

Studienarbeit Nr. 1945

Realisierung des TPC-H-Schemas auf einem Oracle-Datenbanksystem

Thorsten Müller

Studiengang:	Informatik
Prüfer:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer:	Dipl. Inf. Tobias Kraft
begonnen am:	17. März 2004
beendet am:	16. September 2004
CR-Klassifikation:	H.2.4, H.2.7, H.4.2

Institut für Parallele und
Verteilte Systeme (IPVS)
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Aufgabe dieser Arbeit ist die Portierung des TPC-H-Benchmarkschemas aus einem bestehenden DB2-Datenbanksystem auf ein Oracle-Datenbanksystem (Ver.10g). Dies wird in drei Phasen realisiert. In der ersten Phase, an deren Ende die Installation des Oracle-Datenbanksystems steht, erfolgt eine Einarbeitung in den TPC-H-Benchmark und das Oracle-Datenbanksystem. Die zweite Phase umfasst die Datenübertragung und den Datenimport in das Oracle-System. Hierbei werden nicht nur alle Daten sondern auch die Datenbankstruktur (Indizes, Tabellen, Views, ...) übertragen. In der dritten und letzten Phase werden die Leistungsmessungen durchgeführt, die bereits auf dem DB2-System durchgeführt wurden. Darüber hinaus wird in der dritten Phase noch auf Join-Reihenfolgen und das Sammeln von Statistiken eingegangen.

Inhaltsverzeichnis

1	Einleitung	4
2	Grundlagen	7
2.1	Der TPC-H Benchmark	7
2.2	Zugrundeliegende Datenbasis	9
2.3	Das Oracle-Datenbanksystem	13
2.3.1	Oracle Universal Installer (OUI)	13
2.3.2	Database Configuration Assistant (DBCA)	14
2.3.3	Administration Assistent for Windows	14
2.3.4	Database Upgrade Assistant	14
2.3.5	Oracle Net Manager	15
2.3.6	Ultra Search und Ultra Search Administration Tool	15
2.3.7	Oracle Enterprise Manager 10g Database Control	16
2.3.8	SQL*Plus und iSQL*Plus	17
3	Vergleich der Datenübertragungswerkzeuge	20
3.1	Originalimport- und Originalexportwerkzeug	20
3.2	Data Pump	21
3.3	SQL*Loader	22
3.4	Transportable Tablespaces	23
3.5	Transparent Gateways	23
3.6	Ermittlung des passenden Migrationswerkzeugs	24
4	Migration	26
4.1	Installation	26
4.2	Datenübertragung mit SQL*Loader	29
4.2.1	Export der Tabellen (Schritt 1)	29
4.2.2	Control Files erstellen (Schritt 2)	31
4.2.3	Tablespace und Benutzer TPCH4 erstellen (Schritt 3)	32
4.2.4	Die Tabellen im Oracle-System erstellen (Schritt 4)	34
4.2.5	Datenimport mit SQL*Loader (Schritt 5)	36
4.2.6	Primär- und Fremdschlüsselbeziehungen erstellen (Schritt 6)	37
4.2.7	Indizes der Tabellen umbenennen bzw. erstellen (Schritt 7)	38
4.2.8	Views im Oracle-System erstellen (Schritt 8)	38

Inhaltsverzeichnis

5	Leistungsmessungen	40
5.1	Schnittstellen zur Analyse der Ausführungspläne	40
5.1.1	Explain Plan	40
5.1.2	SQL Trace und TKPROF	43
5.1.3	Laufzeitmessungen	46
5.2	Vorbereitungen und Durchführung	47
5.2.1	Vorgegebene Sequenzen	47
5.2.2	Umwandlung der Sequenzen in Oracle SQL	52
5.2.3	Anwendung der Analysewerkzeuge	54
5.3	Auswertung der Leistungsmessungen	55
6	Optimizer Hints	63
6.1	Allgemeine Hints Syntax	64
6.2	Hints zur Festlegung der Join-Reihenfolge	64
6.3	Hints zur Festlegung der Join-Methode	66
7	Manipulationsmöglichkeiten	69
7.1	Art und Verwendungszweck der Statistiken	69
7.2	Sammeln der Statistiken	70
7.3	Histogramme	72
7.4	Exportieren oder Importierung von Statistiken	74
8	Schluss	75
8.1	Zusammenfassung	75
8.2	Fazit und Ausblick	75
	Literaturverzeichnis	77
	Anhang A	79
	Anhang B	101
	Anhang C	110

1 Einleitung

Heutzutage spielen Datenbanken eine immer wichtigere Rolle in den verschiedensten Bereichen der Industrie. Jedes Unternehmen das Informationen über seine Kunden, Produkte oder dergleichen sammeln und verarbeiten will, benötigt Datenbanken um der stetig steigenden Informationsflut Herr zu werden. Um diese Datenbanken und somit die Datenbestände sinnvoll auswerten zu können, werden die verschiedensten Datenbank-Systeme und -Werkzeuge angeboten. Die drei größten Anbieter für den Bereich der relationalen Datenbank-Systeme sind, laut einer Statistik der Gartner Group, der Marktführer Oracle, kurz dahinter IBM, mit ihrem System DB2 und an Platz drei Microsoft.(s.Abbildung 1.1)

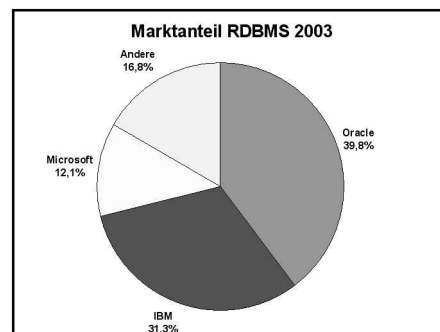


Abbildung 1.1: *relationale Datenbank-Management-Systeme Marktaufteilung 2003*[Quelle:Gartner Group]

Die drei unterschiedlichen Datenbanksysteme der Marktführer besitzen jeweils einen ähnlichen Basisfunktionsumfang, aber jedes der einzelnen DB-Systeme hat seine Funktionen auf eine individuelle Art und Weise implementiert. Zudem setzen die Unternehmen jeweils unterschiedliche Schwerpunkte, um ihre Produkte auch voneinander abzuheben. In verschiedenen Bereichen sollten sich die Hersteller an einen gemeinsamen Standard halten, aber leider sind auch hier Unterschiede festzustellen. Zum Beispiel halten sich die einzelnen Systeme nicht an einen einheitlichen SQL-Standard sondern benutzen jeweils, wenn man einmal von den Grundfunktionalitäten wie z.B. der SELECT...FROM...WHERE-Reihenfolge und ähnlichen Dingen absieht, eine mehr oder minder stark abweichende, individuelle Syntax. Des Weiteren verwenden die unterschiedlichen Hersteller auch unterschiedliche Datenformate, um die jeweiligen Daten abzuspeichern. So gibt es bei DB2 fünf unterschiedliche Datenformate für die Abbildung von numerischen Daten (INTE-

1 Einleitung

GER, SMALLINT, DECIMAL, SINGLE PRECISION, DOUBLE PRECISION), dem gegenüber stehen in Oracle nur zwei unterschiedliche Datenformate für numerische Daten (NUMBER, FLOAT). Diese Unterschiede führen unter anderem dazu, dass die Portierung der Datenbanken, also die Überführung der Datenbasis und der logischen Verknüpfungen von einem System in das andere, meist mit einigen Problemen verbunden ist.

Um die Leistung der unterschiedlichen Systeme in Bezug auf bestimmte Eigenschaften wie Anfragezeit und Performanz miteinander vergleichen zu können, müssen die zu vergleichenden DB-Systeme standardisierten Tests unterzogen werden, was auch Benchmarking genannt wird. Ursprünglich kommt der Begriff Benchmark aus der Landvermessung und bezeichnet einen fixen Punkt in der Landschaft. Mit Benchmarking ist in diesem Zusammenhang das Orientieren an diesem Punkt gemeint. Im Falle der DB-Systeme werden die Ergebnisse des Benchmarking eingesetzt, um die einzelnen Softwareprodukte in eine standardisierte Leistungsskala einordnen zu können. Das Ergebnis des Benchmarking ergibt für jedes getestete Produkt einen sogenannten Benchmark-Wert, der an einer imaginären Messlatte, z.B. dem Wert des auf dem Markt führenden Softwareprodukts, gemessen werden kann. Je nach Höhe des ermittelten Benchmark-Werts kann nun erkannt werden, ob das getestete DB-System höher, niedriger oder gleichhoch wie die zu vergleichende Messlatte in die Leistungsskala eingeordnet werden kann.

Das Transaction Performance Processing Council (kurz. TPC) hat es sich zur Aufgabe gemacht verschiedene Benchmarks zur Verfügung zu stellen, um die Leistungen der unterschiedlichen DB-Systeme zu vergleichen. Die unterschiedlichen Benchmarks des TPC zielen auf unterschiedliche DB-Systeme ab, und werden durch TPC- und einem einzelnen Buchstaben abgekürzt. Die Abkürzung TPC-C steht zum Beispiel für den Benchmark der auf on-line transaction processing systems (online Dialogverarbeitungssysteme) abzielt. Die beiden Benchmarks TPC-A und TPC-B sind obsolet, sie wurden von der Technik eingeholt und sind nicht mehr im Gebrauch. Der Benchmark auf den sich diese Arbeit bezieht ist der TPC BenchmarkH (TPC-H) und ist ein Entscheidungsunterstützung-Benchmarksystem (ad-hoc, decision support benchmark). In Abschnitt 2.1 (*Der TPC-H Benchmark*) wird genauer auf dessen Aufbau und Funktionsweise eingegangen.

Innerhalb des ORBIT-Projekts am IPVS wurde das TPC-H-Schema bereits auf einem Windows-NT-System unter SQL Server, und auf einem SUN-Solaris-System unter DB2 realisiert, und um einige Sichten und Tabellen erweitert [Wag99]. ORBIT steht für *Optimization and IntegRation of Business Intelligence Technology*. Ziel des ORBIT-Projekts ist die Entwicklung neuer Strategien und Techniken, mit deren Hilfe man die Lücke zwischen Business Intelligence und Datenbanksystemen schließen kann. So wurde zum Beispiel bereits ein DSS Optimierer (*Decision Support System Optimizer*) entwickelt, der in der Lage ist, Anfragen an eine Datenbank umzuschreiben und zu verbessern, bevor diese vom Datenbanksystem abgearbeitet werden. Decision Support (also Entscheidungsunterstützung) ermöglicht das Durchführen von Analysen auf der Datenbasis eines Unternehmens und hilft somit beim Fällen von Entscheidungen, die das Unternehmensumfeld betreffen. Mit den von den DSS-Tools generierten Sequenzen von SQL-Statements

1 Einleitung

wurden sowohl unter DB2, als auch für SQL Server umfangreiche Leistungsmessungen durchgeführt. Aufgabe dieser Studienarbeit ist es nun, das TPC-H-Schema und die erweiterten Sichten und Tabellen, die für die Verwendung des DSS-Tools von MicroStrategy notwendig sind, auf eine Oracle-Datenbank zu übertragen. Darüber hinaus sollen die bereits auf den beiden anderen Systemen durchgeführten Messungen auf dem Oracle-System nachvollzogen werden.

In Kapitel 2 werden die Grundlagen behandelt, die für diese Arbeit von Interesse sind. Es wird allgemein auf den TPC-H Benchmark eingegangen und wie dieser erweitert wird um die zugrundeliegende Datenbasis zu bilden. Darüber hinaus wird das zu installierende Oracle-Datenbanksystem und dessen Werkzeuge vorgestellt. Im darauffolgenden 3. Kapitel werden die zur Verfügung stehenden Datenübertragungswerkzeuge betrachtet, und das im Migrationskapitel verwendete Tool ausgesucht. Im 4. Kapitel wird die Installation des Oracle-Datenbanksystems, und die eigentliche Migration der DB2-Datenbank in das Oracle-System beschrieben. Danach werden in Kapitel 5 die Leistungsmessungen beschrieben und auf die Schnittstellen zur Analyse eingegangen. Im vorletzten Kapitel werden die von Oracle zur Verfügung gestellten Werkzeuge zur Anfrageoptimierung, vor allem zur Erzwingung von Join-Reihenfolgen erläutert. Im 7. Kapitel wird auf Systemstatistiken, insbesondere auf Histogramme eingegangen und Manipulationsmöglichkeiten, erläutert. Am Schluß folgt eine kurze Zusammenfassung der Arbeit und ein kurzer Ausblick auf Ansatzpunkte für Folgearbeiten.

2 Grundlagen

In diesem Kapitel wird näher auf den TPC-H Benchmark eingegangen und ein Blick auf dessen Aufbau geworfen. In Abschnitt 2.2 *Zugrundeliegende Datenbasis* werden die dem TPC-H-Schema hinzugefügten Erweiterungen behandelt und kurz aufgezeigt wie diese die eigentliche Datenbasis erweitern und wofür diese benötigt werden. Danach folgt eine Einführung in das Oracle-Datenbanksystem (Version: 10g), und dessen unterschiedliche Werkzeuge.

2.1 Der TPC-H Benchmark

Der TPC BenchmarkH (TPC-H) ist ein Decision-Support-Benchmarksystem. Er besteht aus einer Reihe von businessorientierten ad-hoc Abfragen und parallelablaufenden Datenmanipulationen. Um auf veränderte Anforderungen im DB-Bereich reagieren zu können, werden die Rahmenbedingungen des Benchmarks an die laufende Entwicklung in der Industrie angepasst und weiterentwickelt, das eigentliche Schema (s.u.) bleibt aber bestehen. Die aktuelle Versionsnummer des TPC-H ist 2.1.0. Die Abfragen und die in der Datenbank enthaltenen Daten wurden vom TPC so ausgewählt, dass sie eine möglichst breite branchen-übergreifende Bedeutung haben. Dieser Benchmark illustriert ein Entscheidungsunterstützungssystem, das eine sehr große Datenmenge untersucht. Es werden Abfragen mit einem hohen Komplexitätsgrad ausgeführt und dadurch versucht Antworten auf kritische Geschäftsfragen zu geben.

Die Datenbasis des Benchmarks besteht aus 8 eigenständigen Tabellen von zum Teil sehr unterschiedlicher Größe. Wie die einzelnen Tabellen in Beziehung stehen wird in Abbildung 2.1 *TPC-H Schema* dargestellt. Bei den kursivgedruckten Attributen handelt es sich um die Primärschlüssel der jeweiligen Tabelle. Die Pfeilrichtung gibt die jeweilige Fremdschlüsselbeziehung an. Die Zahl unter dem Tabellennamen gibt die Kardinalität (die Anzahl der Zeilen) der jeweiligen Tabelle an. Die Kardinalitäten, die noch mit dem Faktor **SF** multipliziert werden, hängen vom Scaling Factor (dem Skalierungswert) des Benchmarks ab. Dieser Wert bestimmt die Größe der zu Grunde liegenden Datenbasis. Er kann die folgenden Werte annehmen: SF= 1, 10, 30, 100, 300, 1000, 3000, 10000, 30000, 100000. Bei einem **SF=1** beträgt die geschätzte Größe der Datenbasis etwa 1 GB. Um Datenbanken mit einem größeren Volumen zu simulieren, kann der **SF** entsprechend größer gewählt werden, und ergibt dadurch entsprechend eine Datenbasis von ca. 10GB, 30GB, 100GB, usw. . Für die vorliegende Implementierung wurde ein Scaling Factor von

2.1 Der TPC-H Benchmark

10 gewählt, was eine geschätzte Datenbasis von 10GB ergibt. In diese 10GB ist aber noch kein Speicherplatz für Zugriffspfadstrukturen einberechnet. Auch Recoverymaßnahmen wie Logging oder Datenreplikation können den Umfang der Datenbank zusätzlich vergrößern.

Das Benchmarkschema von TPC-H bildet die Geschäftsprozesse eines imaginären Han-

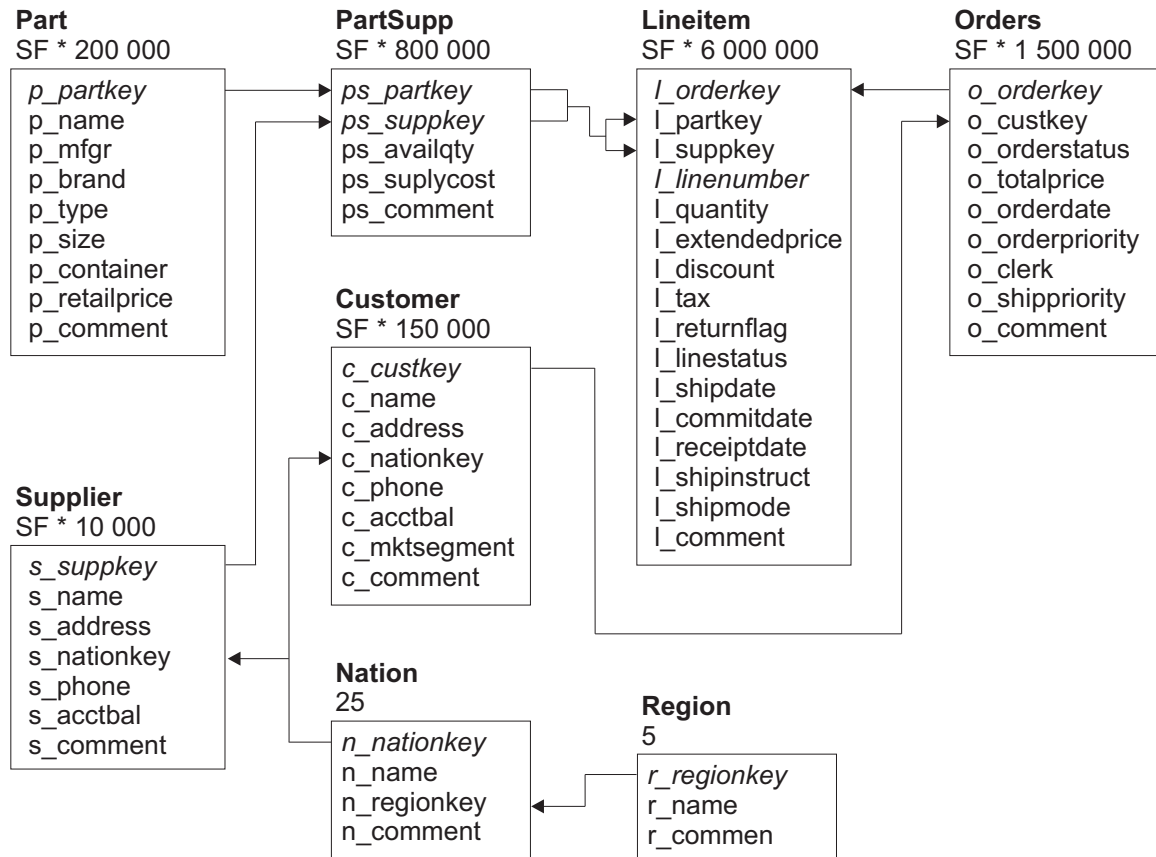


Abbildung 2.1: TPC-H Schema

delsunternehmen ab, das Waren (*Part*) von seinen Lieferanten (*Supplier*) bezieht. *PartSupp* gibt an welche Waren von welchem Lieferanten geliefert werden können, so wie deren Anzahl und Preis. Die Kunden (*Customer*) können Bestellungen (*Order*) aufgeben, welche wiederum aus mehreren Bestellposten (*Lineitem*) bestehen können. Sowohl Kunden als auch Lieferanten können unterschiedlicher Nationalität (*Nation*) sein, und sind somit über die 5 unterschiedlichen Kontinente (*Region*) aufgeteilt[s.Tra99].

Auf dieser Datenbasis werden 22 Decision-Support-Anfragen ausgeführt, die jeweils unterschiedliche geschäftsbezogene Anfragen für das zugrundeliegende Handelsunternehmen darstellen. Dies dient dazu, den betriebswirtschaftlichen Kontext zu verdeutlichen, in des-

2.2 Zugrundeliegende Datenbasis

sen Zusammenhang die Abfrage für ein beliebiges Unternehmen benutzt werden könnte. Über die 22 Anfragen hinaus gibt es noch zwei Refresh-Funktionen. Diese beiden Funktionen fügen durch INSERT-Befehle Daten dem Datenbestand hinzu, bzw. nehmen mittels DELETE-Befehlen Daten aus dem Datenbestand heraus. Es handelt sich bei all diesen Anfragen um sogenannte Ad-hoc Anfragen. Es wird also davon ausgegangen, dass der Benutzer nicht weiß, welche Anfragen an das Datenbanksystem gestellt werden und wie die Daten in der Datenbank abgespeichert sind. Die Anfragen dürfen also nicht mit Hilfe von Kenntnissen über den Datenbestand oder die Anfragerihenfolge optimiert werden. TPC-H definiert drei grundlegende Metriken, die dazu dienen, die getesteten Datenbanksysteme in 3 unterschiedlichen Bereichen in eine Leistungsskala einordnen zu können.

- Eine Anfragen-pro-Stunde (query-per-hour) Metrik (QphH@Size). Sie gibt an, wie viele Anfragen einer bestimmten Größe (@Size) pro Stunde erfolgreich verarbeitet werden können.
- Eine Preis-Leistungs (price-performance) Metrik ($\$/\text{QphH@Size}$ mit $\$$ =Gesamtpreis des Systems). Sie gibt das Preis-Leistungsverhältnis des Datenbanksystems an.
- Eine Systemverfügbarkeits (systems-availability-date) Metrik. Sie gibt an seit wann alle am Test beteiligten Systemkomponenten auf dem Markt frei verfügbar sind.

Diese Performance-Metriken und -Anfragen spielen für diese Arbeit weiter keine Rolle, und sind hier nur der Vollständigkeit halber aufgeführt. Diese Ausarbeitung benutzt nur die Datenbasis und das zugrundeliegende Datenbankschema des TPC-H-Benchmarks. Dieses Schema, bzw. diese Datenbasis wird noch, wie in der Einleitung bereits erwähnt, durch zusätzliche Tabellen, Sichten und deren Daten erweitert. Wie diese Erweiterungen genauer aussehen, und wie die vollständige Datenbasis zustande kommt wird im nächsten Abschnitt *Zugrundeliegende Datenbasis* behandelt. Auf der Internetseite des TPC (www.tpc.org) sowie in [Tra99] sind weitere Informationen zum TPC-H Benchmark zu finden.

2.2 Zugrundeliegende Datenbasis

Wie bereits in der Einleitung angesprochen wurde, handelt es sich bei der zu übertragenden Datenbasis nicht nur um das reine TPC-H Benchmark-Schema, sondern um ein Schema, das um einige Sichten und Tabellen erweitert wurde. Die zugrundeliegende Datenbasis wurde im Verlauf einer Studienarbeit mit dem Titel *Realisierung und Optimierung einer OLAP-Anwendung im Handelsbereich*[Wag99] im Rahmen des ORBIT Projekts erstellt. Ziel dieser Studienarbeit war die Erstellung einer typischen OLAP-Anwendung, um eine Analyse von Unternehmensdaten eines Handelsunternehmens zu ermöglichen. OLAP steht für *Online Analytical Processing* und ist eine analytische Informationssystematik, die mit Hilfe von aufgestellten Hypothesen(komplexen Anfragen), Aussagen über relevante betriebswirtschaftliche Fragen zu gewinnen versucht. Durch OLAP-Werkzeuge

2.2 Zugrundeliegende Datenbasis

kann eine zugrundeliegende Datenbasis benutzerfreundlich analysiert und abgefragt werden[MS0303],[Wag99]. Als Grundlage für die zu erstellende OLAP-Anwendung wurden im Rahmen der Studienarbeit mehrere Fragestellungen ausgearbeitet, die im Kontext eines Handelsunternehmens als geeignet empfunden wurden.

Da die reine Datenbasis des TPC-H-Benchmarks, mit seinen 8 Tabellen, nicht mit den DSS-Tools und den erarbeiteten Fragestellungen bearbeitet werden konnte, musste die TPC-H Datenbasis um 15 Tabellen und 31 Sichten (Views) erweitert werden. Die Views stehen für abgeänderte Tabellen des TPC-H-Schemas, und wurden aus Speicherplatz- und Redundanzgründen als Views auf den bestehenden 8 Tabellen realisiert[Wag99]. Auf Grund des häufigen Zugriffs auf die 15 anderen Tabellen, wurden diese in eigenständigen physischen Tabellen gespeichert, von denen keine mehr als 7000 Tupel besitzt, wodurch sie nicht viel Speicherplatz verbrauchen.

Mit Hilfe der durch die Erweiterungen zustande gekommenen endgültigen Datenbasis, wurde es auch möglich, die für die OLAP-Anwendung und die DSS-Tools generierten Fragestellungen zu bearbeiten. Die folgende Liste zeigt die Fragestellungen, wie sie in der Studienarbeit [Wag99] ausgearbeitet wurden. In Klammern sind die entsprechenden Attribute und Fakten des zugrundeliegenden Datenbankschemas angegeben, die in die Abfragen einbezogen werden müssen.

1. Welches sind die Top 25 Produkte (part), deren Stückzahlenabsatz (quantity) in bestimmten, noch festzulegenden Monaten (ordermonth) gegenüber dem jeweiligen Vormonat am stärksten angestiegen ist?
2. Wie verteilt sich der Stückzahlenabsatz (quantity) von bestimmten, noch festzulegenden Produkten (part) in bestimmten, noch festzulegenden Jahren (orderyear) auf die Länder der Kunden (custnation)?
3. Welches sind die Top 25 Produkte (part), deren in bestimmten, noch festzulegenden Monaten (ordermonth) bestellte Stückzahlen (quantity) am stärksten streuen, d.h. deren Standardabweichung am größten ist?
4. Wer sind die Top 25 Kunden (customer), die in den letzten drei Jahren (orderyear) einen jeweils zu wählenden Mindestumsatz (extendedprice, discount)gebracht haben und deren Standardabweichung des Umsatzes in dieser Zeit am geringsten ist?
5. Wer sind die Top 25 Kunden (customer), deren Reklamationsrate (returnflag, line-number) in bestimmten, noch festzulegenden Monaten (ordermonth) im Verhältnis zum jeweiligen Vormonat am stärksten angestiegen ist?
6. Welches sind die Top 25 Produkte (part), deren Differenz des durchschnittlichen Rabattes (discount) von bestimmten, noch festzulegenden Monaten (ordermonth) zum jeweiligen Vormonat am größten ist, und wie sieht die jeweilige Differenz des Stückzahlenabsatzes (quantity) zwischen den beiden Monaten aus?

2.2 Zugrundeliegende Datenbasis

Die Erweiterungstabellen, dies schließt auch die Tabellen die als Views realisiert sind mit ein, bilden ein Datenbankschema, das nach dem Snowflake-Schema aufgebaut ist. Bei dieser Art von Schema, sind die Tabellen um eine zentrale Faktentabelle (im vorliegenden Fall *lineitem_orders*) angeordnet, und bilden mehrdimensionale Abzweigungen, ausgehend von der zentralen Faktentabelle. Diese Anordnung ähnelt dem Aufbau eines Eiskristalls in einer Schneeflocke (Snowflake). In Abbildung 2.2 ist der Aufbau des Erweiterungsschemas im Star-Schema dargestellt. Um vom Star-Schema, in dem die Tabellen

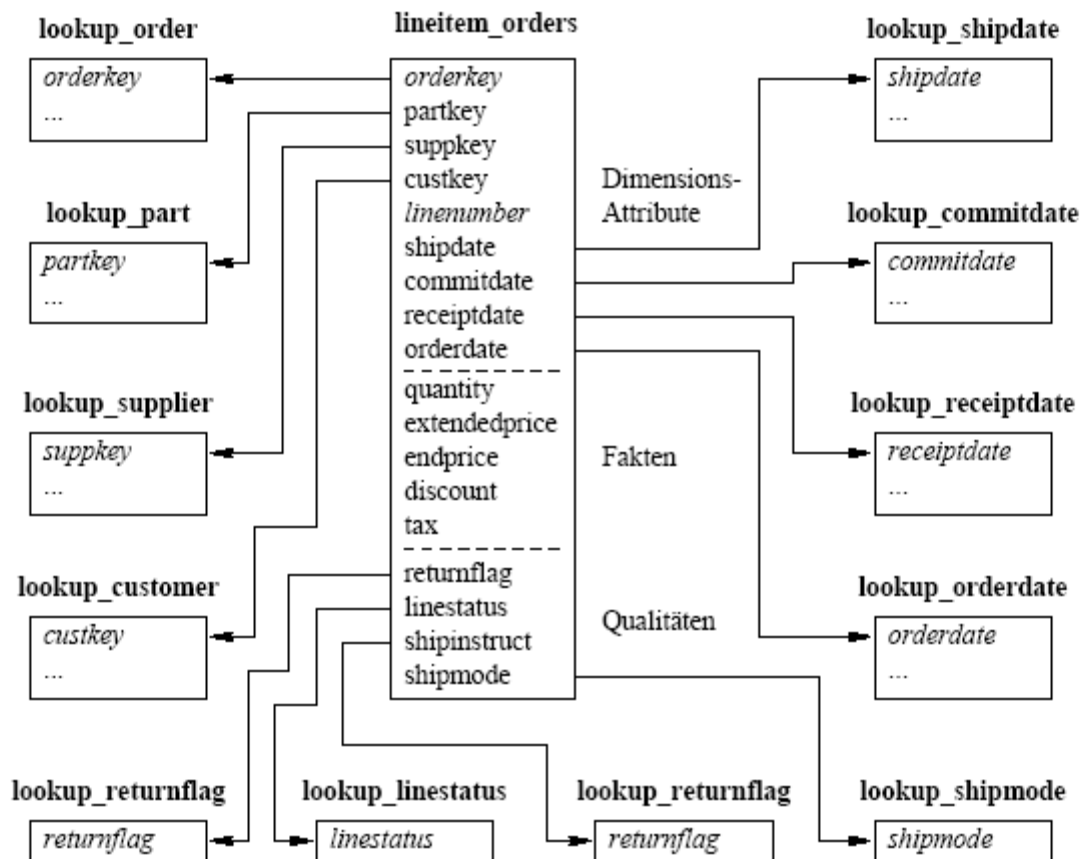


Abbildung 2.2: Erweitertes Schema

sternenförmig um die zentrale Faktentabelle angeordnet sind, zum Snowflake Schema zu kommen, müssen die Dimensionstabellen noch in ihre jeweiligen Hierarchien unterteilt werden. Jede der Dimensionstabellen die in Abbildung 2.2 um *lineitem_orders* angeordnet sind, ist in Hierarchiestufen unterteilt. In Abbildung 2.3 ist die weitere Unterteilung der Dimensionstabelle *lookup_part* dargestellt. Im verwendeten Snowflake-Schema wird *lookup_shipdate* durch die drei Tabellen *lookup_shipday*, *lookup_shipmonth* und *lookup_shipyear* wie in Abbildung 2.4 ersetzt. *lookup_commitdate*, *lookup_receiptdate*

2.2 Zugrundeliegende Datenbasis

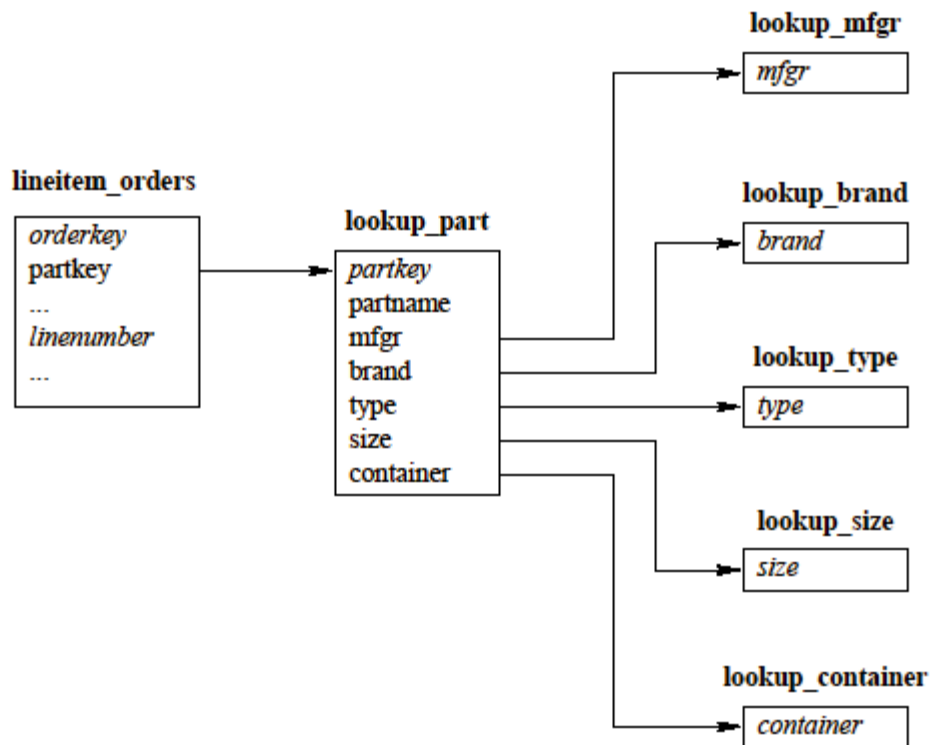


Abbildung 2.3: Hierarchien der Dimension Part

und lookup_orderdate werden ebenfalls durch die drei entsprechenden Tabellen (year, month, day) ersetzt, und bilden 12 der 15 eigenständigen Erweiterungstabellen. Dazu kommen die drei Tabellen relate_date, relate_month und relate_year, diese wurden aus der Tabelle *Orders* abgeleitet, und liefern zu einem bestimmten Bestelldatum, das entsprechende Bestelldatum, Bestelljahr oder den Bestellmonat des Vormonats oder des Vorjahres. Insgesamt umfasst die Datenbasis dieser Ausarbeitung 23 eigenständige, phy-

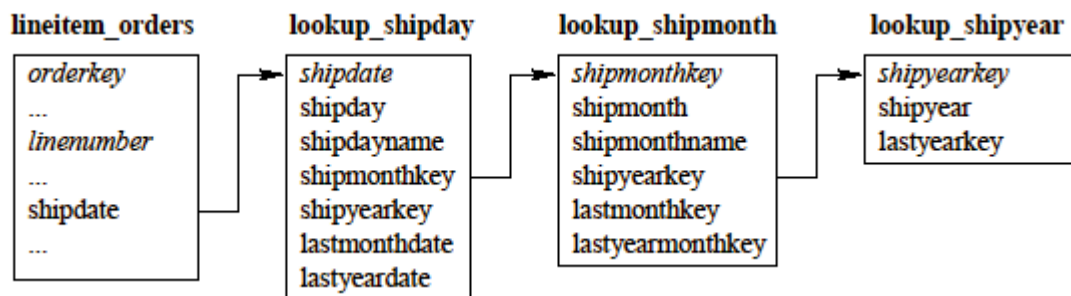


Abbildung 2.4: Hierarchien der Dimension lookup_shipdate

sische Tabellen, und 31 Tabellen, die als Views realisiert sind. Dazu kommen 12 Indizes auf den Tabellen.

2.3 Das Oracle-Datenbanksystem

In dieser Arbeit wird das Oracle® Database System 10g Release 1 (10.1) verwendet, es handelt sich dabei um ein relationales Datenbank Management System, das sich logisch in zwei Bereiche unterteilen lässt. Einerseits in die eigentlichen Datenbankinstanzen und andererseits in den Oracle Database Server, der die Datenbankinstanzen, also die eigentlichen Datenbanken verwaltet und Administrations- bzw. Managementwerkzeuge zur Verfügung stellt. Bei den Oracle-Datenbanken handelt es sich um relationale Datenbanken. In einer relationalen Datenbank werden alle Daten in zweidimensionalen Tabellen abgespeichert, die sich aus Zeilen und Spalten zusammensetzen. Die Oracle-Datenbanksoftware ermöglicht es, verschiedenste Daten in den Tabellen zu speichern und abzurufen. Die Datenbanken bestehen aber nicht nur aus den abgespeicherten Daten, sondern enthalten auch physikalische und logische Strukturen, in denen System-, Benutzer- und Kontrollinformationen abgespeichert werden.

Die folgende Liste gibt einen Überblick über verschiedene Werkzeuge, die Oracle anbietet um mit der Datenbank zu arbeiten. Danach folgt eine kurze Beschreibung der einzelnen Werkzeuge (s. auch [MB+03]).

- Oracle Universal Installer (OUI)
- Database Configuration Assistant (DBCA)
- Administration Assistant for Windows
- Database Upgrade Assistant
- Oracle Net Manager
- Ultra Search und Ultra Search Administration Tool
- Oracle Enterprise Manager 10g Database Control
- SQL*Plus und iSQL*Plus

2.3.1 Oracle Universal Installer (OUI)

Der Oracle Universal Installer ist das Installationswerkzeug, mit dessen Hilfe das ganze Oracle-System installiert werden kann. Er wird auch dazu verwendet, Zusatzsoftware zu installieren oder Installationsoptionen auszuwählen. Des Weiteren fungiert er auch als Deinstallationswerkzeug und ist somit auch für das Deinstallieren beliebiger Komponenten bzw. des ganzen Systems zuständig. Der OUI ist mit einem java-basierten grafischen Benutzerinterface (GUI) realisiert und ist in der Lage, automatisch den Database Configuration Assistant (s. nächster Abschnitt) aufzurufen, um am Ende der Installation

eine Datenbank zu erstellen. Darüber hinaus stellt der OUI auch eine Indexliste aller bereits installierten Oracle-Produkte zur Verfügung. Für eine detailliertere Beschreibung s. [Cho03].

2.3.2 Database Configuration Assistant (DBCA)

Der Database Configuration Assistant (Datenbankkonfigurationsassistenten) ist das Werkzeug zum Erstellen, Löschen und Modifizieren der eigentlichen Datenbanken. Dabei stehen mehrere von Oracle definierte Vorlagen (Templates) für das Anlegen einer neuen Datenbank zur Verfügung (General Purpose, Transaction Processing oder Data Warehouse). Der DBCA wird automatisch vom OUI nach der Installation der Oracle-Software aufgerufen, er kann aber auch im Nachhinein über folgenden Pfad manuell gestartet werden:

```
Start -> Programs -> Oracle - OraDb10g_home1 -> Configuration and  
Migration Tools -> Database Configuration Assistant
```

Eine genaue Bedienungsanleitung und Hilfe bei der Konfiguration einer Datenbank bietet [Cho03].

2.3.3 Administration Assistent for Windows

Der Oracle Administration Assistant for Windows ist ein Werkzeug, das dazu dient, administrative Aufgaben mit Hilfe eines grafischen Benutzerinterfaces im Windows Stil zu erleichtern. Der Administration Assistant ermöglicht die Konfiguration und Authentifizierung von Oracle-Datenbankadministratoren, Operatoren, Benutzern und Rollen durch das Windows-Betriebssystem. Durch diese Authentifizierung ist es möglich einen Betriebssystem-User mit einem Oracle-Datenbank-User gleich zu setzen. Dadurch entfällt die entsprechende Login- und Passwortabfrage beim Interagieren mit der Oracle-Datenbank für den entsprechenden Benutzer. Darüber hinaus bietet der Administration Assistent die Möglichkeit verschiedene Parameter des Oracle-Systems zu modifizieren. Er kann über folgenden Pfad gestartet werden:

```
Start -> Programs -> Oracle - OraDb10g_home1 -> Configuration and  
Migration Tools -> Administration Assistent for Windows
```

2.3.4 Database Upgrade Assistant

Der Database Upgrade Assistant kann dazu benutzt werden, ein bereits existierendes Oracle-Datenbanksystem auf eine neuere Version aufzupgraden. Der Upgrade Assistant wird automatisch gestartet wenn im OUI (s.o.) die Option Upgrade gewählt wird. Er kann aber auch, wie die anderen Tools, manuell gestartet werden:

2.3 Das Oracle-Datenbanksystem

Start -> Programs -> Oracle - OraDb10g_home1 -> Configuration and Migration Tools -> Database Upgrade Assistant

Für mehr Informationen s.[Mor03a].

2.3.5 Oracle Net Manager

Der Oracle Net Manager ist ein weiteres Oracle-Werkzeug mit grafischer Benutzerschnittstelle. Er ermöglicht die Konfiguration der Oracle Net Netzwerkkomponente. Oracle Net dient als Schnittstelle zwischen einer Clientanwendung und einem Oracle-Datenbankserver, und ist für den Verbindungsaufbau und die Überwachung einer Netzwerksession, genauso wie für den Nachrichtenaustausch zwischen den beiden Instanzen verantwortlich. Um all diese Aufgaben erfüllen zu können, muss Oracle Net auf jedem Client und Server im betreffenden Netzwerk installiert sein. Der Oracle Net Manager kann dann dazu eingesetzt werden die jeweiligen Computer zu konfigurieren. Dies kann sowohl vom Client wie auch vom Server aus geschehen. Der Oracle Net Manager ist auch im Oracle Enterprise Manager (s.u.) integriert und kann durch diesen aufgerufen werden. Eine weitere Möglichkeit den Net Manager aufzurufen ist:

Start -> Programs -> Oracle - OraDb10g_home1 -> Configuration and Migration Tools -> Net Manager

Eine detaillierte Anleitung zum Aufbau von Oracle Net und der Bedienung des Oracle Net Manager ist in [Pol03] zu finden.

2.3.6 Ultra Search und Ultra Search Administration Tool

Ultra Search ist ein Oracle-Werkzeug, das zum Suchen und Finden von Textinformationen in den verschiedensten Quellen eingesetzt werden kann. Die Suche kann unter anderem Oracle-Datenbanken, andere ODBC konforme Datenbanken, IMAP Mail Server, HTML Dokumente von einem Web Server und gespeicherte Dateien umfassen. Ultra Search benutzt einen sogenannten 'crawler' (Kriecher), der die angegebenen Quellen durchsucht und einen Index über die durchsuchten Informationen erstellt, und somit auf die Ergebnisse verweist, ohne die eigentlichen Daten zu verschieben. Dadurch ist Ultra Search in der Lage eine Art Suchportal zu erstellen, ohne die IT Topologie verändern zu müssen. Des Weiteren stellt Ultra Search verschiedene APIs zur Verfügung, um Content Management Lösungen zu erstellen. Ultra Search ist als J2EE Anwendung realisiert und kann mittels einer Webschnittstelle über jeden herkömmlichen Browser bedient werden.

Das Ultra Search Administration Tool ist ebenfalls eine J2EE Anwendung und dient zur Administration des Ultra Search crawlers, und ist als Java Server Page realisiert, die wie Ultra Search über einen Browser bedient werden kann. Das Administration Tool ist ein eigenständiges Programm und unabhängig von der eigentlichen Ultra Search Anwendung. Dadurch können die beiden Anwendungen auch auf unterschiedlichen Rechnern

betrieben werden. Weitere Information zum Ultra Search Tool und dem dazugehörigen Administrationstool ist in [Cyr03] zu finden.

2.3.7 Oracle Enterprise Manager 10g Database Control

Das Database Control (Datenbankverwaltungs-) Werkzeug ist ein Element des Oracle Enterprise Managers. Dieser Enterprise Manager ist nach der Installation und dem Erzeugen einer Datenbank, das zentrale Werkzeug um das komplette Oracle-Datenbanksystem zu verwalten und zu bedienen. Der Enterprise Manager umfasst eine Fülle von unterschiedlichen Werkzeugen und bietet somit einen Rahmen für die Verwaltung der gesamten System- und Hardwareumgebung.

Das Database Control Werkzeug ist die webbasierte Schnittstelle des Enterprise Managers und ermöglicht eine benutzerfreundliche Verwaltung des gesamten Datenbanksystems. Wie das Ultra Search Werkzeug (s.o.) ist auch das Enterprise Manager Database Control Werkzeug als J2EE Anwendung realisiert und mittels einer Web-Schnittstelle bedienbar. Durch diese Schnittstelle kann der Enterprise Manager auch remote durch einen beliebigen Browsers bedient werden und bietet so die Möglichkeit, auch räumlichverteilte Datenbanksysteme zentral über das Intra- oder Internet zu administrieren.

Wenn eine Datenbank während des Installationsvorgangs der Oracle-Software erstellt wird (s. Abschnitt 4.1 *Installation*), besteht die Möglichkeit, die Datenbank so zu konfigurieren, dass sie durch das Enterprise Manager 10g Grid Control Werkzeug oder durch das Database Control Werkzeug verwaltet werden soll. Das Grid Control Werkzeug ist ein weiterer Bestandteil des Enterprise Managers und bietet die Möglichkeit, ein gesamtes Rechensystem, mit allen Hosts, Datenbanken, Listenern, Application Servern, HTTP Servern und Webanwendungen zu verwalten. Dabei handelt es sich um Zusatzsoftware, die getrennt installiert werden muss. Da das Ziel dieser Studienarbeit das Übertragen der DB2 Datenbank in Oracle ist, und es sich bei dieser Datenbank um eine einzelne Datenbank handelt, die sich nur auf einem Rechner befindet, ist die Option des Enterprise Manager Grid Control hier unnötig, und es wird nicht weiter darauf eingegangen, sondern die Database Control Variante benutzt. Für weitere Informationen zum Thema Oracle Enterprise Manager Grid Control s. [LaQ03],[Gos03].

Jede Version der Oracle Database 10g Software enthält das Database Control-Werkzeug das bei jeder Installation automatisch mit installiert wird. Die Konfiguration des Werkzeugs ist während oder nach der Installation möglich. Standardmäßig wird die Database Control automatisch mit jedem Systemstart gestartet, das Programm kann aber auch manuell gestartet oder gestoppt werden.

Start der Database Control mit folgendem Befehl:

```
ORACLE_HOME\bin> emctl start dbconsole
```

Stoppen der Database Control mit folgendem Befehl:

2.3 Das Oracle-Datenbanksystem

```
ORACLE_HOME\bin> emctl stop dbconsole
```

Da in der vorliegenden Arbeit nur die SQL-Schnittstelle des Oracle-Systems benutzt wird und die Bedienungsanleitung des Enterprise Managers nicht sehr kompliziert aber sehr umfangreich ist, wird hier nicht weiter darauf eingegangen. Viele der im weiteren Verlauf dieser Arbeit durchgeführten Aktionen, wie zum Beispiel dem Erstellen von Tabellen, Sichten und Indizes, können auch mittels der Enterprise Manager Database Control durchgeführt werden. Wer die grafische Oberfläche des Enterprise Managers der SQL-Schnittstelle vorzieht, kann auch das Database Control Werkzeug für die entsprechenden Aktionen verwenden. Weitere Informationen zum Enterprise Manager gibt es in [LaQ03],[Gos03],[MB+03]. Ein Onlinekurs zur Bedienung des Enterprise Managers ist auf folgender Internetseite zu finden:

http://www.oracle.com/technology/obe/2day_dba/index.html

2.3.8 SQL*Plus und iSQL*Plus

Bei SQL*Plus bzw. iSQL*Plus handelt es sich um die SQL-Schnittstellenwerkzeuge die von Oracle angeboten werden. Diese Studienarbeit macht regen Gebrauch von der SQL-Schnittstelle, weshalb diese auch ausführlicher als die anderen Datenbankwerkzeuge beschrieben wird.

Oracle bietet drei unterschiedliche Benutzeroberflächen für die SQL-Schnittstelle an. Eine Kommandozeilen Oberfläche im Stil von DOS, eine Oberfläche im Windowsstil und dann noch die iSQL*Plus Oberfläche. iSQL*Plus ist ein webbasiertes Userinterface und bietet den meisten Komfort der drei unterschiedlichen Oberflächen. Das Webinterface ist wie das Enterprise Manager Database Control Werkzeug oder das Ultra Search Werkzeug, eine J2EE Anwendung, die als Java Server Page realisiert ist. Dadurch ist iSQL*Plus über eine URL (s. Kapitel 4.1 *Installation*) mit einem beliebigen javafähigen Browser bedienbar. Um iSQL*Plus zu starten müssen folgende Schritte ausgeführt werden:

1. Eingabe der iSQL*Plus URL in den Webbrowser (s.Kapitel 4.1 *Installation*)
2. Drücken der Eingabetaste, danach erscheint der Login Bildschirm
3. Oracle-Datenbank Benutzername und Passwort angeben
4. Den Connection Identifier frei lassen um zur Standarddatenbank zu gelangen, andernfalls Datenbank-Identifizier angeben
5. Login Icon betätigen

Um einen SQL-Befehl in iSQL*Plus ausführen zu lassen, muss dieser in das SQL-Fenster eingetragen werden und auf den Knopf execute (ausführen) gedrückt werden. Des weiteren bietet iSQL*Plus auch die Knöpfe Load Script, Save Script, Cancel und Clear. Load Script ermöglicht es, über ein Browserfenster eine zu ladende Skriptdatei auszuwählen und in das SQL-Fenster zu laden. Save Script ermöglicht das Abspeichern der Kommandos im SQL-Fenster in eine Skriptdatei. Cancel stoppt eine Ausführung und

2.3 Das Oracle-Datenbanksystem

Clear löscht das SQL-Fenster. Leider hatte das für diese Arbeit installierte iSQL*Plus noch einige Bugs, so konnte ein Skript mit dem Load Script Knopf zwar geladen werden, aber nach dem Drücken auf den Execute Knopf wurde ein Java-Page-Fault ausgegeben. Wurde das entsprechende Skript mittels Copy-and-Paste aus einem Editor in das SQL-Fenster kopiert, wurde das Skript fehlerfrei ausgeführt. Darüber hinaus, sind bestimmte Befehle (wie z.B.: GET,SAVE,SPOOL,...) in iSQL*Plus nicht verfügbar und bestimmte Systemoperationen sind aus Sicherheitsgründen nicht ausführbar. Auf Grund dieser Einschränkungen wurde iSQL*Plus in dieser Arbeit nicht verwendet, sondern auf die beiden anderen SQL*Plus Oberflächen zurück gegriffen. Hauptsächlich wurde die Windowsoberfläche der SQL-Schnittstelle verwendet, da diese gegenüber der Kommandozeilenoberfläche etwas mehr Komfort bietet.

Um die Kommandozeilenoberfläche zu starten sind folgende Schritte auszuführen:

1. Öffnen eines Dosfensters (cmd.exe) und Eingabe des Befehls *sqlplus*
2. Eingabe des Oracle-Datenbank Benutzernamens und Passworts
3. Alternativ dazu kann auch *sqlplus benutzername/passwort* eingegeben werden oder *sqlplus benutzername* um das Passwort zu verbergen
4. Um eine andere als die Standarddatenbank auszuwählen, kann folgender Befehl eingegeben werden:

```
sqlplus benutzername/passwort@connect_identifizier
```

Um die Windowsoberfläche zu starten sind folgende Schritte notwendig:

1. Ausführen von:

```
Start-> Programs-> Oracle-OraHomeName->  
Application Development-> SQL Plus
```
2. Eingabe des Oracle-Datenbank Benutzernamens und Passworts
3. Alternativ dazu kann auch ein Dosfenster geöffnet werden und *sqlplusw* eingegeben werden
4. Die Windowsoberfläche öffnet sich und die Login Informationen müssen eingegeben werden (s.o.)

Der Umgang und die eigentliche Eingabe der SQL-Befehle sollte für jemandem mit SQL-Kenntnissen kein Problem darstellen. Da es aber auch SQL*Plus spezifische Kommandos gibt und die Syntaxstellung mancher SQL-Befehle vom Standard abweicht, ist eine Eingewöhnung an das Oracle-SQL unumgänglich. Es folgt eine kleine Aufstellung an Befehlen, die im Verlauf dieser Arbeit als nützlich empfunden wurden.

Eine komplette Liste der SQL*Plus und Oracle-SQL-Befehle, sowie detailliertere Informationen zur Bedienung und Konfiguration der drei unterschiedlichen SQL*Plus Oberflächen sind in [OSQ03] und in [Wat03] zu finden.

Informationen aus bzw. über Datenbanken in SQL*Plus im Oracle-System:

2.3 Das Oracle-Datenbanksystem

Kommando	Erklärung
START (kurz: @)	Führt ein SQL-Skript aus
DESC[RIBE]	Gibt den Aufbau einer Tabelle, eines Views oder eines anderen Objekts an
EXIT/QUIT	Beendet SQL*Plus
HELP	Zeigt die Kommandozeilen Hilfe an
SPOOL	Speicher die Kommandozeilenausgabe in einer Datei
SELECT global_name FROM global_name	hieraus erhält man den DB-Namen
SELECT table_name FROM user_tables	hieraus erhält man alle Tabellennamen der DB *
SELECT view_name FROM user_views	hieraus erhält man alle Viewnamen *
	<i>*auf die der User (mit dem man eingelogged ist) Zugriff hat</i>

Tabelle 2.1: SQL*Plus Kommandos

3 Vergleich der Datenübertragungswerkzeuge

Das Oracle-Datenbanksystem stellt unterschiedliche Datenübertragungswerkzeuge bzw. Datenimportwerkzeuge zur Verfügung. Jedes dieser Werkzeuge ist mehr oder minder für einen bestimmten Aufgabenbereich ausgelegt, dennoch lassen sich für die gleiche Art von Import oft verschiedene dieser Tools verwenden. Je nach Aufgabenstellung und Zielsetzung des Importauftrags, muss individuell ausgewählt werden, welches Werkzeug benutzt werden soll. Ein entsprechendes Datenübertragungswerkzeug wird anhand von Kriterien gewählt, wie z.B.: wie oft der Import durchgeführt werden muss (einmal oder mehrmals), wie groß die zu übertragende Datenmenge ist (kleine, große oder sehr große Datenmenge), oder wie zeitkritisch das Laden der entsprechenden Daten ist. Die unterschiedlichen Tools sind:

- Das ursprüngliche Import/Export-Werkzeug von Oracle
- Oracle Data Pump (gibt es erst seit Oracle 10g)
- Oracle SQL*Loader
- Transportable Tablespaces
- Transparent Gateways von Oracle

3.1 Originalimport- und Originalexportwerkzeug

Das in diesem Abschnitt behandelte Import-Werkzeug ist das ursprüngliche Import- bzw. Exportwerkzeug von Oracle, wie es bisher in den früheren Versionen (bis einschließlich 9i) der Oracle-Datenbank-Software angeboten wurde. Um die alten Werkzeuge von dem in der Version 10g neu hinzugekommenen Data Pump Import bzw. Export (s. nächster Abschnitt) abzugrenzen, werden diese als „ursprüngliches“ oder „Original-“ Export- und Importwerkzeug bezeichnet. Die Originalwerkzeuge werden mit dem Befehl *exp* bzw. *imp* ausgeführt. Sie bieten eine einfache Methode Daten zwischen unterschiedlichen Oracle-Datenbanken auszutauschen. Zu erst muss das Exportwerkzeug dazu benutzt werden, die zu übertragenden Daten aus der Ausgangsdatenbank zu exportieren. Dies wird erreicht, indem die Datenbankobjekte (z.B. Tabellen, Indizes, Views,...) extrahiert werden und in sogenannten 'Dump Files' (Zwischenlager-Dateien) gespeichert werden. Das Importwerkzeug ist dann in der Lage, das Export-Dump File in die Zieldatenbank einzulesen.

3.2 Data Pump

Das Dump File ist in einem binären Oracle-Format abgespeichert und muss auf einem Speichermedium zwischengespeichert werden. Die dump files müssen dann auf irgend einem Wege von der Ausgangsdatenbank zur Zieldatenbank transferiert werden. Dies kann sowohl über Netzwerk als auch über den physikalischen Transport auf einem Datenträger geschehen. Die exportierten Dump Files können nur von einem Oracle-Importwerkzeug gelesen werden, dessen Version nicht älter als das des Exportwerkzeuges ist. Um nur die Daten des Dump Files anzeigen zu lassen, ohne einen eigentlichen Import durchzuführen, wird ebenfalls das Oracle-Importwerkzeug benutzt. Hierfür muss der Import mit dem Befehl *imp SHOW* aufgerufen werden. Weiterführende Informationen sind unter [Ric03] zu finden.

3.2 Data Pump

In diesem Abschnitt wird das nächste Export- bzw. Importwerkzeug von Oracle vorgestellt. Das Tool nennt sich Oracle Data Pump und es handelt sich dabei um eine neuere Version des im vorherigen Abschnitt dargestellten Originalexport- und Importwerkzeugs. Oracle Data Pump ist eine Neuentwicklung von Oracle, die erst seit der Version 10g zur Verfügung steht.

Die Datenmigration lässt sich bei Oracle Data Pump in drei Bereiche unterteilen:

1. Der Kommandozeilenaufruf
2. Das Data Pump API
3. Das Metadata API

Mit dem Kommandozeilenaufruf lassen sich ähnlich wie bei den Originalwerkzeugen, das Data Pump Import Werkzeug mit *impdp* und das Data Pump Export Werkzeug mit *expdp* aufrufen. Ebenso wie im vorigen Abschnitt, wird ein Dump File vom Exportwerkzeug erzeugt und dieses vom Importwerkzeug benutzt um die Daten in die Zieldatenbank zu laden. Das Export- und das Importwerkzeug verwenden Prozeduren des Data Pump oder Metadata API ,die das Exportieren und Importieren von Daten und Metadaten ermöglichen. Dabei ist das Metadata API auch dafür verantwortlich, die Metadaten in der Zieldatenbank wieder richtig zusammen zu führen.

Das Exportieren und Importieren der Daten wird bei Data Pump durch Optimierungsverfahren beschleunigt, weshalb der Export mindestens doppelt, der Import sogar 15-45 Mal so schnell wie bei den Originalwerkzeugen sein kann. Dies gilt allerdings nur für sehr große Datenmengen, da die Initialisierung der Optimierungsverfahren einige Zeit in Anspruch nimmt und so bei einer kurzen Übertragungsdauer kaum zeitliche Gewinne erreicht werden können. Darüber hinaus ist die Verarbeitung von Metadaten nicht viel schneller als bei den alten Export- und Importwerkzeugen, der Geschwindigkeitsvorteil ergibt sich also nur aus der Verarbeitung der Rohdaten. Die Verarbeitung der zu exportierenden Daten geschieht auf dem Ausgangssystem, wofür entsprechend viel Speicher zur

3.3 SQL*Loader

Verfügung stehen sollte, außerdem wird im Gegensatz zum Originalexport bzw. -Import, standardmäßig eine Logdatei erzeugt. Zum Bedienen des Export- und Importwerkzeugs kann neben der Kommandozeilenoberfläche auch die webbasierte Schnittstelle des Oracle Enterprise Managers benutzt werden [Gos03].

Alles in allem ist das neue Data Pump Werkzeug entwickelt worden, um die herkömmlichen Werkzeuge zu ersetzen. Da laut Oracle, Data Pump in fast allen Fällen schneller ist als das ursprüngliche Tool und Data Pump alle Optionen des früheren Exports und Imports unterstützt, spricht ab der Verwendung der Version 10g der Oracle-Software nichts mehr für den Einsatz der alten Werkzeuge.

3.3 SQL*Loader

Der Oracle SQL*Loader ist ein weiteres Importwerkzeug, um Daten in eine Oracle-Datenbank zu laden. Ähnlich wie bei Data Pump oder dem Originalimportwerkzeug, kann SQL*Loader Daten aus einem Dump File in eine Zieldatenbank laden. Im Unterschied zu den vorherigen Werkzeugen, ist SQL*Loader dazu in der Lage, Daten aus Dump Files zu extrahieren, die in den verschiedensten Formaten abgespeichert sind. Dadurch ist der SQL*Loader als einziges der von Oracle zur Verfügung gestellten Importwerkzeuge fähig, direkt Dump Files als Importquelle zu benutzen, die von Nicht-Oracle-Werkzeugen (z.B. DB2 Export) erstellt wurden. Auf Grund dieser Vielseitigkeit, besitzt SQL*Loader auch kein eigenes Exportwerkzeug, da das Abspeichern der Daten in einem bestimmten Format unnötig ist.

Um einen erfolgreichen Import zu ermöglichen, muss bekannt sein, wie die Daten im Dump File abgespeichert sind. Es muss also bekannt sein, wie die einzelnen Datentupel, Tabellenzeilen und -spalten voneinander abgegrenzt sind (z.B. durch Komata, Anführungszeichen, Tabs oder ähnliches). Diese Informationen müssen dem SQL*Loader beim jeweiligen Importvorgang, als Ladeoptionen mitgegeben werden, da die Datentupel nicht automatisch an die vorgesehenen Positionen in der Oracle-Zieldatenbank geladen werden können. Welche Dump File-Formate SQL*Loader unterstützt und wie die entsprechenden Ladeoptionen auszusehen haben, wird in [Ric03] behandelt.

Der SQL*Loader kann durch den Befehl *sqlldr* gefolgt von den Ladeoptionen und der Angabe des Dump Files ausgeführt werden. Damit dieser Kommandozeilenaufruf nicht zu lang und zu komplex wird, können diese Angaben auch in einem sogenannten 'Control File' (Kontrolldatei) abgespeichert werden. Dadurch muss dem *sqlldr*-Befehl nur der Speicherpfad der Kontrolldatei angegeben werden. Ein Aufruf des SQL*Loaders erzeugt standardmäßig ein 'log file' (Protokolldatei), das als Hilfe zur Fehlerkorrektur dienen kann. Insofern Fehler beim Laden auftreten, werden auch noch zwei andere Dateien angelegt, ein 'bad file' und ein 'discard file'. Nähere Angaben zu diesen drei Dateien ist in Kapitel 4.2 dieser Arbeit oder in [Ric03] zu finden.

Das SQL*Loader Werkzeug verfügt über zwei unterschiedliche Modi, um die Daten in

die Zieldatei zu laden. Der sogenannte 'conventional path load' Modus ist sehr flexibel und benutzt normale SQL-INSERT-Befehle um die Daten in die vorgesehenen Tabellen zu importieren. Der zweite, sogenannte 'direct path load' Modus ist nicht so universell einsetzbar wie der erste Modus, aber dafür erreicht der 'direct path load' eine höhere Ladegeschwindigkeit. Dies wird dadurch erreicht, dass die Daten direkt in die Oracle-Dateien geschrieben werden, ohne auf INSERT-Befehle zurück zugreifen. Für welche Importaufträge der 'direct path load' Modus einsetzbar ist und welche Ladeoptionen für den jeweiligen Modus angegeben werden müssen, ist ebenfalls unter [Ric03] zu finden.

3.4 Transportable Tablespaces

Die Transportable Tablespaces von Oracle sind kein Export- oder Importwerkzeug im eigentlichen Sinne, vielmehr handelt es sich hierbei um eine Funktionalität, die Oracle für seine Datenbanken zur Verfügung stellt. Transportable Tablespaces, also bewegbare Tablespaces, bedeutet nichts anderes, als dass ganze Tablespace-Dateien von einer Oracle-Datenbank in eine andere Oracle-Datenbank übertragen werden.

Seit der Version 9i des Oracle-Datenbanksystems, können diese transportierten Tablespaces auch eine unterschiedliche Blockgröße wie das Zielsystem besitzen. Dadurch ist es möglich, ohne große Umwandlungen einen Tablespace über Plattformgrenzen hinweg zu verschieben, also zum Beispiel von einem Linux System auf einer Sun Workstation auf ein Windows System auf einem Intel Pentium, hierzu muss aber ein Teil des Kopiervorgangs mit Hilfe der Data Pump-Funktionalität durchgeführt werden. Dabei ist das Verschieben der Transportable Tablespaces viel schneller, als die gleiche Operation mit Export- und Import (sogar mit Data Pump) durchzuführen. Das liegt daran, dass nur die Metadaten, in das neue System importiert werden müssen, die restlichen Daten im Tablespace können durch einen einfacheren Kopiervorgang in das neue System 'eingeklinkt' werden. Dieser Vorgang ist um einiges schneller als die konventionelle Verwendung von Data Pump. Eine genau Anleitung für die Verwendung von Transportable Tablespaces ist in Kapitel 8 des [Bay03] zu finden.

3.5 Transparent Gateways

Auch die Transparent Gateways, sind kein Export- bzw. Importwerkzeug an sich, sondern stellen ihre Funktionalität anderen Werkzeugen zur Verfügung. Sie werden dazu benutzt, um auf Daten von nicht-Oracle-Datenquellen zugreifen zu können, als ob es sich dabei um Oracle-Datenbanken handeln würde. Die Daten müssen also nicht über eine externe Datei exportiert bzw. importiert werden, sondern können per Datenbankabfrage wie eine Oracle-Datenbank gehandhabt werden. Es soll nicht erkennbar sein, wo die Oracle-Datenbank aufhört und die Datenquelle anfängt. Der Übergang soll also transparent sein, was in diesem Zusammenhang unsichtbar bedeutet. Ziel der Transparent

3.6 Ermittlung des passenden Migrationswerkzeugs

Gateways von Oracle ist es sogenannte föderative Datenbanksystem aufzubauen, die aus den unterschiedlichsten Datenbanken von unterschiedlichen Datenbankanbietern bestehen können, aber nach außen hin als einheitliche Oracle-Datenbank fungieren. Bei den Datenquellen kann es sich um Datenbanken anderer Hersteller, aber auch um nichtrelationale Datenquellen wie Dateisysteme handeln.

Um die Unterschiede zwischen der benutzten Datenquelle und dem Oracle-System zu überbrücken, muss das entsprechende Transparent Gateway genau auf diese Datenquelle und deren Eigenschaften zugeschnitten sein. Es muss sich also um maßgeschneiderte, Ende-zu-Ende Zugänge (Gateways) handeln. Dadurch sind die jeweiligen Lösungen auch wirklich nur für bestimmte Datenquellen verwendbar. Weitere Informationen zu Transparent Gateways, speziell für DRDA Datenbanken sind unter [OTG03] zu finden.

3.6 Ermittlung des passenden Migrationswerkzeugs

Um aus den vorgestellten Werkzeugen, das Passende für die Migration der vorliegenden DB2-Datenbank zu wählen, müssen die Anforderungen dieser Migration betrachtet werden.

1. Die zu exportierende DB2-Datenbank hat eine Größe von über 10GB, was eine erhebliche Datenmenge darstellt. Dadurch sollte ein Werkzeug gewählt werden, dass für sehr große Datenmengen geeignet ist.
2. Da es sich bei der Ausgangsdatenbank um eine DB2-Datenbank, also um eine nicht-Oracle-Datenbank handelt, muss ein Werkzeug benutzt werden, dass auch Daten aus einem Oracle-fremden Format importieren kann.
3. Die vorgegebene Migration ist ein einmaliger Vorgang, der zwar wiederholbar sein soll, aber nicht mehrmals von der gleichen Ausgangs- zur gleichen Zieldatenbank durchgeführt werden soll. Dadurch werden Werkzeuge bevorzugt, die einen möglichst geringen Initialisierungsaufwand erfordern.

Die 1. Anforderung spricht vor allem für Data Pump und die Transportable Tablespaces, da diese einen schnellen Import von großen Datenmengen ermöglichen. Diese Anforderung schließt die anderen Werkzeuge allerdings nicht aus, da diese den entsprechenden Import ebenfalls durchführen könnten, auch wenn sie mehr Zeit dafür benötigen würden. Die 2. Anforderung ist ein Ausscheidungskriterium für fast alle der vorgestellten Werkzeuge, da SQL*Loader das einzige Importwerkzeug ist, dass auch Dump Files in Oracle-fremden Formaten benutzen kann. Mit Hilfe der Transparent Gateways, ist es aber möglich, dass auch Data Pump und die Originalwerkzeuge auf die DB2-Datenbank zugreifen können. Auf die Transportable Tablespaces ist das aber nicht anwendbar, wodurch diese für die Migration nicht in Frage kommen. Die Transparent Gateways ermöglichen es auch, ohne Verwendung eines der Importwerkzeuge auf die entfernten Daten zuzugreifen. Dadurch können die Daten direkt per Anfrage mit INSERT-Befehlen aus dem Quellsys-

3.6 Ermittlung des passenden Migrationswerkzeugs

tem in Tabellen des Zielsystems kopiert werden.

Da Oracle dazu rät, im Allgemeinen Data Pump den original Werkzeugen vorzuziehen (s.Kapitel 20[Ric03]), wird hier nur noch darüber diskutiert, ob Data Pump oder der SQL*Loader für die vorgegebene Migration eingesetzt werden sollen. Die 3. Anforderung spricht im ersten Moment, eher für das Data Pump Werkzeug, da die Importe hier nicht sonderlich viele Optionen erfordern und größten Teils automatisch ablaufen. Da die Ausgangsdatenbank eine DB2-Datenbank ist, müsste die Migration mit Hilfe der Transparent Gateways zwischen Oracle und DB2 realisiert werden (Oracle Transparent Gateway for DB2). Da die Gateway-Software nicht im Lieferumfang der installierten Oracle-Version enthalten ist, und nach Literaturrecherche (s.[OTG03]) festgestellt wurde, dass die Einrichtung des Gateways nicht ganz trivial ist, wurde für den in dieser Arbeit vorgegebenen Migrationsvorgang der Oracle SQL*Loader verwendet.

4 Migration

In diesem Kapitel wird das Oracle-Datenbanksystem auf einem Rechner der Abteilung Anwender Systeme des IPVS installiert. Danach werden sämtliche Tabellen (table), Sichten (view) und Indizes (index) des TPC-H-Benchmarks sowie die durch das ORBIT-Projekt hinzugefügten Erweiterungen (s. Einleitung) auf das neuinstallierte Oracle-System übertragen. Als Grundlage für die Migration wird das bestehende DB2 System benutzt, auf dem auch die ORBIT-Erweiterungen bereits vorhanden sind.

4.1 Installation

Die Installation des Oracle-Datenbanksystems Version 10.0.1g erfolgte auf einem AMD Athlon MP 1800+ mit 2 Prozessoren und 1 GB Arbeitsspeicher. Zunächst war der Rechner mit einer 40 GB Festplatte ausgerüstet. Aus Platzgründen wurde im späteren Verlauf zusätzlich eine 130 GB Festplatte nachgerüstet. Der Name bzw. die Adresse des Rechners lautet: <http://aspc2.informatik.uni-suttgart.de> Leider hat es beim ersten Versuch der Installation einige Probleme mit der neuen Oracle-Software gegeben. Diese war zum Zeitpunkt der Installation nur auf einem englischsprachigen Windows-System fehlerfrei zu installieren. Es handelt sich dabei um den Oracle-Fehlercode 3388817. Dieser Fehler wird durch den ORADIM -NEW Befehl erzeugt, welcher versucht einen neuen Windows System Account für die Gruppe ORA_DBA einzurichten. Der ORADIM Befehl sucht dabei den Account mit dem Namen 'NT AUTHORITY/SYSTEM'. Dieser Account ist aber in jeder lokalisierten Windows Version anders benannt (im Deutschen 'NT Autorität/System', im Französischen 'AUTORITE NT/SYSTEM'). Dadurch tritt der Fehler bei der Installation auf jedem Windows System auf, nur nicht in der englischen Originalversion. Deshalb wurde das deutschsprachige Windows System auf dem vorliegenden Rechner deinstalliert und durch ein englischsprachiges Windows 2000 Betriebssystem ersetzt. Die erneute Installation auf dem englischen System lief dann fehlerfrei ab.

Vor der Installation sollte auf die Mindestanforderungen von Oracle 10g geachtet werden. Diese werden von der Oracle-Installationsroutine automatisch überprüft und falls diese nicht erfüllt werden, wird die Installation automatisch abgebrochen. Laut Oracle benötigt die Windows Version des Oracle-Datenbanksystem 10g folgende Basisanforderungen s.[Sim04]:

- RAM: Minimum von 256 MB , empfohlen 512 MB
- virtueller Arbeitsspeicher: Zweifache des oberen Werts

4.1 Installation

- Festplattenspeicher: zwischen 100 MB und 1,4 GB für die Standard Edition
- Temp-Speicher: 100 MB
- Videoadapter: 256 Farben
- Prozessor: mindestens 200 MHz

Da das vorliegende System diese Anforderungen mit Leichtigkeit erfüllt, konnte die Installation ohne Probleme durchgeführt werden. Für die Installation wurde der Oracle Universal Installer benutzt (s. Abschnitt 2.3). Die Installation des Oracle-Systems ist selbsterklärend und es wird hier nicht weiter auf den genauen Ablauf der Installation sondern nur auf einige Punkte wie den Speicherort oder die Datenbankkonfiguration eingegangen. Eine detaillierte Installationsanleitung ist in [MB+03] zu finden.

In dieser Arbeit wurde die Standard Edition des Oracle-Datenbanksystems verwendet. Die Installation benötigte 1.1 GB der vorhandenen Festplatte und wurde auf dem ersten Laufwerk in die Partition C: installiert. Das Oracle Home Verzeichnis befindet sich in folgendem Speicherpfad:

- Oracle Home Location: C:\Programme\oracle\product\10.1.0\Db_1

Unter diesem Speicherpfad wurden die meisten bei der Installation ausgewählten Komponenten installiert. Die Datenbankkonfigurationsdateien sind in folgendem Speicherpfad zu finden:

- Oracle Home Location: C:\Programme\oracle\product\10.1.0

Es sollte darauf geachtet werden, keine dieser Konfigurationsdateien zu löschen, ansonsten könnte eine Neuinstallation des gesamten Oracle-Systems von Nöten sein.

Im Verlauf der Installation kann entschieden werden, nur die Oracle-Software zu installieren und keine Datenbank anzulegen. Dadurch wird nur die Datenbankverwaltungssoftware von Oracle installiert, aber noch keine Datenbank auf der diese angewandt werden kann. Es ist aber immer noch möglich, nach Beendigung der Installation, eine Datenbank mit Hilfe des Database Configuration Assistant (s. Abschnitt 2.3) anzulegen. Da zum Installationszeitpunkt bereits feststand, dass eine Datenbank angelegt werden soll, wurde direkt in der Installation die Option wahrgenommen eine Datenbank anzulegen. Für die in dieser Arbeit benötigte Datenbank wurde der vorkonfigurierte Datenbankentyp Typ Data Warehouse ausgewählt. Dadurch wird die Datenbank für den Verwendungszweck Data Warehouse optimiert. Für die Datenbank müssen dann noch folgende Optionen angegeben werden:

- Global Database Name: tpch.aspc2.informatik.uni-stuttgart.de
- Oracle System Identifier (SID): tpch
- Datenbank-Passwort: tpchadmin
- File System : Use Oracle-Managed Files
- Speicherpfad der Datenbankdateien: D:\oradata

4.1 Installation

- Specify Flash Recovery Area: ORACLE_BASE\flash_recovery_area
- Flash Recovery Area Size: 2048
- Typical Percentage: 80

Der Global Database Name (globaler Datenbankname) gibt den vollen Namen einer Datenbank an. Er setzt sich aus dem Datenbanknamen und der Datenbank-Domain zusammen. Dadurch wird die Datenbank eindeutig identifiziert und von anderen Datenbanken unterschieden.

Der Oracle System Identifier oder kurz SID identifiziert eine spezielle Instanz einer laufenden Oracle-Datenbank. Das heißt, sobald zwei oder mehr Instanzen der gleichen Datenbank, mit dem gleichen Datenbanknamen, in Betrieb sind, sorgt der SID für eine eindeutige Identifizierung der jeweiligen Instanz. Da die zu installierende Datenbank nur über eine Instanz verfügt, unterscheiden sich in diesem Fall der Oracle System Identifier und der Datenbankname nicht.

Das Datenbank-Passwort steht für das Administratorenpasswort der zu installierenden Datenbank. Dies wird dem System-Account SYSTEM zugewiesen. Es kann für jeden USER der Datenbank ein eigenes Passwort spezifiziert werden. In dieser konkreten Installation wurde der Einfachheit halber die Option 'Use the same Password for All Accounts' gewählt. Dadurch erhält jeder Account bzw. User der Datenbank automatisch das Passwort 'tpchadmin'. Nach der Installation kann ein Passwort eines Benutzers mit dem folgenden SQL-Befehl geändert werden.

```
password <username>
  Changing password for <username>
  Old password:
  New password:
  Retype new password:
```

Im Laufe der Installation wurden die folgenden in Kapitel 2.3 erläuterten J2EE Anwendungen installiert und die benötigten Web-Schnittstellen initialisiert. Die einzelnen Tools sind mittels eines beliebigen Browsers über folgende URLs zu erreichen:

- Ultra Search URL:
<http://aspc2.informatik.uni-stuttgart.de:5620/ultrasearch>
- Ultra Search Administration Tool URL:
<http://aspc2.informatik.uni-stuttgart.de:5620/ultrasearch/admin>
- iSQL*Plus URL:
<http://aspc2.informatik.uni-stuttgart.de:5560/isqlplus>
- Enterprise Manager 10g Database Control URL:
<http://aspc2.informatik.uni-stuttgart.de:5500/em>

Wie bereits im Abschnitt 2.3.7 *Oracle Enterprise Manager 10g Database Control* erläutert, wurde in dieser Installation die Database Manager und nicht die Grid Control Option als Enterprise Manager Konfigurationswerkzeug für die Datenbank ausgewählt.

4.2 Datenübertragung mit SQL*Loader

Um sich mit dem Oracle-Datenbanksystem vertraut zu machen, bietet Oracle die Möglichkeit verschiedene Beispiel (Sample) Datenbanken zu erstellen, auf der mit den verschiedenen Oracle-Werkzeugen experimentiert werden kann. Für weitere Informationen zur Installation bietet es sich an, die folgende Webseite von Oracle zu besuchen und die darauf angebotenen Anleitungen Schritt für Schritt durchzugehen.

http://www.oracle.com/technology/obe/2day_dba/install/install.htm

4.2 Datenübertragung mit SQL*Loader

Wie bereits in Abschnitt 3.6 *Ermittlung des passenden Migrationswerkzeugs* beschrieben, wird in dieser Studienarbeit der SQL*Loader als Migrationswerkzeug von DB2 auf Oracle verwendet. In Abschnitt 3.2 *Der SQL*Loader* wurde bereits auf die Funktionsweise und die Eigenschaften des SQL*Loaders eingegangen. Hier wird nun gezeigt, wie dieser aufgerufen werden kann und in welchen Schritten die Datenübertragung vom Ausgangssystem in DB2 zum Zielsystem in Oracle durchgeführt werden muss. Zu erst wurde in dieser Arbeit versucht, die 'direct path load'-Methode des SQL*Loaders zu verwenden (s.Abschnitt 3.3 *SQL*Loader*). Da hierbei Schwierigkeiten bei der Datenkonvertierung auftraten und die Ladegeschwindigkeit für die vorliegende Migration nicht von entscheidender Bedeutung ist, wurde dieser Ansatz verworfen und die 'conventional path load'-Methode verwendet. Für weitere Informationen zu den beiden Methoden s.[Ric03].

Die einzelnen Schritte der Datenübertragung:

1. Alle Tabellen von DB2 exportieren:
2. Control Files erstellen (bzw. existierende verwenden)
3. Tablespace und Benutzer TPCH4 in Oracle erstellen
4. Mit dem Skript createtable.sql die TPC-H Tabellen erstellen
Mit dem Skript mk_tab_v01.sql die Zusatztabelle erstellen
5. Mit SQL*Loader die Daten in die Tabellen importieren:
6. Mit dem Skript altertable.sql die Primär- und Fremdschlüsselbeziehungen erstellen
7. Mit dem Skript mk_ind_v01.sql Indizes der Tabellen umbenennen bzw. erstellen
8. Mit dem Skript mk_view_v01.sql die Views erstellen

4.2.1 Export der Tabellen (Schritt 1)

Alle Tabellen, sowohl die 8 Basistabellen des TPC-H-Schemas, als auch die 15 durch das ORBIT Projekt hinzugefügten Tabellen, müssen aus dem DB2-System exportiert und auf

4.2 Datenübertragung mit SQL*Loader

dem Zielsystem in Dump Files abgespeichert werden. Dies kann auf zwei unterschiedliche Arten erreicht werden. Entweder die Dateien werden auf dem Rechner auf dem auch das DB2-System läuft abgespeichert und danach auf beliebige Weise vom DB2-Rechner auf den Oracle-Rechner übertragen (z.B. über SCP), oder auf dem Ziel-Rechner wird die Client-Software für das DB2-System (eine DB2-Konsole) installiert und die Daten werden direkt aus dem DB2-System in Dateien auf dem Zielsystem exportiert. Hierfür wird eine Netzwerkverbindung zwischen den beiden Rechnern benötigt, allerdings benötigt die zweite Variante keinen Speicherplatz auf dem DB2-System, da die Dump Files nicht auf dem DB2-System zwischengelagert werden müssen. Für beide Varianten wird ein Zugang mit Exportrechten für das DB2-System benötigt.

Der eigentliche Export der Tabellen in die beschriebenen Dump Files geschieht dann in beiden Fällen auf die gleiche Weise. Nur der jeweilige Speicherort muss berücksichtigt werden. Man sollte im Auge behalten, dass die Dump Files für alle 23 Tabellen zusammen eine Größe von mehr als 14 GB erreichen. Es gilt also für genügend Speicherplatz zu sorgen. Man kann den Speicherplatzbedarf einschränken, indem man die einzelnen Dump Files bereits in die entsprechenden Tabellen des Oracle-Systems importiert, bevor man weitere Tabellen exportiert. Dadurch können die bereits importierten Dump Files wieder gelöscht werden. Dies ist nur bedingt hilfreich, da das Dump File für die Lineitem-Tabelle viel größer als die restlichen Dateien ist und alleine mehr als 10 GB umfasst.

Um die Daten aus dem DB2-System zu exportieren muss sich der USER auf dem DB2-System, im vorliegenden Fall ist dies der Rechner mit dem Namen *dutzend*), mittels einer DB2-Konsole einloggen, was mit folgendem Befehl möglich ist:

```
db2 connect to TPC4 user <username>
```

Danach muss jede der 23 Tabellen in ein Dump File exportiert werden, was mit folgendem Befehl möglich ist:

```
db2 export to tpch4-<tablename>.data of del select * from tpch4.<tablename>
```

Hierbei steht *<tablename>* für den jeweiligen Tabellennamen der zu exportierenden Tabelle (also z.B.: Lineitem). TPC4 ist der Schema-Name der die zu exportierenden Tabellen enthält. *of del* bedeutet, dass das Dump File im delimited ASCII Format abgespeichert wird. In diesem Format werden die einzelnen Tupel durch Kommata getrennt (delimited) im ASCII Zeichensatz abgespeichert. Der Rest des obigen Befehls ist eine einfache SQL-Select-Anfrage.

Mit folgendem Befehl kann ein Skript mit beliebigem Name (*<scriptname>*) auf dem DB2-System ausgeführt werden.

```
db2 -tf <scriptname>
```

Mit dem Skript DB2EXPORT.SQL können alle Tabellen exportiert werden. Es macht nichts anderes, als alle benötigten Tabellen aus dem DB2 System nacheinander zu exportieren. Davor muss durch CONNECT(s.o.), eine Verbindung mit der DB2-Datenbank aufgebaut werden. Wenn die 8 TPC-H-Tabellen und die 15 Zusatztabellen exportiert sind und auf das Zielsystem übertragen wurden, kann mit Schritt 2 weitergemacht werden.

4.2.2 Control Files erstellen (Schritt 2)

Um mit Hilfe des SQL*Loader die Dump Files in die entsprechenden Tabellen zu importieren werden Control Files für den Datenimport benötigt. Hier ist als Beispiel das Control File für den Import in die Tabelle CUSTOMER aufgeführt:

```
LOAD DATA
INFILE 'tpch4-customer.data'
BADFILE 'tpch4-customer.bad'
DISCARDFILE 'tpch4-customer.dsc'
INTO TABLE tpch4.customer
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
( c_custkey,
  c_name,
  c_address,
  c_nationkey,
  c_phone,
  c_acctbal,
  c_mktsegment,
  c_comment)
```

Der Aufbau eines Control Files ist relativ einfach. Der Befehl LOAD DATA zeigt dem SQL*Loader, den Beginn eines neuen Datenimports.

INFILE gibt den Namen der Datei an, aus der die Daten geladen werden sollen.

BADFILE gibt den Dateinamen an, in den die abgelehnten Datenwerte abgespeichert werden sollen.

DISCARDFILE gibt den Dateinamen an, in den alle Datensätze geschrieben werden, welche die Ladebedingungen (WHEN) nicht erfüllen. Der WHEN Befehl gibt eine Bedingung für den Datenimport an. Datensätze die diese Bedingung nicht erfüllen werden nicht importiert und in das DISCARDFILE ausgeschrieben. Folgender Code-Ausschnitt ist ein Beispiel für eine WHEN Ladebedingung in der nur die Datentupel geladen werden in denen das Attribut c_custkey größer als Null ist und die den Kundennamen(c_name) Müller besitzen.

```
.
.
INTO TABLE tpch4.customer
WHEN (c_custkey > '0') AND (c_name = 'Müller')
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
( c_custkey,
  c_name,
  c_address,
.
.
```

4.2 Datenübertragung mit SQL*Loader

Sowohl die Angabe BADFILE als auch DISCARDFILE ist optional, sie können also auch weggelassen werden. Sollten doch Datentupel die Ladebedingung nicht erfüllen oder sollten doch Tupel abgelehnt werden, so werden diese standardmäßig in `<controlfile-name>.dsc` bzw. `<controlfile-name>.bad` ausgeschrieben.

Nach der Zeile DISCARDFILE kann optional noch eine der drei Optionen APPEND, REPLACE oder TRUNCATE stehen. Diese bestimmen wie die Daten in die Tabelle eingefügt werden sollen. APPEND hängt die Daten an die bestehenden Daten an, REPLACE und TRUNCATE löschen die bestehenden Daten und laden die neuen Daten in die leere Tabelle. Hierbei ist TRUNCATE die strengere Löscharte. TRUNCATE ignoriert auch DELETE CASCADE Optionen die auf die Tabelle gesetzt waren. Für mehr Informationen über TRUNCATE s.[Ric03]. Wenn keine Option angegeben wird (wie im obigen Beispiel), können die Daten nur in eine leere Tabelle importiert werden.

INTO TABLE gibt die Tabelle an, in die die Daten importiert werden sollen.

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' gibt an, dass die einzelnen Spalten der Datensätze durch ein Komma (,) von einander getrennt sind, und dass sie optional auch noch von Anführungszeichen eingefasst sein können.

Die Klammer am Ende des Control Files gibt schließlich die jeweiligen Spalten an, in die die Daten geladen werden sollen.

Für mehr Informationen zum Thema Control Files für SQL*Loader s.[Ric03] Kapitel 8.

4.2.3 Tablespace und Benutzer TPCH4 erstellen (Schritt 3)

Damit die Tabellen in Oracle genauso wie bereits im DB2-System, in einem eigenen Schema mit dem Namen TPCH4 angelegt werden können, muss im Oracle-Datenbanksystem ein User (Benutzer) mit dem Namen TPCH4 angelegt werden. Durch das Anlegen eines Benutzers wird in Oracle automatisch ein Schema mit dem selben Namen erzeugt. Es gibt zwar auch den Befehl CREATE SCHEMA in Oracle, aber dieser erzeugt kein wirkliches Schema. Er erlaubt nur, ein bereits durch den USER Befehl erzeugtes Schema mit Tabellen und Views zu bestücken, oder Zugriffsrechte auf diese Objekte zu gewähren, ohne mehrere SQL Statements über verschiedene Transaktionen hinweg angeben zu müssen.

Um die neu anzulegende Datenbank und somit das TPCH4 Schema auch im Speicher-raum von den restlichen Datenbeständen des Oracle-Datenbank-Management-Systems abzugrenzen, sollte das neue Schema in einem eigenen neuangelegten Tablespace abgespeichert werden. Ein Tablespace ist ein logischer Teil eines Oracle-Datenbanksystems, das dazu benutzt wird, Festplattenspeicher für Tabellen und Indizes zu reservieren und zu verwalten. Jeder Tablespace besteht aus einer oder mehreren physikalischen Datenbankdateien, die vom Betriebssystem verwaltet werden. Jede Oracle-Installation besitzt einen Tablespace mit dem Namen SYSTEM. Darüber hinaus kann das Oracle-System noch über weitere Tablespaces verfügen. Meist werden diese zusätzlichen Tablespaces, wie im vorliegenden Fall mit TPCH4, benutzt um Datenstrukturen die in logischem Zusammenhang stehen, auch auf einer physikalischen Speicherebene zu gruppieren.

Es sollte zu erst der neue Tablespace angelegt werden und dann erst der neue USER, da diesem noch der Tablespace als default tablespace (standard tablespace) zugewiesen

4.2 Datenübertragung mit SQL*Loader

werden sollte. Mit folgendem Befehl kann ein Tablespace in Oracle angelegt werden:

```
CREATE TABLESPACE tablespace_name
  DATAFILE Datafile_Optionen Speicher_Optionen ;
```

Datafile_Optionen:

```
'filespec' [AUTOEXTEND OFF]
'filespec' [AUTOEXTEND ON [NEXT int K | M] [MAXSIZE int K | M]]
```

Die Autoextend Maxsize Option nimmt den Wert UNLIMITED an wenn kein Wert spezifiziert wird.

Speicher_Optionen:

```
DEFAULT STORAGE storage_clause
MINIMUM EXTENT int {K|M}
LOGGING | NOLOGGING
ONLINE | OFFLINE
PERMANENT | TEMPORARY
EXTENT MANAGEMENT {DICTIONARY |
  LOCAL {AUTOALLOCATE | UNIFORM [SIZE int K | M]} }
```

Um einen neuen Benutzer (User) und somit ein neues Schema in Oracle anzulegen wird folgender Befehl benötigt:

```
CREATE USER benutzername
  IDENTIFIED {BY password | EXTERNALLY | GLOBALLY AS external_name}
  Optionen
```

Optionen:

```
DEFAULT TABLESPACE tablespace_name
TEMPORARY TABLESPACE tablespace_name
QUOTA int {K | M} ON tablespace_name
QUOTA UNLIMITED ON tablespace_name
PROFILE profile_name
PASSWORD EXPIRE
ACCOUNT {LOCK|UNLOCK}
```

Der neue Tablespace TPCH4 und der gleichlautende Benutzer werden mit folgenden Befehlen erstellt:

```
CREATE TABLESPACE TPCH4 DATAFILE 'speicherpfad/TPCH4.dbf' SIZE 100M AUTOEXTEND ON
NEXT 10M ONLINE PERMANENT EXTENT MANAGEMENT LOCAL;
```

4.2 Datenübertragung mit SQL*Loader

```
CREATE USER TPCH4 IDENTIFIED BY tpchadmin DEFAULT TABLESPACE TPCH4
  TEMPORARY TABLESPACE TEMP QUOTA UNLIMITED ON TPCH4;
```

Nachdem der neue Tablespace und der neue User angelegt wurden, kann zu Schritt 4 übergegangen werden. Um die Indizes der neuen Datenbank in einem anderen physikalischen Speicherbereich als die Tabellen und die restlichen Daten abspeichern zu können, wurde in dieser Arbeit noch ein weiterer Tablespace erstellt (s.u.). Da dieser Tablespace nur die Index-Strukturen der Datenbank enthalten soll, wurde der neue Tablespace TPCH4INDEX genannt. Darüber hinaus muss dem USER TPCH4 noch erlaubt werden auf diesen neuen Tablespace zugreifen und dort abspeichern zu können. Dies wird mit dem ALTER USER Befehl realisiert.

```
CREATE TABLESPACE TPCH4INDEX DATAFILE 'speicherpfad/TPCH4INDEX.dbf' SIZE 100M
  AUTOEXTEND ON NEXT 10M ONLINE PERMANENT EXTENT MANAGEMENT LOCAL;
ALTER USER TPCH4 QUOTA UNLIMITED ON TPCH4INDEX;
```

Die physikalische Trennung zwischen Index- und Tabellen-Tablespace ermöglicht es, den Index-Tablespace auf eine zweite Festplatte zu speichern, was bei einer parallelen Bearbeitung von Indizes und Tabellen zu einer schnelleren Zugriffszeit sorgt, da sich die parallelen Zugriffe auf den beiden unterschiedlichen Platten nicht gegenseitig in die Que-re kommen können.

4.2.4 Die Tabellen im Oracle-System erstellen (Schritt 4)

Um die Daten endgültig aus den Dump Files in die neue Oracle-Datenbank importieren zu können, müssen die 23 Tabellen im Oracle-System angelegt werden. Dazu muss jede einzelne der 8 TPC-H und der 15 Zusatz-Tabellen mit einem CREATE TABLE-Befehl erstellt werden. Hierfür kann man die bereits für das DB2-System vorhandenen Createtable-Skripte benutzen. Da die SQL-Syntax von DB2 und Oracle nicht exakt gleich ist, müssen diese Skripte noch für Oracle umgeschrieben werden. Sowohl die Original-DB2-Skripte als auch die angepassten Oracle-Skripte befinden sich im Anhang dieser Arbeit.

Als Beispiel für die Anpassung der SQL-Create-Befehle soll hier nur das Create-Statement der ersten TPC-H Tabelle (TPCH4.PART) herangezogen werden.

```
-----
Ausschnitt aus DB2-Skript:createtable.sql
-----
```

```
1 CONNECT TO tpch;
2
3 CREATE TABLE tpch4.part (
4   p_partkey          Integer NOT NULL,
5   p_name             Varchar(55),
6   p_mfgr             Char(25),
7   p_brand            Char(10),
```

4.2 Datenübertragung mit SQL*Loader

```
8 p_type          Varchar(25),
9 p_size          Integer,
10 p_container    Char(10),
11 p_retailprice  Decimal(10, 2),
12 p_comment      Varchar(23)
13 ) IN data1;
```

Auch wenn der Unterschied zwischen der Oracle- und der DB2-SQL-Syntax nicht sonderlich gravierend ist, so müssen die Befehle doch umgeschrieben werden, oder sie können im jeweils anderen System nicht ausgeführt werden.

Ausschnitt aus angepasstem Oracle-Skript:createtable.sql

```
1 CONNECT system@tpch;
2
3 CREATE TABLE tpch4.part (
4 p_partkey      Integer NOT NULL,
5 p_name         Varchar(55),
6 p_mfgr        Char(25),
7 p_brand       Char(10),
8 p_type        Varchar(25),
9 p_size        Integer,
10 p_container   Char(10),
11 p_retailprice Decimal(10, 2),
12 p_comment     Varchar(23)
13 ) TABLESPACE TPCH4;
```

Die einzigen Unterschiede in den beiden Ausschnitten sind in Zeile 1 und Zeile 13 zu finden. Die Oracle-Syntax für den CONNECT Befehl unterscheidet sich folgendermaßen von der DB2-Syntax:

Oracle-Schreibweise des Connect-Befehls:

```
CONNECT username/password@sid [AS {SYSDBA|SYSOPER}]
  username = Ein gültiger Oracle Benutzername
  sid = Database System ID (im vorliegenden Fall TPCH)
```

DB2-Schreibweise des Connect-Befehls:

```
CONNECT TO database USER userID USING password
```

Der Unterschied in Zeile 13 bezieht sich auf das CREATE TABLE Statement. Die Angabe *IN data1* bedeutet in DB2, dass die Tabelle im Tablespace data1 abgelegt werden soll.

4.2 Datenübertragung mit SQL*Loader

Das Gleiche wird in Oracle durch *TABLESPACE data1* erreicht. Nur dass hier nicht der Tablespace *data1* sondern *TPCH4* benutzt wird. Eigentlich müsste das CREATE TABLE Statement aber völlig umgeschrieben werden, da Oracle andere Datentypen als DB2 benutzt. Dies ist aber nicht nötig, da Oracle auch das obere CREATE-Statement versteht, und die Datentypen automatisch in die Oracle-Schreibweise umwandelt. Die Tabelle *PART* wird nach dem oberen CREATE-Statement folgendermaßen umgewandelt:

Eigentliche Oracle-Schreibweise

```
3 CREATE TABLE tpch4.part (  
4 p_partkey          NOT NULL NUMBER(38),  
5 p_name            VARCHAR2(55),  
6 p_mfgr           CHAR(25),  
7 p_brand          CHAR(10),  
8 p_type           VARCHAR2(25),  
9 p_size           NUMBER(38),  
10 p_container      CHAR(10),  
11 p_retailprice    NUMBER (10, 2),  
12 p_comment        VARCHAR(23)  
13 ) TABLESPACE TPCH4;
```

Nachdem alle 23 Tabellen in der Oracle-Datenbank erstellt wurden kann zu Schritt 5 übergegangen werden.

4.2.5 Datenimport mit SQL*Loader (Schritt 5)

Um die Daten aus den Dump Files in die neu erzeugten Tabellen zu laden kommt nun der SQL*Loader zum Einsatz. Hierfür wird ein einfaches Kommandofenster benötigt, das unter Windows durch *Start* -> *Ausführen* -> *cmd* aufgerufen werden kann. Danach sollte man in das Verzeichnis wechseln, in dem die 23 Dump Files abgespeichert wurden. Der SQL*Loader lässt sich mit dem Befehl *sqlldr* aufrufen und muss für alle 23 Tabellen mit folgender allgemeiner Syntax aufgerufen werden.

```
SQLldr CONTROL=<filename>.ctl, LOG=<filename>.log, BAD=baz.bad, DATA=etc.dat  
        USERID=<username>/<passwort>, ERRORS=999, LOAD=2000, DISCARD=toss.dsc,  
        DISCARDMAX=5
```

CONTROL gibt den Speicherort des Control Files (s. Schritt 2) an. LOG gibt den Dateinamen an, in den der Log-Bericht abgespeichert werden soll. In diesem Log-Bericht protokolliert der SQL*Loader den Ablauf des ausgeführten Importauftrags. Die LOG-Angabe ist optional. Diese kann auch weggelassen werden. Das Protokoll wird aber auf jeden Fall erstellt und standardmäßig unter *<name des controlfiles>.log* abgespeichert. Als Beispiel eines SQL*Loader Protokolls ist im Anhang das Protokoll für den Datenimport in die Tabelle *TPCH4.CUSTOMER* zu finden. Die Optionen *BAD* und *DATA*

4.2 Datenübertragung mit SQL*Loader

können auch im Control File angegeben werden und sind bereits in Schritt 2 erklärt. USERID gibt den Benutzernamen für den Zugriff auf die Oracle-Datenbank an. Hinter dem / kann das Benutzerpasswort angegeben werden. Falls dieses nicht angegeben wird, folgt nach dem Aufruf und vor dem eigentlichen Laden eine Passwortabfrage. ERRORS gibt die maximale Anzahl an fehlgelaufenen Datensätzen an, bevor der gesamte Ladevorgang abgebrochen wird. LOAD gibt die Anzahl der zu ladenden Datensätze aus dem Dump File an. DISCARD wurde ebenfalls in Schritt 2 erklärt. DISCARDMAX ist wieder eine Abbruchbedingung, die angibt nach wie vielen, in das discard file geschriebenen Datensätzen, der gesamte Ladevorgang abgebrochen wird. Wenn man die Control Files wie in Schritt 2 beschrieben angelegt hat, genügt es, das Control File anzugeben. Der Befehlsaufruf für das Importieren eines der exportierten Dump Files lautet wie folgt:

```
SQLLDR USERID=<userid>@tpch control=tpch4-<tablename>.ctl
```

Wenn dies für alle 23 Tabellen geschehen ist, kann zu Schritt 6 übergegangen werden.

4.2.6 Primär- und Fremdschlüsselbeziehungen erstellen (Schritt 6)

In Schritt 6 geht es darum, die Primär- und Fremdschlüsselbeziehungen zwischen den einzelnen Tabellen herzustellen. Diese Schlüsselbedingungen hätten auch beim Erstellen der einzelnen Tabellen berücksichtigt werden können, dann hätte man sich aber an eine genaue Reihenfolge beim Laden und Erstellen der Tabellen halten müssen, da eine Tabelle, die einen Fremdschlüssel in einer anderen Tabelle besitzt, erst geladen bzw. erstellt werden kann, wenn diese Tabelle bereits geladen bzw. erstellt wurde. Es ist also unproblematischer, erst nach dem vollständigen Import der Tabellendaten die Schlüsselbeziehungen zu erstellen.

Das Verändern einer bereits bestehenden Tabelle kann mit dem Befehl ALTER TABLE geschehen. Mit folgendem Befehl wird der Tabelle ORDERS eine Primärschlüsselbedingung (PRIMARY KEY CONSTRAINT) hinzugefügt. Danach gilt *o_orderkey* als Primärschlüssel für ORDERS.

```
ALTER TABLE tpch4.orders ADD CONSTRAINT c_pk05 PRIMARY KEY (o_orderkey);
```

Eine Fremdschlüsselbeziehung (FOREIGN KEY CONSTRAINT) kann durch folgenden Befehl erzeugt werden.

```
ALTER TABLE tpch4.orders ADD CONSTRAINT c_fk05 FOREIGN KEY (o_custkey)
REFERENCES tpch4.customer (c_custkey);
```

Der Fremdschlüssel benötigt noch einen Bezug (REFERENCES) zu einem Attribut einer anderen Tabelle. Im oberen Beispiel wird dem Attribut *o_custkey* der Tabelle ORDERS eine Fremdschlüsselbeziehung zum Attribut *c_custkey* der Tabelle CUSTOMER zugewiesen.

Befehls-Syntax des ALTER TABLE Befehls von DB2 und Oracle unterscheidet sich nicht, weshalb das DB2-Skript altertable.sql fast 1 zu 1 zum Erstellen aller Primär- und Fremdschlüsselbeziehungen der Oracle-Tabellen herangezogen werden kann. Im Skript musste

nur der CONNECT TO Befehl in die CONNECT Syntax von Oracle umgewandelt werden. Um also die Schlüsselbedingungen aller Tabellen zu erstellen kann einfach das Skript altertable.sql aufgerufen werden. Danach kann zu Schritt 7 übergegangen werden.

4.2.7 Indizes der Tabellen umbenennen bzw. erstellen (Schritt 7)

In Schritt 7 werden die Indizes der Oracle-Tabellen erstellt bzw. umbenannt. Es müssen nicht alle Indizes neu erstellt werden, da Oracle schon beim Erzeugen der Primärschlüsselbeziehungen, automatisch jeweils einen Index für das Primärschlüsselattribut der jeweiligen Tabelle erstellt. Oracle benennt diese automatisch erzeugten Indizes dann nach dem CONSTRAINT Bezeichner, der die Schlüsselbeziehung angibt. Im oberen Beispiel(s.Schritt 6) benennt Oracle den automatisch erzeugten Index für das Primärattribut *o_orderkey* dann *c_pk05*. Im DB2 System lautet der Name für den Primärschlüssel-Index von *o_orderkey* jedoch *sk_o_ok*. Um in Oracle die gleichen Bezeichner wie in DB2 zu verwenden, müssen diese automatisch erzeugten Primärschlüssel umbenannt werden. Dies kann mit dem Befehl ALTER INDEX erreicht werden. Mit folgendem Befehl wird der Index *C_PK05* in *sk_o_ok* umbenannt (RENAME).

```
ALTER INDEX tpch4.C_PK05 RENAME TO sk_o_ok;
```

Danach wird der Index in den extra für die Indizes angelegten Tablespace TPCH4INDEX (s.Schritt 3) verschoben (REBUILD).

```
ALTER INDEX tpch4.sk_o_ok REBUILD TABLESPACE TPCH4INDEX;
```

Da das vorhandene DB2 System aber auch ein paar Indizes auf Fremdschlüssel besitzt, müssen diese noch mittels CREATE INDEX erzeugt werden. Der nachfolgende Befehl erzeugt einen Index mit dem Namen *sk_o_od* auf der Tabelle ORDERS. Er bezieht sich auf das Attribut *o_orderdate* und ist aufsteigend sortiert. Der Index wird im Index-Tablespace TPCH4INDEX erzeugt.

```
CREATE INDEX tpch4.sk_o_od ON tpch4.orders (o_orderdate ASC)
TABLESPACE TPCH4INDEX;
```

Um alle bereits auf dem DB2-System vorhandenen Indizes zu erzeugen, kann das Skript *mk_ind_v01.sql* ausgeführt werden. Danach kann zu Schritt 8 übergegangen werden.

4.2.8 Views im Oracle-System erstellen (Schritt 8)

Im 8. und letzten Schritt werden die 31 Views erzeugt, die bereits auf dem DB2-System vorhanden sind. Ein neuer VIEW wird mittels des CREATE VIEW Befehls erzeugt.

```
CREATE VIEW tpch4.lookup_custnation (
  custnationkey, custnationname, custregionkey) AS
SELECT n_nationkey, n_name, n_regionkey
FROM tpch4.nation;
```

4.2 Datenübertragung mit SQL*Loader

Auch in diesem Fall unterscheidet sich die Syntax des CREATE VIEW Befehl unter DB2 und Oracle nicht und so kann das DB2-Skript ohne große Umwandlungen auch für die Erstellung der Views in Oracle benutzt werden.

5 Leistungsmessungen

In diesem Kapitel werden auf dem neuinstallierten Oracle-Datenbanksystem Leistungsmessungen durchgeführt. Hierbei wird die im Kapitel *Grundlagen* beschriebene und im Migrationskapitel übertragene Datenbasis als Grundlage verwendet. Im ersten Abschnitt werden die möglichen Schnittstellen zur Analyse aufgezeigt und in Abschnitt 2 werden die eigentlichen Leistungsmessungen anhand der beschriebenen Schnittstellen durchgeführt, und schließlich im dritten Abschnitt die Ergebnisse der Analysewerkzeuge ausgewertet.

5.1 Schnittstellen zur Analyse der Ausführungspläne

Auf dem bestehenden DB2-System wurden mehrere Performance-Tests entwickelt und ausgeführt. Dabei handelt es sich um eine Sequenz von SQL-Statements, deren Ausführungspläne analysiert werden sollen. Darüber hinaus soll deren Durchführungsdauer gemessen werden. Hierzu bietet Oracle unterschiedliche Analyseschnittstellen an, deren Funktionsweise und Einsatz hier beschrieben werden.

5.1.1 Explain Plan

Explain Plan ist ein von Oracle zur Verfügung gestelltes Werkzeug zur Analyse von SQL-Statements. EXPLAIN PLAN ist ein Oracle SQL-Befehl, der vor den Befehlen ANALYZE, CREATE, INSERT, SELECT oder UPDATE stehen kann und bietet eine Möglichkeit, den Ausführungsplan und die geschätzten Ausführungskosten des SQL-Befehls anzuzeigen, ohne den Befehl wirklich auszuführen. Die allgemeine Syntax des EXPLAIN PLAN Befehls sieht wie folgt aus:

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

Das Ergebnis des EXPLAIN PLAN Befehls wird nicht auf dem Bildschirm angezeigt, sondern in einer festgelegten Tabelle abgespeichert. Diese Tabelle hat standardmäßig den Namen PLAN_TABLE, und wird automatisch nach der Installation als globale, temporäre Tabelle erzeugt. Wenn der Befehlszusatz INTO weggelassen wird, wird automatisch in PLAN_TABLE geschrieben. Um in eine andere Ausgabetable zu schreiben,

5.1 Schnittstellen zur Analyse der Ausführungspläne

kann `PLAN_TABLE` mit Hilfe des `RENAME`-Befehls umbenannt werden. Um eine lokale `PLAN_TABLE`-Tabelle zu erzeugen kann das Skript *utlxplan.sql* benutzt werden, was eine lokale Tabelle mit dem Namen `PLAN_TABLE` erzeugt. Diese kann wieder mit dem `RENAME`-Befehl umbenannt werden. Mit dem Befehlszusatz `INTO` kann in die umbenannte oder neuerzeugte Tabelle geschrieben werden. Um verschiedene SQL-Anweisungen im `PLAN_TABLE` auseinander halten zu können, muss den zu analysierenden Anweisungen mit `SET STATEMENT_ID` ein eindeutiger Bezeichner gegeben werden, mit dessen Hilfe die Ausgabetablelle gezielt nach den einzelnen Anweisungen durchsucht werden kann. Wenn keine `STATEMENT_ID` angegeben wird, setzt `EXPLAIN PLAN` die ID standardmäßig auf `NULL`. Dadurch überschreiben mehrere hintereinander ausgeführte SQL-Anweisungen ohne ID jeweils die Vorgängeranweisung.

Die `FOR`-Anweisung muss direkt oder in der Zeile vor dem zu analysierenden Befehl stehen, bei dem es sich um ein `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE`, `CREATE INDEX`, oder `ALTER INDEX ... REBUILD` Befehl handeln darf.

Oracle bietet zwei vordefinierte Skripte und eine vorgegebene Prozedur, um auf die Daten eines analysierten Statements im `PLAN_TABLE` zugreifen zu können:

- Das Skript *UTLXPLS.SQL* zeigt den seriellen Ablaufplan der zu letzt analysierten SQL-Anweisung an.
- Das Skript *UTLXPLP.SQL* zeigt den gleichen Ablaufplan an, jedoch erweitert, um Spalten die die parallele Ausführung betreffen.
- Die Prozedur `DBMS_XPLAN.DISPLAY` akzeptiert drei Eingabeparameter um die Ausgabe zu spezifizieren. Der erste Parameter gibt den Namen des `PLAN_TABLE`s an, der zweite Parameter gibt die `STATEMENT_ID` an und der letzte Parameter spezifiziert das Ausgabeformat, das die Werte `BASIC`, `SERIAL`, `TYPICAL` und `ALL` annehmen kann, was die Ausgabe unterschiedlich detaillierter Berichte bewirkt.

Das Folgende Beispiel-Statement, ist eine `INSERT`-Anfrage, aus einer der Sequenzen die für die Performance Messung eingesetzt wurden.

```
EXPLAIN PLAN SET STATEMENT_ID = 'sequence-Insert1'
FOR
INSERT INTO tpch4.temptableC010
SELECT
  a1.custkey,
  sum(a1.endprice)
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey = 1992
```

5.1 Schnittstellen zur Analyse der Ausführungspläne

```
GROUP BY
  a1.custkey;
```

Die angegebene SQL-Anfrage kann durch die DBMS_XPLAN.DISPLAY Prozedur mit dem Ausgabeparameter TYPICAL wie in Zeile 2 des folgenden Aufrufs abgerufen werden.

```
1 SPOOL c:\sqlscripts\c-subsequence\sequence_0\sequenceoutput-explainplan
2 SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY(NULL, 'sequence-Insert1',
'TYPICAL'));
3 SPOOL OFF;
```

Mit Hilfe des SPOOL Befehls in der 1. Zeile wird die Ausgabe in die angegebene Datei ausgeschrieben. SPOOL OFF beendet das Ausschreiben der Befehlsausgabe. SPOOL kann nicht nur mit EXPLAIN PLAN benutzt werden, sondern dient dazu jegliche Kommandozeilenausgabe in die angegebene Datei auszuschreiben. Die Ausgabe der Prozedur sieht dann wie folgt aus.

PLAN_TABLE_OUTPUT

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	INSERT STATEMENT		999K	41M		379K (2)	01:15:55
1	SORT GROUP BY		999K	41M	515M	379K (2)	01:15:55
* 2	HASH JOIN		8569K	351M	83M	338K (1)	01:07:44
* 3	HASH JOIN		2142K	59M		47063 (2)	00:09:25
* 4	TABLE ACCESS FULL	LOOKUP_ORDERDAY	344	4128		6 (0)	00:00:01
5	TABLE ACCESS FULL	ORDERS	15M	243M		46918 (1)	00:09:24
6	TABLE ACCESS FULL	LINEITEM	59M	800M		212K (1)	00:42:30

Predicate Information (identified by operation id):

```
2 - access("L"."L_ORDERKEY"="0"."O_ORDERKEY")
3 - access("A2"."ORDERDATE"="0"."O_ORDERDATE")
4 - filter("A2"."ORDERYEARKEY"=1992)
```

19 rows selected.

Die Erklärung zu den einzelnen Spalten ist der Tabelle 5.1 zu entnehmen. Zusätzlich zu den zwei Skripten und der DBMS_XPLAN.DISPLAY Prozedur, kann der PLAN_TABLE wie jede andere Tabelle auch, mittels eigenen Skripten und SQL-Befehlen durchsucht und abgefragt werden. Dazu sollte aber die Struktur und die Tabellenspalten der PLAN_TABLE Tabelle bekannt sein. Es folgt eine kurze Liste mit den 8 Spalten, die mit der TYPICAL Einstellung der DBMS_XPLAN.DISPLAY Prozedur ausgegeben werden. Die ausführliche Spaltenliste mit mehr als 50 verschiedenen Spalten und deren Erklärung ist im Oracle Performance Tuning Guide [Moo03] Kapitel 19 zu finden.

5.1 Schnittstellen zur Analyse der Ausführungspläne

Spalte	Erklärung
Id	Gibt die Nummer in der Ausführungsreihenfolge an.
Operation	Name der internen Operation die in diesem Schritt ausgeführt wird. Die erste Zeile, gibt den SQL-Befehl an, der mit Hilfe von EXPLAIN PLAN analysiert werden sollte.
Name	Heißt eigentlich OBJECT_NAME und gibt den Tabellen- oder Indexnamen an, der in diesem Schritt Verwendung findet.
Rows	Heißt eigentlich CARDINALITY und gibt die geschätzte Anzahl der Zeilen an, die von dieser Operation zurückgegeben werden.
Bytes	Gibt eine Schätzung der zu bearbeitenden Bytes an.
TempSpc	Heißt eigentlich TEMP_SPACE und gibt eine Schätzung des benötigten temporären Speichers an.
Cost (%CPU)	Gibt einen Kostenschätzwert an und wie viele CPU Zyklen die Operation schätzungsweise benötigt.
Time	Gibt die geschätzte Zeit an, die die jeweilige Operation benötigt.

Tabelle 5.1: Ausgesuchte Spalten der PLAN_TABLE Tabelle

5.1.2 SQL Trace und TKPROF

SQL Trace und TKPROF sind zusammen ein weiteres Werkzeug zur Analyse von Ausführungsplänen. Im Gegensatz zu EXPLAIN PLAN basieren SQL Trace und TKPROF auf wirklich durchgeführten SQL-Anweisungen und nicht auf bloßen Schätzungen der angegebenen SQL-Befehle. Hierbei wird SQL Trace dazu benutzt, die SQL-Statements zu protokollieren und diese Protokolle in Dateien auszuschreiben. Um dies zu bewerkstelligen, muss SQL Trace vor der Ausführung des zu protokollierenden Statements aktiviert werden und während der Ausführung im Hintergrund laufen. SQL Trace kann für eine Session oder eine Instanz aktiviert werden. Dabei protokolliert SQL Trace nicht nur ein einzelnes Statement, sondern alle, also auch mehrere parallellaufende SQL-Operationen, die innerhalb der Session oder Instanz auf die Datenbank zugreifen. Durch die Aktivierung von SQL Trace entsteht ein gewisser Overhead, der die Ergebnisse der Leistungsprotokolle aber in den meisten Fällen nicht verfälscht, aber bei einer stark ausgelasteten Datenbank zu Problemen führen kann[Moo03].

Das TKPROF Werkzeug ist ein Kommandozeilenprogramm, das dazu dient, die abgespeicherten Protokolldateien in ein gewünschtes, aussagekräftiges Format zu wandeln und

5.1 Schnittstellen zur Analyse der Ausführungspläne

in einer getrennten Datei abzuspeichern. Um den Unterschied zwischen den Protokolldateien und den TKPROF-Ausgabedateien zu verdeutlichen, ist hier ein kurzer Ausschnitt aus einer Protokoll- und einer TKPROF-Datei angeführt:

Ausschnitt aus einer Protokolldatei von SQL Trace:

```
*** TRACE DUMP CONTINUED FROM FILE c:\statistiken\tpch_ora_1848.trc ***
EXEC #11:c=15625,e=1566,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=1673443489
XCTEND rlbk=0, rd_only=1
=====
PARSING IN CURSOR #1 len=100 dep=0 uid=5 oct=1 lid=5 tim=1673454592 hv=3909328080 ad='2d4c600c'
CREATE TABLE tpch4.temptableC010(
  custkey INTEGER,
  turnover1992 DECIMAL(31, 6)) TABLESPACE TPCH4
END OF STMT
PARSE #1:c=0,e=1124,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=1673454583
=====
PARSING IN CURSOR #7 len=116 dep=1 uid=0 oct=3 lid=0 tim=1673455888 hv=854877822 ad='322db368'
select o.owner#,o.name,o.namespace,o.remoteowner,o.linkname,o.subname,o.dataobj#,....
END OF STMT
PARSE #7:c=0,e=95,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1673455879
...

```

Ausschnitt aus einer TKPROF-Datei:

```
...
*****
INSERT INTO tpch4.temptableC010
SELECT
  a1.custkey,
  sum(a1.endprice)
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey = 1992
GROUP BY
  a1.custkey

```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.01	0.00	0	0	0	0
Execute	1	260.07	817.75	1495383	1338656	899583	868074
Fetch	0	0.00	0.00	0	0	0	0
total	2	260.09	817.75	1495383	1338656	899583	868074

Misses in library cache during parse: 1

Optimizer mode: ALL_ROWS

Parsing user id: 5

```
Rows      Row Source Operation
-----
868074    SORT GROUP BY (cr=1336166 pr=1495383 pw=163137 time=742924886 us)
```

5.1 Schnittstellen zur Analyse der Ausführungspläne

```
9117084  HASH JOIN  (cr=1336166 pr=1489326 pw=157080 time=704853784 us)
2281205  HASH JOIN  (cr=241242 pr=237366 pw=0 time=114064551 us)
      366    TABLE ACCESS FULL LOOKUP_ORDERDAY (cr=23 pr=0 pw=0 time=545 us)
15000000 TABLE ACCESS FULL ORDERS (cr=241219 pr=237366 pw=0 time=90000086 us)
59986052 TABLE ACCESS FULL LINEITEM (cr=1094924 pr=1094880 pw=0 time=480011780 us)
*****
...

```

Die TKPROF-Ausgabedatei stellt im Gegensatz zur Protokolldatei von SQL Trace eine gut lesbare Informationsquelle dar. Der obere Dateiausschnitt gibt verschiedene Informationen an. Die erste Tabelle zeigt, die Anzahl der Parse-, Execute- und Fetch-Operationen (call), wie viel CPU Zeit diese in Anspruch genommen haben (cpu), wie viel Zeit in Sekunden dabei vergangen ist (elapsed), wie viele Datenblöcke gelesen wurden (disk), wie viele Pufferzugriffe im Consisten Mode (query) und wie viele im Current Mode (current) durchgeführt wurden und wie viele Tabellenzeilen die Operationen produziert haben (rows). Beim Consisten oder auch Query Mode werden alle Daten gelesen wie sie vor Beginn der Query-Ausführung im System vorhanden waren. Eventuelle Änderungen die nach dem Ausführungsbeginn stattfinden werden ignoriert, wodurch keine Dirty Reads auftreten können. Im Gegensatz dazu liest der Current Mode den neuesten Stand der Daten. Unter der Tabelle stehen die Anzahl der Fehlschläge im library cache, die Query Optimizer Methode (s.Kapitel 6 *Optimizer Hints*) und der Ausführungsplan des SQL-Statements. Der Ausführungsplan wurde mit Hilfe der EXPLAIN PLAN-Funktionalität erzeugt, und gibt nicht nur die Ausführungsreihenfolge, sondern auch die Anzahl der Zeilen (Rows), die Anzahl an konsistenten (cr) und physischen (pr) Lesezugriffen, die Anzahl physischer Schreibzugriffe (pw) und die Ausführungsdauer in Mikrosekunden (time) für jede der aufgeführten Operationen an.

Um SQL Trace und TKPROF einsetzen zu können müssen folgenden Schritte durchgeführt werden:

1. Setzen der Initialisierungsparameter für das SQL Trace Protokolldateimanagement.
2. Aktivierung des SQL Trace Werkzeugs.
3. Umwandlung der Protokolldateien durch das TKPROF-Kommandozeilenwerkzeugs.

Im 1. Schritt muss der Speicherort für die Protokolldateien von SQL Trace festgelegt werden. Durch den folgenden SQL-Befehl werden die Trace-Dateien im Verzeichnis `c:\Statistiken` abgespeichert:

```
ALTER SYSTEM SET USER_DUMP_DEST='c:\statistiken';
```

Standardmäßig haben diese Trace-Dateien einen systemgenerierten Namen. Um die Protokolldateien für bestimmte SQL-Anfragen auseinander halten zu können, kann diesem systemgenerierten Namen noch ein eigener Identifikator angehängt werden, was mit folgendem Befehl funktioniert:

```
ALTER SESSION SET TRACEFILE_IDENTIFIER='my_trace_id';
```

5.1 Schnittstellen zur Analyse der Ausführungspläne

Damit SQL Trace auch zeitbezogene Statistiken wie (cpu) und (elapsed) sammelt, müssen diese mit folgendem Befehl aktiviert werden:

```
ALTER SYSTEM SET TIMED_STATISTICS=TRUE;
```

Im 2. Schritt muss SQL Trace für die Session oder die Instanz aktiviert werden, was mit folgendem Befehl für die Session durchgeführt wird:

```
ALTER_SESSION SET SQL_TRACE=TRUE;
```

oder für die gesamte Instanz:

```
SQL_TRACE=TRUE;
```

Durch das Ersetzen von TRUE durch FALSE in den obigen Befehlen, kann SQL Trace wieder beenden werden, was auf Grund des entstehenden Overheads nach jeder Messung getan werden sollte.

Im 3. Schritt müssen die Protokolldateien von SQL-Trace in das anschauliche TKPROF-Format gewandelt werden. Dies wird durch das TKPROF-Werkzeug in der Kommandozeile erreicht. Der folgende Befehl wandelt eine Trace-Datei Namens *tpch_ora_1848_sequence-seq.trc* im Verzeichnis *c:\Statistiken* in eine TKPROF Datei mit dem Namen *sequence-seq.prf* um:

```
TKPROF c:\Statistiken\tpch_ora_1848_sequence-seq.trc c:\Statistiken\sequence-seq
explain=<userid>/<passwort>
```

Die Option `explain` erzeugt mittels EXPLAIN PLAN, den im TKPROF-Dateiausschnitt gezeigten Ausführungsplan. Eine genaue Beschreibung der TKPROF-Syntax und aller Argumente ist in Tabelle 20-2 des [Moo03] zu finden. Nach diesen drei Schritten kann die TKPROF-Datei zur Statistikanalyse herangezogen und ausgewertet werden.

5.1.3 Laufzeitmessungen

Um reine Laufzeitmessungen für die SQL-Befehle in Oracle durchzuführen, bietet Oracle den Kommandozeilenbefehl TIMING an. Durch diesen Befehl können eigene Timer erstellt und manuell gestartet werden (TIMING START timer_name). Wie bei einer Stoppuhr, muss der manuell erstellte Timer auch wieder von Hand gestoppt werden (TIMING STOP timer_name). Um den aktuellen oder gestoppten Stand des Timers auszugeben, wird der Befehl TIMING SHOW benutzt. Die allgemeine Syntax des TIMING Befehls sieht wie folgt aus:

```
TIMI[NG] [START timer_name | SHOW | STOP]
```

Die selbst erstellten Timer können dazu benutzt werden, mehrere SQL-Anweisungen einzufassen, um so einen einzelnen Zeitwert für die gesamte Ablaufdauer der eingefassten Befehle zu erhalten. Um alle Timer wieder zu löschen kann der Befehl CLEAR TIMING eingegeben werden. Um die einzelne Ablaufdauer eines oder mehrerer SQL-Befehle zu erhalten, kann man jeden dieser Befehle mit einem TIMING START und TIMING STOP

5.2 Vorbereitungen und Durchführung

Kommando einfassen und nachher mit `TIMING SHOW` die einzelnen Werte wieder ausgeben. Oracle bietet aber auch eine einfachere Lösung, um die Ablaufdauer jedes einzelnen SQL-Befehls anzugeben. Dabei handelt es sich um den `SET TIMING` Befehl.

```
SET TIMING [ON | OFF]
```

Durch die Aktivierung des `TIMING`-Befehls, wird hinter jedem eingegebenen Befehl, sofort dessen Ablaufdauer ausgegeben.

Um die einzelnen Timing-Werte oder die selbst erstellten Timer festzuhalten, können diese mit dem `SPOOL`-Befehl (s. Abschnitt 5.1.1) in eine Datei ausgeschrieben werden. Dadurch ergibt sich ein simples aber effektives Werkzeug, um Laufzeitmessungen für Oracle SQL-Befehle durchführen zu können [Wat03]. Auch `TKPROF` erzeugt Laufzeitwerte der überwachten SQL-Anweisungen. `TIMING` bietet aber die Möglichkeit, die Zeiten auf einfachere Weise auszugeben und erzeugt im Vergleich zu `TKPROF` keinen nennenswerten Overhead.

5.2 Vorbereitungen und Durchführung

In Abschnitt 2.2 *Zugrundeliegende Datenbasis* wurde bereits darauf eingegangen, dass auf dem Ausgangssystem in DB2 verschiedene Performance Tests durchgeführt wurden. Mit Hilfe der DSS-Tools von Microstrategy wurden die in [Wag99] ausgearbeiteten und in Abschnitt 2.2 aufgeführten Fragestellungen jeweils in eine Sequenz von SQL-Anweisungen umgewandelt. Als Vorgabe für die durchzuführenden Performance-Tests in dieser Arbeit wurden mehrere Varianten der SQL-Sequenz für die 4. Frage aus Abschnitt 2.2 vorgeben. Diese SQL-Sequenzen werden auf die Oracle-Datenbasis angewendet und die in den vorigen Abschnitten vorgestellten Analysewerkzeuge dazu benutzt, um die Eigenschaften der Anfragen zu analysieren. Da die erstellten SQL-Sequenzen im DB2-Format vorliegen, müssen diese vorher in das Oracle-SQL-Format umgewandelt werden.

5.2.1 Vorgegebene Sequenzen

In den vorgegebenen Sequenzen wurden für die 4. Fragestellung mehrere Annahmen getroffen. Daraus ergibt sich die folgende Fragestellung in der keine Variablen mehr vorkommen:

- 4.Frage: Wer sind die Kunden (customer), die in den Jahren 1992, 1993 und 1994 (orderyearkey) einen Mindestumsatz von größer gleich 500000\$ (extendedprice, discount) gebracht haben und deren Standardabweichung des Umsatzes in dieser Zeit unter 66,79% lag?

Bei den vorgegebenen Sequenzen handelt es sich um SQL-Skripte, die in mehrere Kategorien unterteilt sind. Die Sequenz-Skripte sind unter dem Ordner `C:\SQLScripts` abgespeichert und in die folgenden Unterordner aufgeteilt, die den Kategorien entsprechen:

5.2 Vorbereitungen und Durchführung

- *c-sequence*
 - *sequence_0*
 - *sequence_1*
 - *sequence_2*
 - *sequence_3*
 - *sequence_4*
- *c-subsequence*
 - *sequence_0*
 - *sequence_1*
 - *sequence_2*

Die SQL-Sequenz *c-sequence-sequence_0* steht hierbei für die unveränderte Originalsequenz, die eine SQL-Umsetzung der obigen Fragestellung darstellt. Die vier Unterteilungen stehen für verschiedene Optimierungsversuche der gleichen Anfrage. In diesen Optimierungsvarianten werden in aufsteigender Reihenfolge möglichst viele CREATE TABLE und INSERT-Statements der Ausgangssequenz (*sequence_0*) zusammengezogen. Es handelt sich dabei aber immer um die gleiche Fragestellung, die jeweils die selbe Ergebnismenge liefert. Bei der SQL-Sequenz *c-subsequence-sequence_0* handelt es sich um ein Fragment der Originalsequenz. Übersetzt steht *c-subsequence* für einen Teil der oberen Fragestellung:

- Welches sind die Kundennummern (*custkey*), die in den Jahren 1992, 1993 und 1994 (*orderyearkey*) einen Mindestumsatz von größer gleich 500000\$ gebracht haben?

Auch *c-subsequence* ist wie *c-sequence* in Optimierungsvarianten unterteilt, die CREATE TABLE und INSERT-Statements zusammenziehen.

Jede dieser Sequenzen ist aus mehreren CREATE TABLE, INSERT und DROP TABLE-Statements zusammengesetzt, und liegt wiederum in drei Unterschiedlichen Variationen vor. Die drei Variationen sind:

- *sequence.sql*
- *singlequery.sql*
- *with.sql*

Wie diese drei Variationen aussehen, wird hier anhand von *c-sequence-sequence_0*, der unveränderten Originalsequenz erklärt. Das *sequence.sql*-Skript stellt die Standardvariante vor und besteht für *c-sequence-sequence_0* aus 8 CREATE TABLE-Statements und ebenso vielen INSERT-Statements. Bei den ersten 7 Tabellen handelt es sich nur um temporäre Tabellen, in die mit den INSERT-Statements Zwischenergebnisse der Gesamtberechnung eingefügt werden. Zum Löschen dieser Tabellen enthält *sequence.sql* noch 8

5.2 Vorbereitungen und Durchführung

DROP TABLE-Statements. Da die Sequenzen in dieser Arbeit nur zu Performanzzwecken genutzt werden und nicht um die Ergebnisse der Fragen zu erhalten, werden nicht nur die temporäre Tabellen sondern auch die Ergebnistabelle am Ende der Sequenz wieder gelöscht. Die ersten drei Tabellen, die mit CREATE TABLE erzeugt werden sind beinahe identisch, genauso wie die dazugehörigen INSERT-Befehle. Eine der Tabellen ist hier mit ihrem dazugehörigen INSERT-Befehl abgebildet:

```
01 CREATE TABLE tpch4.temptableC010(  
02  custkey INTEGER,  
03  turnover1992 DECIMAL(31, 6)) TABLESPACE TPCH4;  
04  
05 INSERT INTO tpch4.temptableC010  
06 SELECT  
07   a1.custkey,  
08   sum(a1.endprice)  
09 FROM  
10  tpch4.lineitem_orders a1,  
11  tpch4.lookup_orderday a2  
12 WHERE  
13  a2.orderdate = a1.orderdate AND  
14  a2.orderyearkey = 1992  
15 GROUP BY  
16  a1.custkey;
```

Die drei Tabellen unterscheiden sich lediglich in der Jahreszahl. In Zeile 3 und Zeile 14 steht statt 1992 dann 1993 oder 1994. Durch das INSERT-Statement wird der Mindestumsatz des entsprechenden Jahres in die jeweilige Tabelle eingefügt.

Das 4. CREATE-Statement erzeugt eine Tabelle, welche die ersten drei zusammenfasst. Das 4. INSERT-Statement füllt diese Tabelle und überprüft ob der Mindestumsatz jeweils größer gleich 500000 ist.

Die 5. und 6. Tabelle sowie das 5. und 6. INSERT-Statement sind ebenfalls nahezu identisch. In Ihnen wird die Standardabweichung berechnet.

```
01 CREATE TABLE tpch4.temptableC050(  
02  custkey INTEGER,  
03  stddeviation DECIMAL(31, 6)) TABLESPACE TPCH4;  
04  
05 INSERT INTO tpch4.temptableC050  
06 SELECT  
07   a1.custkey,  
08   stddev(a1.endprice)  
09 FROM  
10  tpch4.lineitem_orders a1,  
11  tpch4.lookup_orderday a2,  
12  tpch4.temptableC040 a3
```

5.2 Vorbereitungen und Durchführung

```
13 WHERE
14 a2.orderdate = a1.orderdate AND
15 a2.orderyearkey IN (1992, 1993, 1994) AND
16 a1.custkey = a3.custkey
17 GROUP BY
18 a1.custkey;
```

Die beiden CREATE- und INSERT-Statements unterscheiden sich nur in den folgenden beiden Zeilen:

```
03 stddeviation FLOAT) TABLESPACE TPCH4;
08 stddev(a1.endprice)/(CASE avg(a1.endprice) WHEN
    0 THEN NULL ELSE avg(a1.endprice) END)
```

Das 7. CREATE-Statement erzeugt die letzte Zwischentabelle. Der 7. INSERT-Befehl fügt alle Daten der anderen sechs Tabellen in die 7. Tabelle ein und verbindet diese über einen Join mit der Tabelle *lookup_customer* um den Kundennamen (*customername*) zu den Kundennummern (*customerkey*) zu erfahren.

Das 8. CREATE-Statement erzeugt die Ausgabetable, in der das Gesamtergebnis ausgegeben werden soll. Das 8. INSERT-Statement führt die Daten der 7. Tabelle in die Ausgabetable und überprüft die Standardabweichungsbedingung. Die gesamte *sequence.sql*-Anweisung ist in Anhang B abgebildet.

In der *singlequery.sql*-Sequenz werden die 8 CREATE und INSERT-Statements von *sequence.sql* in einer einzigen verschachtelten INSERT-Query realisiert. Es werden keine temporäre Tabellen, sondern nur die Ausgabetable erzeugt. Das CREATE-Statement für diese Tabelle ist identisch mit dem CREATE-Statement für die 8. Tabelle von *sequence.sql* und der Ausgabetable von *with.sql*:

```
CREATE TABLE tpch4.temptableC080(
    custkey INTEGER,
    custname VARCHAR(25),
    stddev1 DECIMAL(10, 2),
    stddev2 FLOAT,
    turnover1992 FLOAT,
    turnover1993 FLOAT,
    turnover1994 FLOAT) TABLESPACE TPCH4;
```

Das verschachtelte INSERT-Statement benutzt keinerlei temporäre Tabellen, in die Zwischenergebnisse abgespeichert werden können, sondern realisiert die 8 INSERT-Befehle von *sequence.sql* in einer verschachtelten INSERT-SELECT-Anweisung. Da keine temporären Tabellen zur Verfügung stehen, kann kein Zwischenergebnis mehrfach genutzt werden, sondern muss, falls es an einer anderen Stelle ebenfalls benötigt wird, dort erneut angewendet werden. Der berechnete Mindestumsatz für die Jahre 1992, 1993 und 1994 wird für die Beantwortung der Frage in *sequence.sql* an zwei Stellen benötigt. In Tabelle 4 und Tabelle 7. Da Tabelle 4 aber ebenfalls an zwei Stellen verwendet wird, in

5.2 Vorbereitungen und Durchführung

Tabelle 5 und 6, bedeutet dies für die singlequery-Anweisung einen enormen Mehraufwand, da der Mindestumsatz für die drei Jahre jeweils dreimal und Tabelle 4 zweimal berechnet werden müssen.

Die with.sql-Variante ist eine Mischung aus den oberen beiden Sequenzen. Einerseits besteht die with.sql-Sequenz ebenso wie die singlequery.sql-Sequenz nur aus einem CREATE TABLE-Statement für die Ausgabetabelle (s.o.) und einer verschachtelten INSERT-Anweisung, andererseits werden mit Hilfe der WITH-Klausel Zwischenergebnisse in dieser Verschachtelung realisiert, die wie in sequence.sql wiederverwendet werden können. Dies eliminiert den Mehraufwand, der durch die Verschachtelung in singlequery.sql entstanden ist. Die WITH-Klausel weist einer Unteranfrage (subquery) einer verschachtelten Gesamtanfrage einen Namen (query_name) zu. Dadurch kann diese Unteranfrage, durch den zugewiesenen Namen, mehrfach in der restlichen Gesamtanfrage referenziert werden. Seit Oracle 9i kann der Query Optimizer entscheiden, ob die WITH-Klausel entweder als Inline-View oder als Temp-Tabelle aufgelöst wird, dadurch muss bei der Referenzierung einer Unteranfrage, diese nicht erneut Berechnet werden. Die allgemeine Syntax der WITH-Klausel sieht wie folgt aus:

```
WITH query_name AS (subquery) [, query_name AS (subquery) ]...
```

Die WITH-Klausel kann in jeder toplevel SELECT-Anfrage spezifiziert werden und ist auch in der INSERT INTO-Klausel verwendbar [LG03].

Alle Optimierungsvarianten (sequence_1...4) besitzen ebenfalls eine sequence-, singlequery- und with-Ausführung. Die Optimierungsvarianten unterscheiden sich, nur dadurch von den sequence_0-Ausführungen, dass in den Optimierungsvarianten jeweils mehrere Tabellen und INSERT-Statements der sequence_0-sequence.sql zusammengezogen wurden. Hier folgt eine kurze Aufstellung der Veränderungen in den Optimierungsvarianten im Vergleich zur Originalsequenz c-sequence-sequence_0-sequence.sql. Die Optimierungen der c-subsequence-Sequenzen sehen entsprechend aus. Die Optimierungsvariationen enthalten jeweils immer die Veränderungen ihrer Vorgängeroptimierungen:

- sequence_1: CREATE-Statement 5 und 6 sowie INSERT-Statement 5 und 6 werden zusammengefasst.
- sequence_2: CREATE-Statement 1, 2 und 3 sowie INSERT-Statement 1, 2 und 3 werden zusammengefasst.
- sequence_3: CREATE-Statement 4 entfällt, INSERT-Statement 4 wird mit den bereits zusammengefassten 5. und 6. INSERT-Statement zusammengefasst.
- sequence_4: CREATE-Statement 7 und 8 sowie INSERT-Statement 7 und 8 werden zusammengefasst.

Alle Varianten und Optimierungen der Sequenzen wurden im DB2-SQL-Format vorgegeben. Da diese auf der Oracle-Datenbank eingesetzt werden sollen, müssen sie unter Umständen noch an das Oracle-Format angepasst werden.

5.2.2 Umwandlung der Sequenzen in Oracle SQL

Da die DB2-SQL-Syntax und die Oracle-SQL-Syntax in verschiedenen Fällen voneinander abweichen, müssen einige der vorgegebenen DB2-Sequenzen in das Oracle-SQL-Format umgewandelt werden. Wie bereits in Abschnitt 4.2.4 *Die Tabellen im Oracle-System erstellen* gezeigt, unterscheidet sich die CREATE TABLE-Syntax von Oracle von der von DB2. In jeder der vorgegebenen Sequenzen müssen die Tablespace-Zuweisungen der CREATE TABLE-Statements wie in Abschnitt 4.2.4 erklärt, verändert werden. Die nichtverschalteten INSERT-Anweisungen der sequence.sql-Dateien können eins zu eins übernommen werden, die entsprechenden Sequenzen sind nach der CREATE TABLE-Anpassung bereits lauffähig. Die singlequery.sql- und with.sql-Dateien müssen noch weitergehend verändert werden.

Um die verschachtelten INSERT-Anweisungen der singlequery.sql-Sequenzen in das Oracle-Format umzuschreiben, müssen einige kleine Veränderungen vorgenommen werden, da sich die Syntax für SELECT-Anweisung in den beiden Systemen ebenfalls unterscheidet. Als Beispiel ist hier ein Ausschnitt aus sequence_0-singlequery.sql im DB2-Format aufgeführt:

```

01 ...
02 FROM
03     (SELECT
04         a1.custkey,
05         sum(a1.endprice)
06     FROM
07         tpch4.lineitem_orders a1,
08         tpch4.lookup_orderday a2
09     WHERE
10         a2.orderdate = a1.orderdate AND
11         a2.orderyearkey = 1992
12     GROUP BY
13         a1.custkey
14     ) AS a1 (custkey, turnover1992),...
```

Es unterscheidet sich in der Positionierung der Spalten- und des Tabellenaliases vom gleichen Ausschnitt im Oracle-Format:

```

01 ...
02 FROM
03     (SELECT
04         a1.custkey custkey,
05         sum(a1.endprice) turnover1992
06     FROM
07         tpch4.lineitem_orders a1,
08         tpch4.lookup_orderday a2
09     WHERE
```

5.2 Vorbereitungen und Durchführung

```
10         a2.orderdate = a1.orderdate AND
11         a2.orderyearkey = 1992
12     GROUP BY
13         a1.custkey
14     ) a1,...
```

In Zeile 14 des DB2-Ausschnitts ist zu sehen, dass der Tabellenalias der geklammerten SELECT-Anweisung durch AS eingeleitet wird. Dies muss bei Oracle weggelassen werden. Hinter dem Tabellenalias folgen in DB2 in einer Klammer, die Ersatznamen (Aliasse) der Spaltenattribute, die durch die SELECT-Anweisung ausgewählt wurden. Diese Spaltenaliasse müssen in Oracle direkt hinter den Spaltenattributen (Zeile 4+5) angegeben werden. Damit die singlequery-Skripte unter Oracle lauffähig werden, müssen die Umwandlung jeweils für das gesamte INSERT-Statement durchgeführt werden.

Auch die with-Sequenzen müssen angepasst werden, da sich die Syntax der WITH-Klausel in DB2 von der in Oracle unterscheidet. Im Folgenden ist hier ein Ausschnitt aus der INSERT-Anweisung der DB2-with.sql-Sequenz angegeben

```
01 INSERT INTO tpch4.temptableC080
02 WITH
03     temptableC010 (custkey, turnover1992) AS (
04     SELECT
05         a1.custkey,
06         sum(a1.endprice)
07     FROM
08     ...
```

Die DB2-Syntax unterscheidet sich in der Positionierung der Spaltenaliasse in Zeile 3. Die Oracle-Schreibweise setzt den Alias wieder direkt hinter das jeweilige Spaltenattribut in Zeile 5 und 6:

```
01 INSERT INTO tpch4.temptableC080
02 WITH
03     temptableC010 AS (
04     SELECT
05         a1.custkey custkey,
06         sum(a1.endprice) turnover1992
07     FROM
08     ...
```

Wenn dies für die gesamte INSERT-Anweisung durchgeführt wird, ist auch die with-Sequenz in Oracle ausführbar und die Sequenzen können für die Performance-Tests eingesetzt werden.

5.2.3 Anwendung der Analysewerkzeuge

Wie in Abschnitt 5.1.1 *Explain Plan* erklärt, kann mit Hilfe von EXPLAIN PLAN, der Ablaufplan einer SQL-Sequenz angezeigt werden, wie dieser vom Query Optimizer berechnet wurde. Dabei verwendet der Query Optimizer Statistiken wie sie in Kapitel 7 *Manipulationsmöglichkeiten* vorgestellt werden. Für jede der vorgegebenen Sequenzen wurde mit EXPLAIN PLAN ein Ausführungsplan erstellt und mit dem SPOOL-Befehl in eine Datei in ihrem jeweiligen Speicherpfad unter folgendem Namen abgespeichert (s. Abschnitt 5.1.1):

```
<skriptvariante>output-explainplan.LST
```

<skriptvariante> entspricht sequence, singlequery oder with. Zur Durchführung von EXPLAIN PLAN musste jedem der Originalskripte ein SPOOL-Befehl hinzugefügt und jedes INSERT in ein EXPLAIN PLAN-Befehl eingebettet werden. Um die Originalskripte nicht zu verändern wurden diese veränderten Skripte unter folgendem Namen abgespeichert:

```
<skriptvariante>-explainplan.sql
```

Da ein EXPLAIN PLAN-Befehl nur auf ein einzelne INSERT-Statements bezogen werden können, müssen die sequence-explainplan-Skripte besonders behandelt werden, da diese mehr als ein INSERT enthalten. Jede dieser INSERT-Anweisungen muss durch einen separaten EXPLAIN PLAN-Befehl analysiert werden. Da diese INSERT-Anweisungen jeweils auf die Ergebnisse ihrer Vorgänger-INSERTS aufbauen, ist die einfache Einbettung der INSERT-Anweisungen in EXPLAIN PLAN-Anweisungen aber problematisch. Im Beispiel von *c-sequence-sequence_0-sequence.sql*, das wie in Abschnitt 5.2.1 *Vorgegebene Sequenzen* gezeigt, 8 INSERT-Statements enthält, greift das 4. INSERT-Statement auf die Tabellen zu, die von den ersten drei INSERTs gefüllt wurden. Wenn diese drei INSERT-Anweisungen in ein EXPLAIN PLAN-Statement eingebettet sind, werden diese nur analysiert und nicht ausgeführt. Deshalb findet das Auffüllen der temporären Tabellen garnicht statt. Wenn jetzt das 4. INSERT mittels EXPLAIN PLAN analysiert wird, greift dieses auf drei leere und nicht wie bei der realen Ausführung auf die gefüllten Tabellen zu, was die Berechnungen von EXPLAIN PLAN verfälscht. Dieses Problem existiert für jede der sequence.sql-Skripte, und kann nur dadurch umgangen werden, dass die Vorgänger-INSERTS wirklich ausgeführt werden, bevor die Nachfolger-INSERTS durch EXPLAIN PLAN bearbeitet werden. Durch diese Vorgehensweise geht EXPLAIN PLAN jedoch der Vorteil verloren, dass die zu analysierenden Skripte nicht ausgeführt werden müssen. Weshalb für diese Vorgehensweise zusätzlich zu den oberen sequence-explainplan.sql-Skripten jeweils ein Skript mit dem folgenden Namen abgespeichert wurde:

```
sequence-explainplan-allinserts.sql
```

Des Weiteren wurde ein Skript erstellt, das jeweils die sequence-, singlequery- und with-Explainplan-Skripte aufruft. Dieses Skript wurde für jede Optimierungsvariante erstellt und unter *start-explainplan.sql* abgespeichert. Zwei weitere Skript mit den Namen

5.3 Auswertung der Leistungsmessungen

`start-sequence-explainplan.sql` und `start-subsequence-explainplan.sql` wurden erstellt. Diese Skripte rufen alle `start-explainplan`-Skripte in `c-sequence` bzw. `c-subsequence` auf.

Um genaue Aussagen über den Ablauf der Sequence-Skripte treffen zu können und um die Schätzungen von EXPLAIN PLAN mit reellen Zahlen vergleichen zu können, wurden mit SQL Trace und TKPROF wie in Abschnitt 5.1.2 *SQL Trace und TKPROF* beschrieben, TKPROF-Dateien für die drei Sequenzvariationen in `c-sequence-sequence_0` und `c-subsequence-sequence_0` angelegt. Diese wurden mit Hilfe des Skripts `start-sqltrace.sql`, in `c:\statistiken` gespeichert und danach mit dem TKPROF-Kommandozeilenwerkzeug umgewandelt. Das erzeugte Skript macht nichts anderes, als die Schritte die in Abschnitt 5.1.2 beschrieben wurden, für alle Skriptvarianten in den beiden, oben erwähnten Kategorien durchzuführen.

Das in Abschnitt 5.1.3 *Laufzeitmessungen* vorgestellte TIMING-Werkzeug wurde dazu benutzt, einfache Laufzeitmessungen für alle vorgegebenen Sequenzen zu erstellen. Damit eine bessere Aussage über die allgemeine Laufzeit der einzelnen Sequenzen getroffen werden konnte, wurden die Sequenzen jeweils 3 Mal gemessen und der Schnitt aus diesen drei Ausführung als Laufzeitwert genutzt. Zu diesem Zweck wurden wieder mehrere Skripte erstellt, die alle Skripte dreimal nacheinander aufrufen und mittels SPOOL, die Ausgabe in Dateien ausschreiben. Die Skripte haben die gleiche Namensgebung wie die Explain Plan-Skripte, nur dass die TIMING-Skripte anstatt 'explainplan' den Zusatz 'test' enthalten. Die TIMING-Ausgabedateien wurden wie die Explain Plan-Skripte in ihrem jeweiligen Versionsordner (`sequence_0...4`) unter folgendem Namen abgespeichert:

```
skriptvariante>output.LST
```

5.3 Auswertung der Leistungsmessungen

Wenn man die unterschiedlichen Ergebnisse der drei verschiedenen Analysewerkzeuge betrachtet und in Beziehung bringt, können mehrere Aussagen über die Leistungsmessungen und die Effizienz der einzelnen Werkzeuge getroffen werden. Mit Hilfe des Explain Plan-Werkzeugs können ohne die Sequenzen auszuführen bereits erste Aussagen über deren Verhalten getroffen werden. Abbildung 5.1 zeigt die Größe bzw. Komplexität des Ausführungsplans anhand der Anzahl der Ausführungsschritte des `PLAN_TABLES` für die drei unterschiedlichen Varianten der `c-sequence`-Skripte. Die einzelnen Ausführungsschritte des Plans sagen nichts über die Ausführungsdauer der einzelnen Skripte aus, da ein Schritt länger als hundert andere sein kann, aber auch umgekehrt. Trotzdem gibt die Grafik Übersicht darüber, dass die Optimierungen keinerlei Einfluss auf die Länge der `singlequery`-Ausführungspläne haben. Des Weiteren zeigt die Grafik, dass vor allem die ersten beiden Optimierungsschritte Einsparung für die Schrittzahl der `sequence`- und der `with`-Skripte ergeben. Ob dies auch Einsparung in der Ausführungszeit zur Folge hat, muss mit den TIMING-Laufzeitmessungen überprüft werden. Um

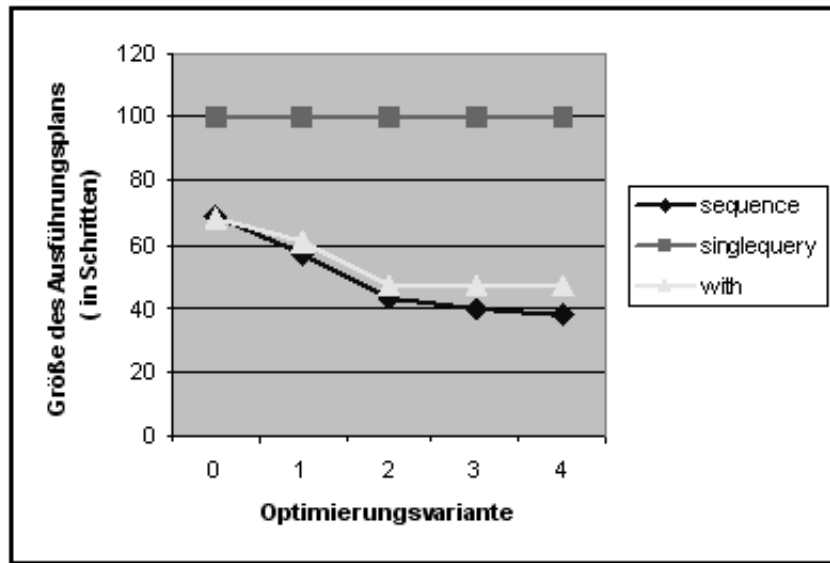


Abbildung 5.1: Größe der Ausführungspläne in Ausführungsschritten für *c-sequence* im *EXPLAIN PLAN*

einen Eindruck über die Art der Ausführungspläne der *sequence*-, *singlequery*- und *with*-Varianten zu bekommen, werden im Folgenden die *EXPLAIN PLAN-PLAN_TABLE*-Ausgaben der *c-subsequence-sequence_1*-Skripte betrachtet. Es wird hier auf das *c-subsequence*-Fragment zurückgegriffen, da der Ausführungsplan entsprechend kürzer ist als der Ausführungsplan der *c-sequence*-Skripte. Es wird auf die erste Optimierungsvariante zurückgegriffen, da die Ausführungspläne für das *singlequery*- und das *with*-Skript in *c-subsequence-sequence_0* identisch sind. Warum dies so ist wird später erläutert. Zum Vergleich ist hier das zweite *INSERT*-Statement der *sequence*-Variante einmal mit der *sequence-explainplan.sql*-Berechnung und einmal mit der *sequence-explainplan-allinserts.sql*-Berechnung aufgeführt. Der Unterschied in den Schätzwerten ist deutlich zu erkennen, vorallem in der Zeilenschätzung ist zu erkennen, dass bei der *allinserts*-Variante die gefüllte Zwischentabelle und nicht nur eine einzelne Zeile wie bei der anderen Variante verwendet wird. Da die *allinserts*-Ausführungspläne die unter „Realbedingungen“ berechneten Ausführungspläne darstellen, wird hier für den späteren vergleich mit *TKPROF* und *TIMING* auch diese Variante herangezogen. Trotzdem ist zu sagen, dass durch die *allinserts*-Berechnung ein Teil des *EXPLAIN PLAN*-Vorteils verloren geht, da in diesem Fall das 1. *INSERT*-Statement berechnet werden muss um die korrekte Aussage für das zweite Statement zu bekommen.

`PLAN_TABLE_OUTPUT SEQUENCE`

5.3 Auswertung der Leistungsmessungen

Id	Operation	Name	Rows	Bytes	TempSpc	Cost(%CPU)	Time
0	INSERT STATEMENT		2121K	86M		562K (2)	01:52:35
1	SORT GROUP BY		2121K	86M	1499M	562K (2)	01:52:35
* 2	HASH JOIN		25M	1054M	251M	346K (1)	01:09:24
* 3	HASH JOIN		6428K	177M		47063 (2)	00:09:25
* 4	TABLE ACCESS FULL	LOOKUP_ORDERDAY	1031	12372		6 (0)	00:00:01
5	TABLE ACCESS FULL	ORDERS	15M	243M		46918 (1)	00:09:24
6	TABLE ACCESS FULL	LINEITEM	59M	800M		212K (1)	00:42:30
0	INSERT STATEMENT		1	117		7 (15)	00:00:01
* 1	HASH JOIN		1	117		7 (15)	00:00:01
2	MERGE JOIN CARTESIAN		1	78		4 (0)	00:00:01
* 3	TABLE ACCESS FULL	TEMPTABLEC010	1	39		2 (0)	00:00:01
4	BUFFER SORT		1	39		2 (0)	00:00:01
* 5	TABLE ACCESS FULL	TEMPTABLEC010	1	39		2 (0)	00:00:01
* 6	TABLE ACCESS FULL	TEMPTABLEC010	1	39		2 (0)	00:00:01

PLAN_TABLE_OUTPUT SEQUENCE ALL_INSERTS

Id	Operation	Name	Rows	Bytes	TempSpc	Cost(%CPU)	Time
0	INSERT STATEMENT		216M	23G		36189 (1)	00:07:15
* 1	HASH JOIN		216M	23G	11M	36189 (1)	00:07:15
* 2	TABLE ACCESS FULL	TEMPTABLEC010	237K	9030K		1610 (4)	00:00:20
* 3	HASH JOIN		6938K	516M	11M	4337 (3)	00:00:53
* 4	TABLE ACCESS FULL	TEMPTABLEC010	228K	8694K		1610 (4)	00:00:20
* 5	TABLE ACCESS FULL	TEMPTABLEC010	231K	8801		1610 (4)	00:00:20

Der Ausführungsplan für die sequence-Variante zeigt, dass hier im ersten INSERT-Statement jeweils einmal auf die Tabellen LINEITEM, ORDERS und LOOKUP_ORDERDAY zugegriffen wird. Das Ergebnis wird in TEMPTABLEC010 abgespeichert, der dann dreimal wieder verwendet wird. Im Gegensatz dazu wird im Ausführungsplan des singlequery-Skripts dreimal der gleiche Zugriff auf LOOKUP_ORDERDAY, ORDERS und die sehr große Tabelle LINEITEM durchgeführt. Dies führt zu einem erheblichen Mehraufwand, der sich auch in der Time- und der Cost-Spalte widerspiegelt. Es wird aber bereits jetzt davon ausgegangen, dass die Time-Spalte nicht die reale Ausführungsdauer der einzelnen Operatoren widerspiegelt, da manche Operationen des 1. INSERT-Statements als viel zu lange eingeschätzt werden. Dies wird später mit dem TKPROF und dem TIMING-Befehl überprüft.

PLAN_TABLE_OUTPUT SINGLEQUERY

Id	Operation	Name	Rows	Bytes	TempSpc	Cost(%CPU)	Time
0	INSERT STATEMENT		35355	4039K		1118K (2)	03:43:43
* 1	HASH JOIN		35355	4039K	1768K	1118K (2)	03:43:43
2	VIEW		35355	1346K		372K (2)	01:14:33
* 3	FILTER						
4	SORT GROUP BY		35355	1484K	463M	372K (2)	01:14:33
* 5	HASH JOIN		8569K	351M	83M	338K (1)	01:07:44

5.3 Auswertung der Leistungsmessungen

* 6	HASH JOIN		2142K	59M		47063 (2)	00:09:25
* 7	TABLE ACCESS FULL	LOOKUP_ORDERDAY	344	4128		6 (0)	00:00:01
8	TABLE ACCESS FULL	ORDERS	15M	243M		46918 (1)	00:09:24
9	TABLE ACCESS FULL	LINEITEM	59M	800M		212K (1)	00:42:30
*10	HASH JOIN		35355	2693K	1768K	745K (2)	02:29:08
11	VIEW		35355	1346K		372K (2)	01:14:33
*12	FILTER						
13	SORT GROUP BY		35355	1484K	463M	372K (2)	01:14:33
*14	HASH JOIN		8569K	351M	83M	338K (1)	01:07:44
*15	HASH JOIN		2142K	59M		47063 (2)	00:09:25
*16	TABLE ACCESS FULL	LOOKUP_ORDERDAY	344	4128		6 (0)	00:00:01
17	TABLE ACCESS FULL	ORDERS	15M	243M		46918 (1)	00:09:24
18	TABLE ACCESS FULL	LINEITEM	59M	800M		212K (1)	00:42:30
19	VIEW		35355	1346K		372K (2)	01:14:33
*20	FILTER						
21	SORT GROUP BY		35355	1484K	463M	372K (2)	01:14:33
*22	HASH JOIN		8569K	351M	83M	338K (1)	01:07:44
*23	HASH JOIN		2142K	59M		47063 (2)	00:09:25
*24	TABLE ACCESS FULL	LOOKUP_ORDERDAY	344	4128		6 (0)	00:00:01
25	TABLE ACCESS FULL	ORDERS	15M	243M		46918 (1)	00:09:24
26	TABLE ACCESS FULL	LINEITEM	59M	800M		212K (1)	00:42:30

Durch den Ausführungsplan der with-Variante wird deutlich, dass hier die 3 großen Tabellen in einer Temp-Tabelle (SYS_TEMP_0FD9D6600) gespeichert werden, die dreimal wiederverwendet wird. Das Verhalten von with ist dem von sequence sehr ähnlich, Zeile 1-6 des ersten INSERT-Statements bei sequence entsprechen Zeile 3-8 bei with. Erst der Zugriff auf die temporäre Tabellen wird unterschiedlich realisiert. Was im Ausführungsplan für sequence.sql als normaler Tabellenzugriff durchgeführt wird, ist bei with als Inline-View realisiert.

PLAN_TABLE_OUTPUT WITH

Id	Operation	Name	Rows	Bytes	TempSpC	Cost(%CPU)	Time
0	INSERT STATEMENT		9545K	1065M		604K (2)	02:00:58
2	LOAD AS SELECT						
3	SORT GROUP BY		2121K	86M	1499M	562K (2)	01:52:35
* 4	HASH JOIN		25M	1054M	251M	346K (1)	01:09:24
* 5	HASH JOIN		6428K	177M		47063 (2)	00:09:25
* 6	TABLE ACCESS FULL	LOOKUP_ORDERDAY	1031	12372		6 (0)	00:00:01
7	TABLE ACCESS FULL	ORDERS	15M	243M		46918 (1)	00:09:24
8	TABLE ACCESS FULL	LINEITEM	59M	800M		212K (1)	00:42:30
* 9	HASH JOIN		9545K	1065M	103M	41949 (1)	00:08:24
*10	VIEW		2121K	78M		2420 (2)	00:00:30
11	TABLE ACCESS FULL	SYS_TEMP_0FD9D6600	2121K	44M		2420 (2)	00:00:30
*12	HASH JOIN		4499K	334M	103M	15143 (1)	00:03:02
*13	VIEW		2121K	78M		2420 (2)	00:00:30
14	TABLE ACCESS FULL	SYS_TEMP_0FD9D6600	2121K	44M		2420 (2)	00:00:30
*15	VIEW		2121K	78M		2420 (2)	00:00:30
16	TABLE ACCESS FULL	SYS_TEMP_0FD9D6600	2121K	44M		2420 (2)	00:00:30

5.3 Auswertung der Leistungsmessungen

Mit Hilfe der Ausführungspläne lassen sich also bereits erste Thesen und Vermutungen aufstellen, die durch die anderen beiden Schnittstellen überprüft werden können. Die folgenden Thesen werden aufgestellt:

1. Durch die sehr ähnlichen Ausführungspläne der sequence- und der with-Variante verhalten sich diese sehr ähnlich. Dadurch sollten sie fast die gleiche Ausführungsdauer besitzen.
2. Durch die Wiederverwendung der temporäre Tabellen sind die sequence- und die with-Variante der singlequery-Variante bezüglich der Ausführungsdauer überlegen. Die Ausführung von singlequery.sql sollte sehr viel länger als bei den anderen beiden Varianten dauern.
3. Die Zeitschätzungen in den Ausführungsplänen sind sehr ungenau.

Die beiden ersten Annahmen können mittels der Laufzeitmessungen die mit dem TIMING-Werkzeug durchgeführt wurden überprüft werden. Die Ergebnisse dieser Laufzeitmessungen sind in Abbildung 5.2 und 5.3 dargestellt. Die dritte These kann mit Hilfe von SQL-Trace und TKPROF beurteilt werden. Abbildung 5.2 zeigt deutlich, dass die 1. These

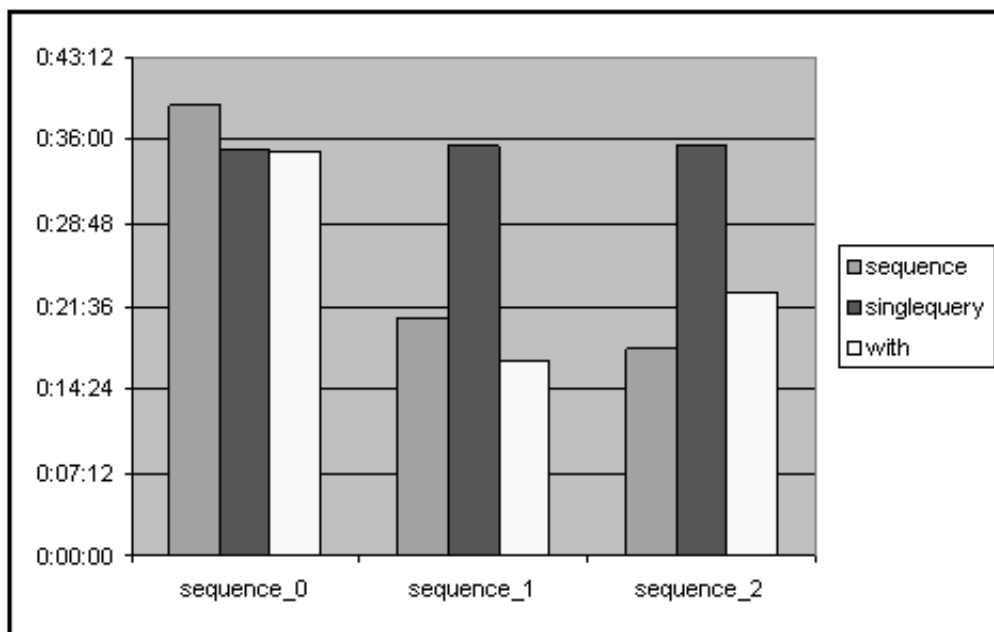


Abbildung 5.2: Ausführungszeiten c-subsequence

zutrifft und das Verhalten und die Ausführungsdauer der beiden Varianten sehr ähnlich ist. Auch die 2. These wird für die 1. und 2. Optimierungsvariante bestätigt. Der Grund dafür, dass sich die Ausführungsdauer der sequence_0-Sequenzen nicht unterscheidet ist,

5.3 Auswertung der Leistungsmessungen

dass dort noch keine Tabellen zusammengezogen wurden, und somit keine der temporäre Tabellen mehrfach verwendet werden kann. Dadurch kommen die Vorteile der sequence- und with-Skripte gegenüber der singlequery-Ausführung nicht zum Tragen. Dies führt zu dem Schluss, dass die Optimierungen hier sehr sinnvoll sind, und auf jeden Fall der Originalversion vorgezogen werden sollte.

Abbildung 5.3 zeigt, dass These 1 und 2 auch auf die c-sequence-Skripte zutrifft und zeigt eindeutig den Mehraufwand der singlequery-Variante, der sich sogar noch vergrößert hat. Die Gründe hierfür sind bereits in Abschnitt 5.2.1 *Vorgegebene Sequenzen* angesprochen worden. Des weiteren zeigt der konstant hohe Zeitaufwand für die singlequery-Skripte, dass keine der Optimierungen einen zeitlichen Vorteil für die verschachtelten INSERT-Befehle erreicht. Was mit der Aussage von Abbildung 5.1 für c-subsequence übereinstimmt. Die TIMING-Messungen zeigen weiter, dass die Ausführungsdauer der sequence-Skripte in der 2. Optimierungsvariante optimal ist. Dies zeigt, dass der dritte Optimierungsschritt (s.Abschnitt 5.2.1) einen Rückschritt bedeutet und das Zusammenziehen des 4. INSERT-Statement mit den bereits zusammengefassten 5. und 6. INSERT-Statements, im Falle von sequence.sql zu einem Mehraufwand führt. Zwischen Optimierung 3 und 4 sind keine wesentlichen Unterschiede festzustellen, die 4. Optimierung ist also unnötig. In der 2. Optimierungsvariante zeigt sich der Unterschied zwischen der sequence- und der with-Realisierung. Dieser Unterschied muss vor allem im zweiten Teil von c-sequence zum Tragen kommen, da die c-subsequence-Ergebnisse noch weitgehend gleich sind. Um

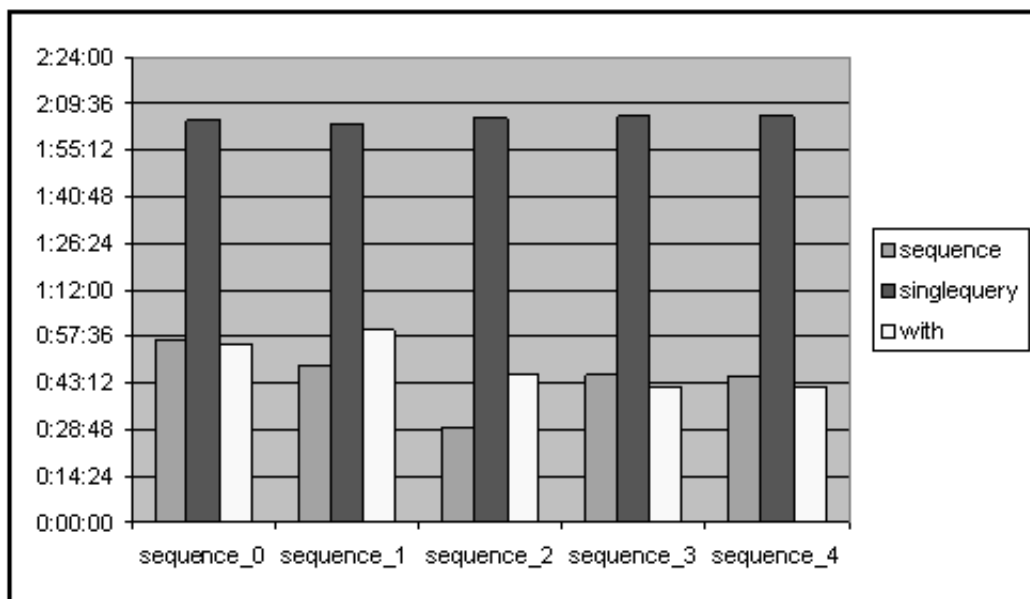


Abbildung 5.3: Ausführungszeiten c-sequence

5.3 Auswertung der Leistungsmessungen

die 3. der oben angegebenen Thesen zu überprüfen werden die detaillierteren Laufzeitergebnisse von TKPROF mit den Schätzungen des Explain Plan-Tools verglichen. Bereits wenn man die mittels TIMING gemessenen Zeiten mit den geschätzten Zeiten vergleicht, wird deutlich, wie stark sich die Schätzungen von den realen Zeiten unterscheiden. Hier sind als Beispiel die geschätzten und gemessenen Zeiten für die c-subsequence-sequence_1-Skripte, die bereits weiter oben als Beispiel gedient haben, tabellarisch aufgeführt. Der

Skriptvariante	gemessene Gesamtzeit	geschätzte Gesamtzeit	Unterschied
sequence.sql	00:21:07	01:59:50	5,7mal höher geschätzt
singlequery.sql	00:35:31	03:43:43	6,3mal höher geschätzt
with.sql	00:16:47	02:00:58	7,2mal höher geschätzt

Tabelle 5.2: Vergleich der geschätzten und der realen Ausführungsdauer der c-subsequence-sequence_1-Skripte

folgende angepasste Ausschnitt aus der TKPROF-Datei ermöglicht einen genauen Vergleich mit dem weiter oben angegebenen PLAN_TABLE, und ermöglicht eine Aussage zur 3. These zu treffen:

```
TKPROF-Datei: c-subsequence-sequence_1:sequence.sql
1. INSERT-Statement
Rows      Row Source Operation
-----
 868074   SORT GROUP BY (cr=1336166 pr=1499210 pw=163137 time=12:23min:sec)
9117084   HASH JOIN (cr=1336166 pr=1493153 pw=157080 time=11:45min:sec)
2281205   HASH JOIN (cr=241242 pr=241192 pw=0 time=01:54min:sec)
   1098    TABLE ACCESS FULL LOOKUP_ORDERDAY (cr=23 pr=0 pw=0 time=00:00min:sec)
15000000  TABLE ACCESS FULL ORDERS (cr=241219 pr=241192 pw=0 time=01:30min:sec)
59986052  TABLE ACCESS FULL LINEITEM (cr=1094924 pr=1094881 pw=0 time=08:00min:sec)
2. INSERT-Statement
Rows      Row Source Operation
-----
 32492    HASH JOIN (cr=6606 pr=806 pw=806 time=00:02min:sec)
257415    TABLE ACCESS FULL TEMPTABLEC010 (cr=2202 pr=0 pw=0 time=00:00min:sec)
 86134    HASH JOIN (cr=4404 pr=403 pw=403 time=00:01min:sec)
257572    TABLE ACCESS FULL TEMPTABLEC010 (cr=2202 pr=0 pw=0 time=00:00min:sec)
257244    TABLE ACCESS FULL TEMPTABLEC010 (cr=2202 pr=0 pw=0 time=00:00min:sec)
```

Die time-Werte geben die reale Ausführungsdauer der einzelnen Operationen an. Sie wurden zur besseren Anschaulichkeit in Minuten:Sekunden umgerechnet, normalerweise liegen diese in Mikrosekunden vor. Alle Operationen die mit 00:00 angegeben wurden, benötigen nicht einmal eine Sekunde zur Ausführung. Wenn man die einzelnen Zeiten vergleicht, ist zu erkennen, dass sich EXPLAIN PLAN bei jeder Operation extrem nach oben verschätzt. Die Zeitschätzungen von EXPLAIN PLAN sind also mit Vorsicht zu genießen. Die Zeilenschätzungen werden in Tabelle 5.3 mit den real-bearbeiteten Zeilen verglichen. Bis auf die Schätzungen der FULL TABLE-Zugriffe auf die Tabellen LOOKUP_ORDERDAY, ORDERS, LINEITEM und die temporäre Tabelle sind auch diese

5.3 Auswertung der Leistungsmessungen

Schätzungen sehr ungenau und lassen keine weiteren Schlüsse zu.

Alles in Allem kann man mit Hilfe von EXPLAIN PLAN bereits einige Aussagen über die auszuführenden SQL-Skripte treffen. Durch die Verwendung von TIMING und TKPROF können dann sehr genaue Schlüsse über die Ausführungsdauer, den Ausführungsplan und die Kosten der einzelnen Operationen getroffen werden. Das Beispiel der hier vorgegebenen SQL-Anfrage und deren Optimierungsversuchen zeigt, dass sich die vorgestellten Analysewerkzeuge gut eignen um Performance-Tests von regelmäßig durchgeführten, komplexen SQL-Anweisungen durchzuführen und um deren Optimierungsversuche zu bewerten. Um die Ausführungspläne der SQL-Anweisungen beeinflussen zu können stellt Oracle die im nächsten Kapitel vorgestellten Optimizer Hints zur Verfügung.

Operation	Schätzung (S)	Realer Wert	Abweichung
SORT GROUP BY	2121K	868074	S 2,4 mal höher
HASH JOIN	25M	9117084	S 2,7 mal höher
HASH JOIN	6428K	2281205	S 2,8 mal höher
TABLE ACCESS FULL LOOKUP_ORDERDAY	1031	1098	S sehr genau
TABLE ACCESS FULL ORDERS	15M	15000000	S exakt
TABLE ACCESS FULL LINEITEM	59M	59986052	S sehr genau
HASH JOIN	216M	32492	S 6648 Mal höher
TABLE ACCESS FULL TEMPTABLEC01O	237K	257415	S sehr genau
HASH JOIN	6938K	86134	S 804 mal höher
TABLE ACCESS FULL TEMPTABLEC01O	228K	257572	S sehr genau
TABLE ACCESS FULL TEMPTABLEC01O	231K	257244	S sehr genau

Tabelle 5.3: Vergleich der Zeilenschätzungen von Explain Plan mit den realen Werten von TKPROF für c-subsequence-sequence_1-sequence.sql

6 Optimizer Hints

In diesem Kapitel wird auf die Optimierungsmöglichkeiten für SQL-Statements in Oracle eingegangen [Moo03]. Es wird untersucht, inwieweit man eine bestimmte Join-Abfolge für ein Oracle-SQL-Statement mit mehreren Joins erzwingen bzw. beeinflussen kann.

Normalerweise wird jede SQL-Anfrage, die an das Oracle-Datenbanksystem gestellt wird, vom Oracle Query Optimizer (Anfrageoptimierer) bearbeitet, um den bestmöglichen Ausführungsplan für diese Anfrage zu erreichen[Moo03]. Diese Optimierung läuft vollautomatisch ab und basiert auf Systemstatistiken und Schätzungen der Ausführungskosten für eine SQL-Anfrage. Dabei kann die Ausführungsreihenfolge der Operatoren von der Auslastung des Datenbanksystems beeinflusst werden, da durch überlastete Systembereiche höhere Kosten für die darauf zugreifenden Operatoren entstehen können. Diese Optimierungen können die Ausführungsdauer einer SQL-Anfrage stark beschleunigen. Da diese Kostenschätzverfahren nicht perfekt sind, kann es aber auch zu Fehloptimierungen kommen, die eine unnötig lange Ausführungsdauer zur Folge haben können. Um diese Fehloptimierungen zu vermeiden, bietet Oracle ein Hilfsmittel an, mit dem man dem Query Optimizer manuell Hinweise geben kann, um die Ausführungsreihenfolge der SQL-Befehle zu beeinflussen.

Die Optimizer Hints von Oracle ermöglichen es, Entscheidungen für den Query Optimizer zu treffen, oder diesen sogar ganz zu übergehen. Die Hints bieten eine Möglichkeit, einen bestimmten Ausführungsplan der SQL-Anfrage zu erzwingen, oder mit Hilfe von Hintergrundwissen über den Datenbankaufbau, die Schätzverfahren des Query Optimizers zu verbessern.

Die Oracle Hints lassen sich nach Art und Verwendungszweck kategorisieren. Die folgende Liste zeigt zu welchen Kategorien Oracle Hints zur Verfügung stellt.

- Hints für Optimierungsansätze und -ziele
- Hints zur Bestimmung des Zugriffspfad
- Hints zur Beeinflussung der Anfragetransformation
- Hints zur Festlegung der Join-Reihenfolge
- Hints zur Festlegung der Join-Methode
- Hints zur parallelen Anfrageausführung
- Weitere Hints

Hier wird nur genauer auf die allgemeine Syntax der Hints und die Hints zum Join-Operator eingegangen. Eine detaillierte Liste aller anderen Hint-Varianten ist im Kapitel 17 des Oracle Performance Tuning Guides [Moo03] zu finden.

6.1 Allgemeine Hints Syntax

Ein Hint wird in Oracle-SQL innerhalb eines Kommentars eingefügt. Die allgemeine Syntax der einzelnen Hints lässt sich in zwei unterschiedlichen Varianten schreiben, da es zwei unterschiedliche Kommentarschreibweisen im Oracle-SQL gibt. Die folgenden beiden Zeilen zeigen die unterschiedliche Schreibweise der allgemeinen Syntax.

```
{DELETE|INSERT|MERGE|SELECT|UPDATE} /*+ hint [text] [hint[text]]... */
```

oder

```
{DELETE|INSERT|MERGE|SELECT|UPDATE} --+ hint [text] [hint[text]]...
```

Ein Hint kann in Oracle nur nach den Angegebenen SQL-Befehlen stehen und kann sich auf 4 unterschiedliche Bereiche beziehen. Das Pluszeichen direkt hinter der Kommentarinleitung signalisiert dem Oracle-System, dass es sich hierbei um einen Kommentar mit Hints handelt.

Entweder der Hint bezieht sich auf eine einzelne Tabelle (Single-table), auf mehrere Tabellen (Multi-table), auf einen Anfrageblock (Query block) oder auf die gesamte SQL-Anfrage (Statement).

In einer Kommentaranweisung können mehrere Hints stehen, die mindestens durch ein Leerzeichen voneinander getrennt sein müssen. Wenn widersprüchliche oder syntaktisch falsche Hints angegeben werden, ignoriert der Query Optimizer diese Hints, beachtet aber sofern vorhanden, die korrekten Hints weiterhin.

Zusammen mit den Hints kann auch normaler Kommentartext [text] angegeben werden, der ebenfalls durch mindestens ein Leerzeichen von den Hints abgegrenzt werden muss. Damit sich ein Hint auf einen Anfrageblock beziehen kann, muss erst der interne Oracle-Name für den Anfrageblock herausgefunden werden. Dies kann mittels Explain Plan in Erfahrung gebracht werden. Alternativ kann einem Anfrageblock mit dem Hint QB_NAME(Queryblock_name) ein Name zugewiesen werden, der dann in einem anderen Hint durch @Queryblock_name referenziert werden kann.[Moo03]

6.2 Hints zur Festlegung der Join-Reihenfolge

Ein wichtiger Verwendungszweck für Hints ist die Festlegung einer bestimmten Join-Reihenfolge für die zu durchsuchenden Datenbanktabellen. Der Join- oder Verbundoperator ist ein Zusammenschluss zweier Tabellen oder Views, der durchgeführt werden muss, sobald mehr als eine Tabelle hinter der FROM Klausel der SQL-Anfrage angegeben wurde.

6.2 Hints zur Festlegung der Join-Reihenfolge

Ein Join hat zur Aufgabe, alle Zeilen der zwei beteiligten Tabellen oder Views zusammenzuführen. Grob gesagt handelt es sich dabei um die Bildung des Kartesischen Produkts der beiden Tabellen, eingeschränkt durch eine Join-Bedingung, die zwischen einer Spalte aus jeder der beteiligten Tabellen besteht.

Wenn mehr als zwei Tabellen verbunden werden müssen, werden diese nacheinander jeweils in Zweierpaaren mit dem Join-Operator verbunden. Es werden nur die Zeilen zusammengeführt, die die Join-Bedingung hinter der WHERE-Klausel erfüllen. Umso mehr Zeilen die Join-Bedingung nicht erfüllen, desto weniger Zeilen müssen insgesamt verarbeitet werden. Es ist also optimal für die Ausführungsdauer der SQL-Anfrage, wenn möglichst viele Zeilen die WHERE-Bedingung nicht erfüllen. Bei einer Beteiligung von mehr als zwei Tabellen, ist es wichtig, möglichst früh die Tabellen zu verbinden, bei denen die meisten Zeilen herausfallen. Dadurch müssen diese Zeilen nicht unnötiger Weise in weiteren Joins verarbeitet werden, um dann erst am Schluss herauszufallen.[LG03] Aufgabe des Query Optimizers ist es, genau diese optimale Join-Reihenfolge herauszufinden und durchzuführen. Wenn der Anfragensteller Wissen über die beteiligten Tabellen besitzt, kann er dem Optimizer mit Hilfe der Hints eine Join-Reihenfolge vorschreiben. Hierfür bietet Oracle zwei unterschiedliche Hints an, die Einfluss auf die Join-Reihenfolge nehmen.

1. LEADING HINT
2. ORDERED HINT

Der LEADING HINT gibt dem Optimizer die Tabellen an, die als erstes verbunden werden sollen. Wenn die angegebenen Tabellen nicht am Anfang verbunden werden können, oder mehrere LEADING HINTS gegeben werden, die mit einander in Konflikt stehen, werden die Hints automatisch ignoriert. Die Syntax des LEADING HINT eingebettet in ein SELECT-Statement sieht wie folgt aus:

```
SELECT /*+ LEADING ( [@queryblock] tablespec [tablespec] ... ) */ ... FROM ...
```

Die LEADING Syntax kann eine oder mehrere Tabellen angeben, angeben. Wenn nur eine Tabelle angegeben ist, wird diese Tabelle als äußere Tabelle für den ersten Join, mit einer beliebigen anderen Tabelle verwendet. Wenn zum Beispiel drei Tabellen angegeben sind, dann werden die ersten beiden Tabellen zu erst mit einem Join verbunden und danach die Ergebnistabelle der ersten Beiden mit der dritten Tabelle verbunden.

Der ORDERED HINT gibt eine genaue, geordnete (ordered) Join-Reihenfolge vor. Die Tabellen werden in der Reihenfolge verbunden, wie sie hinter der FROM-Klausel angegeben wurden. Wenn sowohl LEADING als auch ORDERED HINTs angegeben wurden, hat der ORDERED HINT eine höhere Gewichtung und die LEADING HINTs werden ignoriert. Die Syntax des ORDERED HINTS erfordert keine weiteren Angaben und sieht wie folgt aus:

```
SELECT /*+ORDERED */ ... FROM ...
```

6.3 Hints zur Festlegung der Join-Methode

Oracle bietet die folgenden Methoden an, um die im vorigen Abschnitt behandelten Joins durchzuführen. Standardmäßig wird die jeweilige Methode automatisch vom Query Optimizer ausgewählt, aber diese Auswahl kann mit Hilfe der angegebenen Hints für die ersten drei Join-Varianten beeinflusst werden.

1. Nested Loop Joins
 - USE_NL HINT
 - USE_NL_WITH_INDEX HINT
 - NO_USE_NL HINT
2. Hash Joins
 - USE_HASH HINT
 - NO_USE_HASH HINT
3. Sort Merge Joins
 - USE_MERGE HINT
 - NO_USE_MERGE HINT
4. Cartesian Joins
5. Outer Joins

Der Nested Loop Join realisiert den Verbund durch eine geschachtelte Schleife. Eine der beiden beteiligten Tabellen wird als Ausgangstabelle gewählt und als äußere Tabelle bezeichnet. Die andere Tabelle, die von der ersten Tabelle abhängen sollte, wird als innere Tabelle bezeichnet. Für alle Zeilen der äußeren Tabelle werden alle Zeilen der inneren Tabelle aufgerufen, was durch zwei ineinander geschachtelte Schleifen realisiert wird.

Um dem Query Optimizer vorzuschreiben einen Nested Loop Join zu verwenden, können die angegebenen Hints verwendet werden. Die als erstes angegebene Tabelle im Hint, wird als äußere Tabelle vorgeschrieben. Der NO_USE_NL Hint verbietet dem Optimierer, einen Nested Loop Join zu verwenden. Der USE_NL_WITH_INDEX Hint gibt dem Optimierer noch den zu verwendenden Index an. Wenn einer der angegebenen Hints keinen Sinn macht, oder der Ablauf der SQL-Anfrage dadurch scheitern würde, wird der entsprechende Hint vom Query Optimizer ignoriert.[Moo03]Die Syntax für die Nested Loop Hints sehen wie folgt aus.

```

/--> USE_NL ( [queryblock] tablespec [tablespec]... ) */
/--> NO_USE_NL ( [queryblock] tablespec [tablespec]... ) */
/--> USE_NL_WITH_INDEX ( [queryblock] tablespec [indexspec [indexspec]...] ) */

```

Der Hash Join wird verwendet um große Tabellen miteinander zu verbinden, die über eine Join-Bedingung miteinander verbunden werden. Für die kleinere der beiden Tabellen

6.3 Hints zur Festlegung der Join-Methode

wird eine Hash-Tabelle erzeugt, die sich auf die Spalte der Join-Bedingung bezieht. Die zweite Tabelle wird mittels der Hash-Werte durchsucht und die entsprechenden Zeilen werden miteinander verbunden.[Moo03]

Die Hints `USE_HASH` und `NO_USE_HASH` erzwingen bzw. verbieten, wie beim Nested Loop Join, die Verwendung eines Hash Join. Die entsprechende Syntax sieht wie folgt aus:

```
/*+ USE_HASH ( [@queryblock] tablespec [tablespec]... ) */  
/*+ NO_USE_HASH ( [@queryblock] tablespec [tablespec]... ) */
```

Bei Sort Merge Joins werden die beteiligten Tabellen nach der Spalte der Join Bedingungen sortiert (SORT) und danach mit Hilfe der Sortierung ineinander gemischt (MERGE). Auch hier funktionieren der `USE_MERGE HINT` und der `NO_USE_MERGE HINT` wie in den vorherigen Join-Methoden. `USE_MERGE` erzwingt, `NO_USE_MERGE` verbietet die Verwendung des Sort Merge Joins.[Moo03] Die Syntax für diese Join-Methode sieht wie folgt aus:

```
/*+ USE_MERGE ( [@queryblock] tablespec [tablespec]... ) */  
/*+ NO_USE_MERGE ( [@queryblock] tablespec [tablespec]... ) */
```

Auch für den Nested Loop, den Hash und den Sort Merge Join gilt wie für die Syntax des `LEADING` Hints, dass eine oder mehrere Tabellen angegeben werden können. Um zu zeigen wie in einer Hintanweisung unterschiedliche Join-Verfahren benutzt werden können, wird hier ein Beispiel eines Mehrfach-Joins gezeigt. In diesem Beispiel werden die drei Tabellen A, B und C durch einen Join verbunden. A und B werden durch einen Nested Loop Join verbunden, das Ergebnis dieses Joins wird mittels eines Hash Joins mit der Tabelle C verbunden:

```
SELECT /*+ LEADING(A B C) USE_NL(A B) USE_HASH(C)*/ *  
FROM tablea A, tableb B, tablec C  
WHERE ...
```

Die `LEADING`-Anweisung gibt hier die Reihenfolge der Joins an, in diesem Fall hätte hier auch die `ORDERED`-Anweisung stehen können. `USE_NL` und `USE_HASH` geben dann jeweils die zu verwendenden Join-Varianten an.

Der Cartesian Join ist die Standard-Join-Methode die angewandt wird, wenn die beiden Tabellen ohne Join-Bedingung vereinigt werden sollen. Hierbei werden alle Zeilen der einen Tabelle mit allen Zeilen der anderen Tabelle verbunden. Dadurch entsteht das Kartesische Produkt der beiden Tabellen. Da es sich hierbei um die Standardmethode handelt, gibt es für diese Join-Variante keine Optimizer hints. Ein kartesischer Verbund kann einfach durch das Weglassen der Join-Bedingung hinter der `WHERE`-Klausel erreicht werden.

Schließlich gibt es noch den Outer Join, von dem es einen Left Outer Join und einen Right Outer Join gibt. Bei dieser Join-Methode, werden alle Tabellenzeilen der linken (left) oder der rechten (right) Tabelle in die Ergebnistabelle übernommen, egal ob sie die Join-Bedingung erfüllen oder nicht. Nur die Zeilen der anderen Tabelle, die die Join-Bedingung nicht erfüllen fallen aus der Ergebnistabelle heraus. Auch diese Join-Methode wird nicht durch Hints erzeugt, sondern wird in der jeweiligen SQL-Anfrage durch `LEFT`

6.3 Hints zur Festlegung der Join-Methode

OUTER JOIN bzw. RIGHT OUTER JOIN spezifiziert. Eine Variante des Outer Joins ist der Full Outer Joins, der wie der Outer Join funktioniert, aber keine Tabellenzeilen wegfallen lässt. Er funktioniert also als wenn ein Outer Join von beiden Richtungen gleichzeitig durchgeführt wird. Auch für den Full Outer Join gibt es keine Hints, aber er muss speziell wie der Right oder Left Outer Join in der SQL-Anfrage angegeben werden. Das folgende Beispiel stellt einen Full Outer Join dar. Wenn man den Parameter FULL durch RIGHT oder LEFT ersetzt, erhält man einen entsprechenden Outer Join.

```
SELECT s_address, n_nationkey
FROM tpch4.supplier s
FULL OUTER JOIN tpch4.nation n
ON s_nationkey = n_nationkey
```

Der Outer Join kann mit den bereits gezeigten Join-Verfahren, zum Beispiel zum Sort Merge Outer Join oder zum Nested Loop Outer Join kombiniert werden, wobei dann auch wieder die entsprechenden Hints verwendet werden können.

Alles in allem macht es aber nur Sinn den Optimizer mit Hilfe der verschiedenen Hints zu beeinflussen, wenn man Hintergrundwissen bezüglich der Datenbankstruktur besitzt, oder spezielle Ergebnisse erreichen will.

7 Manipulationsmöglichkeiten

In diesem Kapitel wird das Sammeln von Datenbankstatistiken behandelt und wie diese Statistiken manipuliert werden können. Es wird erläutert, welche Statistiktypen Oracle verwendet und insbesondere auf einen Bereich der Spaltenstatistiken eingegangen. Dieser Bereich umfasst die Datenverteilung der Spaltenattribute, die auch als Histogramme bezeichnet wird.

7.1 Art und Verwendungszweck der Statistiken

Oracle sammelt eine Menge von Statistiken, die das gesamte Datenbanksystem betreffen und sich in zwei Bereiche unterteilen lassen. In den ersten Bereich fallen Statistiken die angelegt werden, um den Systemzustand zu messen. Sie werden auch dynamische Leistungsstatistiken (dynamic performance statistics) genannt, da sie vor allem Informationen über das Leistungsverhalten (performance) des Datenbanksystems enthalten. Es handelt sich dabei um kumulative Statistiken, die nicht manipuliert- oder löscherbar sind, auch nicht für den Datenbankadministrator.[OOR03]

Auf Basis dieser gesammelten Daten erzeugt Oracle Metriken, die ein Maß für den Gesundheitszustand des Systems darstellen[OEM03]. Metriken werden dazu benutzt, die Veränderungsrate in den kumulativen Statistiken darzustellen. Ein Beispiel für eine Metrik ist die Anzahl der Datenbankaufrufe pro Sekunde. Mit Hilfe von Alerts (Warnsignalen) können diese Metriken überwacht werden, indem ein Alert aktiviert wird, sobald der Wert einer Metrik über oder unter einen bestimmten systemkritischen Schwellenwert steigt bzw. fällt.[Moo03] Oracle ermöglicht es die gesammelten Statistiken dieses ersten Bereichs in verschiedenen System Views (V\$ Views) zu betrachten. Da diese Statistiken nicht veränderbar sind, wird hier nicht weiter darauf eingegangen, sondern der zweite Bereich der Statistiken behandelt.

Der zweite Bereich der Statistiken umfasst das Sammeln von Daten, die Aufschluss über den genauen Aufbau der eigentlichen Datenbank und deren Datenbankobjekte geben. Diese Statistiken werden vom Query Optimizer benutzt, um die Entscheidungen über den Ausführungsplan einer SQL-Anweisung zu fällen. Je nachdem, wie man diese Statistiken manipuliert bzw. auf welche der Statistiken der Schwerpunkt gelegt wird, kann die Entscheidung für eine bestimmte Ausführungsreihenfolge unterschiedlich ausfallen. Die Query Optimizer Statistiken lassen sich in vier unterschiedliche Kategorien einteilen.[Moo03]

7.2 Sammeln der Statistiken

1. Statistiken zu den Tabellenstrukturen
(z.B.: Anzahl der Zeilen/Blöcke oder die durchschnittliche Zeilenlänge)
2. Statistiken zu den Datenbankspalten (Histogramme)
(z.B.: Anzahl an unterschiedlichen Datenwerten oder die Verteilung der Datenwerte in einer Spalte)
3. Statistiken über die Datenbankindizes
(z.B.: Anzahl der Levels, Wert des Clustering Faktor)
4. Systemstatistiken (z.B.:I/O und CPU Auslastung)

Da nur auf das Sammeln der Statistiken aus der zweiten Kategorie Einfluss genommen werden kann, wird in diesem Kapitel nur auf diese Optimizer-Statistiken, ins Besondere auf die Histogramme und auf die allgemeine Manipulation der einzelnen Statistiken eingegangen, und wie diese automatisch bzw. manuell gesammelt werden können.

Da sich die Statistiken auf Parameter beziehen, die im Verlauf des Datenbankbetriebs mehr oder minder schnell verändert werden, müssen die Statistiken regelmäßig aktualisiert und neu gesammelt werden, um aussagekräftig zu bleiben.

7.2 Sammeln der Statistiken

Standardmäßig werden die Optimizer-Statistiken in Oracle automatisch für jedes Datenbankobjekt erstellt, dieser Automatismus kann aber auch für die gesamte Datenbank oder bestimmte Objekte abgeschaltet werden. Der GATHER_STATS_JOB Dienst von Oracle sammelt automatisch Daten für alle Datenbankobjekte, die noch keine Statistiken besitzen, oder deren Statistiken als veraltet gelten, da mehr als 10% der zugrundeliegenden Daten verändert wurden. GATHER_STATS_JOB wird vom Datenbank-Scheduler aufgerufen, sobald das Wartungsintervall beginnt. Das Wartungsintervall bzw. -fenster ist standardmäßig auf 22 bis 6 Uhr und das gesamte Wochenende gesetzt, es kann vom Datenbankadministrator beliebig verändert werden. [Bay03]

Wenn man sich dafür entscheidet, das Sammeln der Statistikdaten manuell durchzuführen, muss der GATHER_STATS_JOB durch den folgenden Befehl beendet werden.

```
DBMS_SCHEDULER.DISABLE('GATHER_STATS_JOB');
```

Für das manuelle Erzeugen von aktuellen Statistiken muss selbständig entschieden werden, wann und für welche Datenbankobjekte Daten gesammelt werden sollen. Dies kann mit Hilfe des DBMS_STATS Pakets von Oracle durchgeführt werden. Das DBMS_STATS Paket bietet verschiedene Prozeduren an, um die unterschiedlichen Statistikfunktionen zu ermöglichen. Auch die GATHER_STATS_JOB Prozedur greift auf die Prozeduren dieses Pakets zurück, um die automatischen Statistiken zu erstellen. DBMS_STATS bietet fünf grundlegende Prozeduren, die zum Sammeln der Statistikdaten benötigt werden. Eine ausführliche Beschreibung des DBMS_STATS Pakets mit allen Pro-

7.2 Sammeln der Statistiken

zeduren und deren Syntax ist unter [Rap03] zu finden.

Mit GATHER_TABLE_STATS werden Statistiken über eine Tabelle und deren Da-

Prozedur	Sammelt
GATHER_INDEX_STATS	Indexstatistikdaten
GATHER_TABLE_STATS	Tabellen-, Spalten- und Indexstatistikdaten
GATHER_SCHEMA_STATS	Statistikdaten für alle Objekte in einem Schema
GATHER_DICTIONARY_STATS	Statistikdaten für alle Dictionaryobjekte
GATHER_DATABASE_STATS	Statistikdaten für alle Objekte in einer Datenbank

Tabelle 7.1: DBMS_STATS Prozeduren zum Sammeln von Statistikdaten[Mor03b]

ten erzeugt. Dabei werden auch grundlegende Informationen über die Verteilung der Datenwerte, wie das Maximum bzw. Minimum eines Attributes bzw. einer Spalte angelegt. Um aussagekräftigere Statistiken über die Datenverteilung zu erhalten, können die fünf angegebenen Prozeduren mit der Erweiterung METHOD_OPT aufgerufen werden, um Histogramme zu erstellen. In der folgenden Tabelle wird beispielhaft dargestellt, welche Statistiken Oracle automatisch mit GATHER_TABLE_STATS für eine Tabelle und deren Spalten anlegt und wie die entsprechende Spalte im Data Dictionary-View bezeichnet wird [Mor03b]. Die Statistikdaten können auch manipuliert werden, indem

Bezeichner	Erklärung
NUM_ROWS	Anzahl der Tabellenzeilen
BLOCKS	Anzahl der vorformatierten Blöcke, schließt auch leere Datenblöcke mit ein
EMPTY_BLOCKS	Anzahl der leeren Blöcke die der Tabelle zugewiesen, aber nie belegt wurden
AVG_ROW_LEN	Durchschnittliche Zeilenlänge in Bytes für die Tabelle und alle Spalten
NUM_DISTINCT	Anzahl unterschiedlicher Datenwerte
LOW_VALUE	Niedrigster Datenwert
HIGH_VALUE	Höchster Datenwert
DENSITY	Dichte
NUM_NULLS	Anzahl der Spalten die einen Nullwert haben

Tabelle 7.2: Automatisch gesammelte Statistikdaten

die Attributewerte dieser Spalten manuell mit Hilfe der SET_*_STATS-Prozeduren von DBMS_STATS gesetzt werden. Für jede der oben aufgeführten GATHER_*_STATS-Prozeduren gibt es eine entsprechende SET_*_STATS-Prozedur, mit der deren gesammelte Werte überschrieben werden können.

7.3 Histogramme

Ein Histogramm ist eine Darstellung der Häufigkeitsverteilung von Messwerten. In Oracle zeigen Histogramme die Häufigkeitsverteilung von Datenwerten einer Datenbanktabellenspalte an. Oracle verfügt über zwei unterschiedliche Arten von Histogrammen. Height-Balanced (größenbalancierte) und Frequency (Häufigkeits-) Histogramme.

Größenbalancierte Histogramme teilen die Menge der Datenwerte in eine bestimmte Anzahl möglichst gleichgroßer Tupelmengen, sogenannter Buckets(Töpfe) ein. Anhand der Beschriftung dieser Buckets kann festgestellt werden, wie viel Prozent der Datenmenge einen bestimmten Datenwert annehmen. Wenn eine Menge von 100 Datenwerten, die einen Wert von 1 bis 100 annehmen können, in 10 Buckets unterteilt wird, dann fasst jeder Bucket 10 Datenwerte. Das folgende Histogramm zeigt die Datenverteilung der einzelnen Datenwerte:

Bucketnr.:	1	2	3	4	5	6	7	8	9	10	
	I_____I	I_____I	I_____I	I_____I	I_____I	I_____I	I_____I	I_____I	I_____I	I_____I	
Wert:	1	30	50	60	70	80	90	95	95	95	100

Aus dem obigen Histogramm ist zum Beispiel erkennbar, dass nur 10 Datenwerte einen Wert zwischen 1 und 30 annehmen, oder zum Beispiel, dass 20 Datenwerte einen Wert von 95 annehmen.

Frequenz- oder Häufigkeitshistogramme basieren auf dem gleichen Prinzip wie die größenbalancierten Histogramme, nur dass hier jeder Bucket einen einzelnen Wert repräsentiert und die Größe der einzelnen Buckets angibt, wie viele Datenwerte, den entsprechenden Bucket-Wert besitzen. Ein größenbalanciertes Histogramm wird von Oracle automatisch in ein Frequenzhistogramm umgewandelt, wenn die Anzahl der unterschiedlichen Werte kleiner als die Anzahl der angegebenen Buckets ist.

Mit folgendem Befehl, wird ein Histogramm für die Spalte `l_extendedprice`, mit 100 Buckets für die Tabelle `LINEITEM` aus dem Schema `TPCH4` angelegt.

```
begin
  dbms_stats.gather_table_stats(ownname => 'TPCH4',
    tabname => 'LINEITEM', method_opt => 'for columns size 100
l_extendedprice',);
end;
```

Die Werte `ownname` und `tabname` geben das Tabellenschema und den Tabellennamen an. Die `method_opt`-Option akzeptiert folgende Werte:

```
FOR ALL [INDEXED | HIDDEN] COLUMNS SIZE {integer | REPEAT | AUTO | SKEWONLY}
oder
FOR COLUMNS SIZE {integer | REPEAT | AUTO | SKEWONLY} column|
  attribute SIZE {integer | REPEAT | AUTO | SKEWONLY} [,column|
  attribute SIZE {integer | REPEAT | AUTO | SKEWONLY}...]
```

7.3 Histogramme

Der Parameter *integer* gibt die Anzahl der Buckets an und muss kleiner gleich 1254 sein. Durch REPEAT werden nur Histogramme von Spalten erstellt, die bereits über frühere Histogramme verfügen. Der Parameter AUTO ist der Standardwert und lässt Oracle die Spalten auswählen, für die Histogramme erstellt werden sollen. Der Parameter SKEWONLY führt dazu, dass nur für die Spalten ein Histogramm erzeugt werden, die eine sehr ungleiche Datenverteilung besitzen.

Die Statistiken und Histogramme über Tabellen, Indizes und Spalten werden im Data Dictionary abgespeichert und können mit verschiedenen Views abgerufen werden. Das folgende Beispiel gibt das für die Tabelle TPCH4.LINEITEM erzeugte Histogramm aus.

```
SELECT endpoint_number, endpoint_value
FROM USER_TAB_HISTOGRAMS
WHERE table_name='LINEITEM' AND column_name='l_extendedprice'
ORDER BY endpoint_number;
```

```
ENDPOINT_NUMBER  ENDPOINT_VALUE
-----
                0                0
                1                27
                2                42
                3                57
                . .
                . .
                . .
                97                149
                98                175
                99                202
                100               353
```

Weitere Views, die auf das Data Dictionary zugreifen sind unter anderem DBA_TABLES, DBA_OBJECT_TABLES, DBA_TAB_STATISTICS, DBA_TAB_COL_STATISTICS,... . Die folgende Tabelle zeigt die verfügbaren Attribute von USER_TAB_HISTOGRAMS an: Weitere Views und aus welchen Attributen sich diese Views zusamm-

Bezeichner	Datentyp	Erklärung
OWNER	VARCHAR2(30)	Besitzer der Tabelle
TABLE_NAME	VARCHAR2(30)	Tabellenname
COLUMN_NAME	VARCHAR2(4000)	Spalten- oder Attributsnamen
ENDPOINT_NUMBER	NUMBER	Histogram Bucket-Nummer
ENDPOINT_VALUE	NUMBER	Normalisierter Endpunktwert des Buckets
ENDPOINT_ACTUAL_VALUE	VARCHAR2(1000)	Nicht normalisierter Endpunktwert

Tabelle 7.3: Attribute des ALL/DBA/USER_HISTOGRAMS-Views[Mor03b]

mensetzen ist in [Mor03b] zu finden.

7.4 Exportieren oder Importierung von Statistiken

Oracle ermöglicht mit Hilfe seiner Export- und Importwerkzeuge (s. Kapitel 3) das Exportieren oder Importieren der gesammelten Statistiken in eigens dafür erstellte Tabellen. Die Prozedur `DBMS_STATS.CREATE_STAT_TABLE` erzeugt die eigenen Statistiktabelle. Sobald eine oder mehrere dieser Tabellen erstellt wurden, kann mittels `DBMS_STATS.EXPORT_*_STATS` der Exportvorgang in die neuen Statistiktabelle durchgeführt werden. Dabei realisiert die `EXPORT_*_STATS` Prozedur den Export aus der Originaltabelle mit dem Oracle EXP-Tool und den Import in die neue Tabelle mit Oracle IMP. Auf diese Weise können die Statistiken zwischen verschiedenen Datenbanken ausgetauscht werden, um zum Beispiel die gesammelten Statistiken einer Produktionsdatenbank auf ein Testsystem zu kopieren, damit sich dieses wie die Produktionsumgebung verhält und so sinnvollere Ergebnisse liefert.[Moo03]

Eine Ausführliche Beschreibung des `DBMS_STATS` Paketes ist in [Rap03] zu finden. Weitergehende Informationen so wie eine detaillierte Beschreibung der Oracle-Statistiken steht in Kapitel 5 und 15 des Oracle Performance Tuning Guides(s.[Moo03]).

8 Schluss

8.1 Zusammenfassung

Im Laufe dieser Arbeit wurde gezeigt, dass es sich beim Oracle® Database System 10g um ein komplexes, vielseitig einsetzbares relationales Datenbank Management-System handelt. Die Aufgabenstellung dieser Arbeit war eine Datenbank aus einem DB2-Datenbanksystem in das Oracle-System zu migrieren. Dabei wurde erklärt, wie die Datenbasis der vorgegebenen Datenbank zustande kommt und wie diese mit dem TPC-H-Benchmark-Schema zusammenhängt. Es wurde auf verschiedene Werkzeuge eingegangen, die Oracle zur Verfügung stellt, um mit der Datenbank zu interagieren. Dabei wurde hauptsächlich auf die SQL-Schnittstelle SQL*Plus eingegangen, da diese als Grundlage für die weitere Arbeit benutzt wurde. Es wurden verschiedene Migrationswerkzeuge vorgestellt und begründet, warum SQL*Loader zur Migration der vorgegebenen DB2-Datenbank verwendet wurde. Darüber hinaus wurde eine detaillierte Anleitung zur Installation des Oracle-Systems gegeben, in der auch auf Probleme während der Installation eingegangen wurde. Danach folgte eine Migrationsanleitung die in mehreren Schritten mit Hilfe des vorgestellten SQL*Loader-Werkzeugs und mit angelegten SQL-Skripten durchgeführt wurde. Dabei wurde unter anderem auf die Syntaxunterschiede der SQL-Versionen von DB2 und Oracle eingegangen. Danach wurden Analyse-Tools von Oracle verwendet um vorgegebene SQL-Skripte zu untersuchen und die Verwendung der jeweiligen Analysewerkzeuge demonstriert. Es wurden detaillierte Aussagen über die vom Query Optimizer aufgestellten Ausführungspläne der einzelnen Skripte getroffen. Darüberhinaus wurde gezeigt, wie diese Ausführungspläne mit manuellen Befehlen manipuliert werden können. Dabei wurde vor allem auf Join-Reihenfolgen und Join-Methoden eingegangen. Es wurde gezeigt, dass die Berechnung des Query Optimizers auf Statistiken basieren, die vom Oracle-Datenbanksystem automatisch angelegt werden, aber auch wie dieser Automatismus beeinflusst und abgeschaltet werden kann, um die Statistikdaten manuell zu sammeln.

8.2 Fazit und Ausblick

Bei der Arbeit mit dem Oracle-System hat sich gezeigt, dass die Version 10g noch einige Bugs enthält, die den Ablauf der Arbeit das ein oder andere Mal zum Stocken gebracht haben. Dies fing beim Scheitern der ersten Installation an und trat in Fällen wie der fehlerhaften iSQL*Plus-Schnittstelle und anderen kleineren nicht erklärbaren Fehlern,

8.2 Fazit und Ausblick

die oft nach einem Neustart des Systems verschwunden waren, immer wieder zu Tage. Nach einer gewissen Eingewöhnungsphase ließ sich das Oracle-System, vor allem die SQL*Plus-Schnittstelle und der Oracle Enterprise Manager, jedoch gut bedienen. Auf die Bedienungsanleitung des Enterprise Managers wurde in dieser Arbeit verzichtet, da er zur Ausführung der gezeigten Aufgaben nicht notwendig ist. Die Grundfunktionen seines Interfaces sind jedoch leicht zu erlernen und zu bedienen und vereinfachen manche der hier aufgeführten Arbeiten. Deshalb sei hier nochmals auf die Literatur zum Enterprise Manager (s. [Gos03],[MB+03]) und auf die entsprechenden Onlinekurse von Oracle verwiesen(s. http://www.oracle.com/technology/obe/2day_dba/index.html).

Als Ausblick für die Zukunft können einige Anknüpfungspunkte für Folgearbeiten und Folgeuntersuchen betreffend des Oracle-Systems gegeben werden. Wie in Abbildung 5.3 deutlich wird, tritt eine Verschlechterung der Ausführungszeit für die sequence-Skripte von Optimierungsvariante 2 nach Variante 3 auf. Es könnte untersucht werden, warum dieser Effekt auftritt. Des Weiteren könnte dabei untersucht werden, in wie fern sich die sequence- von der with-Variante unterscheidet, und warum die 2. Optimierung beim with-Skript nicht ähnlich gute Resultate wie bei sequence.sql liefert. Ein weiterer Punkt der von Folgearbeiten untersucht werden könnte sind die genauen Schätzmethode des Query Optimizers, warum zum Beispiel die Zeitschätzungen so stark abweichen oder warum die Zeilenschätzungen für die singlequery-Variante sehr gut, für die beiden anderen Varianten aber sehr schlecht sind. Dabei müsste genauer auf die in Kapitel 7 vorgestellten Optimizer Statistiken eingegangen werden, wobei untersucht werden könnte ob und wie die Abschätzungen verbessert werden könnten, um die Ergebnisse zukünftiger EXPLAIN PLAN-Tabellen zu optimieren.

Literaturverzeichnis

- [Bay03] Baylis, R.: Oracle® Database Administrator's Guide 10g Release 1 10.1 Part No. B10739-01, December 2003.
- [Cho03] Choi, P.: Oracle® Universal Installer Concepts Guide 10g Release 1 10.1 Part No. B12140-01, December 2003.
- [Cyr03] Cyran, M.: Oracle® Ultra Search User's Guide 10g Release 1 10.1 Part No. B10731-01, December 2003.
- [Gos03] Gosselin, J.: Oracle® Enterprise Manager Concepts 10g Release 1 10.1 Part No. B12016-01, December 2003.
- [LaQ03] LaQuerre, P.: Oracle® Enterprise Manager Advanced Configuration 10g Release 1 10.1 Part No. B12013-01, December 2003.
- [LG03] Lorentz, D.; Gregoire, J.: Oracle® Database SQL Reference 10g Release 1 10.1 Part Number B10759-01, December 2003.
- [MB+03] McGregor, C.; Baylis R.; Kumar, S.; Romero, A.; Austin, D.; Cyran, M.: Oracle® Database 2 Day DBA 10g Release 1 10.1 Part No. B10742-01, December 2003.
- [Moo03] Moore, V.: Oracle® Database Performance Tuning Guide 10g Release 1 10.1 Part No. B10752-01, December 2003.
- [Mor03a] Morales, T.: Oracle® Database Upgrade Guide 10g Release 1 10.1 Part No. B10763-01, December 2003.
- [Mor03b] Morales, T.: Oracle® Database Reference 10g Release 1 10.1 Part Number B10755-01, December 2003.
- [MS03] Messerschmidt, H.; Schweinsberg, K.: OLAP mit dem SQL-Server. Eine Einführung in Theorie und Praxis, 2003.
- [OOR03] Oracle® OLAP Reference 10g Release 1 10.1 Part Number B10334-02, December 2003.
- [OSQ03] Oracle® SQL*Plus® Quick Reference Release 10.1 Part No. B12171-01, December 2003.

Literaturverzeichnis

- [OTG03] Oracle® Transparent Gateway for DRDA Installation and User's Guide 10g Release 1 10.1.0.2.0 for Microsoft Windows Part Number B12010-01, March 2004.
- [Pol04] Polk, J.: Oracle® Database Net Services Administrator's Guide 10 g Release 1 10.1 Part No. B10775-01, January 2004.
- [Rap03] Raphaely, D.: PL/SQL Packages and Types Reference 10g Release 1 10.1 Part No. B10802-01, December 2003.
- [Ric03] Rich, K.: Oracle® Database Utilities 10g Release 1 10.1 Part No. B10825-01, December 2003.
- [Sim04] Simmons, J.: Oracle® Database Installation Guide 10g Release 1 10.1.0.2.0 for Windows Part No. B10130-01, 2004.
- [Tra99] Transaction Processing Performance Council. TPC Benchmark H. Technical Report Standard Specification Revision 1.2.1, Transaction Processing Performance Council, 1999.
- [Wag99] Wagner, R.: Realisierung und Optimierung einer OLAP-Anwendung im Handelsbereich, Studienarbeit-Nr. 1770 am IVPR, 1999.
- [Wat03] Watt, S.: Oracle® SQL*Plus® User's Guide and Reference Release 10.1 Part No. B12170-01, December 2003.

Anhang A:

A.1 Migrationskripte

A.1.1 altertable.sql

Oracle-Version identisch mit DB2-Version

```
-- FILE      : altertable.sql
-- CREATED   : 2004-06-04
-- AUTHOR    : Thorsten Müller (originally from Ralf Rantza)
-- PURPOSE   : TPC-H schema according to Standard Specification Revision 1.1.0
--           : replace tpchx by target schema
-----
-- Primary key constraints.
ALTER TABLE tpch4.part      ADD CONSTRAINT c_pk01 PRIMARY KEY (p_partkey);
ALTER TABLE tpch4.supplier  ADD CONSTRAINT c_pk02 PRIMARY KEY (s_suppkey);
ALTER TABLE tpch4.partsupp  ADD CONSTRAINT c_pk03 PRIMARY KEY (ps_partkey, ps_suppkey);
ALTER TABLE tpch4.customer  ADD CONSTRAINT c_pk04 PRIMARY KEY (c_custkey);
ALTER TABLE tpch4.orders    ADD CONSTRAINT c_pk05 PRIMARY KEY (o_orderkey);
ALTER TABLE tpch4.lineitem  ADD CONSTRAINT c_pk06 PRIMARY KEY (l_orderkey, l_linenum);
ALTER TABLE tpch4.nation    ADD CONSTRAINT c_pk07 PRIMARY KEY (n_nationkey);
ALTER TABLE tpch4.region    ADD CONSTRAINT c_pk08 PRIMARY KEY (r_regionkey);
-- Foreign key constraints.
ALTER TABLE tpch4.supplier  ADD CONSTRAINT c_fk01 FOREIGN KEY (s_nationkey)
    REFERENCES tpch4.nation (n_nationkey);
ALTER TABLE tpch4.partsupp  ADD CONSTRAINT c_fk02 FOREIGN KEY (ps_partkey)
    REFERENCES tpch4.part (p_partkey);
ALTER TABLE tpch4.partsupp  ADD CONSTRAINT c_fk03 FOREIGN KEY (ps_suppkey)
    REFERENCES tpch4.supplier (s_suppkey);
ALTER TABLE tpch4.customer  ADD CONSTRAINT c_fk04 FOREIGN KEY (c_nationkey)
    REFERENCES tpch4.nation (n_nationkey);
ALTER TABLE tpch4.orders    ADD CONSTRAINT c_fk05 FOREIGN KEY (o_custkey)
    REFERENCES tpch4.customer (c_custkey);
ALTER TABLE tpch4.lineitem  ADD CONSTRAINT c_fk06 FOREIGN KEY (l_orderkey)
    REFERENCES tpch4.orders (o_orderkey);
ALTER TABLE tpch4.lineitem  ADD CONSTRAINT c_fk07 FOREIGN KEY (l_partkey)
    REFERENCES tpch4.part (p_partkey);
ALTER TABLE tpch4.lineitem  ADD CONSTRAINT c_fk08 FOREIGN KEY (l_suppkey)
    REFERENCES tpch4.supplier (s_suppkey);
ALTER TABLE tpch4.lineitem  ADD CONSTRAINT c_fk09 FOREIGN KEY (l_partkey, l_suppkey)
    REFERENCES tpch4.partsupp (ps_partkey, ps_suppkey);
ALTER TABLE tpch4.nation    ADD CONSTRAINT c_fk10 FOREIGN KEY (n_regionkey)
    REFERENCES tpch4.region (r_regionkey);
```

A.1.2 createtable.sql

Oracle-Version

```
-- FILE      : createtable.sql
-- CREATED   : 1999-10-13
-- AUTHOR    : Ralf Rantza
-- PURPOSE   : TPC-H schema according to Standard Specification Revision 1.1.0
--           : replace tpchx by target schema
-- CHANGES  : 2000-04-11: HS: changes for different schemas
--           : 2004-06-17: Thorsten Müller: changes for Oracle
```

```
-----
CONNECT system@tpch;
CREATE TABLE tpch4.part (
  p_partkey      Integer NOT NULL,
  p_name         Varchar(55),
  p_mfgr        Char(25),
  p_brand        Char(10),
  p_type         Varchar(25),
  p_size         Integer,
  p_container    Char(10),
  p_retailprice  Decimal(10, 2),
  p_comment      Varchar(23)
) TABLESPACE TPCH4;

CREATE TABLE tpch4.supplier (
  s_suppkey      Integer NOT NULL,
  s_name         Char(25),
  s_address      Varchar(40),
  s_nationkey    Integer,
  s_phone        Char(15),
  s_acctbal      Decimal(10, 2),
  s_comment      Varchar(101)
) TABLESPACE TPCH4;

CREATE TABLE tpch4.partsupp (
  ps_partkey     Integer NOT NULL,
  ps_suppkey     Integer NOT NULL,
  ps_availqty    Integer,
  ps_supplycost  Decimal(10, 2),
  ps_comment     Varchar(199)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.customer (
  c_custkey      Integer NOT NULL,
  c_name         Varchar(25),
  c_address      Varchar(40),
  c_nationkey    Integer,
  c_phone        Char(15),
  c_acctbal      Decimal(10, 2),
  c_mktsegment   Char(10),
  c_comment      Varchar(117)
) TABLESPACE TPCH4;

CREATE TABLE tpch4.orders (
  o_orderkey     Integer NOT NULL,
  o_custkey      Integer,
  o_orderstatus  Char(1),
  o_totalprice   Decimal(10, 2),
  o_orderdate    Date,
  o_orderpriority Char(15),
```

Literaturverzeichnis

```
o_clerk          Char(15),
o_shippriority   Integer,
o_comment        Varchar(79)
) TABLESPACE TPCH4;

CREATE TABLE tpch4.lineitem (
  l_orderkey      Integer NOT NULL,
  l_partkey       Integer,
  l_suppkey       Integer,
  l_linenumbers   Integer NOT NULL,
  l_quantity      Decimal(10, 2),
  l_extendedprice Decimal(10, 2),
  l_discount      Decimal(10, 2),
  l_tax          Decimal(10, 2),
  l_returnflag    Char(1),
  l_linestatus    Char(1),
  l_shipdate      Date,
  l_commitdate    Date,
  l_receiptdate   Date,
  l_shipinstruct  Char(25),
  l_shipmode      Char(10),
  l_comment       Varchar(44)
) TABLESPACE TPCH4;

CREATE TABLE tpch4.nation (
  n_nationkey     Integer NOT NULL,
  n_name          Char(25),
  n_regionkey     Integer,
  n_comment       Varchar(152)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.region (
  r_regionkey     Integer NOT NULL,
  r_name          Char(25),
  r_comment       Varchar(152)
)TABLESPACE TPCH4;

CONNECT RESET;
```

Original-DB2-Version

```
-- FILE      : createtable.sql
-- CREATED   : 1999-10-13
-- AUTHOR    : Ralf Rantza
-- PURPOSE   : TPC-H schema according to Standard Specification Revision 1.1.0
--           : replace tpchx by target schema
-----
CONNECT TO tpch;

CREATE TABLE tpch4.part (
  p_partkey      Integer NOT NULL,
  p_name         Varchar(55),
  p_mfgr        Char(25),
  p_brand        Char(10),
  p_type         Varchar(25),
  p_size        Integer,
  p_container    Char(10),
  p_retailprice  Decimal(10, 2),
  p_comment      Varchar(23)
) IN data1;
```

Literaturverzeichnis

```
CREATE TABLE tpch4.supplier (  
  s_suppkey      Integer NOT NULL,  
  s_name         Char(25),  
  s_address     Varchar(40),  
  s_nationkey   Integer,  
  s_phone       Char(15),  
  s_acctbal     Decimal(10, 2),  
  s_comment     Varchar(101)  
) IN data1;  
  
CREATE TABLE tpch4.partsupp (  
  ps_partkey    Integer NOT NULL,  
  ps_suppkey    Integer NOT NULL,  
  ps_availqty   Integer,  
  ps_supplycost Decimal(10, 2),  
  ps_comment    Varchar(199)  
) IN data1;  
  
CREATE TABLE tpch4.customer (  
  c_custkey     Integer NOT NULL,  
  c_name        Varchar(25),  
  c_address     Varchar(40),  
  c_nationkey   Integer,  
  c_phone       Char(15),  
  c_acctbal     Decimal(10, 2),  
  c_mktsegment  Char(10),  
  c_comment     Varchar(117)  
) IN data1;  
  
CREATE TABLE tpch4.orders (  
  o_orderkey    Integer NOT NULL,  
  o_custkey     Integer,  
  o_orderstatus Char(1),  
  o_totalprice  Decimal(10, 2),  
  o_orderdate   Date,  
  o_orderpriority Char(15),  
  o_clerk       Char(15),  
  o_shippriority Integer,  
  o_comment     Varchar(79)  
) IN data1;  
  
CREATE TABLE tpch4.lineitem (  
  l_orderkey    Integer NOT NULL,  
  l_partkey     Integer,  
  l_suppkey     Integer,  
  l_linenumber  Integer NOT NULL,  
  l_quantity    Decimal(10, 2),  
  l_extendedprice Decimal(10, 2),  
  l_discount    Decimal(10, 2),  
  l_tax         Decimal(10, 2),  
  l_returnflag  Char(1),  
  l_linestatus  Char(1),  
  l_shipdate    Date,  
  l_commitdate  Date,  
  l_receiptdate Date,  
  l_shipinstruct Char(25),  
  l_shipmode    Char(10),  
  l_comment     Varchar(44)  
) IN data1;  
  
CREATE TABLE tpch4.nation (  
  n_nationkey   Integer NOT NULL,
```

Literaturverzeichnis

```
n_name          Char(25),
n_regionkey     Integer,
n_comment       Varchar(152)
) IN data1;
```

```
CREATE TABLE tpch4.region (
  r_regionkey   Integer NOT NULL,
  r_name        Char(25),
  r_comment     Varchar(152)
) IN data1;
```

```
CONNECT RESET;
```

A.1.3 mk_ind_v01.sql

Oracle-Version

```
-- file: mk_ind_v01.sql
-- author: Holger Schwarz
-- last update: 17.06.2004 Thorsten Müller
-- content: create additional indexes on benchmark schema+ analyze tables
-----
CONNECT TO TPCH;
ALTER INDEX tpch4.C_PK05 RENAME TO sk_o_ok;
ALTER INDEX tpch4.sk_o_ok REBUILD TABLESPACE TPCH4INDEX;
ALTER INDEX tpch4.C_PK06 RENAME TO sk_l_ok;
ALTER INDEX tpch4.sk_l_ok REBUILD TABLESPACE TPCH4INDEX;
CREATE INDEX tpch4.sk_o_od ON tpch4.orders (o_orderdate ASC)TABLESPACE TPCH4INDEX;

ALTER INDEX tpch4.C_PK01 RENAME TO sk_p_pk;
ALTER INDEX tpch4.sk_p_pk REBUILD TABLESPACE TPCH4INDEX;

CREATE INDEX tpch4.sk_o_ck ON tpch4.orders (o_custkey ASC)TABLESPACE TPCH4INDEX;
CREATE INDEX tpch4.sk_l_pk ON tpch4.lineitem (l_partkey ASC)TABLESPACE TPCH4INDEX;
ALTER INDEX tpch4.C_PK04 RENAME TO sk_c_ck;
ALTER INDEX tpch4.sk_c_ck REBUILD TABLESPACE TPCH4INDEX;

CREATE INDEX tpch4.sk_l_rf ON tpch4.lineitem (l_returnflag ASC)TABLESPACE TPCH4INDEX;

ANALYZE TABLE tpch4.lineitem COMPUTE STATISTICS;
ANALYZE TABLE tpch4.orders COMPUTE STATISTICS;
ANALYZE TABLE tpch4.customer COMPUTE STATISTICS;
ANALYZE TABLE tpch4.supplier COMPUTE STATISTICS;
ANALYZE TABLE tpch4.partsupp COMPUTE STATISTICS;
ANALYZE TABLE tpch4.nation COMPUTE STATISTICS;
ANALYZE TABLE tpch4.region COMPUTE STATISTICS;
ANALYZE TABLE tpch4.part COMPUTE STATISTICS;
CONNECT RESET;
```

Original-DB2-Version

```
-- file:    mk_ind_v01.sql
-- author:   Holger Schwarz
-- last update: 07.11.2000 HS
-- content:  create additional indexes on benchmark schema
--          + runstats on tables
-----
```

```
CONNECT TO TPCH;
```

Literaturverzeichnis

```
CREATE INDEX tpch4.sk_o_ok ON tpch4.orders (o_orderkey ASC) CLUSTER;
CREATE INDEX tpch4.sk_l_ok ON tpch4.lineitem (l_orderkey ASC, l_linenummer ASC)
CLUSTER;
CREATE INDEX tpch4.sk_o_od ON tpch4.orders (o_orderdate ASC);
CREATE INDEX tpch4.sk_p_pk ON tpch4.part (p_partkey ASC) CLUSTER;
CREATE INDEX tpch4.sk_o_ck ON tpch4.orders (o_custkey ASC);
CREATE INDEX tpch4.sk_l_pk ON tpch4.lineitem (l_partkey ASC);
CREATE INDEX tpch4.sk_c_ck ON tpch4.customer (c_custkey ASC) CLUSTER;
CREATE INDEX tpch4.sk_l_rf ON tpch4.lineitem (l_returnflag ASC);

RUNSTATS ON TABLE tpch4.lineitem
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.orders
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.customer
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.supplier
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.partsupp
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.nation
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.region
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.part
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
CONNECT RESET;
```

A.1.4 mk_tab_v01.sql

Oracle-Version

```
-- file: mk_tab_v01.sql
-- author: Holger Schwarz
-- last update: 17.06.2004 von Thorsten Müller
-- content: create additional tables for benchmark schema + Analyze tables
-----
CONNECT TPCH;
CREATE TABLE tpch4.relate_date (
  basisdate      DATE NOT NULL,
  lastmonthdate  DATE,
  lastyeardate   DATE,
  CONSTRAINT pk_reld PRIMARY KEY (basisdate)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.relate_month (
  basismonthkey  INTEGER NOT NULL,
  lastmonthkey   INTEGER,
  lastyearmonthkey  INTEGER,
  CONSTRAINT pk_relm PRIMARY KEY (basismonthkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.relate_year (
  basisyearkey   INTEGER NOT NULL,
  lastyearkey    INTEGER,
  CONSTRAINT pk_rely PRIMARY KEY (basisyearkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_shipday (
```

Literaturverzeichnis

```
    shipdate          DATE NOT NULL,
    shipday           INTEGER,
    shipdayname       VARCHAR(10),
    shipmonthkey      INTEGER,
    shipyearkey       INTEGER,
    lastmonthdate     DATE,
    lastyeardate      DATE,
    CONSTRAINT pk_sd PRIMARY KEY (shipdate)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_shipmonth (
    shipmonthkey      INTEGER NOT NULL,
    shipmonth         INTEGER,
    shipmonthname     VARCHAR(10),
    shipyearkey       INTEGER,
    lastmonthkey      INTEGER,
    lastyearmonthkey  INTEGER,
    CONSTRAINT pk_sm PRIMARY KEY (shipmonthkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_shipyear (
    shipyearkey       INTEGER NOT NULL,
    shipyear          INTEGER,
    lastyearkey       INTEGER,
    CONSTRAINT pk_sy PRIMARY KEY (shipyearkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_commitday (
    commitdate        DATE NOT NULL,
    commitday         INTEGER,
    commitdayname     VARCHAR(10),
    commitmonthkey    INTEGER,
    commityearkey     INTEGER,
    lastmonthdate     DATE,
    lastyeardate      DATE,
    CONSTRAINT pk_cd PRIMARY KEY (commitdate)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_commitmonth (
    commitmonthkey    INTEGER NOT NULL,
    commitmonth       INTEGER,
    commitmonthname   VARCHAR(10),
    commityearkey     INTEGER,
    lastmonthkey      INTEGER,
    lastyearmonthkey  INTEGER,
    CONSTRAINT pk_cm PRIMARY KEY (commitmonthkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_commityear (
    commityearkey     INTEGER NOT NULL,
    commityear        INTEGER,
    lastyearkey       INTEGER,
    CONSTRAINT pk_cy PRIMARY KEY (commityearkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_receiptday (
    receiptdate       DATE NOT NULL,
    receiptday        INTEGER,
    receiptdayname    VARCHAR(10),
    receiptmonthkey   INTEGER,
    receiptyearkey    INTEGER,
    lastmonthdate     DATE,
```

Literaturverzeichnis

```
    lastyeardate    DATE,
    CONSTRAINT pk_rd PRIMARY KEY (receiptdate)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_receiptmonth (
    receiptmonthkey  INTEGER NOT NULL,
    receiptmonth     INTEGER,
    receiptmonthname VARCHAR(10),
    receiptyearkey   INTEGER,
    lastmonthkey     INTEGER,
    lastyearmonthkey INTEGER,
    CONSTRAINT pk_rm PRIMARY KEY (receiptmonthkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_receiptyear (
    receiptyearkey  INTEGER NOT NULL,
    receiptyear     INTEGER,
    lastyearkey     INTEGER,
    CONSTRAINT pk_ry PRIMARY KEY (receiptyearkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_orderday (
    orderdate       DATE NOT NULL,
    orderday        INTEGER,
    orderdayname    VARCHAR(10),
    ordermonthkey   INTEGER,
    orderyearkey    INTEGER,
    lastmonthdate   DATE,
    lastyeardate    DATE,
    CONSTRAINT pk_od PRIMARY KEY (orderdate)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_ordermonth (
    ordermonthkey   INTEGER NOT NULL,
    ordermonth      INTEGER,
    ordermonthname  VARCHAR(10),
    orderyearkey    INTEGER,
    lastmonthkey    INTEGER,
    lastyearmonthkey INTEGER,
    CONSTRAINT pk_om PRIMARY KEY (ordermonthkey)
)TABLESPACE TPCH4;

CREATE TABLE tpch4.lookup_orderyear (
    orderyearkey    INTEGER NOT NULL,
    orderyear       INTEGER,
    lastyearkey     INTEGER,
    CONSTRAINT pk_oy PRIMARY KEY (orderyearkey)
)TABLESPACE TPCH4;

ANALYZE TABLE tpch4.lookup_shipday COMPUTE STATISTICS;
ANALYZE TABLE tpch4.lookup_shipmonth COMPUTE STATISTICS;
ANALYZE TABLE tpch4.lookup_shipyear COMPUTE STATISTICS;

ANALYZE TABLE tpch4.lookup_commitday COMPUTE STATISTICS;
ANALYZE TABLE tpch4.lookup_commitmonth COMPUTE STATISTICS;
ANALYZE TABLE tpch4.lookup_commityear COMPUTE STATISTICS;

ANALYZE TABLE tpch4.lookup_receiptday COMPUTE STATISTICS;
ANALYZE TABLE tpch4.lookup_receiptmonth COMPUTE STATISTICS;
ANALYZE TABLE tpch4.lookup_receiptyear COMPUTE STATISTICS;

ANALYZE TABLE tpch4.lookup_orderday COMPUTE STATISTICS;
```

Literaturverzeichnis

```
ANALYZE TABLE tpch4.lookup_ordermonth COMPUTE STATISTICS;  
ANALYZE TABLE tpch4.lookup_orderyear COMPUTE STATISTICS;
```

```
ANALYZE TABLE tpch4.relate_date COMPUTE STATISTICS;  
ANALYZE TABLE tpch4.relate_month COMPUTE STATISTICS;  
ANALYZE TABLE tpch4.relate_year COMPUTE STATISTICS;  
CONNECT RESET;
```

Original-DB2-Version

```
-- file: mk_tab_v01.sql  
-- author: Holger Schwarz  
-- last update: 07.11.2000 HS  
-- content: create additional tables for benchmark schema  
--           + runstats on created tables  
-----  
CONNECT TO TPCH;  
  
CREATE TABLE tpch4.relate_date (  
    basisdate          DATE NOT NULL,  
    lastmonthdate     DATE,  
    lastyeardate      DATE,  
    CONSTRAINT pk_reld PRIMARY KEY (basisdate)  
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;  
  
ALTER TABLE tpch4.relate_date ACTIVATE NOT LOGGED INITIALLY;  
  
-- start date 01/01/1992, end date 12/31/1998  
-- use import (pp. 644) with option INSERT_UPDATE or REPLACE  
IMPORT FROM dates.del OF DEL  
    MESSAGES import.msg  
    INSERT_UPDATE INTO tpch4.relate_date;  
  
CREATE TABLE tpch4.relate_month (  
    basismonthkey     INTEGER NOT NULL,  
    lastmonthkey      INTEGER,  
    lastyearmonthkey  INTEGER,  
    CONSTRAINT pk_relm PRIMARY KEY (basismonthkey)  
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;  
  
ALTER TABLE tpch4.relate_month ACTIVATE NOT LOGGED INITIALLY;  
  
-- start date 01/01/1992, end date 12/31/1998  
-- use import (pp. 644) with option INSERT_UPDATE or REPLACE  
IMPORT FROM months.del OF DEL  
    MESSAGES import.msg  
    INSERT_UPDATE INTO tpch4.relate_month;  
  
CREATE TABLE tpch4.relate_year (  
    basisyearkey      INTEGER NOT NULL,  
    lastyearkey       INTEGER,  
    CONSTRAINT pk_rely PRIMARY KEY (basisyearkey)  
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;  
  
ALTER TABLE tpch4.relate_year ACTIVATE NOT LOGGED INITIALLY;  
  
-- start date 01/01/1992, end date 12/31/1998  
-- use import (pp. 644) with option INSERT_UPDATE or REPLACE  
IMPORT FROM years.del OF DEL  
    MESSAGES import.msg  
    INSERT_UPDATE INTO tpch4.relate_year;
```

Literaturverzeichnis

```
-- BEGIN H.S. 31.07.2000

DROP TABLE tpch4.temp01;

CREATE TABLE tpch4.temp01 (
  shipdate DATE NOT NULL,
  CONSTRAINT pk_sd PRIMARY KEY (shipdate)
) IN data1 INDEX IN data1;

INSERT INTO tpch4.temp01
  SELECT DISTINCT l.l_shipdate
  FROM tpch4.lineitem l;

-- END H.S.

CREATE TABLE tpch4.lookup_shipday (
  shipdate DATE NOT NULL,
  shipday INTEGER,
  shipdayname VARCHAR(10),
  shipmonthkey INTEGER,
  shipyearkey INTEGER,
  lastmonthdate DATE,
  lastyeardate DATE,
  CONSTRAINT pk_sd PRIMARY KEY (shipdate)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_shipday ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_shipday
  SELECT DISTINCT l.shipdate,
  DAY(l.shipdate),
  DAYNAME(l.shipdate),
  YEAR(l.shipdate)*100+MONTH(l.shipdate),
  YEAR(l.shipdate),
  rd.lastmonthdate,
  rd.lastyeardate
  FROM tpch4.temp01 l, tpch4.relate_date rd
  WHERE l.shipdate = rd.basisdate;

-- END H.S.

CREATE TABLE tpch4.lookup_shipmonth (
  shipmonthkey INTEGER NOT NULL,
  shipmonth INTEGER,
  shipmonthname VARCHAR(10),
  shipyearkey INTEGER,
  lastmonthkey INTEGER,
  lastyearmonthkey INTEGER,
  CONSTRAINT pk_sm PRIMARY KEY (shipmonthkey)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_shipmonth ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_shipmonth
  SELECT DISTINCT YEAR(l.shipdate)*100+MONTH(l.shipdate),
  MONTH(l.shipdate),
  MONTHNAME(l.shipdate),
```

Literaturverzeichnis

```
YEAR(l.shipdate),
rm.lastmonthkey,
rm.lastyearmonthkey
FROM tpch4.temp01 l, tpch4.lookup_shipday lsd, tpch4.relate_month rm
WHERE l.l.shipdate = lsd.shipdate AND lsd.shipmonthkey = rm.basismonthkey;

-- END H.S.

CREATE TABLE tpch4.lookup_shipyear (
  shipyearkey INTEGER NOT NULL,
  shipyear INTEGER,
  lastyearkey INTEGER,
  CONSTRAINT pk_sy PRIMARY KEY (shipyearkey)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_shipyear ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_shipyear
  SELECT DISTINCT YEAR(l.shipdate),
  YEAR(l.shipdate),
  ry.lastyearkey
  FROM tpch4.temp01 l, tpch4.lookup_shipday lsd, tpch4.relate_year ry
  WHERE l.shipdate = lsd.shipdate AND lsd.shipyearkey = ry.basisyearkey;

-- END

-- BEGIN H.S. 31.07.2000

DROP TABLE tpch4.temp01;

CREATE TABLE tpch4.temp01 (
  commitdate DATE NOT NULL,
  CONSTRAINT pk_sd PRIMARY KEY (commitdate)
) IN data1 INDEX IN data1;

INSERT INTO tpch4.temp01
  SELECT DISTINCT l.l_commitdate
  FROM tpch4.lineitem l;

-- END H.S.

CREATE TABLE tpch4.lookup_commitday (
  commitdate DATE NOT NULL,
  commitday INTEGER,
  commitdayname VARCHAR(10),
  commitmonthkey INTEGER,
  commityearkey INTEGER,
  lastmonthdate DATE,
  lastyeardate DATE,
  CONSTRAINT pk_cd PRIMARY KEY (commitdate)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_commitday ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_commitday
  SELECT DISTINCT l.commitdate,
  DAY(l.commitdate),
  DAYNAME(l.commitdate),
```

Literaturverzeichnis

```
YEAR(l.commitdate)*100+MONTH(l.commitdate),
YEAR(l.commitdate),
rd.lastmonthdate,
rd.lastyeardate
FROM tpch4.temp01 l, tpch4.relate_date rd
WHERE l.commitdate = rd.basisdate;

-- END H.S.

CREATE TABLE tpch4.lookup_commitmonth (
  commitmonthkey INTEGER NOT NULL,
  commitmonth INTEGER,
  commitmonthname VARCHAR(10),
  commityearkey INTEGER,
  lastmonthkey INTEGER,
  lastyearmonthkey INTEGER,
  CONSTRAINT pk_cm PRIMARY KEY (commitmonthkey)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_commitmonth ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_commitmonth
  SELECT DISTINCT YEAR(l.commitdate)*100+MONTH(l.commitdate),
  MONTH(l.commitdate),
  MONTHNAME(l.commitdate),
  YEAR(l.commitdate),
  rm.lastmonthkey,
  rm.lastyearmonthkey
  FROM tpch4.temp01 l, tpch4.lookup_commitday lcd, tpch4.relate_month rm
  WHERE l.commitdate = lcd.commitdate
  AND lcd.commitmonthkey = rm.basismonthkey;

-- END H.S.

CREATE TABLE tpch4.lookup_commityear (
  commityearkey INTEGER NOT NULL,
  commityear INTEGER,
  lastyearkey INTEGER,
  CONSTRAINT pk_cy PRIMARY KEY (commityearkey)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_commityear ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_commityear
  SELECT DISTINCT YEAR(l.commitdate),
  YEAR(l.commitdate),
  ry.lastyearkey
  FROM tpch4.temp01 l, tpch4.lookup_commitday lcd, tpch4.relate_year ry
  WHERE l.commitdate = lcd.commitdate
  AND lcd.commityearkey = ry.basisyearkey;

-- END H.S.

-- BEGIN H.S. 31.07.2000

DROP TABLE tpch4.temp01;

CREATE TABLE tpch4.temp01 (
```

Literaturverzeichnis

```
receiptdate DATE NOT NULL,
CONSTRAINT pk_sd PRIMARY KEY (receiptdate)
) IN data1 INDEX IN data1;

INSERT INTO tpch4.temp01
SELECT DISTINCT l.l_receiptdate
FROM tpch4.lineitem l;

-- END H.S.

CREATE TABLE tpch4.lookup_receiptday (
receiptdate DATE NOT NULL,
receiptday INTEGER,
receiptdayname VARCHAR(10),
receiptmonthkey INTEGER,
receiptyearkey INTEGER,
lastmonthdate DATE,
lastyeardate DATE,
CONSTRAINT pk_rd PRIMARY KEY (receiptdate)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_receiptday ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_receiptday
SELECT DISTINCT l_receiptdate,
DAY(l_receiptdate),
DAYNAME(l_receiptdate),
YEAR(l_receiptdate)*100+MONTH(l_receiptdate),
YEAR(l_receiptdate),
rd.lastmonthdate,
rd.lastyeardate
FROM tpch4.temp01 l, tpch4.relate_date rd
WHERE l_receiptdate = rd.basisdate;

-- END H.S.

CREATE TABLE tpch4.lookup_receiptmonth (
receiptmonthkey INTEGER NOT NULL,
receiptmonth INTEGER,
receiptmonthname VARCHAR(10),
receiptyearkey INTEGER,
lastmonthkey INTEGER,
lastyearmonthkey INTEGER,
CONSTRAINT pk_rm PRIMARY KEY (receiptmonthkey)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_receiptmonth ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_receiptmonth
SELECT DISTINCT YEAR(l_receiptdate)*100+MONTH(l_receiptdate),
MONTH(l_receiptdate),
MONTHNAME(l_receiptdate),
YEAR(l_receiptdate),
rm.lastmonthkey,
rm.lastyearmonthkey
FROM tpch4.temp01 l, tpch4.lookup_receiptday lrd, tpch4.relate_month rm
WHERE l_receiptdate = lrd_receiptdate
AND lrd_receiptmonthkey = rm.basismonthkey;
```

Literaturverzeichnis

```
-- END H.S.

CREATE TABLE tpch4.lookup_receiptyear (
  receiptyearkey INTEGER NOT NULL,
  receiptyear INTEGER,
  lastyearkey INTEGER,
  CONSTRAINT pk_ry PRIMARY KEY (receiptyearkey)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_receiptyear ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_receiptyear
  SELECT DISTINCT YEAR(l.receiptdate),
  YEAR(l.receiptdate),
  ry.lastyearkey
  FROM tpch4.temp01 l, tpch4.lookup_receiptday lrd, tpch4.relate_year ry
  WHERE l.receiptdate = lrd.receiptdate
  AND lrd.receiptyearkey = ry.basisyearkey;

-- END H.S.

-- BEGIN H.S. 31.07.2000

DROP TABLE tpch4.temp01;

CREATE TABLE tpch4.temp01 (
  orderdate DATE NOT NULL,
  CONSTRAINT pk_sd PRIMARY KEY (orderdate)
) IN data1 INDEX IN data1;

INSERT INTO tpch4.temp01
  SELECT DISTINCT o.o_orderdate
  FROM tpch4.orders o;

-- END H.S.

CREATE TABLE tpch4.lookup_orderday (
  orderdate DATE NOT NULL,
  orderday INTEGER,
  orderdayname VARCHAR(10),
  ordermonthkey INTEGER,
  orderyearkey INTEGER,
  lastmonthdate DATE,
  lastyeardate DATE,
  CONSTRAINT pk_od PRIMARY KEY (orderdate)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_orderday ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_orderday
  SELECT DISTINCT o.orderdate,
  DAY(o.orderdate),
  DAYNAME(o.orderdate),
  YEAR(o.orderdate)*100+MONTH(o.orderdate),
  YEAR(o.orderdate),
  rd.lastmonthdate,
  rd.lastyeardate
```

Literaturverzeichnis

```
FROM tpch4.temp01 o, tpch4.relate_date rd
WHERE o.orderdate = rd.basisdate;

-- END H.S.

CREATE TABLE tpch4.lookup_ordermonth (
  ordermonthkey INTEGER NOT NULL,
  ordermonth INTEGER,
  ordermonthname VARCHAR(10),
  orderyearkey INTEGER,
  lastmonthkey INTEGER,
  lastyearmonthkey INTEGER,
  CONSTRAINT pk_om PRIMARY KEY (ordermonthkey)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_ordermonth ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_ordermonth
  SELECT DISTINCT YEAR(o.orderdate)*100+MONTH(o.orderdate),
  MONTH(o.orderdate),
  MONTHNAME(o.orderdate),
  YEAR(o.orderdate),
  rm.lastmonthkey,
  rm.lastyearmonthkey
  FROM tpch4.temp01 o, tpch4.lookup_orderday lod, tpch4.relate_month rm
  WHERE o.orderdate = lod.orderdate AND lod.ordermonthkey = rm.basismonthkey;

-- END H.S.

CREATE TABLE tpch4.lookup_orderyear (
  orderyearkey INTEGER NOT NULL,
  orderyear INTEGER,
  lastyearkey INTEGER,
  CONSTRAINT pk_oy PRIMARY KEY (orderyearkey)
) IN data1 INDEX IN data1 NOT LOGGED INITIALLY;

ALTER TABLE tpch4.lookup_orderyear ACTIVATE NOT LOGGED INITIALLY;

-- BEGIN H.S. 31.07.2000

INSERT INTO tpch4.lookup_orderyear
  SELECT DISTINCT YEAR(o.orderdate),
  YEAR(o.orderdate),
  ry.lastyearkey
  FROM tpch4.temp01 o, tpch4.lookup_orderday lod, tpch4.relate_year ry
  WHERE o.orderdate = lod.orderdate AND lod.orderyearkey = ry.basisyearkey;

-- END H.S.

RUNSTATS ON TABLE tpch4.lookup_shipday
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.lookup_shipmonth
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.lookup_shipyear
WITH DISTRIBUTION AND DETAILED INDEXES ALL;

RUNSTATS ON TABLE tpch4.lookup_commitday
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.lookup_commitmonth
```

Literaturverzeichnis

```
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.lookup_commityear
WITH DISTRIBUTION AND DETAILED INDEXES ALL;

RUNSTATS ON TABLE tpch4.lookup_receiptday
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.lookup_receiptmonth
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.lookup_receiptyear
WITH DISTRIBUTION AND DETAILED INDEXES ALL;

RUNSTATS ON TABLE tpch4.lookup_orderday
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.lookup_ordermonth
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.lookup_orderyear
WITH DISTRIBUTION AND DETAILED INDEXES ALL;

RUNSTATS ON TABLE tpch4.relate_date
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.relate_month
WITH DISTRIBUTION AND DETAILED INDEXES ALL;
RUNSTATS ON TABLE tpch4.relate_year
WITH DISTRIBUTION AND DETAILED INDEXES ALL;

CONNECT RESET;
```

A.1.5 mk_view_v01.sql

Oracle-Version

```
-- file: mk_view_v01.sql
-- author: Holger Schwarz
-- last update: 07.11.2000 HS
-- content: create views for benchmark schema
-----
CONNECT TPCH;
CREATE VIEW tpch4.lineitem_orders (
  orderkey, partkey, suppkey, custkey, linenumber,
  shipdate, commitdate, receiptdate, orderdate,
  quantity, extendedprice, discount, tax, endprice,
  returnflag, linestatus, shipinstruct, shipmode) AS
  SELECT l.l_orderkey, l.l_partkey, l.l_suppkey, o.o_custkey, l.l_linenumber,
  l.l_shipdate, l.l_commitdate, l.l_receiptdate, o.o_orderdate,
  l.l_quantity, l.l_extendedprice, l.l_discount, l.l_tax,
  l.l_extendedprice*(1- l.l_discount)*(1+l.l_tax),
  l.l_returnflag, l.l_linestatus, l.l_shipinstruct, l.l_shipmode
  FROM tpch4.lineitem l, tpch4.orders o
  WHERE l.l_orderkey = o.o_orderkey;

CREATE VIEW tpch4.lookup_orders (
  orderkey, orderstatus, orderpriority, shippriority, clerk) AS
  SELECT o_orderkey, o_orderstatus, o_orderpriority, o_shippriority, o_clerk
  FROM tpch4.orders;

CREATE VIEW tpch4.lookup_orderstatus (orderstatus) AS
  SELECT DISTINCT o_orderstatus FROM tpch4.orders;

CREATE VIEW tpch4.lookup_orderpriority (orderpriority) AS
  SELECT DISTINCT o_orderpriority FROM tpch4.orders;
```

Literaturverzeichnis

```
CREATE VIEW tpch4.lookup_shippriority (shippriority) AS
  SELECT DISTINCT o_shippriority FROM tpch4.orders;

CREATE VIEW tpch4.lookup_clerk (clerk) AS
  SELECT DISTINCT o_clerk FROM tpch4.orders;

CREATE VIEW tpch4.customer_acctbal (custkey, custacctbal) AS
  SELECT c_custkey, c_acctbal FROM tpch4.customer;

CREATE VIEW tpch4.lookup_customer (
  custkey, custname, custnationkey, custregionkey,
  custaddress, custphone, mktsegment) AS
  SELECT c.c_custkey, c.c_name, c.c_nationkey, n.n_regionkey,
  c.c_address, c.c_phone, c.c_mktsegment
  FROM tpch4.customer c, tpch4.nation n
  WHERE c.c_nationkey = n.n_nationkey;

CREATE VIEW tpch4.lookup_custnation (
  custnationkey, custnationname, custregionkey) AS
  SELECT n_nationkey, n_name, n_regionkey
  FROM tpch4.nation;

CREATE VIEW tpch4.lookup_custregion (custregionkey, custregionname) AS
  SELECT r_regionkey, r_name FROM tpch4.region;

CREATE VIEW tpch4.lookup_custaddress (custaddress) AS
  SELECT DISTINCT c_address FROM tpch4.customer;

CREATE VIEW tpch4.lookup_custphone (custphone) AS
  SELECT DISTINCT c_phone FROM tpch4.customer;

CREATE VIEW tpch4.lookup_mktsegment (mktsegment) AS
  SELECT DISTINCT c_mktsegment FROM tpch4.customer;

CREATE VIEW tpch4.lookup_shipmode (shipmode) AS
  SELECT DISTINCT l_shipmode FROM tpch4.lineitem;

CREATE VIEW tpch4.lookup_shipinstruct (shipinstruct) AS
  SELECT DISTINCT l_shipinstruct FROM tpch4.lineitem;

CREATE VIEW tpch4.lookup_linestatus (linestatus) AS
  SELECT DISTINCT l_linestatus FROM tpch4.lineitem;

CREATE VIEW tpch4.lookup_returnflag (returnflag) AS
  SELECT DISTINCT l_returnflag FROM tpch4.lineitem;

CREATE VIEW tpch4.part_supplier (
  partkey, suppkey, availqty, supplycost) AS
  SELECT ps_partkey, ps_suppkey, ps_availqty, ps_supplycost
  FROM tpch4.partsupp;

CREATE VIEW tpch4.part_price (partkey, retailprice) AS
  SELECT p_partkey, p_retailprice FROM tpch4.part;

CREATE VIEW tpch4.lookup_part (
  partkey, partname, mfgr, brand, type, size, container) AS
  SELECT p_partkey, p_name, p_mfgr, p_brand, p_type, p_size, p_container
  FROM tpch4.part;

CREATE VIEW tpch4.lookup_mfgr (mfgr) AS
  SELECT DISTINCT p_mfgr FROM tpch4.part;
```

Literaturverzeichnis

```
CREATE VIEW tpch4.lookup_brand (brand) AS
  SELECT DISTINCT p_brand FROM tpch4.part;

CREATE VIEW tpch4.lookup_type (type) AS
  SELECT DISTINCT p_type FROM tpch4.part;

CREATE VIEW tpch4.lookup_size (size) AS
  SELECT DISTINCT p_size FROM tpch4.part;

CREATE VIEW tpch4.lookup_container (container) AS
  SELECT DISTINCT p_container FROM tpch4.part;

CREATE VIEW tpch4.supplier_acctbal (suppkey, suppacctbal) AS
  SELECT s_suppkey, s_acctbal FROM tpch4.supplier;

CREATE VIEW tpch4.lookup_supplier (
  suppkey, suppname, suppnationkey, suppreigionkey,
  suppaddress, suppphone) AS
  SELECT s.s_suppkey, s.s_name, s.s_nationkey, n.n_nationkey,
  s.s_address, s.s_phone
  FROM tpch4.supplier s, tpch4.nation n
  WHERE s.s_nationkey = n.n_nationkey;

CREATE VIEW tpch4.lookup_suppnation (
  suppnationkey, suppnationname, suppreigionkey) AS
  SELECT n_nationkey, n_name, n_regionkey
  FROM tpch4.nation;

CREATE VIEW tpch4.lookup_suppreigion (suppreigionkey, suppreigionname) AS
  SELECT r_regionkey, r_name FROM tpch4.region;

CREATE VIEW tpch4.lookup_suppaddress (suppaddress) AS
  SELECT DISTINCT s_address FROM tpch4.supplier;

CREATE VIEW tpch4.lookup_suppphone (suppphone) AS
  SELECT DISTINCT s_phone FROM tpch4.supplier;
CONNECT RESET;
```

Original-DB2-Version

```
-- file: mk_view_v01.sql
-- author: Holger Schwarz
-- last update: 07.11.2000 HS
-- content: create views for benchmark schema
-----
CONNECT TO TPCH;

CREATE VIEW tpch4.lineitem_orders (
  orderkey, partkey, suppkey, custkey, linenumber,
  shipdate, commitdate, receiptdate, orderdate,
  quantity, extendedprice, discount, tax, endprice,
  returnflag, linestatus, shipinstruct, shipmode) AS
  SELECT l.l_orderkey, l.l_partkey, l.l_suppkey, o.o_custkey, l.l_linenumber,
  l.l_shipdate, l.l_commitdate, l.l_receiptdate, o.o_orderdate,
  l.l_quantity, l.l_extendedprice, l.l_discount, l.l_tax,
  l.l_extendedprice*(1- l.l_discount)*(1+l.l_tax),
  l.l_returnflag, l.l_linestatus, l.l_shipinstruct, l.l_shipmode
  FROM tpch4.lineitem l, tpch4.orders o
  WHERE l.l_orderkey = o.o_orderkey;
```

Literaturverzeichnis

```
CREATE VIEW tpch4.lookup_orders (
  orderkey, orderstatus, orderpriority, shippriority, clerk) AS
  SELECT o_orderkey, o_orderstatus, o_orderpriority, o_shippriority, o_clerk
  FROM tpch4.orders;

CREATE VIEW tpch4.lookup_orderstatus (orderstatus) AS
  SELECT DISTINCT o_orderstatus FROM tpch4.orders;

CREATE VIEW tpch4.lookup_orderpriority (orderpriority) AS
  SELECT DISTINCT o_orderpriority FROM tpch4.orders;

CREATE VIEW tpch4.lookup_shippriority (shippriority) AS
  SELECT DISTINCT o_shippriority FROM tpch4.orders;

CREATE VIEW tpch4.lookup_clerk (clerk) AS
  SELECT DISTINCT o_clerk FROM tpch4.orders;

CREATE VIEW tpch4.customer_acctbal (custkey, custacctbal) AS
  SELECT c_custkey, c_acctbal FROM tpch4.customer;

CREATE VIEW tpch4.lookup_customer (
  custkey, custname, custnationkey, custregionkey,
  custaddress, custphone, mktsegment) AS
  SELECT c.c_custkey, c.c_name, c.c_nationkey, n.n_regionkey,
  c.c_address, c.c_phone, c.c_mktsegment
  FROM tpch4.customer c, tpch4.nation n
  WHERE c.c_nationkey = n.n_nationkey;

CREATE VIEW tpch4.lookup_custnation (
  custnationkey, custnationname, custregionkey) AS
  SELECT n_nationkey, n_name, n_regionkey
  FROM tpch4.nation;

CREATE VIEW tpch4.lookup_custregion (custregionkey, custregionname) AS
  SELECT r_regionkey, r_name FROM tpch4.region;

CREATE VIEW tpch4.lookup_custaddress (custaddress) AS
  SELECT DISTINCT c_address FROM tpch4.customer;

CREATE VIEW tpch4.lookup_custphone (custphone) AS
  SELECT DISTINCT c_phone FROM tpch4.customer;

CREATE VIEW tpch4.lookup_mktsegment (mktsegment) AS
  SELECT DISTINCT c_mktsegment FROM tpch4.customer;

CREATE VIEW tpch4.lookup_shipmode (shipmode) AS
  SELECT DISTINCT l_shipmode FROM tpch4.lineitem;

CREATE VIEW tpch4.lookup_shipinstruct (shipinstruct) AS
  SELECT DISTINCT l_shipinstruct FROM tpch4.lineitem;

CREATE VIEW tpch4.lookup_linestatus (linestatus) AS
  SELECT DISTINCT l_linestatus FROM tpch4.lineitem;

CREATE VIEW tpch4.lookup_returnflag (returnflag) AS
  SELECT DISTINCT l_returnflag FROM tpch4.lineitem;

CREATE VIEW tpch4.part_supplier (
  partkey, suppkey, availqty, supplycost) AS
  SELECT ps_partkey, ps_suppkey, ps_availqty, ps_supplycost
  FROM tpch4.partsupp;
```

Literaturverzeichnis

```
CREATE VIEW tpch4.part_price (partkey, retailprice) AS
  SELECT p_partkey, p_retailprice FROM tpch4.part;

CREATE VIEW tpch4.lookup_part (
  partkey, partname, mfg, brand, type, size, container) AS
  SELECT p_partkey, p_name, p_mfg, p_brand, p_type, p_size, p_container
  FROM tpch4.part;

CREATE VIEW tpch4.lookup_mfg (mfg) AS
  SELECT DISTINCT p_mfg FROM tpch4.part;

CREATE VIEW tpch4.lookup_brand (brand) AS
  SELECT DISTINCT p_brand FROM tpch4.part;

CREATE VIEW tpch4.lookup_type (type) AS
  SELECT DISTINCT p_type FROM tpch4.part;

CREATE VIEW tpch4.lookup_size (size) AS
  SELECT DISTINCT p_size FROM tpch4.part;

CREATE VIEW tpch4.lookup_container (container) AS
  SELECT DISTINCT p_container FROM tpch4.part;

CREATE VIEW tpch4.supplier_acctbal (suppkey, suppacctbal) AS
  SELECT s_suppkey, s_acctbal FROM tpch4.supplier;

CREATE VIEW tpch4.lookup_supplier (
  suppkey, suppname, suppnationkey, suppregionkey,
  suppaddress, suppphone) AS
  SELECT s.s_suppkey, s.s_name, s.s_nationkey, n.n_nationkey,
  s.s_address, s.s_phone
  FROM tpch4.supplier s, tpch4.nation n
  WHERE s.s_nationkey = n.n_nationkey;

CREATE VIEW tpch4.lookup_suppnation (
  suppnationkey, suppnationname, suppregionkey) AS
  SELECT n_nationkey, n_name, n_regionkey
  FROM tpch4.nation;

CREATE VIEW tpch4.lookup_suppregion (suppregionkey, suppregionname) AS
  SELECT r_regionkey, r_name FROM tpch4.region;

CREATE VIEW tpch4.lookup_suppaddress (suppaddress) AS
  SELECT DISTINCT s_address FROM tpch4.supplier;

CREATE VIEW tpch4.lookup_suppphone (suppphone) AS
  SELECT DISTINCT s_phone FROM tpch4.supplier;

CONNECT RESET;
```

A.2 SQL*Loader Control- und Logdatei

```
-----
-- file: tpch4-customerctl
-- author: Thorsten Müller
-----

LOAD DATA
INFILE 'tpch4-customer.data'
INTO TABLE tpch4.customer
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
```

Literaturverzeichnis

```
( c_custkey,  
  c_name,  
  c_address,  
  c_nationkey,  
  c_phone,  
  c_acctbal,  
  c_mktsegment,  
  c_comment)
```

```
-----  
-- file: tpch4-customer.log  
-- author: Thorsten Müller  
-----
```

SQL*Loader: Release 10.1.0.2.0 - Production on Tue Jun 22 11:26:04 2004

Copyright (c) 1982, 2004, Oracle. All rights reserved.

```
Control File:  tpch4-customer.ctl  
Data File:    tpch4-customer.data  
  Bad File:   tpch4-customer.bad  
  Discard File: none specified
```

(Allow all discards)

```
Number to load: ALL  
Number to skip: 0  
Errors allowed: 50  
Bind array:    64 rows, maximum of 256000 bytes  
Continuation:  none specified  
Path used:    Conventional
```

Table TPCH4.CUSTOMER, loaded from every logical record.
Insert option in effect for this table: INSERT

Column Name	Position	Len	Term	Encl	Datatype
C_CUSTKEY	FIRST	*	,	0(")	CHARACTER
C_NAME	NEXT	*	,	0(")	CHARACTER
C_ADDRESS	NEXT	*	,	0(")	CHARACTER
C_NATIONKEY	NEXT	*	,	0(")	CHARACTER
C_PHONE	NEXT	*	,	0(")	CHARACTER
C_ACCTBAL	NEXT	*	,	0(")	CHARACTER
C_MKTSEGMENT	NEXT	*	,	0(")	CHARACTER
C_COMMENT	NEXT	*	,	0(")	CHARACTER

Table TPCH4.CUSTOMER:
1500000 Rows successfully loaded.
0 Rows not loaded due to data errors.
0 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.

```
Space allocated for bind array:          132096 bytes(64 rows)  
Read  buffer bytes: 1048576
```

```
Total logical records skipped:          0  
Total logical records read:             1500000  
Total logical records rejected:          0  
Total logical records discarded:         0
```

```
Run began on Tue Jun 22 11:26:04 2004  
Run ended on Tue Jun 22 11:30:01 2004
```

Literaturverzeichnis

Elapsed time was: 00:03:56.72
CPU time was: 00:00:32.79

Anhang B:

B.1 Performance Sequenzen

B.1.1 c-sequence-sequence_0-sequence.sql

```
-- FILE      : \c-sequence\sequence_0\sequence.sql
-- CREATED   : 2004-08-10
-- AUTHOR    : Thorsten Müller
-- PURPOSE   : TPC-H Schema Performance test
```

```
-----
CREATE TABLE tpch4.temptableC010(
  custkey INTEGER,
  turnover1992 DECIMAL(31, 6)) TABLESPACE TPCH4;
```

```
CREATE TABLE tpch4.temptableC020(
  custkey INTEGER,
  turnover1993 DECIMAL(31, 6)) TABLESPACE TPCH4;
```

```
CREATE TABLE tpch4.temptableC030(
  custkey INTEGER,
  turnover1994 DECIMAL(31, 6)) TABLESPACE TPCH4;
```

```
CREATE TABLE tpch4.temptableC040(
  custkey INTEGER,
  turnover1992 FLOAT,
  turnover1993 FLOAT,
  turnover1994 FLOAT) TABLESPACE TPCH4;
```

```
CREATE TABLE tpch4.temptableC050(
  custkey INTEGER,
  stddeviation DECIMAL(31, 6)) TABLESPACE TPCH4;
```

```
CREATE TABLE tpch4.temptableC060(
  custkey INTEGER,
  stddeviation FLOAT) TABLESPACE TPCH4;
```

```
CREATE TABLE tpch4.temptableC070(
  custkey INTEGER,
  custname VARCHAR(25),
  stddev1 DECIMAL(10, 2),
  stddev2 FLOAT,
  turnover1992 FLOAT,
  turnover1993 FLOAT,
  turnover1994 FLOAT) TABLESPACE TPCH4;
```

```
CREATE TABLE tpch4.temptableC080(
  custkey INTEGER,
  custname VARCHAR(25),
  stddev1 DECIMAL(10, 2),
```

Literaturverzeichnis

```
stddev2 FLOAT,  
turnover1992 FLOAT,  
turnover1993 FLOAT,  
turnover1994 FLOAT) TABLESPACE TPCH4;  
  
INSERT INTO tpch4.temptableC010  
SELECT  
  a1.custkey,  
  sum(a1.endprice)  
FROM  
  tpch4.lineitem_orders a1,  
  tpch4.lookup_orderday a2  
WHERE  
  a2.orderdate = a1.orderdate AND  
  a2.orderyearkey = 1992  
GROUP BY  
  a1.custkey;  
  
INSERT INTO tpch4.temptableC020  
SELECT  
  a1.custkey,  
  sum(a1.endprice)  
FROM  
  tpch4.lineitem_orders a1,  
  tpch4.lookup_orderday a2  
WHERE  
  a2.orderdate = a1.orderdate AND  
  a2.orderyearkey = 1993  
GROUP BY  
  a1.custkey;  
  
INSERT INTO tpch4.temptableC030  
SELECT  
  a1.custkey,  
  sum(a1.endprice)  
FROM  
  tpch4.lineitem_orders a1,  
  tpch4.lookup_orderday a2  
WHERE  
  a2.orderdate = a1.orderdate AND  
  a2.orderyearkey = 1994  
GROUP BY  
  a1.custkey;  
  
INSERT INTO tpch4.temptableC040  
SELECT  
  a1.custkey,  
  a1.turnover1992,  
  a2.turnover1993,  
  a3.turnover1994  
FROM  
  tpch4.temptableC010 a1,  
  tpch4.temptableC020 a2,  
  tpch4.temptableC030 a3  
WHERE  
  a1.custkey = a2.custkey AND  
  a1.custkey = a3.custkey AND  
  a1.turnover1992 >= 500000 AND  
  a2.turnover1993 >= 500000 AND  
  a3.turnover1994 >= 500000;  
  
INSERT INTO tpch4.temptableC050
```

Literaturverzeichnis

```
SELECT
  a1.custkey,
  stddev(a1.endprice)
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2,
  tpch4.temptableC040 a3
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey IN (1992, 1993, 1994) AND
  a1.custkey = a3.custkey
GROUP BY
  a1.custkey;

INSERT INTO tpch4.temptableC060
SELECT
  a1.custkey,
  stddev(a1.endprice)/(CASE avg(a1.endprice) WHEN 0
    THEN NULL ELSE avg(a1.endprice) END)
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2,
  tpch4.temptableC040 a3
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey IN (1992, 1993, 1994) AND
  a1.custkey = a3.custkey
GROUP BY
  a1.custkey;

INSERT INTO tpch4.temptableC070
SELECT
  a6.custkey,
  a6.custname,
  a4.stddeviation,
  a5.stddeviation,
  a1.turnover1992,
  a2.turnover1993,
  a3.turnover1994
FROM
  tpch4.temptableC010 a1,
  tpch4.temptableC020 a2,
  tpch4.temptableC030 a3,
  tpch4.temptableC050 a4,
  tpch4.temptableC060 a5,
  tpch4.lookup_customer a6
WHERE
  a4.custkey = a1.custkey AND
  a4.custkey = a2.custkey AND
  a4.custkey = a3.custkey AND
  a4.custkey = a5.custkey AND
  a4.custkey = a6.custkey;

INSERT INTO tpch4.temptableC080
SELECT
  a1.custkey,
  a1.custname,
  a1.stddev1,
  a1.stddev2,
  a1.turnover1992,
  a1.turnover1993,
  a1.turnover1994
```

```
FROM
  tpch4.temptableC070 a1
WHERE
  a1.stddev2 <= 0.66794004454646;

DROP TABLE tpch4.temptableC010;
DROP TABLE tpch4.temptableC020;
DROP TABLE tpch4.temptableC030;
DROP TABLE tpch4.temptableC040;
DROP TABLE tpch4.temptableC050;
DROP TABLE tpch4.temptableC060;
DROP TABLE tpch4.temptableC070;
DROP TABLE tpch4.temptableC080;
```

B.1.2 c-sequence-sequence_0-singlequery.sql

```
-- FILE      : \c-sequence\sequence_0\singlequery.sql
-- CREATED   : 2004-08-10
-- AUTHOR    : Thorsten Müller
-- PURPOSE   : TPC-H Schema Performance test
-----
```

```
CREATE TABLE tpch4.temptableC080(
  custkey INTEGER,
  custname VARCHAR(25),
  stddev1 DECIMAL(10, 2),
  stddev2 FLOAT,
  turnover1992 FLOAT,
  turnover1993 FLOAT,
  turnover1994 FLOAT) TABLESPACE TPCH4;
```

```
INSERT INTO tpch4.temptableC080
SELECT
```

```
  a1.custkey,
  a1.custname,
  a1.stddev1,
  a1.stddev2,
  a1.turnover1992,
  a1.turnover1993,
  a1.turnover1994
FROM
  (SELECT
    a6.custkey custkey,
    a6.custname custname,
    a4.stddeviation stddev1,
    a5.stddeviation stddev2,
    a1.turnover1992 turnover1992,
    a2.turnover1993 turnover1993,
    a3.turnover1994 turnover1994
  FROM
    (SELECT
      a1.custkey custkey,
      sum(a1.endprice) turnover1992
    FROM
      tpch4.lineitem_orders a1,
      tpch4.lookup_orderday a2
    WHERE
      a2.orderdate = a1.orderdate AND
      a2.orderyearkey = 1992
    GROUP BY
      a1.custkey
```


Literaturverzeichnis

```
        a1.custkey custkey,
        sum(a1.endprice) turnover1994
FROM
    tpch4.lineitem_orders a1,
    tpch4.lookup_orderday a2
WHERE
    a2.orderdate = a1.orderdate AND
    a2.orderyearkey = 1994
GROUP BY
    a1.custkey
) a3
WHERE
    a1.custkey = a2.custkey AND
    a1.custkey = a3.custkey AND
    a1.turnover1992 >= 500000 AND
    a2.turnover1993 >= 500000 AND
    a3.turnover1994 >= 500000
) a3
WHERE
    a2.orderdate = a1.orderdate AND
    a2.orderyearkey IN (1992, 1993, 1994) AND
    a1.custkey = a3.custkey
GROUP BY
    a1.custkey
) a4,
(SELECT
    a1.custkey custkey,
    stddev(a1.endprice)/(CASE avg(a1.endprice) WHEN 0
        THEN NULL ELSE avg(a1.endprice) END)  stddeviation
FROM
    tpch4.lineitem_orders a1,
    tpch4.lookup_orderday a2,
    (SELECT
        a1.custkey custkey,
        a1.turnover1992  turnover1992,
        a2.turnover1993  turnover1993,
        a3.turnover1994  turnover1994
    FROM
        (SELECT
            a1.custkey custkey,
            sum(a1.endprice) turnover1992
        FROM
            tpch4.lineitem_orders a1,
            tpch4.lookup_orderday a2
        WHERE
            a2.orderdate = a1.orderdate AND
            a2.orderyearkey = 1992
        GROUP BY
            a1.custkey
        ) a1,
        (SELECT
            a1.custkey custkey,
            sum(a1.endprice) turnover1993
        FROM
            tpch4.lineitem_orders a1,
            tpch4.lookup_orderday a2
        WHERE
            a2.orderdate = a1.orderdate AND
            a2.orderyearkey = 1993
        GROUP BY
            a1.custkey
        ) a2,
```

```
(SELECT
  a1.custkey custkey,
  sum(a1.endprice) turnover1994
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey = 1994
GROUP BY
  a1.custkey
) a3
WHERE
  a1.custkey = a2.custkey AND
  a1.custkey = a3.custkey AND
  a1.turnover1992 >= 500000 AND
  a2.turnover1993 >= 500000 AND
  a3.turnover1994 >= 500000
) a3
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey IN (1992, 1993, 1994) AND
  a1.custkey = a3.custkey
GROUP BY
  a1.custkey
) a5,
tpch4.lookup_customer a6
WHERE
  a4.custkey = a1.custkey AND
  a4.custkey = a2.custkey AND
  a4.custkey = a3.custkey AND
  a4.custkey = a5.custkey AND
  a4.custkey = a6.custkey
) a1
WHERE
  a1.stddev2 <= 0.66794004454646;

DROP TABLE tpch4 temptableC080;
```

B.1.3 c-sequence-sequence_0-with.sql

```
-- FILE      : \c-sequence\sequence_0\with.sql
-- CREATED   : 2004-08-10
-- AUTHOR    : Thorsten Müller
-- PURPOSE   : TPC-H Schema Performance test
```

```
-----
CREATE TABLE tpch4.temptableC080(
  custkey INTEGER,
  custname VARCHAR(25),
  stddev1 DECIMAL(10, 2),
  stddev2 FLOAT,
  turnover1992 FLOAT,
  turnover1993 FLOAT,
  turnover1994 FLOAT) TABLESPACE TPCH4;
```

```
INSERT INTO tpch4.temptableC080
WITH
  temptableC010 AS (
  SELECT
    a1.custkey custkey,
```

Literaturverzeichnis

```
    sum(a1.endprice) turnover1992
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey = 1992
GROUP BY
  a1.custkey),
temptableC020 AS (
SELECT
  a1.custkey custkey,
  sum(a1.endprice) turnover1993
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey = 1993
GROUP BY
  a1.custkey),
temptableC030 AS (
SELECT
  a1.custkey custkey,
  sum(a1.endprice) turnover1994
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey = 1994
GROUP BY
  a1.custkey),
temptableC040 AS (
SELECT
  a1.custkey custkey,
  a1.turnover1992 turnover1992,
  a2.turnover1993 turnover1993,
  a3.turnover1994 turnover1994
FROM
  temptableC010 a1,
  temptableC020 a2,
  temptableC030 a3
WHERE
  a1.custkey = a2.custkey AND
  a1.custkey = a3.custkey AND
  a1.turnover1992 >= 500000 AND
  a2.turnover1993 >= 500000 AND
  a3.turnover1994 >= 500000),
temptableC050 AS (
SELECT
  a1.custkey custkey,
  stddev(a1.endprice) stddeviation
FROM
  tpch4.lineitem_orders a1,
  tpch4.lookup_orderday a2,
  temptableC040 a3
WHERE
  a2.orderdate = a1.orderdate AND
  a2.orderyearkey IN (1992, 1993, 1994) AND
  a1.custkey = a3.custkey
GROUP BY
```

Literaturverzeichnis

```
    a1.custkey),
temptableC060 AS (
  SELECT
    a1.custkey custkey,
    stddev(a1.endprice)/(CASE avg(a1.endprice) WHEN 0
      THEN NULL ELSE avg(a1.endprice) END) stddeviation
  FROM
    tpch4.lineitem_orders a1,
    tpch4.lookup_orderday a2,
    temptableC040 a3
  WHERE
    a2.orderdate = a1.orderdate AND
    a2.orderyearkey IN (1992, 1993, 1994) AND
    a1.custkey = a3.custkey
  GROUP BY
    a1.custkey),
temptableC070 AS (
  SELECT
    a6.custkey custkey,
    a6.custname custname,
    a4.stddeviation stddev1,
    a5.stddeviation stddev2,
    a1.turnover1992 turnover1992,
    a2.turnover1993 turnover1993,
    a3.turnover1994 turnover1994
  FROM
    temptableC010 a1,
    temptableC020 a2,
    temptableC030 a3,
    temptableC050 a4,
    temptableC060 a5,
    tpch4.lookup_customer a6
  WHERE
    a4.custkey = a1.custkey AND
    a4.custkey = a2.custkey AND
    a4.custkey = a3.custkey AND
    a4.custkey = a5.custkey AND
    a4.custkey = a6.custkey)
SELECT
  a1.custkey,
  a1.custname,
  a1.stddev1,
  a1.stddev2,
  a1.turnover1992,
  a1.turnover1993,
  a1.turnover1994
FROM
  temptableC070 a1
WHERE
  a1.stddev2 <= 0.66794004454646;

DROP TABLE tpch4.temptableC080;
```

Anhang C:

C.1 Zeitmessungen der einzelnen Skriptvarianten

Skriptvariante	1. Zeitmessung	2. Zeitmessung	3. Zeitmessung	Schnitt
seq_0:sequence.sql	0:39:12	0:38:48	0:38:48	0:38:56
seq_0:singlequery.sql	0:35:22	0:35:14	0:35:02	0:35:13
seq_0:with.sql	0:34:50	0:35:00	0:34:48	0:34:53
seq_1:sequence.sql	0:21:10	0:20:21	0:20:34	0:20:42
seq_1:singlequery.sql	0:35:28	0:35:35	0:35:29	0:35:31
seq_1:with.sql	0:16:48	0:16:48	0:16:46	0:16:47
seq_2:sequence.sql	0:18:02	0:18:05	0:17:58	0:18:02
seq_2:singlequery.sql	0:35:25	0:35:33	0:35:34	0:35:31
seq_2:with.sql	0:35:20	0:16:47	0:16:42	0:22:56

Tabelle .1: Einzelzeitmessungen der Subsequence-Skripte

Skriptvariante	1. Zeitmessung	2. Zeitmessung	3. Zeitmessung	Schnitt
seq_0:sequence.sql	0:56:12	0:56:40	0:56:27	0:56:26
seq_0:singlequery.sql	2:04:04	2:04:10	2:04:06	2:04:07
seq_0:with.sql	0:54:47	0:54:56	0:54:56	0:54:53
seq_1:sequence.sql	0:48:35	0:48:27	0:48:51	0:48:38
seq_1:singlequery.sql	2:02:44	2:03:30	2:03:34	2:03:16
seq_1:with.sql	0:59:24	0:59:08	1:00:01	0:59:31
seq_2:sequence.sql	0:29:13	0:29:48	0:29:14	0:29:25
seq_2:singlequery.sql	2:04:48	2:04:53	2:05:02	2:04:54
seq_2:with.sql	0:55:09	0:41:15	0:41:08	0:45:51
seq_2:sequence.sql	0:47:33	0:44:58	0:45:02	0:45:51
seq_2:singlequery.sql	2:05:58	2:05:21	2:05:45	2:05:41
seq_2:with.sql	0:41:38	0:41:46	0:41:40	0:41:41
seq_2:sequence.sql	0:45:09	0:45:21	0:45:21	0:45:17
seq_2:singlequery.sql	2:05:51	2:05:51	2:06:06	2:05:56
seq_2:with.sql	0:42:04	0:42:00	0:42:04	0:42:03

Tabelle .2: Einzelzeitmessungen der Sequence-Skripte

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Thorsten Müller)