht in F übernommen. Attrib2 und attrib3 werden mit Zufallszahlen von 0-100 belegt. Die Tabelle enthält nach der Ausführung 10000 Tuppel.

```
INSERT INTO F
WITH a(attrib1, attrib2, attrib3)
      AS (VALUES(1, int(rand()*100), int(rand()*100))
      UNION ALL
      SELECT attrib1+1, int(rand()*100, int (rand()*100)
      FROM a
      WHERE attrib1<10000)
SELECT attrib2, attrib3
FROM a
```

Abb. 10: „Quick Create“ – rekursives SQL

2.4.2 Indizes

Bei den Testszenarien Merge Where und Where To Group kann eine Indizierung der Attribute der Basistabelle vorgenommen werden. Bei Where To Group können auch die Attribute der neuen gemeinsamen Tabelle Indiziert werden.
Für die Indizierung wird die im Kapitel 2.1 beschrieben Stored Procedure verwendet. Beim Anwählen einer Indizierung, erscheint eine Liste der Attribute, welche indizierbar sind. In dieser Liste ist die Auswahl mehrer Attribute zum Indizieren möglich. Dies kann durch halten der „Strg"-Taste erreicht werden. Bei der Auswahl von mehreren Indizes wird für jedes Attribut ein eigener Index angelegt.

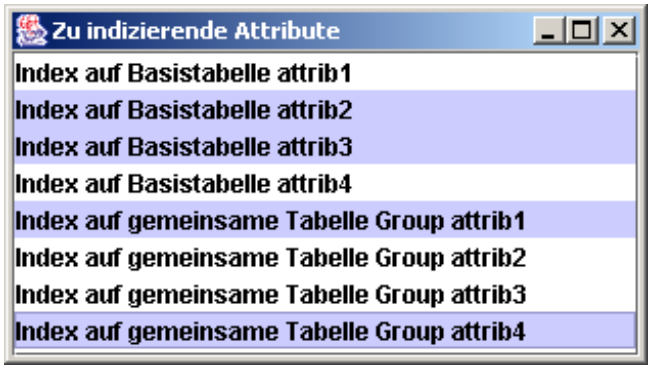

Abb. 11: Auswahl von Indizes

2.4.3 Automatische Anfragengenerierung

Auf den Oberflächen der Testszenarien können automatisch Anfragen generiert werden. Die entsprechenden Parameter müssen im Feld „INSERT Anweisungen" vorgegeben werden.
Variable Parameter stellen hier Anzahl der Attribute, Anzahl der Aggregate und Anzahl der GROUP BY Attribute dar. In den jeweiligen Testszenarien stehen nur relevante Parameter zur Verfügung. Darüber hinaus kann bei manchen Tests noch zwischen besonderen Formen der Anfragen und/oder besonderen Wertebreichen für die Attribute in der Basistabelle gewählt werden.

Zur Übersichtlichkeit der generierten Statements, wurde die Anzahl der GROUP-BY Attribute und Aggregat-Attribute auf jeweils 5 begrenzt. Diese Anzahl stellt jedoch keine Einschränkung dar, da nötigenfalls die generierten Statements manuell erweitert werden können.
Die Software übernimmt automatisch Kontrollfunktionen und korrigiert Eingaben, bedingt durch Abhängigkeiten der Parameter.
Folgende Abhängigkeiten werden entsprechend der Testszenarien beachtet:

- Anzahl der Anfrageattribute <= Anzahl der Basistabellenattribute.
- Anzahl der Anfrageattribute wird entsprechend des Anfragetyps der Anzahl der Aggregatattribute angepasst.
- Anzahl der Anfrageattribute wird entsprechend des Anfragetyps der Anzahl der GROUP BY Attribute angepasst.

Durch wählen des Knopfes „Erzeuge Anweisungen“ werden automatisch zwei, auf den Parametern basierende, Anweisungen erzeugt und dargestellt.
Diese Anweisungen können zusätzlich manuell verändert werden. Vor dem Ausführen der Tests werden diese Anweisungen durch Senden an die Datenbank auf ihre Ausführbarkeit überprüft.

Durch wählen des Knopfs „Hinweise“ wird eine kurze Zusammenfassung der Randbedingungen für die gewählte Rewrite-Regel angezeigt.

2.4.3.1 Merge Select

Auf der zum Testszenario Merge Select gehörenden Oberfläche, können folgende Einstellungen, zur automatischen Generierung der Anfragen, vorgenommen werden:

- Anzahl der Attribute.
- Anzahl der Aggregate.
- Anzahl der GROUP-BY Attribute.

Randbedingungen hierbei sind (Anzahl der Attribute >=2) und ([Anzahl der Aggregate + Anzahl der GROUP-BY Attribute = Anzahl der Attribute] oder ([Anzahl der Aggregate + Anzahl der GROUP-BY Attribute = 0]).

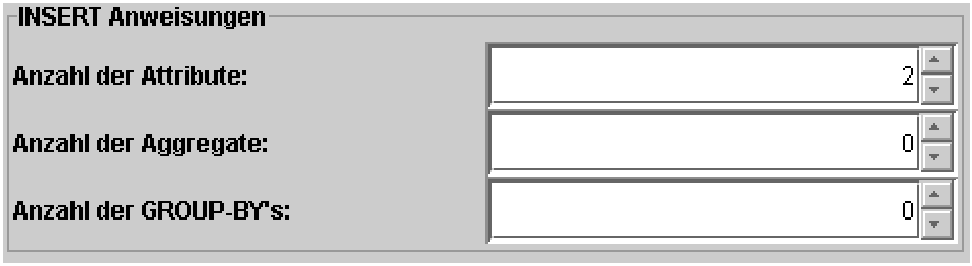

Abb. 12: Merge Select - Parameter

2.4.3.2 Merge Where

Auf der zum Testszenario Merge Where gehörenden Oberfläche, können folgende Einstellungen, zur automatischen Generierung der Anfragen, vorgenommen werden:

- Anzahl der Attribute.
- Anzahl der Aggregate.
- Verhalten der WHERE Klauseln
  - attrib1=s1 & attrib1=s2
  - attrib1=s1 & attrib2=s2
  - attrib1<s1 & attrib1<s2
  - attrib1<s1 & attrib2<s2
- Werte von s1 und s2.
- Randbedingungen zum Füllen der Basistabelle mit s1 und s2
  - Operator der Wertedefinition
  - Häufigkeit der Wertedefinition

  (Soll sich der Wert von s1 vom Wert der Wertedefinition unterscheiden, so muss s1 manuell in der Anfrage verändert werden).

Randbedingungen hierbei sind (Anzahl der Attribute >=1) und ([Anzahl der Aggregate = Anzahl der Attribute] oder ([Anzahl der Aggregate = 0]).

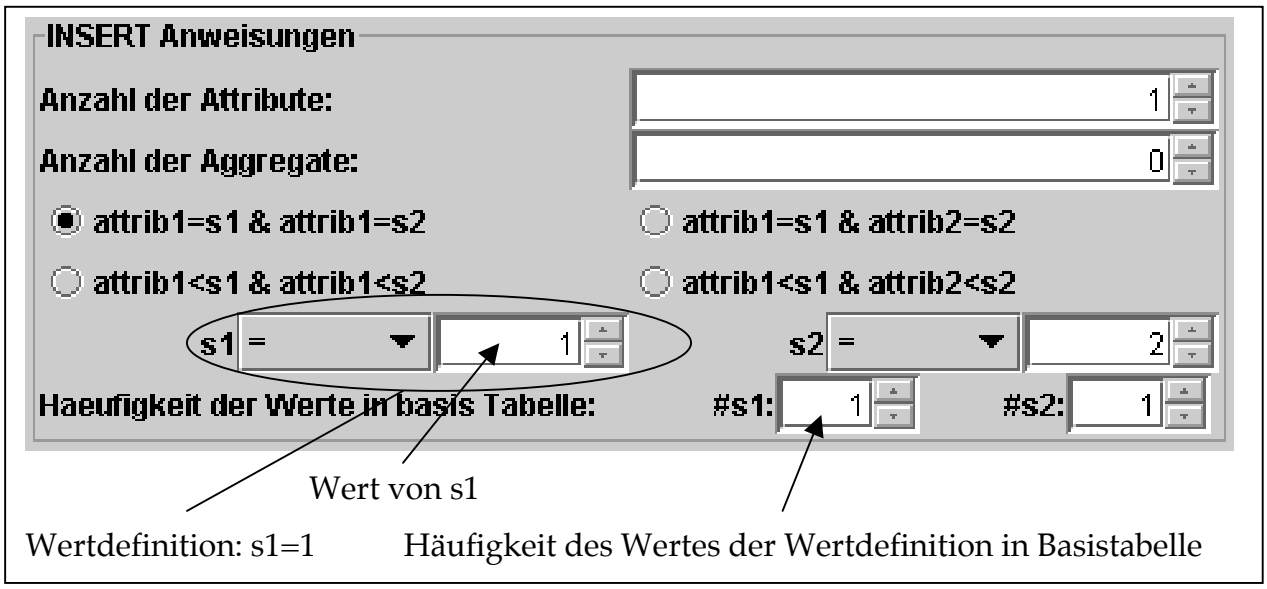

Abb. 13: Merge Where - Parameter

2.4.3.3 Merge Having

Auf der zum Testszenario Merge Having gehörenden Oberfläche, können folgende Einstellungen, zur automatischen Generierung der Anfragen, vorgenommen werden:

- Anzahl der Attribute.
- Anzahl der Aggregate.
- Anzahl der GROUP-BY Attribute.
- Verhalten der HAVING Klauseln
  - attrib1=s1 & attrib1=s2
  - attrib1=s1 & attrib2=s2
- Werte von s1 und s2.
- Randbedingungen zum Füllen der Basistabelle mit s1 und s2
  - Operator der Wertedefinition
  - Häufigkeit der Wertedefinition

  (Soll sich der Wert von s1 vom Wert der Wertedefinition unterscheiden, so muss s1 manuell in der Anfrage verändert werden).

Randbedingungen hierbei sind (Anzahl der Attribute >=1) , (Anzahl der Attribute = Anzahl der Aggregate + Anzahl der GROUP-BY Attribute) und (Anzahl der GROUP-BY Attribute >=1).

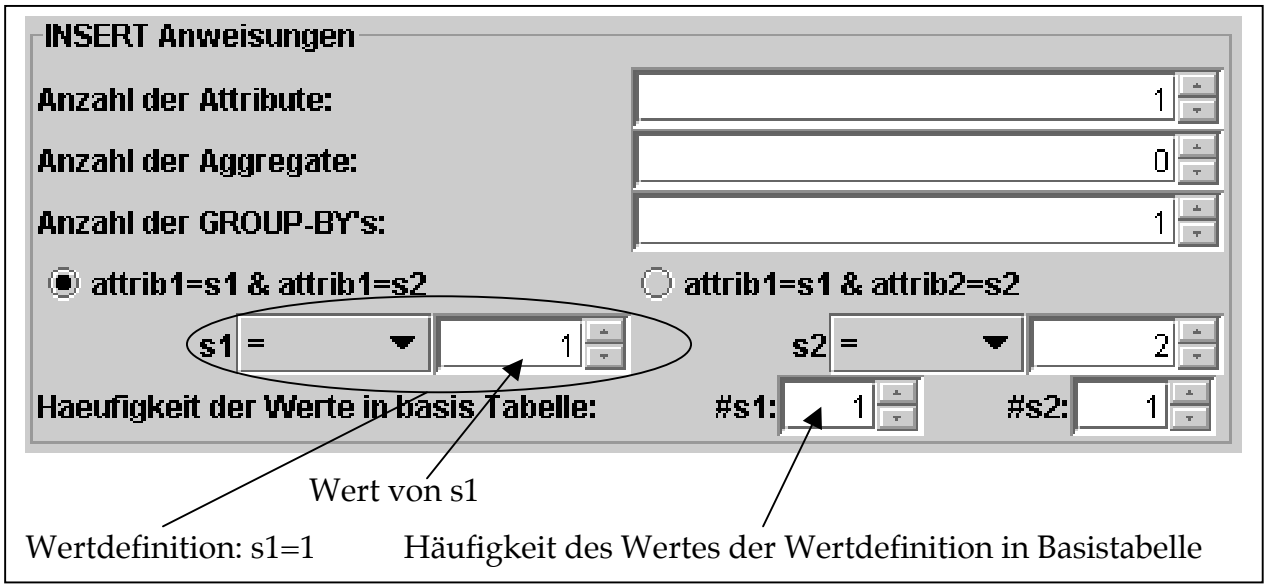

Abb. 14: Merge Having – Parameter

2.4.3.4 Where To Group

Auf der zum Testszenario Where To Group gehörenden Oberfläche, können folgende Einstellungen, zur automatischen Generierung der Anfragen, vorgenommen werden:

- Anzahl der Attribute.
- Anzahl der Aggregate.
- Anzahl der GROUP BY Attribute.
- Verhalten der HAVING Klauseln
  - attrib1=s1 & attrib1=s2
  - attrib1=s1 & attrib2=s2
- Werte von s1 und s2.
- Randbedingungen zum Füllen der Basistabelle mit s1 und s2
  - Operator der Wertedefinition

- Häufigkeit der Wertedefinition
  (Soll sich der Wert von s1 vom Wert der Wertedefinition unterscheiden, so muss s1 manuell in der Anfrage verändert werden).
- Indizierung der GROUP-BY Attribute.

Randbedingungen hierbei sind (Anzahl der Attribute >=2) und (Anzahl der Aggregate + Anzahl der GROUP BY Attribute= Anzahl der Attribute).

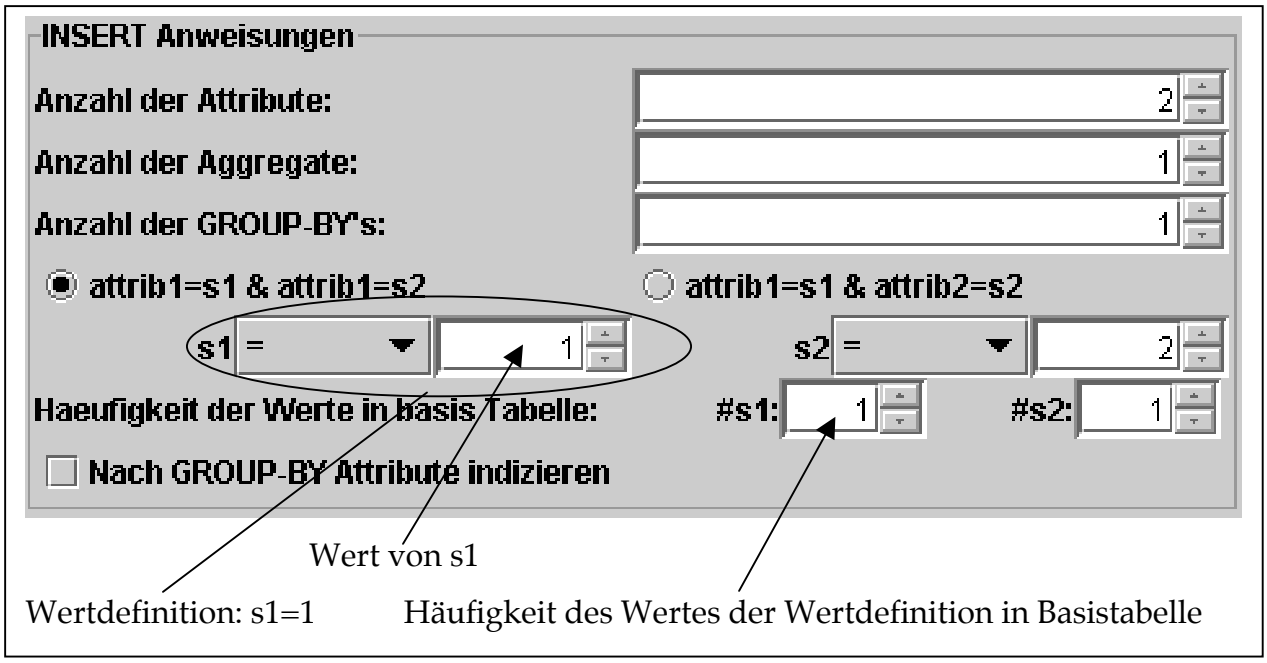

Abb. 15: Where To Group – Parameter

2.4.3.5 Join To Table

Auf der zum Testszenario Join To Table gehörenden Oberfläche können folgende Einstellungen, zur automatischen Generierung der Anfragen, vorgenommen werden:

- Anzahl der Attribute.
- Anzahl der Join's.
- Verhalten der WHERE Klauseln
  - (F.a=G.a) & (F.a=G.a)
  - (F.a=G.a & F.a=H.a) & (F.a=G.a & G.a=H.a)

Randbedingungen hierbei sind (Anzahl der Attribute >=1) und (3 <= Anzahl der Join's <=5).

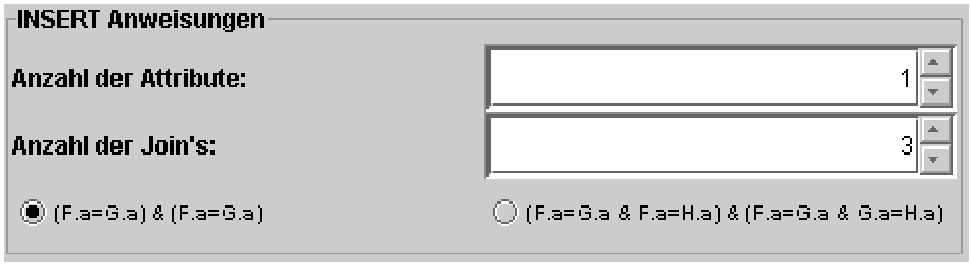

Abb. 16: Join To Table – Parameter

2.4.4 Teststart und Protokollfenster

Nachdem die zu testenden Anfragen generiert und gegebenenfalls manuell abgeändert worden sind, kann durch drücken des „Start" Knopfs der Test begonnen werden.
Am unteren Fensterrand befindet sich eine Statuszeile, welche die momentan von der Software ausgeführte Aktion beschreibt.
Den Verlauf und die Ergebnisse des Test werden in einem separaten Protokollfenster ausgegeben. Hier werden alle wichtigen Schritte der Software erwähnt. Die Ergebnisse werden ebenfalls in Textform in diesem Fenster ausgegeben.
Sollte während der Abarbeitung des Test ein Fehler auftreten, so wird dieser hier ebenfalls protokolliert.
Der Test läuft wie in Kapitel 2 beschrieben ab und produziert eine Ausgabe wie sie beispielhaft in Abb. 17 zu sehen ist.
Sollte bei überprüfen der Anfragen, beim Start des Test, ein Fehler auftreten, so wird der Testlauf abgebrochen. Tritt im weiteren Verlauf ein Fehler auf, so werden alle weitern Anfragen abgesendet und produzieren Folgefehler. Diese Folgefehler, benötigen nicht viel Zeit und führen nicht zu fatalen Fehlern, wie ein Datenbank- oder Softwareabsturz. Da der Testablauf durch dieses Folgefehler nicht gestört wird, werden am Ende des Tests alle erzeugten Tabellen auch wieder gelöscht.

```
IndexUeberpruefe Statements...
Ueberpruefung abgeschlossen!
Erzeuge und fuelle Basistabelle...

Ausfuehrung von erstem Statement...
CREATE TABLE tab1 (attrib1 INTEGER, attrib2 INTEGER)
insert into tab1(attrib1, attrib2) select attrib1, attrib2 from f where attrib1<10
Erstes Statement benoetigte 1300 millisekunden.

Ausfuehrung von zweitem Statement...
CREATE TABLE tab2 (attrib1 INTEGER, attrib2 INTEGER)
insert into tab2(attrib1, attrib2) select attrib1, attrib2 from f where attrib2<10
Zweites Statement benoetigte 1295 millisekunden.

Gewinnen von Informationen aus erstem Statement...
CREATE TABLE tab12 (attrib1 INTEGER, attrib2 INTEGER)
INSERT INTO tab12 SELECT * FROM Tab1
Gewinnen von INformationen aus erstem Statement benoetigte 1214 millisekunden.

Gewinnen von Informationen aus zweitem Statement...
CREATE TABLE tab22 (attrib1 INTEGER, attrib2 INTEGER)
INSERT INTO tab22 SELECT * FROM Tab2
Gewinnen von Informationen aus zweitem Statement benoetigte 1665 millisekunden.

Ausfuehrung des gemeinsamen Statements...
CREATE TABLE tab1tab2 (attrib1 INTEGER, attrib2 INTEGER)
INSERT INTO tab1tab2(attrib1, attrib2) SELECT attrib1, attrib2 FROM F WHERE (attrib1<10) OR
(attrib2<10)
Erzeugung des gemeinsames Statements benoetigte 1636 millisekunden.

Ausfuehrung des Statements zum wiedergewinnen der Information fuer Statement 1 aus gemeinsamer
Tablle...
CREATE TABLE tab1tab2tab1 (attrib1 INTEGER, attrib2 INTEGER)
INSERT INTO tab1tab2tab1 SELECT attrib1, attrib2 FROM tab1tab2 WHERE attrib1<10
Erzeugung des Statements 1 aus gemeinsamen Statement benoetigte 1719 millisekunden.

Ausfuehrung des Statements zum wiedergewinnen der Information fuer Statement 2 aus gemeinsamer
Tablle...
CREATE TABLE tab1tab2tab2 (attrib1 INTEGER, attrib2 INTEGER)
INSERT INTO tab1tab2tab2 SELECT attrib1, attrib2 FROM tab1tab2 WHERE attrib2<10
Erzeugung des Statements 2 aus gemeinsamen Statement benoetigte 2011 millisekunden.

Ende!
```

Abb. 17: Protokoll eines Testlaufes

3. Testreihen

Im diesem Kapitel wird eine kurze Testreihe, zum veranschaulichen der Auswertungsmöglichkeiten, unter zur Hilfenahme der Software, dargestellt.
Für diese Testläufe wurde das Szenario Merge Where ausgewählt.
Dabei sollte die Basistabelle 10.000 und 100.000 Tuppel aus Zufallszahlen zwischen 0 und 20 beinhalten.
Die Anfragen sollten zwei Attribute der Basistabelle selektieren, wobei die erste Anfrage mit der WHERE Bedingung „attrib1<10“ und die zweite Anfrage mit der WHERE Bedingung „attrib2<10“ versehen waren.
Hieraus ergeben sich folgende Anfragen:

- INSERT INTO Tab1(attrib1, attrib2) SELECT attrib1, attrib2 FROM F WHERE attrib1<10
- INSERT INTO Tab2(attrib1, attrib2) SELECT attrib1, attrib2 FROM F WHERE attrib2<10

Die gemeinsame Anfrage nach Umformung durch die entsprechende Rewrite-Regel hat folgende Form:

- INSERT INTO Tab1Tab2(attrib1, attrib2) SELECT attrib1, attrib2 FROM F WHERE (attrib1<10) OR (attrib2<10)

Zur Gewinnung der Informationen aus der geinsamen Tabelle Tab1Tab2 werden folgende Anfragen verwendet:

- INSERT INTO Tab1Tab2Tab1 SELECT attrib1, attrib2 FROM Tab1Tab2 WHERE attrib1<10
- INSERT INTO Tab1Tab2Tab2 SELECT attrib1, attrib2 FROM Tab1Tab2 WHERE attrib2<10

Getestet wurden auf Basistabellen, welche die entsprechenden WHERE Bedingungen zu bestimmten Prozentsätzen erfüllten.

Bei den Prozentsätzen (attrib1 / attrib2) handelte es sich um folgende Kombinationen:

- 10% / 10%; 20% / 20%; ... ;90% / 90%; 100% / 100%
- 10% / 90%; 20% / 80%; 30% / 70%; 40% / 60%; 50% / 50%

Alle Verteilungen wurden nach folgenden Indizierungen der Basistabellenattribute getestet:

- keine Indizierung der Attribute.
- Indizierung von attrib1.
- Indizierung von attrib1 und attrib2.

3.1 Theoretische Überlegungen und Erwartungen

Unabhängig von der Indizierung der Attribute, wird nach theoretischen Überlegungen eine Verbesserung der Anfragezeiten, zur Gewinnung der Informationen aus der gemeinsamen Tabelle, im Vergleich zur Gewinnung aus den gegebenen Tabellen, erwartet.

3.1.1 Prozentuale Gleichverteilung

Im Falle einer prozentualen Gleichverteilung, das heißt sowohl attrib1 als auch attrib2 treten in den Daten der Basistabelle, mit dem gewünschten Wert, gleich häufig auf, wird erwartet, dass das Erzeugen der gemeinsamen Tabelle weniger Zeit beansprucht als das Erzeugen der Tabellen für die gegebenen Anfragen. Dies kann daraus gefolgert werden, dass die Basistabelle für das Erstellen der Rewrite-Regel-Tabelle nur einmal durchsucht werden muss. Im Vergleich dazu muss zum Erstellen der beiden Tabellen für die gegebenen Anfragen die Basistabelle zweimal durchsucht werden. Es kann also eine zeitliche Verbesserung von 50% erwartet werden.

Im besten Fall sind die Tuppel, welche die Bedingung attrib1<10 erfüllen genau die gleichen Tuppel welche auch die Bedingung attrib2<10 erfüllen. Dies bedeutet zum Beispiel bei einer Verteilung von 10%/10% und einer Basistabellengröße von 10.000 Tuppeln, dass die neue Tabelle nach den Rewrite-Regeln (Tab1Tab2), genauso wie jeweils die Tabellen für die gegebenen Anfragen(Tab1 und Tab2), eine Größe von 1000 Einträgen hat.
Hieraus kann man folgern, dass das Auslesen der Rewrite-Regel-Tabelle genau die selbe Zeit wie für das Auslesen von Tab1 und Tab2 benötigt.

Im schlechtesten Fall sind die Tuppel welche die Bedingung attrib1<10 erfüllen nicht die Tuppel welche die Bedingung attrib2<10 erfüllen. Hieraus folgt, bei oben angenommener Verteilung von 10%/10%, dass Tab1Tab2 eine Größe von 2000 Einträgen hat, wobei Tab1 und Tab2 weiterhin 1000 Tuppel beinhalten.
Unter der Annahme, dass die Zeit für das Auslesen, linear zur Anzahl der Datensätze wächst, würde dies eine zeitliche Verschlechterung von 100% bedeuten.

Wenn angenommen wird, dass pro gelesenes Tuppel eine Zeiteinheit verbraucht wird und sich das Auslesen linear zur Anzahl der Tuppel verhält, können folgende erwartete Zeitverbesserungen berechnet werden:

Bester Fall:

	Erz. der Tabelle(n)	Lesen der Tabelle(n)	Summe
Gegeb. Anfragen	2*10.000	2*1.000	22.000
Rewrite-Regel	1*10.000	2*1.000	12.000

Schlechtester Fall:

	Erz. der Tabelle(n)	Lesen der Tabelle(n)	Summe
Gegeb. Anfragen	2*10.000	2*1.000	22.000
Rewrite-Regel	1*10.000	2*2.000	14.000

Somit ist eine Verbesserung der Anfragezeiten, nach theoretischen Überlegungen, im besten Fall von 54% zu erwarten. Diese Zeiteinsparung entsteht beim Erzeugen der Tabellen und wird dadurch, dass jede Ausleseoperation gleichviel Zeit benötigt, erhalten.
Im schlechtesten Fall ergibt sich ebenfalls eine Zeiteinsparung beim Erzeugen der Tabellen. Diese Einsparung kann aber nur für fünf Ausleseoperationen gehalten werden, da das Auslesen der Rewrite-Regel-Tabelle doppelt so viel Zeit benötigt wie das Auslesen von Tab1 und Tab2.

3.1.2 Verteilung im Verhältnis x%=100%-y%

Im Falle einer Verteilung im Verhältnis x%=100%-y%, das heißt attrib1 tritt mit einer Häuffigkeit von x% und attrib2 mit einer Häuffigkeit von y% in den Daten der Basistabelle, mit dem gewünschten Wert, auf, wird erwartet, dass das Erzeugen der gemeinsamen Tabelle weniger Zeit beansprucht als das Erzeugen der Tabellen für die gegebenen Anfragen. Dies kann daraus gefolgert werden, dass die Basistabelle für das Erstellen der Rewrite-Regel-Tabelle nur einmal durchsucht werden muss. Im Vergleich dazu muss zum erstellen der beiden Tabellen für die gegebenen Anfragen die Basistabelle zweimal durchsucht werden. Es kann also eine zeitliche Verbesserung von 50% erwartet werden.

Im besten Fall stimmen alle Tuppel, welche die Bedingung attrib1<10 erfüllen, mit Tuppeln, welche die Bedingung attrib2<10 erfüllen, überein. Dies bedeutet zum Beispiel bei einer Verteilung von 40%/60% und einer Basistabellengröße von 10.000 Tuppeln, dass die neue Tabelle nach den Rewrite-Regeln (Tab1Tab2) eine Größe von 6000 und die Tabellen für die gegebenen Anfragen(Tab1 und Tab2) eine Größe von 4000 und 6000 Einträgen haben.
Hieraus kann man folgern, dass das Auslesen der Rewrite-Regel-Tabelle 40% weniger Zeit in Anspruch nimmt als das Auslesen von Tab1 und Tab2.

Im schlechtesten Fall stimmen die Tuppel, welche die Bedingung attrib1<10 erfüllen, nicht mit den Tuppel, welche die Bedingung attrib2<10 erfüllen, überein. Dies bedeutet zum Beispiel bei der oben angenommenen Verteilung und Basistabellengröße, dass Tab1Tab2 eine Größe von 10.000, Tab1 eine Größe von 4000 und Tab2 eine Größe von 6000 Tuppeln hat. Hieraus kann man folgern, dass das Auslesen der Rewrite-Regel-Tabelle genau die selbe Zeit wie für das Auslesen von Tab1 und Tab2 benötigt.

Wenn angenommen wird, dass pro gelesenes Tuppel eine Zeiteinheit verbraucht wird und sich das Auslesen linear zur Anzahl der Tuppel verhält, können folgende erwartete Zeitverbesserungen berechnet werden:

Bester Fall:

	Erz. der Tabelle(n)	Lesen der Tabelle(n)	Summe
Gegeb. Anfragen	2*10.000	4.000 + 6.000	30.000
Rewrite-Regel	1*10.000	2*6.000	22.000

Schlechtester Fall:

	Erz. der Tabelle(n)	Lesen der Tabelle(n)	Summe
Gegeb. Anfragen	2*10.000	4.000 + 6.000	30.000
Rewrite-Regel	1*10.000	2*10.000	30.000

Somit ist keine Verbesserung der Anfragezeiten, nach theoretischen Überlegungen, im schlechtesten Fall zu erwarten. Es entsteht eine Zeiteinsparung beim Erzeugen der Tabellen. Diese Einsparung wird aber dadurch, dass jede Ausleseoperation von Tab1Tab2 doppelt soviel Zeit benötigt wie das Auslesen von Tab1 und Tab2, nach dem ersten Auslesen aufgehoben.
Im besten Fall ergibt sich eine Zeiteinsparung beim Erzeugen der Tabellen. Diese Einsparung kann aber nur für fünf Ausleseoperationen gehalten werden, da das Auslesen der Rewrite-Regel-Tabelle mehr als doppelt so viel Zeit benötigt wie das Auslesen von Tab1 und Tab2.

3.2 Testreihen und ihre Auswertung

Im Folgenden werden die mit der Software gewonnen Daten, zu den gewählten Testreihen, vorgestellt.
Alle Test wurden dreifach ausgeführt und ihre Ergebnisse arithmetisch gemittelt, um etwaige Leistungsschwankungen der Datenbank, zu bereinigen.

3.2.1 Prozentuale Gleichverteilung

Die im folgenden besprochenen Abbildungen 18 und 19 beziehen sich auf eine Testreihe mit 10.000 Tuppeln in der Basistabelle und einer prozentualen Gleichverteilung. Keines der beiden Basistabellenattribute wurde indiziert.
Beim Erzeugen der Tabellen zu den gegebenen Anfragen und der Tabelle nach der Rewrite-Regel ergaben sich folgende Ergebnisse:

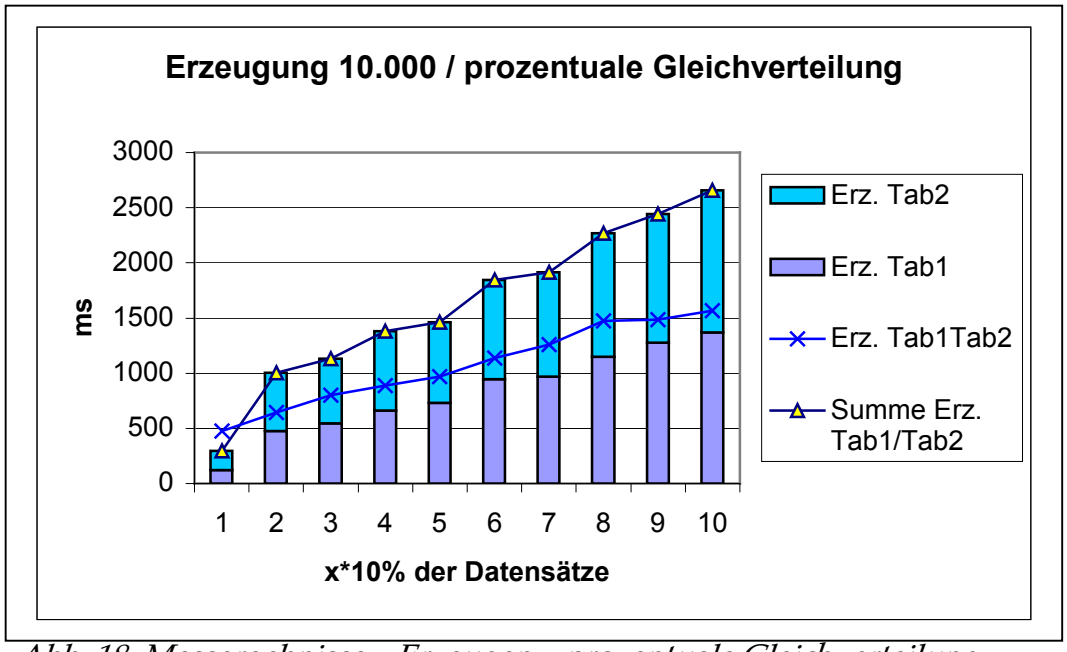

Abb. 18: Messergebnisse – Erzeugen – prozentuale Gleichverteilung

Wie in den theoretischen Überlegungen erwartet, führt das Erzeugen der Tabelle nach der Rewrite-Regel zu einer Zeitverbesserung. Diese Einsparung liegt im getesteten Fall bei $\approx 40\%$.
Beim Auslesen der Tabellen wurde erwartet, dass die Einsparung für mindestens 5 Anfragen gehalten werden kann. Wie aus Abb. 19 in Verbindung mit Abb. 18 ersichtlich wird kann die Einsparung, innerhalb dieser Testreihe, nur für rund 2 Anfragen gehalten werden.
Damit liegen die realen Ergebnisse der Testreihe unter den theoretischen Erwartungen.

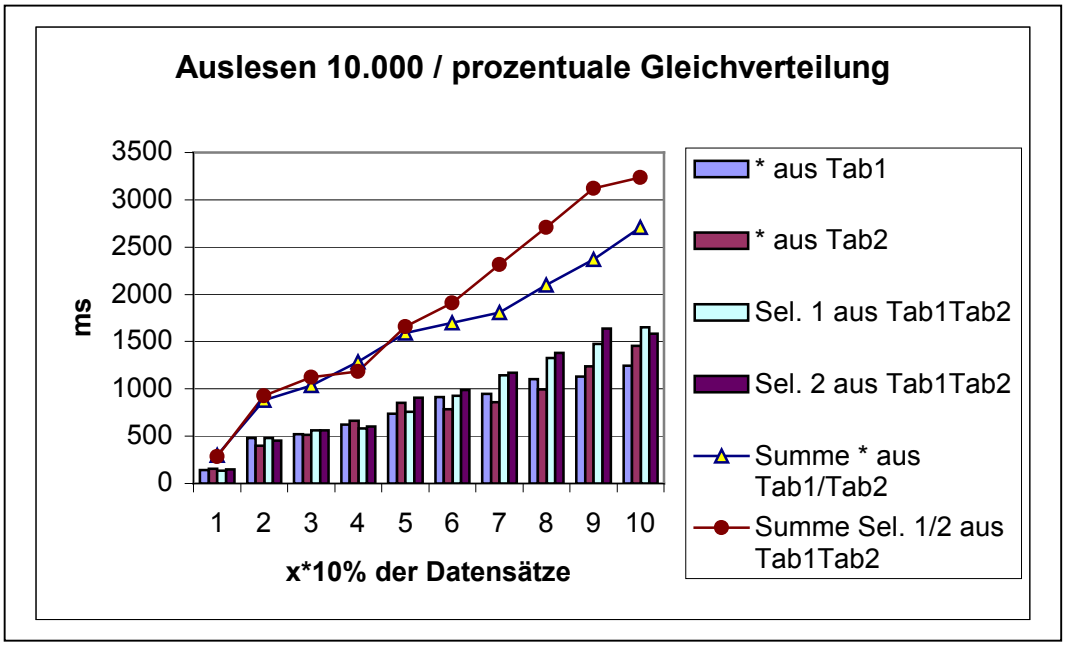

Abb. 19: Messergebnisse – Auslesen – prozentuale Gleichverteilung

Wie in Abbildung 20 gezeigt wird, führt das Erzeugen von Indizes auf der Basistabelle zu keine signifikanten Verbesserungen oder Verschlechterung der Messergebnisse.

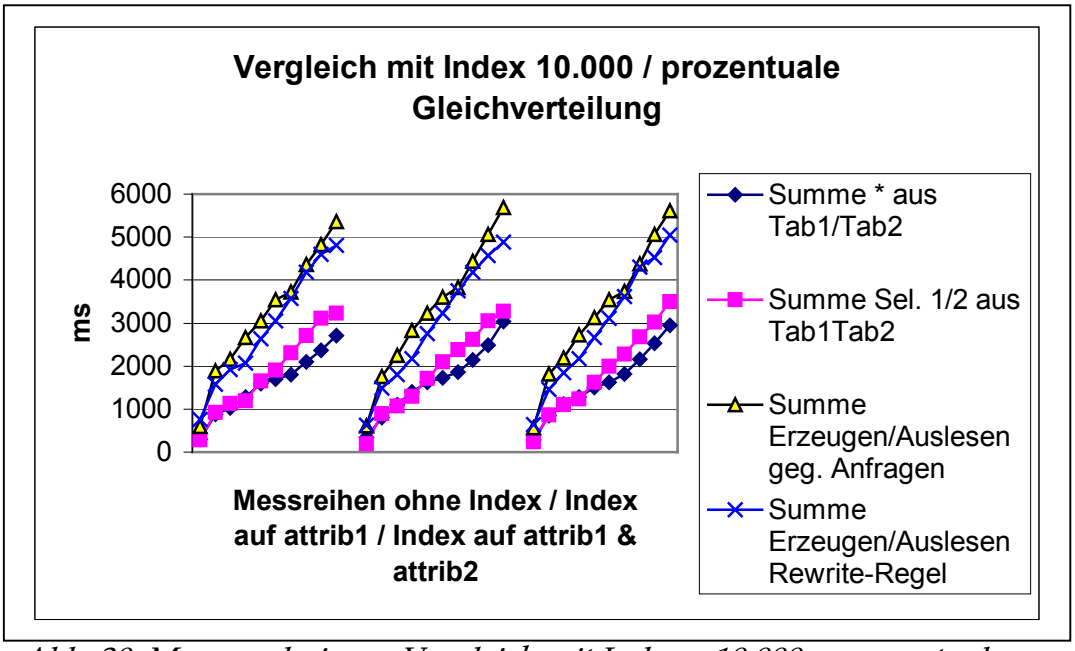

Abb. 20: Messergebnisse – Vergleich mit Index – 10.000 – prozentuale Gleichverteilung

Bei der selben Testreihe mit 100.000 Tuppeln in der Basistabelle kann in Abb. 21 abgelesen werden, dass die theoretisch errechnete Unterschwelle, vom Erhalten der Zeitersparnis beim Erzeugen der Tabelle Tab1Tab2, im Rahmen der Messgenauigkeit, erreicht wird.

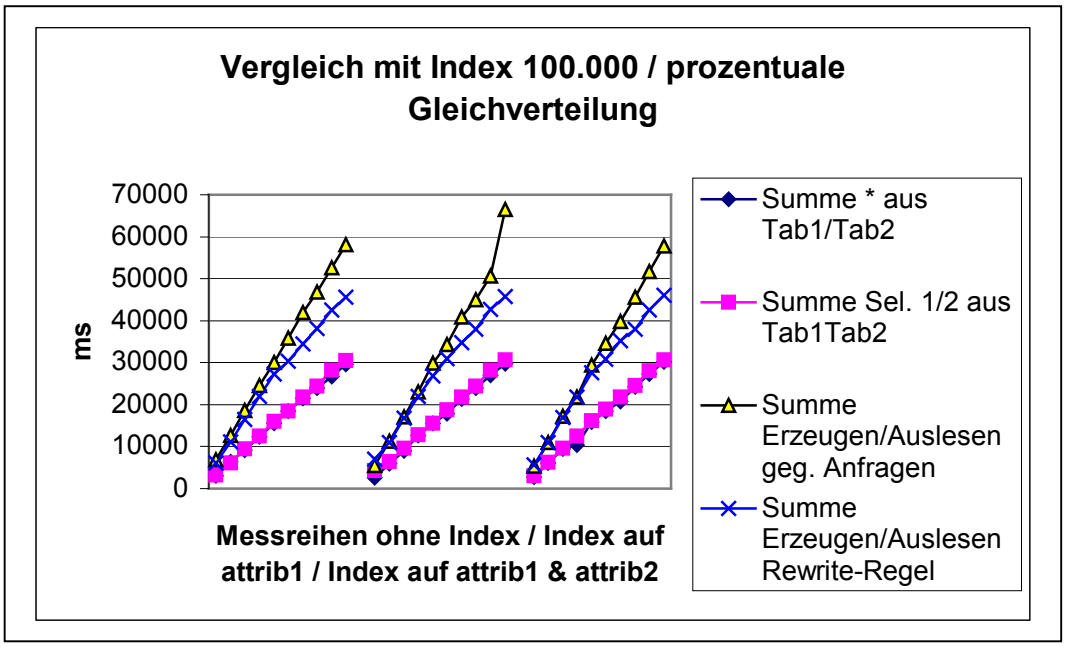

Abb. 21: Messergebnisse – Vergleich mit Index – 100.000 – prozentuale Gleichverteilung

3.2.2 Verteilung im Verhältnis x%=100%-y%

Die im folgenden besprochenen Abbildungen 22 und 23 beziehen sich auf eine Testreihe mit 10.000 Tuppeln in der Basistabelle und einer Verteilung im Verhältnis x%=100%-y%. Keines der beiden Basistabellenattribute wurde indiziert.
Beim Erzeugen der Tabellen zu den gegebenen Anfragen und der Tabelle nach der Rewrite-Regel ergaben sich folgende Ergebnisse:

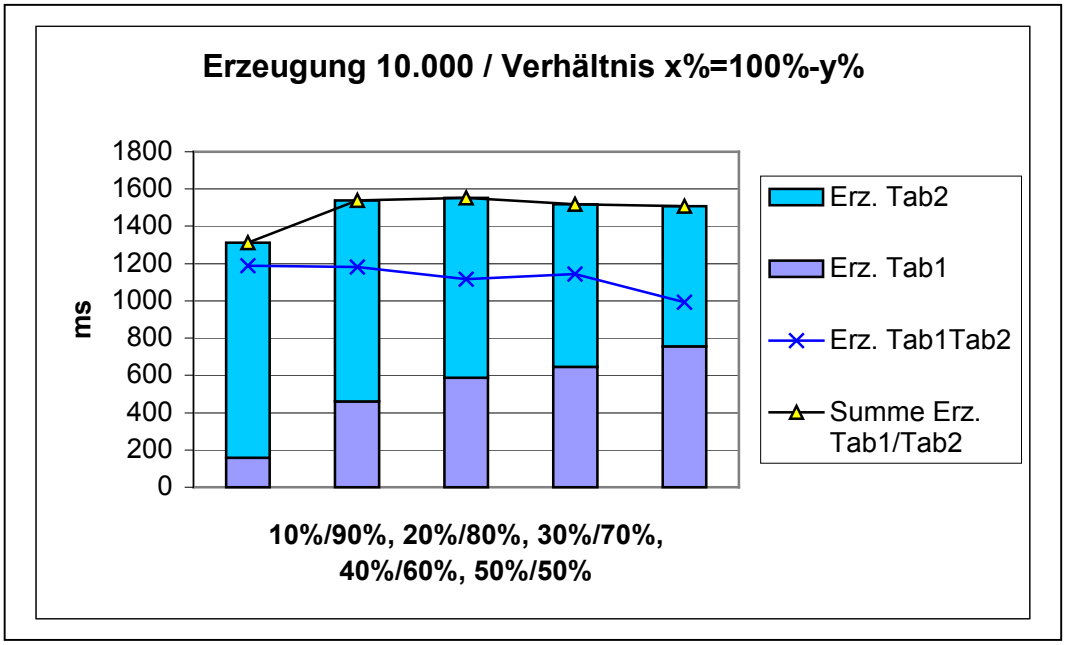

Abb. 22: Messergebnisse – Erzeugen – Verhältnis x%=100%-y%

Wie in den theoretischen Überlegungen erwartet, führt das Erzeugen der Tabelle nach der Rewrite-Regel zu einer Zeitverbesserung. Diese Einsparung liegt im getesteten Fall bei ≈ 25%.
Beim Auslesen der Tabellen wurde erwartet, dass die Einsparung für mindestens 5 Anfragen gehalten werden kann. Wie aus Abb. 23 in Verbindung mit Abb. 22 ersichtlich wird kann die Einsparung, innerhalb dieser Testreihe, nur für rund 2 Anfragen gehalten werden.
Damit liegen die realen Ergebnisse der Testreihe unter den theoretischen Erwartungen.

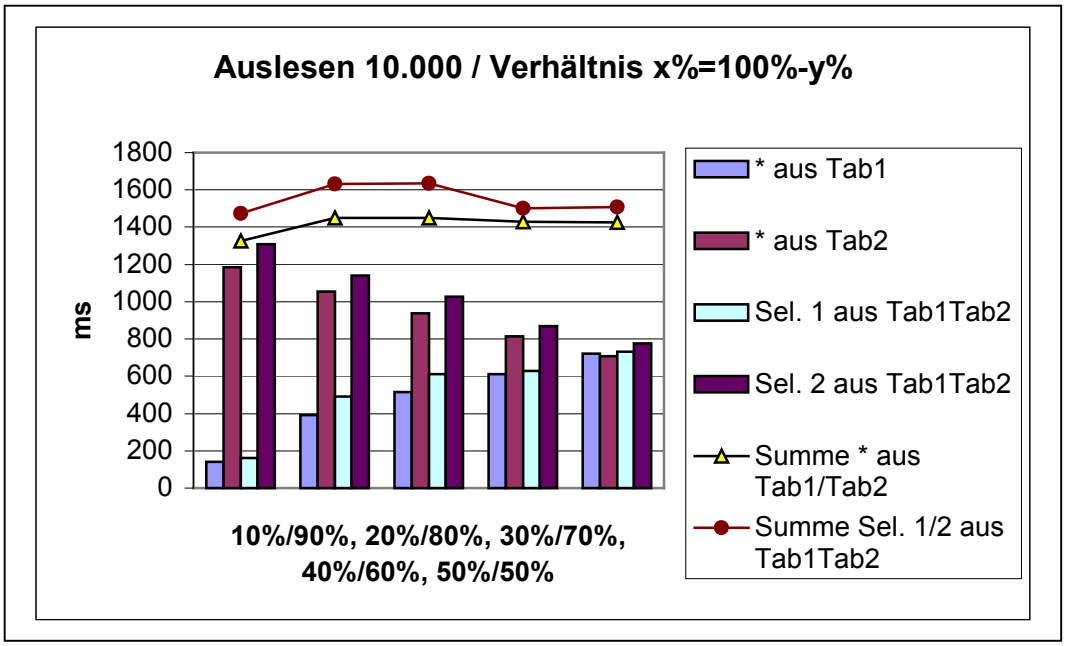

Abb. 23: Messergebnisse – Auslesen – Verhältnis x%=100%-y%

Wie in Abbildung 24 gezeigt wird, führt das Erzeugen von Indizes auf der Basistabelle zu keine signifikanten Verbesserungen oder Verschlechterung der Messergebnisse.

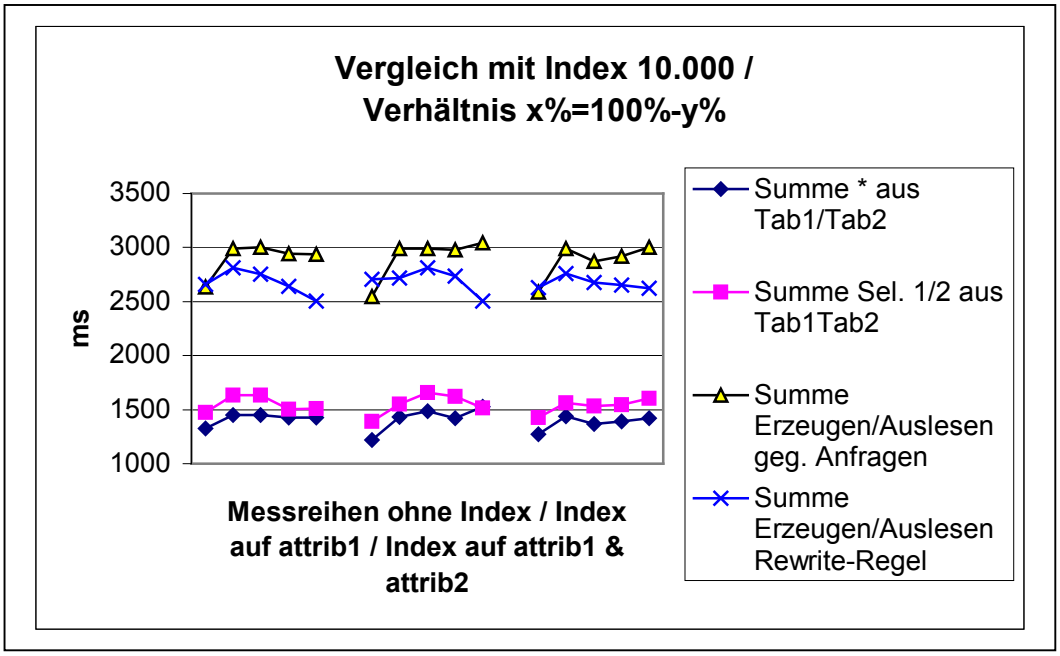

Abb. 24: Messergebnisse – Vergleich mit Index – 10.000 – Verhältnis x%=100%-y%

Bei der gleichen Testreihe mit 100.000 Tuppeln in der Basistabelle kann in Abb. 25 abgelesen werden, dass die theoretisch errechnete Unterschwelle, vom Erhalten der Zeitersparnis beim Erzeugen der Tabelle Tab1Tab2, im Rahmen der Messgenauigkeit, erreicht wird, sofern eine Zeiteinsparung vorliegt.

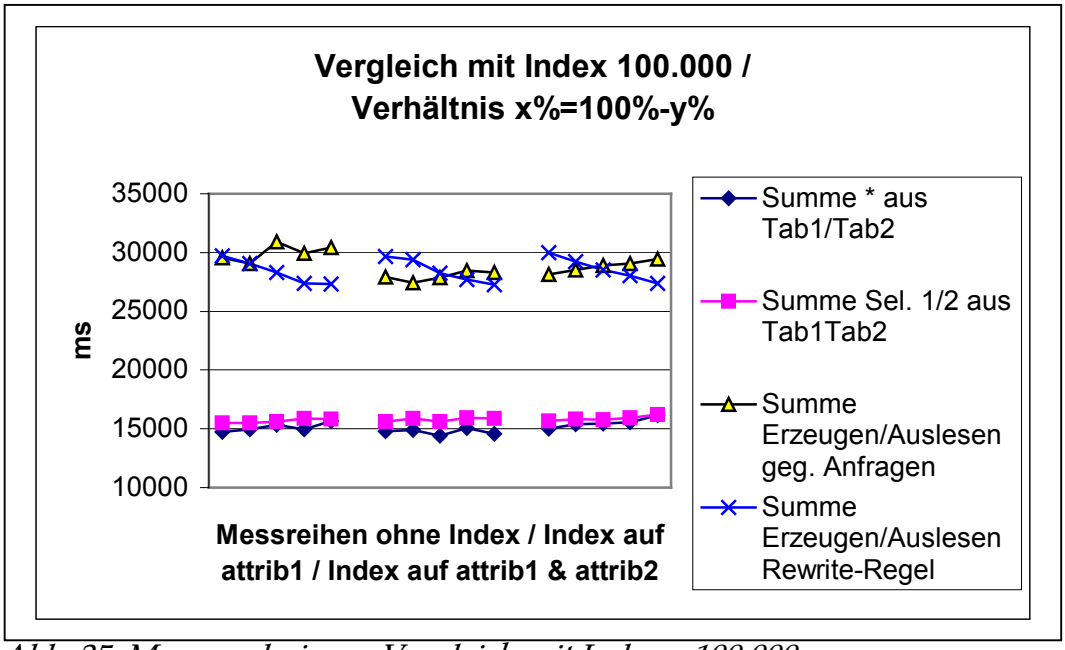

Abb. 25: Messergebnisse – Vergleich mit Index – 100.000 – Verhältnis x%=100%-y%

4. Zusammenfassung und Ausblick

Im Rahmen dieser Studienarbeit wurde eine Testumgebung für Anfragesequenzen erstellt.
Zur Optimierung dieser Anfragesequenzen werden Rewrite-Regeln eingesetzt. Diese Rewrite-Regeln sind jedoch nicht allgemeingültig, d.h. je nach Konstellation der Anfragesequenz kann deren Anwendung zu einer Verbesserung, aber auch zu einer Verschlechterung der Ausführungszeiten führen. Innerhalb dieser Studienarbeit wurde eine Testumgebung erstellt, die für die gegebenen Rewrite-Regeln jeweils automatisch verschiedene Varianten von Anfragesequenzen generiert und für diese und die zugehörigen transformierten Sequenzen die Ausführungszeiten erfasst. Die Messergebnisse geben darüber Aufschluss, in welchen Fällen eine Verbesserung bzw. Verschlechterung eintritt und welche Faktoren hierfür relevant sind.

Im Rahmen einer kleinen Testreihe wurden die Auswertungsmöglichkeiten aufgezeigt. Hierbei zeigte sich, dass nicht alle theoretischen Überlegungen, bezüglich der Zeitersparnis, durch Test bestätigt werden können. Das heißt das genau geplante Testreihen durchgeführt werden müssen, um die Wirksamkeit von Rewrite-Regeln in verschiedenen Fällen auf der Datenbank zu zeigen.

Anmerkungen und Erweiterungen:
Da die Testläufe sehr zeitaufwändige sind, ist als Erweiterung der Software eine Script-Schnittstelle vorstellbar, welche die Eingaben für verschiedene Testläufe übernimmt und automatisiert.

© IPVR, Universität Stuttgart, 2004

5. Quellennachweis

[1] *Tobias Kraft: Rewrite-Strategien für generierte Anfragesequenzen im Online Analytical Processing, Diplomarbeit, Universität Stuttgart, 2001*

[2] *Jim Melton, Alan R. Simon: SQL:1999 Understanding Relational Language Components, Morgan Kauffman, 2002*

[3] *Michael Stillger: Johann-Christoph Freytag, Testing the Quality of a Query Optimizer, IEEE Data Eng. Bull. 18(3): 41-48, 1995*

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

Tobias Flaig

© IPVR, Universität Stuttgart, 2004