Diplomarbeit

Thema:	**Lösungsansätze zur Laufzeitverbesserung kommerzieller Datenbankanwendungen am Beispiel der Softwarelösung „C-Logistic“**
Vorgelegt am:	17.08.2009
Von:	Denny Schubert Torfgasse 3 04808 Röcknitz
Studienrichtung/Studiengang:	Informationstechnik/Netzwerk- und Medientechnik
Seminargruppe:	IT06/2
Matrikelnummer:	4060112
Praxispartner:	C-Informationssysteme GmbH Schützstraße 4 04808 Wurzen
Gutachter:	Dipl.-Ing. Achim Kröber Dr.-Ing. Wolfgang Oehme

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
GLAUCHAU
UNIVERSITY OF COOPERATIVE EDUCATION

Studiengang Informationstechnik / Netzwerk- und Medientechnik

Themenblatt Diplomarbeit

Student: **Schubert, Denny** **SG:** **IT06/2** **Matr.-Nr.:** **4060112**

Bildungsstätte:	C-Informationssysteme GmbH
Anschrift:	Schützstraße 4 04808 Wurzen
Gutachter/Betreuer:	Herr Dipl.-Ing. Achim Kröber
Gutachter (Studienakademie):	Herr Dr.-Ing. Wolfgang Oehme

Thema der Diplomarbeit

Lösungsansätze zur Laufzeitverbesserung kommerzieller Datenbankanwendungen am Beispiel der Softwarelösung "C-Logistic"

Bearbeitungsschwerpunkte:

- Software- und benutzerseitige Schwachstellenanalyse
- Erarbeitung, Beurteilung und Gegenüberstellung von Lösungsansätzen zur Performancesteigerung unter Einbeziehung notwendiger System- und Programmierkosten
- Untersuchung verschiedener Varianten einer Datenarchivierung

Ausgabe des Themas: **18. Mai 2009**

Abgabe der Arbeit an die SR am: **17. August 2009**

Berufsakademie Sachsen
Staatl. Studienakademie Glauchau
Kopernikusstr. 51-53/ Pf 173
08371 Glauchau
Tel. (03763) 173-141
Fax (03763) 173-180

Prof. Christian Reinhold
Leiter des Studienganges
Informationstechnik

www.ba-glauchau.de

I Inhaltsverzeichnis

II Abbildungs- und Tabellenverzeichnis

III Abkürzungsverzeichnis

ADDM	Automatic Database Diagnostic Monitoring
ADO	ActiveX Data Objects
ATO	Automatic Tuning Optimizer
AWR	Automatic Workload Repository
BDE	Borland Database Engine
BLOB	Binary Large Object
CBO	Cost based optimizer
CIS	C-Informationssysteme
CLOB	Character Large Object
CPU	Central Processing Unit
DBA	Datenbankadministrator
DBWR	Database Writer
DOA	Direct Oracle Access
EMT64T	Extended Memory 64 Technology
ERD	Entity Relationship Diagramm
I/O	Input/Output
LGWR	Log Writer
MyDAC	Data Access Components for MySQL
ODAC	Oracle Data Access Components
PGA	Program Global Area
PL/SQL	Procedural Language/SQL
RAID	Redundant Array of Independent Disks
RAM	Random Access memory
RBO	Rule based optimizer
RDBMS	Relationales Datenbankmanagementsystem
SGA	System Global Area
SQL	Structed Query Language
UML	Unified Modeling Language

1. Definition der Anforderungen

C-Logistic ist eine umfassende Softwarelösung der C-Informationssysteme GmbH in Wurzen, die auf die Anforderungen mittelständiger bis großer Speditions-, Transport- und Logistikunternehmen ausgerichtet ist. Mit Hilfe von C-Logistic sind Unternehmen in der Lage, ihre Anforderungen in einer einzigen Softwarelösung abzubilden. C-Logistic unterstützt hierbei Stammdatenerfassung, Fuhrparkmanagement, Lagerverwaltung sowie das Auftrags- und Rechnungswesen. Weiterhin können neben den Standardanforderungen, wie bspw. der Sendungserfassung im Schüttgut-, Ladungs- und Kombiverkehr oder der Personalverwaltung, zahlreiche weitere Aspekte über verschiedenste Zusatzmodule abgedeckt werden. So ermöglicht es C-Logistic bspw. weiterhin Spesenabrechnungen, Schadenfälle, Werkstattaufträge oder Kassenbücher zu erfassen. Die Spanne der C-Logistic-Kunden ist ebenso vielfältig wie die Softwarelösung selbst und reicht von kleinen Unternehmen mit wenigen Mitarbeitern bis hin zu Unternehmen mit mehreren Niederlassungen, mehr als 500 Mitarbeitern und ca. 400 Fahrzeugen, sodass in diesen Fällen an die Leistungsfähigkeit von C-Logistic höchste Ansprüche gestellt werden.

Ausgehend von der vorhandenen Softwarelösung sollen mit Hilfe dieser Diplomarbeit Schwachstellen im System analysiert und Lösungsansätze zur Performanceverbesserung für die Arbeit der Anwender im System erarbeitet werden. Somit ist es notwendig, verschiedenste leistungsbeeinflussende Faktoren zu untersuchen. Zur Beurteilung softwareseitiger Parameter sollen bspw. Datenbankzugriffe sowie eingesetzte Softwarekomponenten auf ihre Eignung bzw. Optimierung hin untersucht werden. Als grobe Rahmenbedingung sind die Verwendung des Delphi-Compilers und der Oracledatenbank vorgegeben.

Zu Beginn dieser Diplomarbeit sollen performancerelevante Einflussgrößen ermittelt und nachfolgend deren Relevanz für C-Logistic festgestellt werden. Hierfür ist es weiterhin notwendig, grundlegende Problembereiche in C-Logistic zu identifizieren. Folgend sollen ggf. Lösungsansätze zur Umsetzung der erarbeiteten Einflussgrößen dargestellt werden.

Neben den bereits genannten Schwerpunkten sollen weiterhin im abschließenden Teil dieser Diplomarbeit verschiedene Varianten einer Datenarchivierung untersucht werden. Ziel ist es hierbei, das beträchtliche Datenvolumen einiger Kunden auf einen

angemessenen Produktivdatenbestand zu reduzieren, woraus sich ggf. ebenfalls Performancesteigerungen erwarten lassen könnten. Zu beachten ist dabei, dass C-Logistic sich ständig in der Weiterentwicklung befindet. Hierdurch kommen häufig neue Programmversionen mit unterschiedlichen Tabellenstrukturanforderungen zum Einsatz. Bei der Archivierung muss jedoch nach Möglichkeit sichergestellt werden, dass ein schneller Zugriff auf den Datenbestand zu Recherchezwecken weiterhin jederzeit möglich ist. Das besondere Problem wird sich hierbei somit aus den unterschiedlichen Programmversionen und der dazugehörigen variierenden Datenstruktur ergeben, sodass es zwingend notwendig ist, ein Konzept zu erarbeiten, welches diesen Umstand beachtet.

Im folgenden Gliederungspunkt sollen jedoch vorerst, wie bereits erwähnt, performancerelevante Einflussgrößen bzw. Optimierungsansätze ermittelt und folgend deren Relevanz anhand von Problembereichen in C-Logistic festgestellt werden.

2. Performancerelevante Einflussgrößen

Grundlegend soll Tuning der Sicherstellung angemessener Antwortzeiten für geschäftsrelevante Prozesse dienen [AHR06]. Tuning ist besonders dann Erfolg versprechend, wenn es auf vorher identifizierte Schwachstellen angewandt wird. Für C-Logistic können spontan als performancerelevante Einflussgrößen Faktoren wie Hardware, Betriebssystem, Datenbank, Datensatzanzahl, verwendete Entwicklungsumgebung bzw. Entwicklungskomponenten sowie die Arbeitsweise des Anwenders gefunden werden.

2.1 Hardware

Bei den meisten C-Logistic-Kunden wird Oracle in den Enterpriseversionen 10g sowie bei wenigen Kunden in der entsprechenden 10g Expressedition und in der Version 11g verwendet. Die Expresseditionen haben hierbei die Einschränkung, dass maximal 4 GB Daten gespeichert und nicht mehr als 1 GB Arbeitsspeicher sowie max. ein CPU[1]-Kern genutzt werden kann. Diese Einschränkungen sind bei der Einführung von Neukunden vorausschauend zu beachten. Für bisherige Bestandskunden sind diese Daten jedoch nicht relevant, da betreffende Kunden meist ein sehr geringes Auftragsvolumen besitzen. Deutlich relevanter sind die Systemanforderungen der Enterpriseversionen, da diese bei Großkunden zum Einsatz kommen, bei denen, im Gegensatz zu den zuvor genannten, Performanceprobleme auftreten können. Die Minimalanforderungen beider Enterpriseversionen liegen bei 512 MB RAM[2] und 200 MHz CPU bzw. 1.024 MB RAM und 1 GHz CPU. Bei Betrachtung dieser Werte fällt selbstverständlich sofort auf, dass sämtliche Kunden diese Werte bei weitem übertreffen. Im Gegenzug definiert Oracle keine empfohlenen Systemkonfigurationen unter verschiedenen Rahmenbedingungen (bspw. Datenaufkommen oder Nutzerzahl), wie es für eine erfolgreiche Gegenüberstellung wünschenswert wäre.

Grundlegend ist es möglich, Antwortzeiten in CPU- und Wartezeiten zu unterteilen. CPU-Zeiten entstehen bspw. beim Parsing von SQL[3]-Anweisungen. Es ist davon auszugehen, dass bei einem Datenbankserver schnelle Prozessoren, möglichst viel Arbeitsspeicher und ein leistungsfähiges Festplattensystem zum Erreichen des höchstmöglichen Datendurchsatzes Grundbestandteile darstellen sollten. Für den

[1] CPU – Central Processing Unit
[2] RAM – Random Access memory
[3] SQL – Structed Query Language

Einsatz in einem Oracleserver sind laut Oracle „wenige schnelle“ Prozessoren, „vielen langsameren“ vorzuziehen [LON05]. Oracle begründet diese Aussage zum einen aus Lizenzierungsgründen und zum anderen mit den Grenzen der Skalierbarkeit von Betriebssystem und Anwendungssoftware. Erhöhte Wartezeiten können sich somit bspw. bei der Bereitstellung von Ressourcen durch I/O[4]-Operationen ergeben. Im Client-Serverbetrieb kommen oft RAID[5]-Systeme zum Einsatz, die I/O-Operationen beschleunigen oder verlangsamen können. Eine Verlangsamung ist hierbei bei RAID 5 denkbar. RAID 5 bietet zwar einen gesteigerten Datendurchsatz beim Lesen von Daten, im Gegensatz nimmt jedoch der Schreibdurchsatz durch das zweiphasige Schreibverfahren deutlich ab. Somit ist ein RAID-5-Einsatz in einer schreibintensiven Umgebung, wie bspw. einem Datenbankserver, ungeeignet. Prinzipiell ist die Nutzung mehrerer Festplatten aus Sicht des gesteigerten Datendurchsatzes bzw. zur Lastverteilung einer einzelnen Platte vorzuziehen.

Erfahrungsgemäß hat sich serverseitig bei kleineren Unternehmen eine Konfiguration aus einem Pentium 4 (> 2 GHz) mit über 2 GB RAM als empfehlenswert erwiesen. Bei größeren Unternehmen sind entsprechend leitungsfähigere CPU’s mit 4-8 GB RAM zu empfehlen. Als leistungsfähiges Festplattensystem ist bspw. der Einsatz von Raid 10 empfehlenswert. Bei einigen C-Logistic-Kunden kann jedoch nicht sichergestellt werden, dass der Server nur für die Datenbank verwendet wird. Oftmals kommen hier Einsatzszenarien zum Tragen, bei denen bspw. durch zusätzlichen Einsatz als Terminalserver kein Einsatz des Servers als reiner Datenbankserver sichergestellt werden kann. Hierdurch kann im Problemfall zusätzlich eine konkrete Analyse deutlich erschwert werden. C-Logistic stellt clientseitig keine speziellen Anforderungen, da ein Großteil der Berechnungen fast ausschließlich über Oraclefunktionen auf dem Server ausgeführt wird. Als Richtwert kann hierbei 1 GHz CPU und 1 GB RAM genannt werden.

2.2 Serverbetriebssystem

Betriebssystemseitig kommt Oracle zur Verwendung mit Windows oder Linux in Frage. Bei CIS[6] und den meisten anderen C-Logistic-Anwendern wird Oracle in Verbindung mit einer Windowsplattform eingesetzt. Im Folgenden soll betrachtet werden,

[4] I/O – Input/Output
[5] RAID – Redundant Array of Independent Disks
[6] CIS – C-Informationssysteme

ob auf Windows- oder Linuxservern Geschwindigkeitsunterschiede erkennbar sind. Hierbei ist ebenfalls zu beachten, dass Oracle vor allem mit einer deutlichen Performancesteigerung durch die Verwendung von 64-Bit-Technik wirbt.

An dieser Stelle soll folgend kein eigener Vergleich beider Betriebssysteme durchgeführt werden, sondern vielmehr dient eine Gegenüberstellung der Performance Tuning Corporation als Grundlage aller folgenden Betrachtungen. In dieser Gegenüberstellung wurde Microsoft Windows 2003 Server Enterprise Edition x64 und Red Hat Enterprise Linux x86_64 in Verwendung mit Oracle 10gR2 verglichen. Für diesen Vergleich wurde die Performance auf zwei gleichen Serverstrukturen mit Intel EM64T[7] Xeon Prozessoren und den beiden unterschiedlichen Betriebssystemen mittels SwingBench ermittelt. Hierbei sollte in verschiedenen Tests geprüft werden, ob sich unter gleichen Voraussetzungen eines der beiden Betriebssysteme performanceseitig kontinuierlich absetzen kann. Um das Ergebnis dieser Studie zu verdeutlichen, wurden die folgenden Diagramme [CWL06] entnommen.

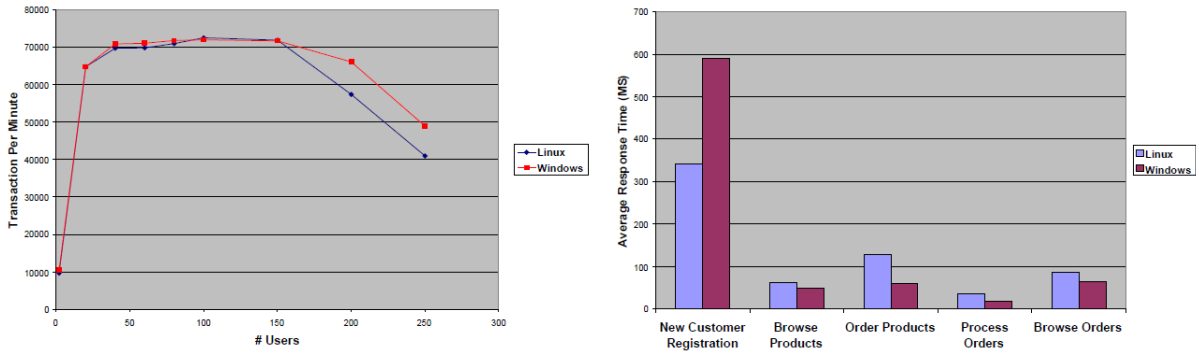


Abbildung 1: Stresstestergebnisse

In den beiden letzten Diagrammen ist erkennbar, dass spürbare performanceseitige Unterschiede aus Sicht der Anzahl möglicher Transaktionen zwischen Linux und Windows lediglich bei sehr hoher Nutzerzahl sowie aus Sicht der durchschnittlichen Reaktionszeit bei der Nutzeranmeldung erkennbar sind. Bei der nicht abgebildeten Gegenüberstellung des Datendurchsatzes beim Zugriff auf einen Tablespace sind die Unterschiede anfänglich gering. Folgend ist bei ca. 150 Benutzern Linux ca. 10 % schneller, bevor mit steigender Useranzahl sich Windows spürbar absetzen kann. Auffällig ist ebenfalls, dass der Speicherbedarf unter Linux bis zu 40 % höher liegt, dennoch werden 30 % aller Sortiervorgänge nicht im Speicher, sondern auf den

[7] EMT64T – Extended Memory 64 Technology

Laufwerken durchgeführt. Die CPU-Last blieb, laut Studie, während der gesamten Testdurchführung bei beiden Betriebssystemen oberhalb von 90 %.

Insgesamt ist festzustellen, dass bei durchschnittlicher Nutzeranzahl Windows und Linux fast identische Leistungswerte an den Tag legen. Mit steigenden Anforderungen bspw. durch eine hohe Nutzerzahl lässt sich aber feststellen, dass Windows zusehends mit höherer Performance bspw. durch eine effektivere Speicherverwaltung bei Sortiervorgängen überzeugt. Im Zusammenhang mit einer weiteren Studie der Performance Tuning Corporation wurde ebenfalls festgestellt, dass sich ein spürbarer Performancevorteil durch die 64-Bit-Technik erst bei der Verwendung von mehr als 8 GB RAM feststellen lässt [PED05]. Maßgeblich profitiert Oracle, wie die meisten anderen Anwendungen, beim Einsatz der 64-Bit-Technik von den höheren bzw. effektiveren Adressierungs- und Verwaltungsmöglichkeiten von großen Speichermengen. Zur Verdeutlichung sei an dieser Stelle erklärt, dass zur Unterstützung großer Nutzerzahlen bspw. ein größerer Shared Pool sowie eine größere PGA[8] benötigt werden. Die PGA stellt einen reservierten Speicherbereich für jeden Benutzer dar. Dieser Speicherbereich ist jedoch beim Einsatz von 32-Bit-Technik auf lediglich 1-2 GB begrenzt, wodurch die Performance bei hoher Nutzeranzahl maßgeblich beeinträchtigt werden kann.

Da neben den bisher genannten Bezeichnungen im weiteren Verlauf noch weitere oraclespezifische Begriffe, wie bspw. LGWR[9], erwähnt werden, soll folgend eine Abbildung zur Struktur des Oracle-RDBMS[10] aus [FRO94] angeführt werden. Mit dieser sollen die Zusammenhänge der unterschiedlichen Komponenten des Oracle-RDBMS verdeutlicht werden.

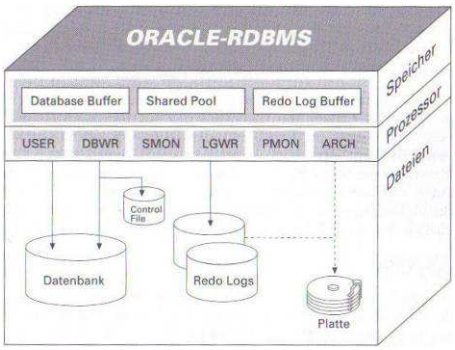


Abbildung 2: Struktur des Oracle-RDBMS

[8] PGA – Program Global Area
[9] LGWR – Log Writer
[10] RDBMS – Relationales Datenbankmanagementsystem

2.3 Datenbank

Die Architektur von Datenbanksystemen sieht eine Normalisierung der Relationen vor. Dies dient einerseits der Vermeidung von Redundanzen und andererseits dem Schutz vor Inkonsistenzen. In der Praxis jedoch kann ein zu stark theoretischer Datenbankentwurf wiederum Probleme seitens der Performance sowie der Anwenderfreundlichkeit bedeuten. Aus Sicht der Anwenderfreundlichkeit sei an dieser Stelle bspw. die Vielzahl der Datensatzsperren genannt, die innerhalb von Transaktionen bei einem stark theoretischen Entwurf durchgeführt werden müssen. Aus Performancesicht gilt weiterhin, dass jede normalisierte Relation ggf. bereits schon bei einfachen Anfragen bspw. über Verbunde wieder zusammengefügt werden muss. Somit sollte beim Entwurf einer Datenbankstruktur nicht die perfekte Normalisierung, sondern vielmehr der Anwender im Mittelpunkt stehen. Hierbei ist es für den reinen Anwender ohnehin irrelevant, wie die Daten strukturiert wurden, ihm kommt es vielmehr auf eine schnelle Erledigung seiner Aufgabe an.

In Bezug auf Datenbanken allgemein ist noch anzuführen, dass das Öffnen einer Datenbankverbindung eine der langsamsten Datenbankoperationen darstellt. Somit lässt sich als ein weiteres grundlegendes Designziel ableiten, dass vermieden werden sollte, dass bei jedem Zugriff erneut eine Verbindung aufgebaut wird. Hierbei ist es designtechnisch sinnvoll, eine einmal aufgebaute Datenbankverbindung kontinuierlich zu nutzen.

2.3.1 Allgemeine SQL-Hinweise

Da bereits in diesem Einstieg Verbundoperationen angesprochen wurden, soll folgend auf grundlegende Optimierungsansätze in Verbindung mit SQL und Oracle eingegangen werden. Für die deskriptive Programmiersprache SQL gilt, dass der Programmierer lediglich beschreiben kann, was er als Ergebnis erwartet. Ein Einwirken auf die interne Abarbeitung ist hierbei weniger möglich, sodass ein Programmierer zumindest einige der folgend erläuterten Grundregeln beachten sollte.

Bei Verbundoperation gilt, dass Oracle die Operationen Merge Join, Nested Loops und Hash Join für die Verarbeitung zur Verfügung stehen. Bei einem Merge Join werden die Relationen separat verarbeitet, sortiert, und erst dann wird die Verknüpfung der Relationen durchgeführt. Dies ist häufig der Fall, wenn keine treffenden Indizes zur Verfügung stehen. Besonders bei kleinen Relationen ist ein Merge Join

sehr effektiv. Nachteil ist, dass Daten erst zur Verfügung gestellt werden können, wenn alle Zeilen vollständig verarbeitet wurden. Nested Loops führen Verknüpfungen über Schleifenmethoden aus und verbrauchen die wenigsten Ressourcen. Hierbei werden Datensätze der ersten Relation abgeholt, und mit diesen wird folgend index-basiert auf Datensätze der zweiten Relation zugegriffen. Somit ist es möglich, schnell erste Datensätze bereitzustellen. Zu beachten ist aber auch, dass sich dieser Vorteil verliert, wenn zusätzliche Sortierungen notwendig sind. Bei Hash-Join-Operationen werden zwei Relationen im Arbeitsspeicher miteinander verglichen. Hierbei wird die erste Relation über Hashing-Funktionen gescannt. Folgend werden die Werte der zweiten Relation gelesen und über die Hashing-Funktion die Werte der ersten mit denen der zweiten Relation verglichen. Hash-Join-Operationen sind besonders bei der Verknüpfung einer kleinen mit einer großen Relation sinnvoll. Im Gegensatz ist der Einsatz bei mehreren großen Relationen als problematisch einzustufen.

Der Optimizer ist unter Oracle für die Ermittlung eines optimalen Zugriffspfads während des Parsing-Vorgangs einer SQL-Anweisung zuständig. Weiterführend lässt sich sagen, dass bei einer Verknüpfung mehrerer Relationen der Optimizer diese als eine Menge mehrerer Verbunde behandelt. Ein Verbund aus drei Relationen bedeutet demzufolge, dass erst ein Verbund der ersten beiden Relationen ausgeführt wird, bevor die Ergebnismenge mit der dritten Relation verknüpft werden kann. Hieraus lässt sich aus Performancesicht ableiten, dass der Umfang bzw. die Selektivität des ersten Verbundes unmittelbare Auswirkungen auf den Aufwand der Folgeoperationen hat. Ziel muss es sein, einen Verbund aus mehreren Relationen so zu gestalten, dass nach Möglichkeit im ersten Verbund bereits eine möglichst geringe Rückgabemenge erreicht werden kann. Folgend können dann alle Abfragearbeiten auf ein Mindestmaß reduziert werden.

Als weiterer Optimierungsansatz in Bezug auf SQL-Anweisungen lässt sich die Vermeidung unnötiger bzw. übermäßiger Sortierungen anführen. Sortierungen benötigen bei Attributen ohne Index deutlich mehr Zeit, da hierbei erst das gesamte Zeilenset sortiert werden muss, bevor das Ergebnis ausgegeben werden kann. Sofern sich Sortiervorgänge nicht vermeiden lassen, muss versucht werden, dass nach Möglichkeit sichergestellt werden kann, dass diese im Speicher ausgeführt werden. Somit kann erreicht werden, dass Swapping- und Pagingvorgänge vermieden werden

können. Das Verhältnis der Sortiervorgänge zwischen Laufwerk und Speicher kann über den View V$SYSSTAT ermittelt werden.

Unter Oracle dienen V$-Views zur Bereitstellung von Performancewerten. Dazu gehören neben dem bereits genannten View zahlreiche weitere Views[11] wie V$SESSION_EVENT, V$METRICNAME oder V$EVENT_NAME mit denen bspw. kumulativ die Anzahl und Dauer aller Wartezustände protokolliert oder kategorisiert werden kann. Im Gegensatz zu Data-Dictionary-Views basieren diese Views nicht auf festen Relationen, sondern auf speziellen X$-Performance-Tabellen, in denen bspw. die Speicherstrukturen der SGA[12] abgebildet werden. Erläuternd sei gesagt, dass die SGA in Oracle den gemeinsamen Speicher für alle Datenbankoperationen bereitstellt. Sie enthält den Datenbank- und Logpuffer sowie Informationen über das Data-Dictionary. Bei den Performance-Relationen ist zu beachten, dass diese zugänglich sind, jedoch von Oracle jederzeit verändert werden können. Somit ist ein Zugriff über die bereitgestellten Views vorzuziehen. Mit Hilfe der in der folgenden Abbildung dargestellten Anweisung kann bspw. ermittelt werden, ob der Sortierbereich ausreichend dimensioniert wurde. Weiterhin wird dargestellt, wie sich aus dem View die Trefferrate des Caches bestimmen lässt.

```
SELECT SORT_PLATTE, SORT_SPEICHER, ROUND(1-SORT_PLATTE/SORT_SPEICHER, 5) SORT_SPEICHER_RATE,
       LESEN_PLATTE, LESEN_CACHE, ROUND(1-LESEN_PLATTE/LESEN_CACHE, 5) TREFFER_CACHE FROM
(
  SELECT SUM(DECODE(NAME, 'sorts (disk)', VALUE, 0)) SORT_PLATTE,
         SUM(DECODE(NAME, 'sorts (memory)', VALUE, 0)) SORT_SPEICHER,
         SUM(DECODE(NAME, 'physical reads', VALUE, 0)) LESEN_PLATTE,
         SUM(DECODE(NAME, 'db block gets', VALUE, DECODE(NAME, 'consistent gets', VALUE, 0))) LESEN_CACHE
  FROM V$SYSSTAT
  WHERE NAME = 'sorts (disk)'
     OR NAME = 'sorts (memory)'
     OR NAME = 'physical reads'
     OR NAME = 'db block gets'
     OR NAME = 'consistent gets'
)
```

#	SORT_PLATTE	SORT_SPEICHER	SORT_SPEICHER_RATE	LESEN_PLATTE	LESEN_CACHE	TREFFER_CACHE
1	33	5873589	,99999	38017060	2882751089	,98681

Abbildung 3: Verhältnis der Sortiervorgänge und Trefferrate des Caches

Sofern das ermittelte Sortierverhältnis einen Wert unter 95 % ergibt, sollte der Sortierbereich entsprechend umdimensioniert werden. Über den Pga_aggregate_size-Parameter wird die Gesamtspeichermenge des Hauptspeichers festgelegt, der Serverprozessen bspw. bei Sortier- und Hashalgorithmen zugewiesen werden kann. Hieraus werden folgend automatisch die Parameter Sort_area_size und Hash_area_size gesetzt.

[11] Übersicht: siehe [AHR06] S.578 f.
[12] SGA – System Global Area

In Bezug auf Sortieranforderungen lässt sich ähnliches auch für die Vereinigung verschiedener Relationen feststellen. Währenddessen die Anweisung UNION prinzipiell eine Sortierung zur Unterdrückung doppelter Datensätze ausführt, ist dies bei UNION ALL nicht der Fall. Somit lässt sich die Regel definieren, dass nach Möglichkeit die UNION ALL-Anweisung der UNION-Anweisung vorzuziehen ist. Neben der UNION-Anweisung können zur Vermeidung zusätzlicher Sortieranforderungen auch die DISTINCT-, MINUS- und INTERSECT-Anweisungen angeführt werden. Hierbei verwendet Oracle zum Auflösen dieser Anweisungen prinzipiell eine SORT UNIQUE NOSORT-Operation. Als letztes sollen an dieser Stelle noch die Aggregatfunktionen MIN, MAX und COUNT auf Attribute ohne Index angeführt werden, da diese in Oracle ebenfalls den Einsatz der SORT AGGREGATE- bzw. der SORT GROUP BY-Funktion erfordern. Somit ist es auch bei diesen Funktionen notwendig, dass erst alle Datensätze verarbeitet werden müssen, bevor eine Ergebnisrückgabe stattfinden kann.

2.3.2 Minimierung physikalischer und logischer Zugriffe

In einigen Fällen lässt sich nachweisen, dass die Ursache für Performanceprobleme nicht in der ausgeführten Anweisung selbst, sondern vielmehr im Umfang bzw. der Häufigkeit der Ausführung zu finden ist. Somit muss in diesen Fällen die Minimierung physikalischer Zugriffe das Ziel der Optimierung darstellen. Abfragen lassen sich hierfür, bspw. durch den Einsatz von Indizes, effektiver gestalten, wodurch Zugriffe mit einem kompletten Scan der Tabelle bzw. Table Access Full-Situationen umgangen werden können. Neben einem Full Table Scan verfügt Oracle noch über weitere Zugriffstrategien, wie die folgende Tabelle aus [ANB06] verdeutlicht.

Zugriffsstrategie	**Gut bei …**	**Schlecht bei …**
Full Table Scan	kleinen Relationen	größeren Relationen, mit nur wenigen Datensätzen als Ergebnismenge
Unique Index Scan	Zugriff auf genau einen Datensatz	Daten, die sehr oft innerhalb eines Joins aufgerufen werden
Index Range Scan	wenigen Datensätzen als Ergebnismenge	großen Ergebnismengen und Weiterverarbeitung der Daten
Index Full Scan	ORDER BY auf die Index-Attribute	bestimmten Voreinstellungen
Fast Full Index Scan	großen Ergebnismengen	relativ kleinen Mengen, die sortiert vorliegen sollen

Tabelle 1: Bewertung der Zugriffsstrategien

Allgemein betrachtet ist das Anlegen eines Indizes auf Attribute mit wenig variierenden Feldinhalten und das Anlegen eines Indizes auf Attribute, die auf NULL geprüft werden, wenig Erfolg versprechend. Sofern sich Abfragen mit NULL oder auf wenig

selektive Daten nicht vermeiden lassen, ist der Einsatz von Bitmap-Indizes empfehlenswert. Im Versuch konnte durch den Einsatz von Bitmap-Indizes keine deutliche Zeiteinsparung festgestellt werden. Dennoch lässt sich erkennen, dass eine Senkung des Verarbeitungsaufwands erreicht werden kann, wie folgend dargestellt.

#	ID	PID	Operation	Name	Rows	Bytes	Cost	CPU Cost	IO Cost
1	0		SELECT STATEMENT		9820	83M	380	92M	376
2	1	0	TABLE ACCESS FULL	SENDPOS	9820	83M	380	92M	376

#	ID	PID	Operation	Name	Rows	Bytes	Cost	CPU Cost	IO Cost
1	0		SELECT STATEMENT		8628	73M	302	6228762	302
2	1	0	TABLE ACCESS BY INDEX ROWID	SENDPOS_KOPIE	8628	73M	302	6228762	302
3	2	1	BITMAP CONVERSION TO ROWIDS						
4	3	2	BITMAP INDEX SINGLE VALUE	BIDX_SENDPOS_KOPIE_FZNR					

Abbildung 4: Auswirkungen Bitmap-Index

Bei Attributen, die mit Funktionen wie bspw. LOWER oder UPPER verwendet werden, ist es zwingend notwendig, einen funktionsbasierten Index anzulegen. Eine weitere Optimierung beim Einsatz von Indizes lässt sich durch die Festlegung der maximalen Anzahl von Datensätzen pro Oracle Block bspw. über den Befehl ALTER TABLE SENDPOS MINIMIZE RECORDS_PER_BLOCK erreichen. Die schnellste Zugriffsmöglichkeit stellt dennoch der Direktzugriff über die ROWID dar, da die ROWID der physischen Adresse des Datenblocks entspricht. Diese Zugriffsvariante ist jedoch in den seltensten Fällen umsetzbar. Als Grundregel für INSERT-, UPDATE- oder DELETE-Anweisungen gilt, dass diese in Abhängigkeit von Betriebssystemumgebung, verwendeter Datenstruktur und Sortierungsgrad durch Indizes ungefähr dreimal so viel Zeit benötigen.

In einigen Fällen kann es erwünscht sein, dass bei der Ergebnisrückgabe die Anzahl der potentiellen Übereinstimmungen maximiert wird. Ein typisches Beispiel ist eine Suchanfrage, bei der im Anwendungscode Wildcards vorangesetzt oder angefügt werden, um zusätzliche Übereinstimmungen zu ermitteln. Hierbei können gewöhnliche B^*-Baum-Indizes an ihre Grenzen geraten. Oracle bietet hierfür die Möglichkeit, dass auch Textindizes angelegt werden können. Mit diesen können folgende aufwändige Full Text Scans vollständig vermieden werden, wodurch auch die Anzahl physischer und logischer Zugriffe auf die Suchrelation minimiert werden kann.

Um eine übermäßige Ansammlung ungenutzter Indizes zu vermeiden ist es möglich, über den ALTER INDEX-Befehl eine Überwachung der Indizes zu aktivieren. Sämtli-

che überwachte Indizes werden bei Oracle dann im View V$OBJECT_USAGE hinterlegt, wie es in der folgenden Abbildung erkennbar ist.

```
--ALTER INDEX IDX_PERSONAL_AENDERDATUM MONITORING USAGE;
--ALTER INDEX IDX_PERSONAL_AENDERDATUM NOMONITORING USAGE;

SELECT * FROM V$OBJECT_USAGE
```

#	INDEX_NAME	TABLE_NAME	MONITORING	USED	START_MONITORING	END_MONITORING
1	PK_DOKUMENT	DOKUMENT	YES	YES	04/15/2008 16:21:32	
2	IDX_EINHEITEN_VPEART	EINHEITEN	YES	YES	04/15/2008 16:21:22	
3	PK_ENTLADEBERICHT_ELL_NR	ENTLADEBERICHT	YES	YES	04/15/2008 16:21:24	
4	PK_ENTLADEBERICHT_PACKSTUECKID	ENTLADEBERICHT_PACKSTUECK	YES	NO	04/15/2008 16:21:24	
5	IDX_FAHRERKARTE_PERSONAL	FAHRERKARTE	YES	YES	04/15/2008 16:21:25	
6	IDX_FAHRERKARTE_ID	FAHRERKARTE	YES	YES	04/15/2008 16:21:24	
7	IDX_LFDNR	FAHRPLAN	YES	YES	04/15/2008 16:21:25	

Abbildung 5: Überwachung der Indexnutzung

Die Häufigkeit von physikalischen Zugriffen kann ebenfalls über die Dimensionierung von Buffer-Cache, Shared-Pool und PGA-Pool beeinflusst werden. Aktuelle Dimensionsangaben können bspw. im View V$BUFFER_POOL abgefragt werden. Empfehlenswerte Richtwerte stellen weiterhin die Views VDB_CACHE_ADVICE, VSHARED_POOL_ADVICE und V$PGA_TARGET_ADVICE_HISTOGRAM bereit. Als Anwendungsbeispiel soll die folgende Abbildung dienen, mit der das Laufzeitverhalten in Abhängigkeit der Größe des entsprechenden Pools bewertet werden kann.

```
SELECT SIZE_FOR_ESTIMATE AS GROESSE, SIZE_FACTOR AS FAKTOR,
       BUFFERS_FOR_ESTIMATE AS PUFFER,
       ESTD_PHYSICAL_READ_FACTOR AS FAKTOR_LESEN,
       ESTD_PHYSICAL_READS AS PHY_ZUGRIFFE,
       ESTD_PHYSICAL_READ_TIME AS PHY_ZUGRIFFSZEIT
FROM V$DB_CACHE_ADVICE
WHERE BLOCK_SIZE = 8192
```

#	GROESSE	FAKTOR	PUFFER	FAKTOR_LESEN	PHY_ZUGRIFFE	PHY_ZUGRIFFSZEIT
1	96	,0882	11874	2,1301	48958515	163567
2	192	,1765	23748	1,9288	44331879	144000
3	288	,2647	35622	1,7442	40088672	126055
4	384	,3529	47496	1,5819	36357975	110278
5	480	,4412	59370	1,4434	33175752	96820
6	576	,5294	71244	1,3275	30510793	85550
7	672	,6176	83118	1,2331	28342180	76379
8	768	,7059	94992	1,1611	26686243	69375
9	864	,7941	106866	1,1021	25331939	63648
10	960	,8824	118740	1,054	24224540	58965
11	1056	,9706	130614	1,0128	23278428	54964
12	1088	1	134572	1	22984437	53720
13	1152	1,0588	142488	,9752	22415507	51314
14	1248	1,1471	154362	,9394	21591524	47830
15	1344	1,2353	166236	,9036	20769112	44351
16	1440	1,3235	178110	,8657	19898263	40669
17	1536	1,4118	189984	,8251	18963813	36717
18	1632	1,5	201858	,7825	17986427	32583

Abbildung 6: Anwendung View V$DB_CACHE_ADVICE

Hierbei wurden die Werte für den Buffer-Cache mit 8 KB-Blockgröße ermittelt. In den Attributen GROESSE, FAKTOR und PUFFER werden die betrachteten Cachegrößen angegeben, denen folgend die Werte der physischen Zugriffe angefügt werden. Erkennbar ist weiterhin, dass die 1,5fache Buffer-Cache-Größe die Anzahl logischer Zugriffe auf 78 % reduzieren kann. Ähnliche Erhebungen lassen sich ebenfalls anhand der beiden anderen genannten Views erstellen.

Neben der Minimierung physikalischer Zugriffe sollte ebenfalls die Vermeidung übermäßiger logischer Zugriffe eine der Zielsetzungen bei der Programmierung sein, da diese meist physikalische Zugriffe erzwingen. Ein logischer Zugriff bezeichnet hierbei einen Datenzugriff, der ohne physikalische Festplattenzugriffe beantwortet werden kann. In diesem Zusammenhang kann die Vermeidung von Datenbankausflügen einen weiteren Optimierungsansatz darstellen. Hierfür kann es sich bspw. als sinnvoll erweisen, innerhalb einer Erfassungsmaske nach Möglichkeit mehrere Abfragen zu einer Prozedur bzw. in einem Package zusammenzuführen, wodurch nebenläufig die Netzlast ebenfalls reduziert werden kann. Ein Package stellt hierbei die Zusammenfassung logisch zusammenhängender Funktionen oder Prozeduren dar. Die Suche nach zusammengehörigen Anweisungen könnte ggf. auf gemeinsame Rückgabewerte basieren. In Packages und Prozeduren ist ebenfalls darauf zu achten, dass unnötige Datenbankzugriffe vermieden werden, und somit sind bspw. häufig bzw. wiederholt genutzte Werte in lokalen Variablen zu speichern.

Oracle erzeugt bei der Datenabfrage ein lesekonsistentes Abbild der Daten. Verändert ein anderer Nutzer zwischenzeitlich Daten, ist über das Undo-Segment sichergestellt, dass die Daten weiterhin wie zu Abfragebeginn dargestellt werden können. Beim Mehrbenutzerbetrieb ist dies eine häufige Situation, die den Arbeitsaufwand des Servers durch logische und physikalische Zugriffe erhöht. Da C-Logistic auf den Mehrbenutzerbetrieb ausgelegt ist, ist es designtechnisch nicht möglich, diese Situation zu umgehen. Vielmehr kann hier lediglich die Empfehlung an den DBA[13] gegeben werden, dass das entsprechende Segment ausreichend zu bemessen ist, um Fehler bzw. Engpässe zu vermeiden. In einem ähnlichen Zusammenhang sind die Rollback-Segmente zu sehen. Hierbei handelt es sich um temporäre Segmente, in denen sich geänderte Daten befinden, deren Bearbeitung noch nicht durch ein Transaktionsende abgeschlossen wurde. Views zur Bewertung der Rollback-Segmente sind V$ROLLSTAT, V$WAITSTAT. Zur Minimierung von Wartezeiten sollten bspw. mehr Rollback-Segmente zugewiesen werden, wenn die Anweisung SELECT ROUND(SUM(WAITS)/SUM(GETS),5) FROM V$ROLLSTAT einen Wert größer 1 % zurückliefert.

[13] DBA – Datenbankadministrator

2.3.3 Maximierter Datendurchsatz

Ideal betrachtet sind keine Daten außerhalb der Datenbank abzufragen und alle Daten bleiben die ganze Zeit im Arbeitsspeicher. Real betrachtet ist dies bei vielen Datenbanken nicht umsetzbar. Somit müssen Möglichkeiten gefunden werden, um den Datendurchsatz zu maximieren. Ein erhöhter Datendurchsatz ist hierbei besonders in einer Multiuserumgebung wichtig, um sicherzustellen, dass möglichst viele Anfragen in einem bestimmten Zeitintervall beantwortet werden können. Laut Oracle beträgt die Trefferrate für den Datenbankpuffercache bei häufig benutzten Blöcken über 90 %, sodass nur in 10 % aller Anfragen auf das Laufwerk zugegriffen werden muss. Somit lässt sich standardgemäß bereits aus dem Disk Caching eine Performancesteigerung erwarten. Beim Anlegen einer neuen Datenbank empfiehlt Oracle nach Möglichkeit die größte verfügbare Blockgröße zu wählen und verspricht dadurch eine höhere Performance der eigenen Applikation [LON05]. Begründet wird dies damit, dass so bspw. der Aufenthalt großer Datenmengen im Arbeitsspeicher unterstützt werden kann. Speziell bei kleinen Relationen ist es mit Hilfe der Buffer_pool-Klausel möglich, beim Erstellen oder Ändern der Relation festzulegen, dass diese im Hauptcache bzw. Arbeitsspeicher gehalten werden soll. Sofern der Speicherbereich ausreichend dimensioniert ist, ist es somit möglich, jeweils bis zum Herunterfahren der Datenbank sicherzustellen, dass diese Daten im deutlich schnelleren Hauptspeicher verbleiben können.

Die Unterstützung vieler gleichzeitiger UPDATE- oder INSERT-Anweisungen lässt sich durch den Pctfree-Parameter beeinflussen. Standardgemäß wird der Pctfree-Wert auf 10 gesetzt. Dies bedeutet, dass ein Block folgend bis 90 % gefüllt wird und die verbleibenden 10 % für die Aktualisierung der Daten reserviert werden. Somit ist es möglich, die Datenblockausnutzung festzulegen und ggf. eine effizientere Ausnutzung des Arbeitsspeichers zu erreichen. Sofern die Datensätze in der Datenbank nicht aktualisiert werden, kann der Pctfree-Wert so niedrig wie möglich gesetzt werden.

Bei Erreichen der zuvor genannten 90 % wird der Block gleichzeitig aus der Freelist entfernt. Mit Hilfe des Pctused-Parameters kann festgelegt werden, ab welchem Punkt ein Block bspw. nach Löschvorgängen wieder in die Freelist aufgenommen wird. Standardgemäß ist dieser Wert auf 40 gesetzt. Bei Applikationen mit einem

hohen Aufkommen an Lösch- oder Einfügevorgängen ist ein geringer Pctused-Wert unpraktisch. Dadurch werden einerseits ständig neue Blöcke belegt und andererseits alte Blöcke erst wieder als frei gekennzeichnet, wenn sie bspw. nur noch zu 40 % belegt sind. In diesem Fall ist es empfehlenswert, den Wert so zu erhöhen, dass Blöcke bedeutend eher wieder als frei gekennzeichnet werden.

Sofern sich aufwändige Datenbankoperationen nicht vermeiden lassen, kann die Aufteilung in kleine verwaltbare Einheiten mit Hilfe von Partitionen ebenfalls einen Optimierungserfolg nach sich ziehen. Für den Endanwender lässt sich hieraus zum einen aufgrund des gezielteren Zugriffs eine erhöhte Abfrageperformance und zum anderen eine erhöhte Verfügbarkeit der Datenbank erreichen. Die erhöhte Verfügbarkeit lässt sich bspw. damit begründen, dass sich administrative Funktionen ebenfalls gezielt auf einzelne Partitionen durchführen lassen. Somit profitieren neben dem Anwender ggf. auch Supportmitarbeiter und der DBA von einer Partitionierung. In einem ähnlichen Zusammenhang ist die Verwendung verschiedener Tablespaces zu sehen. So ist es bspw. empfehlenswert, lokal verwaltete Tablespaces zu erstellen, um Probleme der Speicherplatzverwaltung zu vermeiden. Hierbei zeichnen sich lokal verwaltete Tablespaces im Gegensatz zu Dictionary-verwalteten Tablespaces vor allem durch einen verringerten Verwaltungsaufwand aus. Weitere Empfehlungen in Bezug auf Tablespaces sind:

- Aufteilung von Relationen und Indizes auf verschiedene Tablespaces
- Auslagerung von sehr großen Objekten und Rollback-Segmenten auf einen eigenen Tablespace

Besonders empfehlenswert ist in diesem Zusammenhang die getrennte Speicherung von Indizes und deren referenzierte Relationen auf unterschiedlichen Festplatten.

Da vorhergehend die Beeinflussung der Organisation der Daten durch die Verwendung von Partitionen erwähnt wurde, soll folgend noch kurz auf das Clustering unter Oracle eingegangen werden. Beim Clustering werden Relationen, die miteinander in Beziehung stehen oder häufig miteinander verknüpft werden, in einem gemeinsamen Plattenbereich abgelegt. Hierbei kann sich Clustering ebenfalls bei häufig gruppierten Attributen als durchaus sinnvoll erweisen. Der hierfür benötigte Cluster-Schlüssel besteht bspw. aus den Attributen, über welche die Relationen häufig verknüpft oder gruppiert werden. In diesem Schlüssel wird unter Oracle jeder eindeutige Wert nur

einmal gespeichert. Die Daten selbst werden tatsächlich folgend an einem Standort abgelegt, als ob es sich bei dem Cluster um eine einzige große Relation handelt, wodurch ein optimierter Datenzugriff erreicht werden kann. Zu beachten ist im Gegensatz jedoch auch, dass sich beim Clustering die Zugriffzeit auf eine der geclusterten Relationen stark erhöhen kann.

Ferner lässt sich die Größe der benötigten Rollback-Segmente und somit der Ressourcenbedarf über den Arraysize-Parameter, bspw. bei Transaktionen mit einem COPY-Befehl, einschränken. Zur Erläuterung sei an dieser Stelle gesagt, dass bei einer vollständigen Kopie einer Relation mit CREATE TABLE FAHRZEUG_KOPIE AS SELECT * FROM FAHRZEUG standardgemäß die Kopie als einzige Transaktion abgearbeitet wird. Hierfür muss ein vollständiges Datenabbild in den Rollback-Segmenten gehalten werden, bis die Daten festgeschrieben werden. Die Transaktionsgröße kann über den Arraysize-Parameter folgend so festgelegt werden, dass bspw. während des Kopiervorgangs die Daten aller 1.000 Datensätze festgeschrieben werden, wodurch sich bei umfangreichen Kopiervorgängen zusätzliche Geschwindigkeitsvorteile ergeben können.

Abschließend soll in diesem Kapitel noch auf die Verwendung von Bind-Variablen und dem damit verbundenen Cursor_sharing eingegangen werden. Zur grundlegenden Klärung des Begriffs einer Bind-Variablen soll der folgende Ausschnitt dienen.

```
DECLARE
   GEBORT VARCHAR2(40) := 'Berlin';
BEGIN
   EXECUTE IMMEDIATE 'DELETE FROM PERSONAL WHERE GEBURTSORT = :GEBORT'
   USING GEBORT;
END;
```

Im dargestellten Listing stellt :GEBORT eine Bind-Variable dar, durch deren Einsatz es nicht mehr notwendig ist, dass der Befehl bei jedem Aufruf erneut geparst werden muss. Hierdurch ist es möglich, Platz im Arbeitsspeicher zu sparen, da der Befehl mehrfach verwendet werden kann und nicht jede Anweisung einzeln abgespeichert werden muss. Ein ähnliches Vorgehen lässt sich vom DBA auch über den Initialisierungsparameter Cursor_sharing erzwingen. Die Ausführung einer SQL-Anweisung geschieht über die Phasen Open, Parse, Execute, Fetch und Close. Der Cursor bezeichnet in der Open- und Close-Phase eine Datenstruktur, vergleichbar mit einem

Zeiger, der die aktuelle Position angibt. Der Parameter Cursor_sharing kann auf force, similar oder exact gesetzt werden und bestimmt damit, wie ähnliche SQL-Anweisungen einen Cursor gemeinsam nutzen können. Exact erzwingt hierbei, dass ein Cursor nur für exakt gleiche Anweisungen genutzt werden kann. Die empfohlene Einstellung ist force oder similar, wodurch Cursor bei übereinstimmenden Zeichenfolgen mehrfach genutzt und somit die Parse-Phase eingespart werden kann. Force verwendet hierbei jedoch im Vergleich zu similar zusätzlich selbständig eingefügte Bind-Variablen [ANB06].

Bei der Betrachtung der Möglichkeiten zur Erhöhung des maximalen Datendurchsatzes wurde an dieser Stelle bewusst nicht auf den Einfluss des eigenen Netzwerkes eingegangen, da dieses in den seltensten Fällen eine Limitierung darstellt.

2.3.4 Materialisierte Views

Unter einem materialisierten View versteht man eine Kopie von Daten, die verteilte Daten nutzt oder bspw. basierend auf GROUP BY-Operationen Summentabellen einrichtet [LON05], sodass Benutzeranfragen umgeleitet werden können. Hierbei ist es möglich, ganze Relationen oder Teile einer Relation zu replizieren. Weiterhin können Abfrageergebnisse auf mehrere Relationen verteilt werden. Die Aktualisierung dieser Daten kann in Oracle 10 innerhalb von vorgegebenen Zeitintervallen selbstständig ausgeführt werden. In der einfachsten Form lässt sich ein materialisierter View, bspw. für eine Fahrzeugrelation, bereits aus folgender SQL-Anweisung erstellen.

```
CREATE MATERIALIZED VIEW VIEW_FAHRZEUG REFRESH FORCE
START WITH SYSDATE NEXT SYSDATE+1
AS
  SELECT * FROM FAHRZEUG
```

Aufgrund der Tatsache, dass bei einem materialisierten View Synchronisationen erforderlich sind, wirkt diese Umsetzung evtl. im ersten Anschein wenig Erfolg versprechend, jedoch lässt sich dieser Aufwand ebenfalls zusätzlich beeinflussen. So ist es möglich, vollständige Aktualisierungen zu verhindern und über transaktionsbasierte Refreshes sicherzustellen, dass immer nur die in der Masterrelation geänderten Zeilen angepasst werden, so wie es im dargestellten Beispiel der Fall ist. Bei einem aktualisierbaren materialisierten View lässt sich der Aufwand nicht so stark verringern, da sichergestellt werden muss, dass Änderungen in der Masterrelation sowie

im View synchronisiert werden. Weiterhin muss festgelegt werden, wie potenzielle Konflikte gelöst werden. So ist es angenommen denkbar, dass ein Datensatz im View gelöscht wird, währenddessen auf der Masterseite gerade ein Datensatz angelegt wird, der diesen Datensatz über einen Fremdschlüssel referenziert.

Als Refreshregeln stehen hierbei fast, complete und force zur Verfügung. Bei der Refreshregel force wird, sofern Oracle die Zeilen der Basisrelation denen des materialisierten Views direkt zuordnen kann[14], ein fast, sonst ein complete refresh durchgeführt. Ebenfalls ist es möglich, über QUERY REWRITE festzulegen, dass der Optimizer selbständig entscheiden kann, ob er die Daten aus dem View oder direkt aus den darunter liegenden Relationen ermittelt. Nicht zu vergessen ist weiterhin, dass der materialisierte View der Performancesteigerung dienen soll. Folglich sollte nicht versäumt werden, auch auf diesen entsprechend den an ihn gestellten Anforderungen Indizes anzulegen.

Insgesamt sollte besonders bei komplexen Abfragen die Performance mit materialisierten Views erheblich gesteigert werden können. Durch den Einsatz dieser Views ist es möglich, Datenanpassungen auf die Basisrelationen ausführen zu lassen, währenddessen Abfragen auf den anderweitig positionierten View umgeleitet werden. Weiterhin ist es möglich, auf den Basisrelationen und dem View unterschiedliche Indizes anzulegen, wodurch eine weitere Optimierung auf den betreffenden Einsatzzweck möglich ist. Weniger geeignet ist diese Technik selbstverständlich für den Einsatz mit sich ständig ändernden Daten.

Nachdem vorhergehend die Optimierung durch Einsatz von materialisierten Views angesprochen wurde, soll folgend auf Optimierungsansätze in gewöhnlichen Views eingegangen werden. Bei Selektionen aus einem View kombiniert der Optimizer standardgemäß die Kriterien der Abfrage mit dem Abfragetext des Views, sodass der View keine Auswirkung auf die Abfrageperformance hat. Dies ist jedoch stark vom Inhalt des Views abhängig. Sobald innerhalb des Views gruppiert wird, ist die Zusammenführung mit der Abfrage nicht mehr möglich, sodass sich ggf. daraus Performancenachteile erwarten lassen, da erst der vollständige View aufgelöst werden muss, bevor die Einschränkungen durchgeführt werden können. Durch die Entfernung von Sortier- und Gruppierungsfunktionen lässt sich somit die Wahrscheinlich-

[14] wie dies bei einfachen Abfragen der Fall ist

keit erhöhen, dass der Viewtext mit dem Abfragetext kombiniert werden kann, wodurch sich weiterhin die Abfrageperformance verbessern kann. Dies lässt sich auch dadurch rechtfertigen, dass durch die Kombination für den Optimizer die Anzahl der möglichen Indizes bzw. Datenzugriffpfade erhöht wird.

Oracle kann Operationen parallelisieren und mehrere Prozessoren an der Verarbeitung beteiligen. Typische Operationen sind hierfür ein Table Access Full und verschiedenste Sortieroperationen. Als Standardeinsatzfall kann hierfür der bereits erläuterte Merge Join angeführt werden, da dieser standardgemäß bei umfangreichen Relationen ein Table Access Full sowie Sortieroperationen nach sich zieht. Weiterhin lassen sich aber auch bspw. Einfügeoperationen oder das Anlegen von Objekten parallelisieren. Grundvoraussetzung ist jedoch immer, dass entsprechende Systemressourcen zur Verfügung stehen.

Neben allen bisherigen Betrachtungen kann auch eine durchgeführte Migration ein Ausgangspunkt für eine schlechte Performance darstellen. Zur Erläuterung sollte an dieser Stelle erwähnt werden, dass unter Oracle ein kostenbasierter Optimizer oder ein nicht mehr weiterentwickelter regelbasierter Optimizer zum Einsatz kommen kann. Ein RBO[15] verwendet vordefinierte Regeln, währenddessen ein CBO[16] die Ausführung anhand von Statistiken umsetzt. Bei einer Migration kann es bspw. geschehen, dass ein CBO gegen einen RBO ausgetauscht wird oder verschiedene Versionen des Oracle-Kernels zum Einsatz kommen. Um den negativen Auswirkungen einer Migration entgegenzuwirken, hat Oracle gespeicherte Outlines eingeführt. Outlines bestehen hierbei aus einer Reihe von Optimizer-Hints, die für jede Ausführung zum Einsatz kommen, sodass gleiche Ausführungspläne garantiert werden können.

2.3.5 Statistiken, Optimizer-Hints und Prozesse

Oracle bietet in der Version 10g diverse Tools zur automatisierten Performanceverwaltung an. Diese umfassen die Automatic Statistics Collection, das ADDM[17] und das Automatic SQL Tuning. Um eine Übersicht in den oracleseitigen Begriffsdschungel zu bringen, sollen einige der vorhandenen Tools folgend tabellarisch dargestellt werden.

[15] RBO – Rule based optimizer
[16] CBO – Cost based optimizer
[17] ADDM – Automatic Database Diagnostic Monitoring

Tool/Komponente	Erläuterung
Automatic Statistics Collection	Automatisiert die Sammlung von Optimizer-Statistiken für verschiedene Objekte
Automatic Workload Repository	Sammlung und Verwaltung von Performancestatistiken zur Problemermittlung und zum Selbsttuning Ergebnisse werden in Data-Dictionary-Views mit DBA_HIST_* oder über Data-Dictionary-Relationen mit WRH$_* und WRM$_* bereitgestellt Wertermittlung geschieht durch automatische/regelmäßige Snapshots der Performance-Views → ermöglicht Erstellung von Statistiken über beliebige Zeiträume
ADDM	Dient der Aufwandsreduzierung bei der Problemdiagnose und beim Tuning von Oracle Systemen Baut auf die vom AWR[18] ermittelten Werte auf Werte können in den Views DBA_ADVISOR_* und den Relationen WRI$_* abgerufen werden.
Automatic Tuning Optimizer und Automatic SQL Advisor	Der kostenbasierenden Optimizer ist ebenfalls in der Lage, Maßnahmen zur weiteren Verbesserung aus dem Ausführungsplänen[19] abzuleiten → ATO[20] als Expertensystem mit einer Schnittstelle, um Tuningempfehlungen an Nutzer weiterzuleiten. Beides ermöglicht: die schnelle und effiziente Optimierung von SQL-Anweisungen, wobei der Advisor die nötige Schnittstelle bezeichnet

Tabelle 2: Tools/Komponenten zur Unterstützung des automatisierten Tunings

Zentrales Element bei der Unterstützung eines DBAs stellt meist der Enterprise Manager dar. Neben diesem existieren jedoch auch zahlreiche weitere Tools, wie bspw. Spotlight und SQLab Vision von Quest Software oder Hora von Keeptool. Beim Einsatz derartiger Produkte in Mehrkernumgebungen ist jedoch teilweise mit erheblichen Lizenzkosten zu rechnen. Eine durchaus sinnvolle Alternative kann im Gegensatz das von Oracle bereitgestellte Statspack darstellen, das ähnlich dem erläuterten Automatic Workload Repository funktioniert und verschiedenste Statistiken beinhaltet.

Der Enterprise Manager stellt eine visuelle Benutzeroberfläche bereit und ermöglicht neben der Datenbankwartung und –administration ebenfalls die Performanceüberwachung. Hierzu ermöglicht er bspw. die Identifikation von:

- Top Aktivitäten/Consumern,
- blockierenden Sessions,
- ähnlichen SQL-Anweisungen.

Ein Großteil der bisher erläuterten Funktionen, Einstellungen und Parameter können im Enterprise Manager überwacht bzw. eingestellt werden. Bei der Optimierung ist vor allem das in der folgenden Abbildung dargestellte Register Performance interessant.

[18] AWR – Automatic Workload Repository
[19] Meist als explain plan bezeichnet
[20] ATO – Automatic Tuning Optimizer

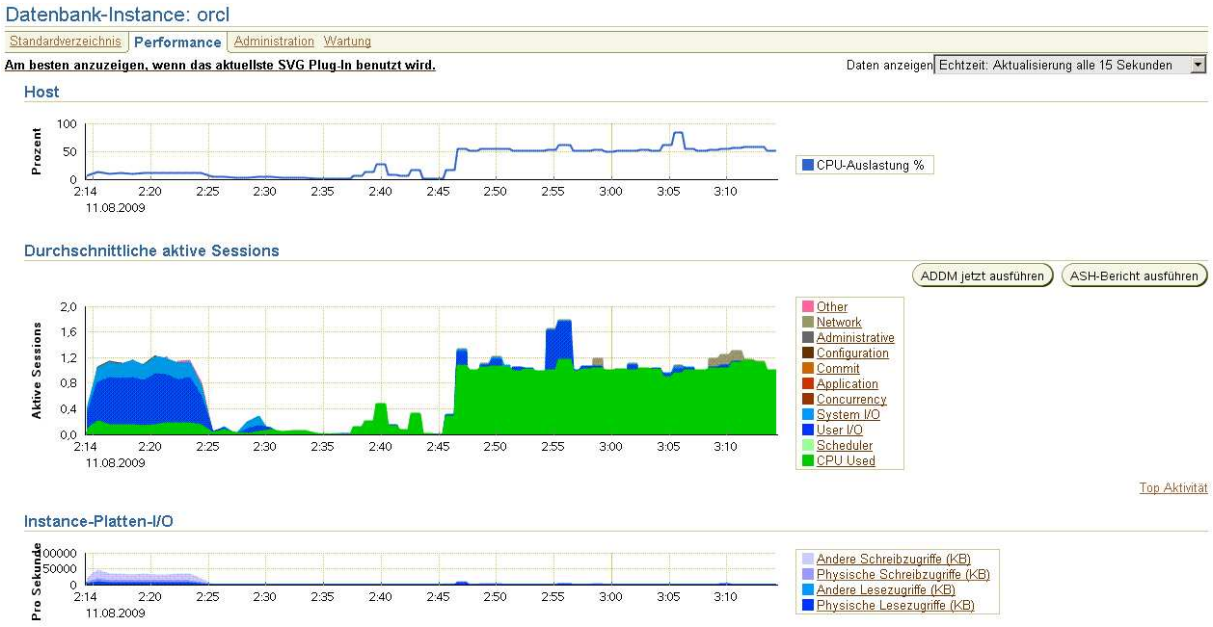


Abbildung 7: Enterprise Manager: Register Performance

In dem abgebildeten Register können neben den verschiedenen Verläufen auch zahlreiche weitere Aktivitäten angezeigt werden. Im Bezug auf SQL-Anweisungen bietet hierbei Oracle bspw. die Möglichkeit, über den SQL Tuning Advisor Tuningvorschläge erstellen zu lassen. Über den Enterprise Manager können weiterhin bspw. Prognosen, Statistiken und Ausführungspläne abgerufen werden, wie die folgende Abbildung verdeutlichen soll.

Abbildung 8: Statistiken und Empfehlungen im Enterprise Manager

In ähnlichem Zusammenhang kann auch Tracing die Identifikation von Quellen übermäßiger Belastung, wie bspw. teure SQL-Anweisungen, ermöglichen. Tracing

beschreibt hierbei die Aufzeichnung von SQL-Anweisungen und deren Ausführungsstatistiken und kann als Session-, Client- oder Datenbanktracing eingesetzt werden. Zur Aufbereitung des Inhalts der Tracedateien wird das Werkzeug TKPROF benötigt.

Im Register Administration ist es bspw. möglich, sich einen Überblick über alle Relationen, Indizes, Tablespaces, materialisierte Views, Datenbankparameter usw. zu verschaffen. Im Register Wartung sind folglich bspw. die Bereiche Backup, Recovery, Import/Export und Patching der Datenbank zu finden.

Oracle ermöglicht es seit Version 9i Caches und Pools im laufenden Betrieb anzupassen. In Version 10g können diese Eingriffe selbstständig vom System ausgeführt werden. Zur Aktivierung des Automatic Shared Memory Managements ist es im Enterprise Manager notwendig, den Initialisierungsparameter Sga_target auf einen Wert ungleich Null und den Statistics_level-Parameter auf typical bzw. all zu setzen.

Erwähnt werden sollte an dieser Stelle noch, dass es einerseits wichtig ist, aktuelle Statistiken zu besitzen, andererseits kann jedoch die Erstellung von Statistiken signifikante Auswirkungen auf die Gesamtperformance haben. Somit sollten nach Möglichkeit die automatischen Sammelfunktionen der Statistikdaten auf einen Zeitpunkt außerhalb der Kernarbeitszeiten gelegt werden. Weiterhin lässt sich bei manueller Durchführung der Batch-Prozess mit dem Befehl ALTER SESSION und mit höheren Werten in den Parametern Db_file_multiblock_read_count und Sort_area_size im Bezug auf Sortieranforderungen und Full Table Scans drastisch beschleunigen. Der Parameter Db_file_multiblock_read_count bestimmt, wie viele Blöcke im Rahmen eines Full Table Scans jeweils je Betriebssystem-I/O geholt werden können.

Weiterhin bietet Oracle die Möglichkeit, über Optimizer-Hints die Ausführung gezielt zu beeinflussen. In der folgenden Übersicht sollen einige der verfügbaren Optimizer-Hints angeführt werden, die zur Beurteilung bzw. für den Performancevergleich unterschiedlicher Ausführungsstränge zur Verfügung stehen. Optimizer-Hints können in jede Anweisung bspw. als SELECT /*+ INDEX(IDX_FAHRZEUG) */ eingebunden werden. Fehlerhafte Optimizer-Hints werden im Gegenzug automatisch ignoriert. Besondere Obacht ist bei der Verwendung von Aliasnamen angebracht, da ein in der SQL-Anweisung verwendeter Alias auch im Hint angewandt werden muss.

Befehl	Erläuterung
FIRST_ROWS	Anwendbar, wenn es unnötig ist, dass vor Rückgabe der Ergebnismenge alle Zeilen sortiert sein müssen → Primäres Ziel: schnellstmögliche Bereitstellung erster Ergebnisse
ALL_ROWS	Optimierung auf Gewinnung der gesamten Ergebnismenge
ORDERED/LEADING	Join nach Reihenfolge der From-Klausel oder mit definierter treibender Relation ausführen
MERGE/NO_MERGE	Zusammenführung von Abfrage- und Viewtext erzwingen/unterbinden
USE_HASH, USE_MERGE, USE_NL	Hash Join, Merge Join oder Nested Loops erzwingen
FULL	Indexnutzung unterbinden, Anforderung eines Full Table Scans (bspw. sinnvoll, wenn zugegriffener Wertebereich einen Großteil der Relation ausmacht)
INDEX INDEX_FFS/ NO_INDEX	Bestimmte Indexnutzung (bspw. mit Index Fast Full Scan) erzwingen/unterbinden
PARALLEL/ NO_PARALLEL	Festlegung der maximalen Anzahl Serverprozesse (Parallelisierungsgrad) bzw. sequentiellen Abarbeitung erzwingen
CLUSTER/HASH	Anforderung eines Cluster- oder Hash-Scans
PUSH_SUBQ	Sorgt dafür, dass korrelierende Unterabfragen im Ausführungspfad möglichst früh ausgewertet werden (korrelierende Unterabfragen sonst an letzter Stelle)
USE_CONCAT	Erzwingung einer UNION ALL-Operation bei kombinierten OR-Bedingungen einer WHERE-Klausel
RULE, CHOOSE	Optimizernutzung: regelbasiert oder nach Wahl (regelbasiert, sofern Relationen nicht analysiert)

Tabelle 3: Optimizer-Hints Oracle 10g

Einer der grundlegendsten Oraclehintergrundprozesse ist der DBWR[21], der für den Datenaustausch zwischen der SGA und den Datenbankdateien und somit für die Bufferverwaltung zuständig ist. Durch den DBWR wird synchron alle 3 Sekunden der LGWR angesprochen. Der LGWR ist für die Pflege der Redo files und somit für die Sicherstellung eines möglichen Recoverys verantwortlich. Aus Sicht der I/O-Optimierung wäre in Verbindung mit dem DBWR und dem LGWR eine Umstellung auf ein asynchrones Zeitintervall denkbar. In Verbindung mit Änderungs- oder Erstellungsoperationen von Relationen und Indizes kann der Redo-Pflegeaufwand zusätzlich über den Nologging-Parameter verringert werden. Somit kann zusätzlich Geschwindigkeit gewonnen werden, währenddessen jedoch Sicherheit in Bezug auf eine Wiederherstellung verloren gehen könnte.

2.3.6 Auflistung der datenbankseitigen Optimierungsansätze

Nachdem in den vorhergehenden Kapiteln verschiedenste Möglichkeiten zur Performancesteigerung innerhalb der Oracledatenbank zusammengetragen wurden, sollen diese folgend zur besseren Übersicht nochmals tabellarisch aufgelistet werden.

[21] DBWR – Database Writer (vgl.: Abbildung 2: Struktur des Oracle-RDBMS)

Bereich	Erläuterung
Datenbank-verbindung	Der Verbindungsaufbau zählt zu den langsamsten Vorgängen beim Datenbankzugriff, somit sollten bestehende Verbindungen kontinuierlich genutzt bzw. aufrechterhalten werden.
Datendurchsatz	Einsatz von RAID-Systemen Beim Anlegen der Datenbank die größtmögliche Blockgröße wählen Partitionierung, um kleinere verwaltbarere Einheiten zu schaffen Tablespaces: Trennung von Relationen und Indizes, Einsatz temporärer Tablespaces, Auslagerung großer Objekte/Rollback-Segmente in einen eigenen Tablespace
Migration	Verwendung von Outlines, um nach einer Umstellung gleiche Ausführungspfade zu garantieren, die sicherstellen, dass keine Performanceeinbrüche zu erwarten sind
Normalisierung	Den Designmittelpunkt sollte nicht die perfekte Normalisierung, sondern vielmehr der Anwender bilden.
Oracleparameter	Buffer_pool-Klausel: für kleinere/wichtige Relationen um diese konstant im Arbeitsspeicher zu halten Db_block_size: Festlegung der Blockgröße (typisch 4 KB und 8 KB) Pctfree-/Pctused-Parameter: Festlegung der Datenblockausnutzung (Effizienz) Arraysize-Parameter: zur Verkleinerung des Rollback-Segments bei umfangreichen Transaktionen Sga_max_size: Maximalgröße der SGA Shared_pool_size: definiert die Größe des gemeinsam genutzten Pools, bestimmt somit den Data Dictionary-Cache und Library Cache Db_cache_size: Festlegung der Default-Buffergröße Db_keep_cache_size: Speicherbereichsgröße für Relationen, die im RAM gehalten werden sollen[22] Large_pool_size: Größendefinition des optionalen Speicherbereichs der SGA, der bspw. beim Back-up/Restore genutzt werden kann Pga_aggregate_size: bestimmt Gesamtspeichermenge des Hauptspeichers, der Serverprozessen zugewiesen wird (Bsp.: genutzt für Sortier- und Hashalgorithmen), bestimmt somit indirekt die Häufigkeit zusätzlicher Auslagerungsvorgänge, Alternative: Parameter Sort_area_size und der damit verbundene Parameter Hash_area_size Max_dispatchers: Bestimmt die Höchstanzahl an Dispatcherprozessen, die simultan laufen dürfen Nologging-Parameter: Verringerung des Wiederherstellungspflegeaufwands
Oracleprozesse	Mit jeder neuen Version werden die Möglichkeiten zur Beeinflussung Oracle eigener Prozesse verringert. In Version 10g lässt es Oracle aber bspw. noch zu, den Abfragezyklus zwischen DBWR und LGWR auf asynchron umzusetzen. Parameter Sga_target auf einen Wert ungleich 0 und Parameter Statistics_level auf typical/all setzen, um das Automatic Shared Memory Managements zu aktivieren
Sortierungen	Sortierungen beeinflussen die Geschwindigkeit der Ergebnisrückgabe und werden bei verschiedensten Operationen teils unbemerkt eingesetzt. Empfehlungen: − Sortierungen nach Möglichkeit auf Indizes anwenden − Sofern Sortierungen unvermeidbar sind, diese nach Möglichkeit im Arbeitsspeicher ausführen − UNION ALL der UNION-Anweisung vorziehen, da diese keine Sortierung zur Unterdrückung doppelter Datensätze nach sich zieht − Wenn möglich Vermeidung von DISTINCT-, MINUS- und INTERSECT-Anweisungen, da diese ebenfalls prinzipiell Sortiervorgänge erfordern − Aggregatfunktionen: MIN, MAX und COUNT möglichst auf indizierte Attribute anwenden
Statistiken	Sollten möglichst aktuell sein, jedoch benötigt deren Erstellung teils erhebliche Ressourcen, weshalb die Datensammlung außerhalb der Kernarbeitszeit stattfinden sollte. Beschleunigung der manuellen Statistikerstellung über Db_file_multiblock_read_count- und Sort_area_size-Parameter möglich
Verbund-operationen	Verbunde sollten nach Möglichkeit von der kleinsten bzw. selektivsten Relation zur größeren hin entwickelt werden, um den Aufwand der Folgeoperationen zu minimieren.
Views	Einsatz materialisierter Views, um Benutzeranfragen auf lokale Datenkopien umzuleiten Für gewöhnliche Views gilt, dass diese nach Möglichkeit bspw. durch Einsparung von Gruppier- und Sortierfunktionen so zu organisieren sind, dass eine Kombination von View- und Abfragetext möglich ist.
Zugriffe	Performanceprobleme sind oft nicht in der Anweisung selbst, sondern in der Häufigkeit der nötigen Zugriffe zu finden. Ziel sollte somit die Minimierung bzw. Erhöhung der Effektivität physischer Zugriffe bspw. durch Anlegen von Indizes sein. Häufige Zugriffe können auch durch Zusammenfassung verschiedener Anweisungen in Prozeduren, Funktionen oder Packages eingeschränkt werden. Hierbei ist auch zwangläufig darauf zu achten, dass sich Zugriffe ebenfalls durch Zwischenspeicherung in Variablen verringern lassen.

Tabelle 4: Auflistung der Oracle-Optimierungsansätze

Bei allen aufgelisteten Maßnahmen ist zu bedenken, dass voraussichtlich keine der Maßnahmen prinzipiell eine drastische Performancesteigerung nach sich zieht. Viel-

[22] siehe: Buffer_pool-Klausel

mehr ist es die Summe der Einzelmaßnahmen, die bspw. in Abhängigkeit der Nutzerzahl eine spürbare Verbesserung nach sich ziehen kann. Hierbei ist weiterhin zu bedenken, dass für eine erfolgreiche Optimierung einer Applikation ein Erfolg meist nur durch die Zusammenarbeit von Entwickler, Designer, DBA und Endanwender ermöglicht werden kann.

2.4 Datensatz- und Attributanzahl

Neben der Datensatzanzahl hat auch die Attributanzahl unmittelbare Auswirkungen auf die Performance des Gesamtsystems. So ist es bei entsprechenden umfangreichen Relationen nach Möglichkeit zwangsläufig zu umgehen, dass in den Erfassungsmasken alle Datensätze oder auch alle Attribute zur Verfügung gestellt werden. Zur Verdeutlichung dieser Abhängigkeiten sollen die folgenden Diagramme dienen, in denen die Abfragedauer in Abhängigkeit zur Attribut- und Datensatzanzahl dargestellt wird. Bei der Attributanzahl wurden jeweils ca. 470 Datensätze aus der Relation FAHRZEUG abgefragt. Die Ermittlung der Werte für die Datensatzanzahl wurde mit Hilfe der Relation SENDUNG durchgeführt.

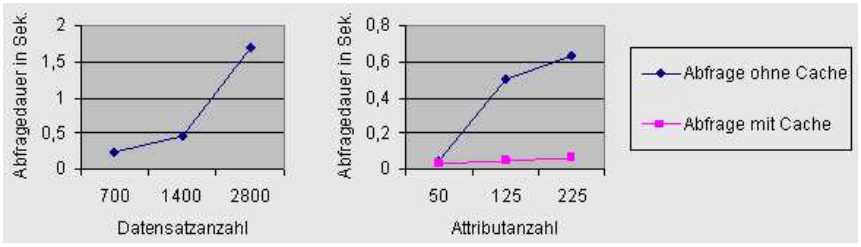


Abbildung 9: Abfragedauer in Abhängigkeit zur Attribut- und Datensatzanzahl

Aus Sicht der Attributanzahl ist in Abbildung 9 eindeutig zu erkennen, dass die 470 Datensätze mit 50 Attributen bereits in 0,05 Sekunden ermittelt werden können. Im Gegensatz dazu benötigt die 2,5fache Attributanzahl ca. 10mal so lange für die gleiche Anzahl Datensätze. Hierbei wurde selbstverständlich darauf geachtet, dass bei den zusätzlichen Attributen keine Datentypen wie bspw. CLOB[23] oder BLOB[24] enthalten sind, die das Ergebnis verfälschen könnten. Eine weitere Erhöhung der Attributanzahl hat folgend nicht annähernd derartig gravierenden Auswirkungen, was bspw. auf die automatische oracleseitige Parallelisierung der Operation sowie den Einsatz anderer Zugriffsstrategien zurückzuführen ist. In einem der vorhergehenden Gliederungspunkte wurde die Leistungsfähigkeit des Datenbankcaches angesprochen so-

[23] CLOB – Character Large Object
[24] BLOB – Binary Large Object

wie die Ermittlung der Trefferrate dargestellt. Bei erneuter Ausführung der betreffenden Anweisung lässt sich dies, wie in der Abbildung dargestellt, durch die Rückgabezeit des Ergebnisses bestätigen.

Weiterhin können große Datenmengen einer der Gründe sein, warum Anwendungen im Laufe der Zeit spürbar langsamer werden. In Verbindung mit Oracle ist so bspw. denkbar, dass bei Binärbäumen für Indizes zu Relationen mit einer großen Datenmenge die Eintragsebenen gesplittet werden müssen, wodurch unter Umständen die Zugriffszeiten erheblich ansteigen können. Der Zustand des Index-Baumes wird durch Einfüge- und Löschoperationen zusätzlich beeinflusst und lässt sich unter Oracle ebenfalls beurteilen. Über die SQL-Anweisung ANALYZE INDEX IDX_SENDPOS_FZ_KENNUNG VALIDATE STRUCTURE können bspw. für den Index IDX_SENDPOS_FZ_KENNUNG die Werte ermittelt werden. Die ermittelten Werte können dann über den View INDEX_STATS abgefragt werden. Die Beurteilung des Änderungsgrads des Indizes kann dann über die Anweisung SELECT NAME, ROUND((DEL_LF_ROWS_LEN/LF_ROWS_LEN)*100,2) FROM INDEX_STATS durchgeführt werden. Hierbei bezeichnet LF_ROWS_LEN die Länge aller Datensätze, währenddessen DEL_LF_ROWS_LEN die Länge aller gelöschten Datensätze beinhaltet. So ist es ab einem Änderungsgrad von ca. 15 % bspw. sinnvoll, den Index mit dem REBUILT-Schlüsselwort reorganisieren zu lassen [ANB06]. Analog könnte auch der Index entfernt und erneut angelegt werden. Im Falle einer Umbenennung oder Umwandlung von einem B*-Index in einen Bitmap-Index ist dies sogar unumgänglich. Zu beachten ist hierbei jedoch, dass der Index dann einerseits zeitweise nicht verfügbar ist und andererseits eine erneute Sortierung erforderlich wird.

Die eben angesprochenen Reorganisationsläufe können sich jedoch auch ebenfalls bei Relationen als sinnvoll erweisen. Hierfür sei zur Erläuterung gesagt, dass zur Speicherung von Zeichenketten CHAR und VARCHAR2 in Frage kommen. Währenddessen bei CHAR prinzipiell durch die Auffüllung mit Leerzeichen der vollständige Speicherplatz belegt wird, wird bei VARCHAR2 immer nur der soeben benötigte Speicherplatz belegt. Die Art der Speicherung bei VARCHAR2 kommt zwar der Abfrageperformance entgegen, jedoch führt es bei umfangreichen Änderungen ggf. zu verteilter Speicherung des Feldinhalts. Im Versuch bei CIS konnte dennoch keine

Performancesteigerung nach Reorganisation der Relationen SENDUNG und KUNDEN festgestellt werden.

Insgesamt ist folgend an dieser Stelle eher der Ansatz zur Performancesteigerung in Verbindung mit einer Archivierung zu untersuchen, was im abschließenden Teil dieser Diplomarbeit dargestellt werden soll. Bei der konkreten Umsetzung der Archivierungslösungen könnten ggf. auch die eben genannten Aspekte beachtet werden. Mit den letzten Ausführungen sollte hauptsächlich verdeutlicht werden, dass große Datenmengen zahlreiche weitere Probleme verursachen können, die bereits bei der Planung und Umsetzung beachtet werden sollten.

2.5 Entwicklungskomponenten

Wie bereits eingangs erwähnt dient Delphi 5 als Programmierumgebung für C-Logistic. Als Entwicklungskomponenten kommen für die Datenbankanbindung fast ausschließlich ODAC[25]- Komponenten zum Einsatz. Weiterhin werden an verschiedenen Stellen MyDAC[26]- und Ado[27]-Komponenten verwendet. Im Folgenden sollen verschiedene Datenbankkomponenten über ein eigenes, in der folgenden Abbildung dargestelltes, Testprojekt miteinander verglichen werden, um zu ermitteln, ob spürbare Zeitunterschiede erkennbar sind.

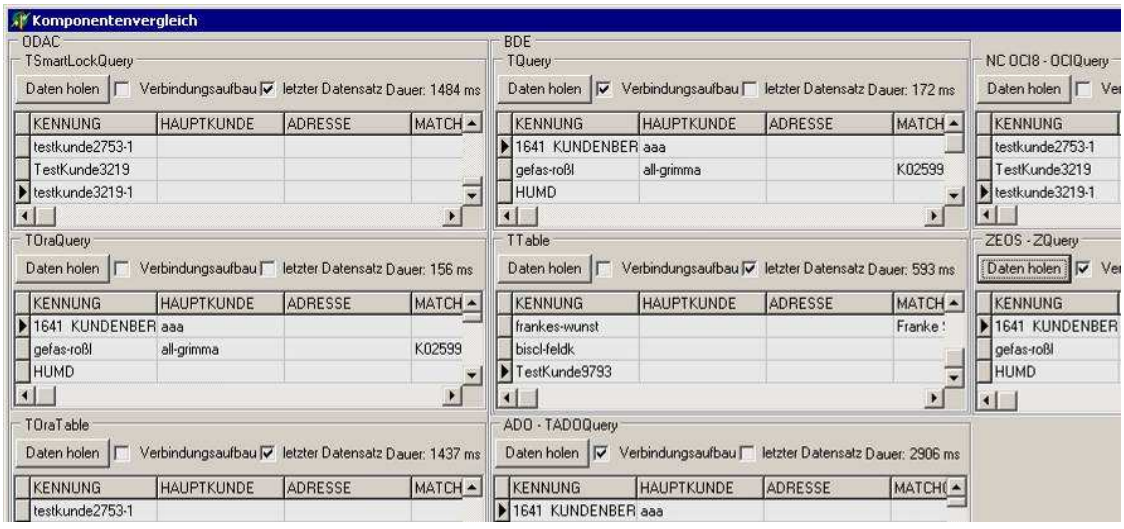


Abbildung 10: Testprojekt für Datenbankkomponenten

Neben dem dargestellten Testprojekt aus Delphi 5 wurde ein weiteres Testprojekt in Delphi 2007 entworfen, um bspw. die dbExpress-Komponenten gegenüberstellen zu

[25] ODAC – Oracle Data Access Components
[26] MyDAC – Data Access Components for MySQL
[27] ADO – ActiveX Data Objects

können. Mit Hilfe der erstellten Testprojekte konnten teilweise beträchtliche Zeitunterschiede ermittelt werden. Eine komplette Auflistung der ermittelten Werte ist in folgender Tabelle dargestellt.

	ODAC	**BDE**	**ADO**	**DOA**	**NC OCI8**	**ZEOS**	**SQLDirect**	**dbExpress**
Bereitstellung der Daten	SmartLockQuery 162 ms OraQuery 158 ms OraTable 167ms SmartQuery 156 ms	Query 141 ms Table 160 ms	ADOQuery 2833 ms	OracleDataSet 23474 ms	OCIQuery 167 ms	ZQuery 3682 ms	SDQuery 318 ms SDTable 130 ms	SQLDataSet 131ms SQLQuery 100 ms SQLTable 125 ms
Bereitstellung mit vorherigem Verbindungsaufbau	SmartLockQuery 195 ms OraQuery 188 ms OraTable 209 ms SmartQuery 188 ms	Query 143 ms Table 188 ms	ADOQuery 2875 ms	OracleDataSet 23578 ms	OCIQuery 203 ms	ZQuery 3839 ms	SDQuery 420 ms SDTable 276 ms	SQLDataSet 1854 ms SQLQuery 219 ms SQLTable 1875 ms
Bereitstellung mit Sprung auf letzten Datensatz	SmartLockQuery 1260 ms OraQuery 1280 ms OraTable 1287 ms SmartQuery 1260 ms	Query 1281 ms Table 547 ms	ADOQuery 2849 ms	OracleDataSet 23427 ms	OCIQuery 2448 ms	ZQuery 3693 ms	SDQuery 5135 ms SDTable 4949 ms	SQLDataSet 2887 ms SQLQuery 2849 ms SQLTable 2838 ms
Bereitstellung, Verbindungsaufbau und Sprung letzter Datensatz	SmartLockQuery 1328 ms OraQuery 1290 ms OraTable 1297 ms SmartQuery 1302 ms	Query 1276 ms Table 563 ms	ADOQuery 2895 ms	OracleDataSet 23495 ms	OCIQuery 2479 ms	ZQuery 3808 ms	SDQuery 5204 ms SDTable 5083 ms	SQLDataSet 4526 ms SQLQuery 2886 ms SQLTable 4573 ms

Tabelle 5: Vergleich unterschiedlicher Datenbankkomponenten

Bei den allgemeinen Tipps zur Optimierung der Oracledatenbank wurde empfohlen, eine Verbindung kontinuierlich zu nutzen, da der Verbindungsaufbau ebenfalls einen beträchtlichen Zeitaufwand erfordert. Dieser Umstand kann mit Hilfe der obigen Auflistung nachweislich bspw. bei der Verwendung mit ODAC-Komponenten bestätigt werden. Weniger gravierend sind die Auswirkungen bei den BDE[28]-Komponenten, die die Verbindung über vordefinierte Aliase umsetzen. Für die Verwendung mit C-Logistic empfiehlt sich weiterhin die Verwendung von ODAC, da diese Komponenten zum einen kontinuierlich weiterentwickelt werden und zum anderen im Vergleich am schnellsten die Daten bereitstellen können.

Folgend sollen noch kurz eine paar Erläuterungen zu der DOA[29]-Komponente angeführt werden, da diese performanceseitig im Vergleich zu anderen Komponenten deutlich abfiel. Währenddessen einige Umsetzungen, wie bspw. die ODAC-

[28] BDE – Borland Database Engine
[29] DOA – Direct Oracle Access

Komponenten oder die OCIQuery-Komponente, nur eine geringe Datensatzzahl unmittelbar bereitstellen und die restlichen Datensätze erst bei Bedarf schrittweise nachladen, werden beim OracleDataSet prinzipiell alle Datensätze geladen und bereitgestellt. Dennoch bieten DOA-Komponenten, ähnlich wie Oracle[30], die Möglichkeit, Optimierungsfestlegungen zu treffen. Hierbei kann ebenfalls gewählt werden, ob das primäre Ziel ein vollständiges Ergebnis oder die möglichst schnelle Rückgabe erster Ergebnisse darstellen soll. Entgegen möglicher Erwartungen beträgt jedoch der Zeitunterschied für die Ergebnisrückgabe lediglich ca. 1 Sekunde, sodass von einer spürbaren Beschleunigung unter Berücksichtigung der Gesamtdauer keine Rede sein kann. Neben der schlechten Antwortzeit bereits bei einer einfachen Abfrage enttäuscht vor allem der enorme Speicherbedarf, sodass ein Einsatz in C-Logistic bspw. bei mehreren offenen Erfassungsmasken undenkbar ist. Dass die Methode der vollständigen Datenbereitstellung nicht zwangsläufig enorm zeitaufwändig sein muss, beweist die ADOQuery-Komponente. Diese stellt ebenfalls sofort alle Datensätze in einem Bruchteil der Zeit bereit. Dennoch überzeugen auch unter diesem Gesichtspunkt ebenfalls die ODAC-Komponenten, da diese wiederum bspw. beim Sprung auf den letzten Datensatz teilweise weniger als die Hälfte der Zeit benötigen.

Insgesamt sollte der durchgeführte Vergleich primär als grober Überblick dienen. Bei konkretem Einsatz der betreffenden Komponenten sind noch weitere kleine Optimierungen, wie bspw. das feste Einfügen der kompletten Feldliste, denkbar. Die versuchsweise Umsetzung eines Kopiervorgangs von ca. 50 Sendungen und ca. 200 Sendungspositionen über eine ODAC-SmartLockQuery ergab hierbei bspw. einen Zeitaufwand von 7625 ms im Vergleich zu 10854 ms beim Zugriff über „FieldByName“, was einer Einsparung von ca. 30 % entspricht.

Den BDE-Komponenten ist ebenfalls eher ein symbolischer Charakter zuzuordnen, da diese nicht mehr weiterentwickelt bzw. in aktuelleren Delphiversionen durch dbExpress-Komponenten ersetzt wurden. Die dbExpress-Komponenten sind mit den BDE-Komponenten nur sehr begrenzt vergleichbar. Dies liegt bspw. daran, dass diese nur noch auf den unidirektionalen Datenverkehr ausgerichtet sind. Hierdurch werden keine Daten mehr gepuffert, sodass eine Bearbeitung von Datensätzen nur noch über explizite UPDATE-Anweisungen oder ein zusätzliches ClientDataSet möglich

[30] vgl.: Optimizer-Hints FIRST_ROWS und ALL_ROWS

ist. Weiterhin stehen zum Navigieren bspw. im SQLDataSet innerhalb der Ergebnismenge nur noch die Methoden First und Next zur Verfügung, sodass bei der Gegenüberstellung der Sprung auf den letzten Datensatz über die Next-Methode umgesetzt werden musste. Auffällig ist ebenfalls, dass der Verbindungsaufbau mit dbExpress-Komponenten im Gegensatz zu den BDE-Komponenten extrem zeitaufwändig ist.

Neben den dargestellten Lösungen sollte ebenfalls die kommerzielle Lösung SQLDirect unter Delphi 5 betrachtet werden. Bei dieser war es jedoch aufgrund eines schwerwiegenden Fehlers nicht möglich, eine Verbindung vom Testprogramm zur Oracledatenbank unter Delphi 5 aufzubauen. Somit wurden die SQLDirect-Komponenten genauso wie die dbExpress-Komponenten unter Delphi 2007 getestet. Die beiden unter Delphi 2007 getesteten Lösungen sind ebenfalls in der vorhergehenden Tabelle aufgelistet. Auf einen erneuten Vergleich der BDE- oder ODAC-Komponenten unter Delphi 2007 wurde an dieser Stelle verzichtet.

Ein Großteil der C-Logistic-Anwender arbeitet gut acht Stunden täglich mit der Softwarelösung. Hierbei werden verschiedenste Erfassungsmasken und somit verschiedenste Methoden, Funktionen, Prozeduren usw. aufgerufen. Wichtig ist hierbei, dass für die Ausführung belegter Speicherplatz bspw. bei einer dynamisch erzeugten Komponente oder Stringliste vollständig wieder freigegeben wird, sodass keine zusätzlichen Engpässe provoziert werden. Hierfür gibt es für Delphi verschiedene „Plug-Ins“, wie bspw. FastMM4, die selbständig „Speicherlöcher“ erkennen. Weiterhin wird durch die Verwendung des FastMM4-Speichermanagers ebenfalls ein Performancevorteil versprochen.

Im Zuge einer Umstellung des Ortsverzeichnisses von einer Paradox-Tabelle auf die Oracledatenbank wurden Varianten zur Beschleunigung des Einfügens neuer Datensätze untersucht. Ausgangspunkt der Betrachtung stellt hierbei die Ausführung einzelner INSERT-Anweisungen aus einem externen Programm, wie bspw. dem Abfragetool Golden32 von Benthic Software, dar. Als zweites wurden die INSERT-Anweisungen einzeln über Datenbankkomponenten ausgeführt. Diesen beiden Varianten soll folgend eine Variante mit der Verwendung eines DML-Arrays gegenübergestellt werden. In der folgenden Abbildung ist der Zeitunterschied für die drei genannten Varianten beim Einfügen von 10.000 Datensätzen dargestellt.

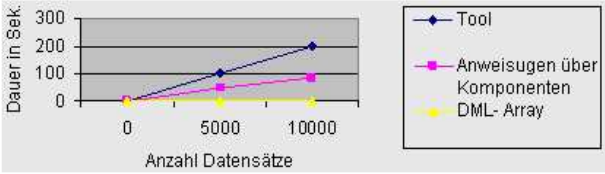


Abbildung 11: Zeitbedarf DML-Array

Als langsamste Variante stellt sich hierbei erwartungsgemäß die Ausführung einzelner INSERT-Anweisungen aus einem externen Programm heraus. Weniger als die Hälfte der Zeit benötigen im Gegensatz die einzelnen Anweisungen bei der Ausführung über entsprechende Datenbankkomponenten. Obwohl sich die Anweisungen mit und ohne DML-Array nicht unterscheiden, ist es mit Hilfe des DML-Arrays möglich, alle 10.000 Datensätze in weniger als 2 Sekunden anzulegen. Der Unterschied besteht hierbei darin, dass durch die Arraygröße angegeben wird, wie viele Anweisungen gemeinsam abgeschickt bzw. ausgeführt werden sollen. Im konkreten Fall bedeutet dies bei einer Arraygröße von 50, dass bei 10.000 Datensätzen lediglich 200 Anforderungen ausgeführt werden müssen. Die komplette Datenübernahme der 477.000 Orte lässt sich somit von 2,5 Stunden im schlechtesten Fall auf rund 2 Minuten minimieren. Die Verwendung eines DML-Arrays stellt somit eine deutliche Zeiteinsparung dar, die in C-Logistic bspw. bei umfangreichen Verbuchungsvorgängen Anwendung finden könnte.

Insgesamt betrachtet gibt es an der Programmiersprache Delphi bzw. am Compiler selbst wenige Ansätze zur Optimierung. Interessanter sind hier allgemeine Hinweise, die ein Programmierer beachten sollte. So ist bspw. darauf zu achten, dass in Datenbankkomponenten und SQL-Anweisungen nach Möglichkeit jeweils nur die Felder hinzugefügt werden, die auch für die Bearbeitung gebraucht werden. Wichtig ist hierbei ebenfalls, dass der Zugriff direkt auf fest hinzugefügte Attribute und nicht bspw. über „FieldByName“ umgesetzt wird. Somit kann eine ständige Suche in der Feldliste vermieden werden. Nur sehr selten benötigte Abfragen/Komponenten müssen nicht zwangsläufig fest auf einem Datenmodul hinterlegt und schlimmstenfalls bei jedem Öffnen des Datenmoduls mit geöffnet werden, sondern können vielmehr auch dynamisch zur Laufzeit erzeugt werden. Bei mehrmaligen aufeinander folgenden Änderungen der Daten einer Relation, die bspw. gleichzeitig in einem DBGrid dargestellt wird, kann für die Dauer der Änderung mit den Methoden DisableControls und EnableControls eine ständige Anpassung der Darstellung unterbunden werden.

In ODAC sowie in einigen weiteren Umsetzungen stehen bspw. neben den TQuery-Komponenten auch Skriptkomponenten bereit, die für reine INSERT-, UPDATE- oder DELETE-Anweisungen genutzt werden können. Weiterhin kann ein Großteil der Funktionalität direkt auf den Server in Prozeduren oder Packages verlagert werden, sodass zur Ausführung in Verbindung mit ODAC-Komponenten bspw. nur noch eine StoredProc-Komponente benötigt wird. Hierbei kann allgemein betrachtet der Einsatz von PL/SQL[31] als Ersatz ineffizienter SQL-Statements durch effizientere Prozeduren angesehen werden. So lassen sich bspw. ineffiziente DECODE-Anweisungen durch Anweisungen ersetzen, in denen gezielt Indizes genutzt werden können. Abfragen, bei denen es lediglich darum geht festzustellen, ob Datensätze vorhanden sind, müssen bspw. in Verbindung mit einem Existenzquantor nicht zwangsläufig eine Attributauswahl enthalten. Hier wäre im einfachsten Fall bereits bspw. ein SELECT 1 FROM NIEDERLASSUNGEN völlig ausreichend. Neben den bisherigen Beispielen ist es auch wichtig, dass vor allem bei umfangreichen Relationen zeitaufwändige Refreshvorgänge nicht übermäßig eingesetzt werden. All dies sind Beispiele für Gewohnheiten, die der Programmierer im Laufe der Zeit in seinem eigenen Programmierstil verinnerlichen sollte, um zusätzliche Engpässe zu vermeiden. Auf eine Auflistung weiterer Ansätze, die der Programmierer während seiner täglichen Arbeit beachten sollte, vermag an dieser Stelle verzichtet werden. Vielmehr kann bei der konkreten Analyse in C-Logistic an betreffenden Stellen ggf. darauf eingegangen werden.

2.6 Entwicklungsumgebung

Im Zuge der vorhergehenden Studienarbeit mit dem Thema „Möglichkeiten einer teilautomatisierten Fehlerprüfung und -analyse von Softwareupdates am Beispiel von C-Logistic“ wurden verschiedene Varianten teilautomatisierter Softwaretests zur Verwendung mit C-Logistic untersucht. Hierbei wurden Testszenarien mit TestComplete 3 entworfen. TestComplete bietet die Möglichkeit, durch den Performanceprofiler AQTime ergänzt zu werden. Die Firma AutomatedQA hat sich mir ihren Produkten TestComplete und AQTime auf teilautomatisierte Softwaretests und auf die Erkennung von Speicher- und Leistungsengpässen spezialisiert. Mittlerweile wirbt das Unternehmen damit, dass selbst Borland zur Beseitigung von Leistungsengpässen in Delphi 2006 auf AQTime vertraut hat. C-Logistic wird mit Delphi 5 entwickelt, das

[31] PL/SQL – Procedural Language/SQL

bereits 1999 veröffentlicht wurde. In der Zwischenzeit wurde Delphi um eine Vielzahl neuer Funktionen erweitert, und es wurden auch gezielt, wie zuvor erwähnt, performanceseitige Schwachstellen beseitigt.

In diesem Kapitel soll untersucht werden, ob evtl. aus dem Umstieg auf eine aktuelle Delphiversion performanceseitige Verbesserungen erwartet werden könnten. Aktuell ist bei CIS ein Umstieg auf Delphi 2009 in Planung, aus diesem Grund existiert neben der gewöhnlichen Entwicklerversion noch eine weitere C-Logistic-Version, die mit Delphi 2009 erstellt wurde. Im Folgenden sollen einige Tests durchgeführt werden, um festzustellen, ob zwischen den beiden genannten Versionen Unterschiede feststellbar sind.

Für die folgende Gegenüberstellung soll C-Logistic in bekannten Problembereichen mit AQTime in einem vorgegebenen Szenario betrachtet werden. Aus Performancesicht lassen sich hierbei bspw. die Erfassung und Optimierung von Transportaufträgen, die graphische Disposition und die debitorische und kreditorische Faktura als Problembereiche identifizieren. Somit sollte sich bei gezielten Untersuchungen vorerst auf diese Bereiche konzentriert werden, da hier die deutlichsten Effekte zu erwarten sind. Weiterhin ließen sich mit teilautomatisierten Softwaretests ggf. Stressszenarien definieren, mit denen es ermöglicht werden könnte, Optimierungen an der Datenbank erfolgreich zu beurteilen. Ein Stresstest kann Aspekte aufzeigen, die sich nicht mit Einzelnutzern simulieren lassen. So ist es zur Vermeidung von Engpässen aus Datenbanksicht bspw. wichtig festzustellen, wann:

- Nutzer mit Deadlocks konfrontiert werden,
- Konsistenzprobleme auftreten,
- bzw. allgemein gesagt Performanceeinschränkungen auftreten.

Weiterhin könnten derartige Testszenarien zur Identifikation von Bereichen dienen, in denen häufig die bereits erwähnten Undo-Segmente eingesetzt werden, welche die Performance zusätzlich spürbar beeinträchtigen. Hierbei ist jedoch nicht zu vergessen, dass auch ein derartiges Testszenario meist nicht ausreicht, um den Einsatz beim Kunden zu simulieren.

Für den folgenden Vergleich sollen vorerst mit dem Performanceprofiler aus AQTime gezielt einzelne Funktionen bspw. in der Disposition oder der Sendungserfassung betrachtet werden. Hierfür bietet AQTime die Möglichkeit zu ermitteln, wie lange sich

das Programm innerhalb einer bestimmten Funktion aufhält. In der folgenden Abbildung ist beispielhaft ein entsprechender Aufrufgraph bei der Initialisierung des Geocoders beim Öffnen der Disposition dargestellt.

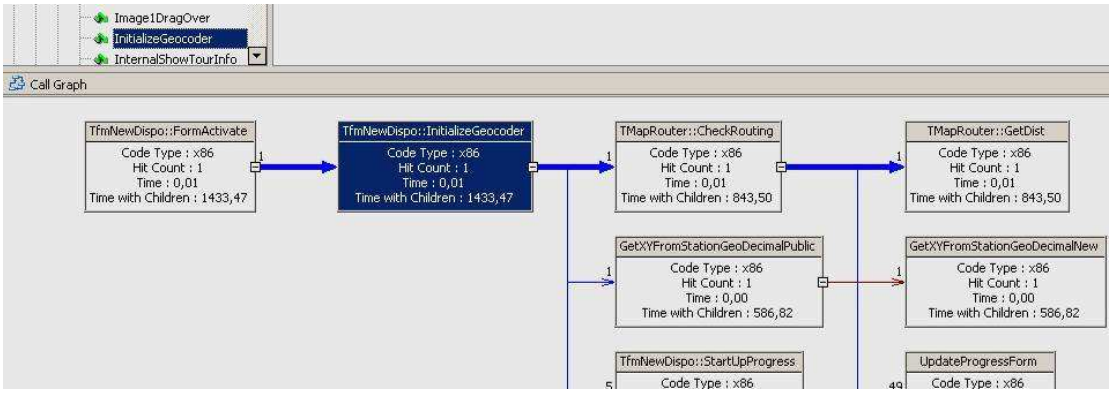


Abbildung 12: Aufrufgraph AQTime

In der vorhergehenden Abbildung ist erkennbar, dass neben dem Zeitaufwand der einzelnen Funktionen bzw. Methoden auch die Häufigkeit des Aufrufes erkennbar ist, sodass sich daraus bereits weitere Optimierungsansätze ableiten lassen könnten. Vorerst soll jedoch exemplarisch folgend der Zeitaufwand verschiedener Funktionen beim Aufruf der Disposition und der Sendungsmaske gegenüberstellt werden, um ggf. Performanceunterschiede zwischen den beiden Programmierumgebungen ermitteln zu können.

Funktion	**Delphi 5**	**Delphi 2009**
CloseOffeneSP (Disposition)	*195,27 ms*	253,74 ms
CreateDayView (Disposition)	*287,73 ms*	302,71 ms
CloseTourSearchForm (Disposition)	169,09 ms	*166,88 ms*
FormCreate (Sendung)	*446,23 ms*	454,38 ms
SourceDataChange (Sendung)	75,18 ms	*53,99 ms*
ShowSendungserfassungFernverkehr (Sendung)	2.309,02 ms	*2.012,56 ms*

Tabelle 6: Gegenüberstellung der Delphiversionen

Entgegen möglicher Versprechungen lässt die vorhergehende Tabelle erkennen, dass beim Vergleich beider Programmierumgebungen unter den derzeitigen Bedingungen keine grundlegende Empfehlung für Delphi 2009 abgegeben werden kann. Begründen lässt sich diese Feststellung bspw. damit, dass C-Logistic vorwiegend auf den Einsatz von ODAC-Datenbankkomponenten beruht, die vermutlich in beiden Entwicklungsumgebungen gleiche Performance zeigen. Erkennbar ist aber auch, dass Standardmethoden wie bspw. das DataChange in der Sendungserfassung in der aktuellen Delphiversion durchaus schneller umgesetzt werden können.

2.7 Arbeitsweise der Anwender

Es ist von Beginn an notwendig, dem Anwender eine gewisse Sensibilisierung im Umgang mit großen Datenmengen beizubringen. Dies ist momentan bedauerlicherweise bei einigen Kunden nicht der Fall. Zusätzlich ist es empfehlenswert, ggf. mit Hilfe von selbstdefinierbaren Vorfiltern den sichtbaren Datenbestand einzuschränken. Ein weiterer Ansatz ist das mittlerweile teilweise umgesetzte Vorgehen in C-Logistic, bei dem der Nutzer beim Öffnen einer Erfassungsmaske zu Beginn lediglich die Erfassungsmaske zu sehen bekommt. Danach hat er erst die Möglichkeit, sich Daten anzeigen zu lassen bzw. deren Umfang zu wählen. Weithin kann mit Hilfe einer aufschlussreichen Dokumentation dem Anwender ein empfohlenes Standardvorgehen, bspw. im Bezug auf Vorfilter, nahe gebracht werden. Diese Dokumentation könnte gleichzeitig auch als eine Orientierungshilfe für die eigenen Mitarbeiter dienen. So könnte in einer derartigen Dokumentation bspw. weiterhin die Empfehlung gegeben werden, dass umfangreiche Bewertungsvorgänge nicht zu Stoßzeiten ausgeführt werden sollten. Hier wäre es sicherlich für alle Nutzer sinnvoller, derartige Vorgänge auf weniger belastete Zeiten, wie bspw. einer einheitlichen Mittagspause, zu verlegen. Selbstverständlich ist hierbei nicht davon auszugehen, dass für die Arbeitsweise aller Kunden Empfehlungen abgegeben werden können, da hierzu die Kundenanforderungen zu vielfältig gestaltet sind.

Zur Beurteilung der Arbeitsweise der Anwender können neben persönlichen Gesprächen ggf. auch die Oraclestatistiken Aufschluss geben. So ist es, wie bereits erwähnt, vorgesehen, dass Erfassungsmasken mit einem hohen Datenvolumen nur mit angepassten Filtern aufgerufen werden. Anwender, die diese bewusst umgehen, könnten anhand der ressourcenintensiven SQL-Anweisungen erfolgreich identifiziert werden. Zur Beurteilung des Datenumfangs und den damit verbundenen auftretenden Performanceproblemen sind die Oracledaten ein unerlässliches Hilfsmittel. Ein sinnbildliches Analysevorgehen könnte folgendermaßen gestaltet werden:

1. Überblick durch Oraclehilfsmittel verschaffen/Untersuchung der Datenbankstatistiken während akzeptabler Performance
2. Beurteilung und Vergleich mit Problemzeiten um vorhandene Wartezeiten zu ermitteln
3. Optimierungsschwerpunkte festlegen (Bsp.: Lese- und Schreibanforderungen im Zusammenhang mit Plattenperformance, SQL-Optimierung…)

Da C-Logistic eine Vielzahl an Anforderungen abdecken muss, existiert eine Vielzahl an Modulen, die teilweise kundenspezifische Anpassungen beinhalten. Während der Erstellung dieser Diplomarbeit hat sich bspw. beiläufig eine seit langer Zeit implementierte kundenspezifische Auswertung als ressourcenintensiv herausgestellt. Diese Auswertung ist bei CIS ebenfalls vorhanden, jedoch wird sie in den seltensten Fällen eingesetzt, sodass der Schwachpunkt nicht festgestellt werden konnte. Seitens CIS ist es aus zeitlichen und teils datenschutzrechtlichen Gründen nicht möglich, die Daten aller Kunden kontinuierlich zu überwachen und zu beurteilen. Vielmehr sollte der DBA einer betreffenden Firma diese Aufgabe übernehmen. Bei Auffälligkeiten bspw. nach Updates könnte dann mit gezielten Fragen an CIS herangetreten werden. Sofern kein geschulter DBA vorhanden ist, sollten entsprechende Zuständigkeiten festgelegt werden. Besonders der Enterprise Manager ermöglicht hierzu einen einfachen Einstieg, bietet aber auch eine Vielzahl an möglichen Hinweisen, die in gemeinsamer Absprache umgesetzt werden könnten. Weiterführend könnte bspw. das AWR durch die Snapshoterstellung eine Zustandsbeurteilung vor und nach einem Update ermöglichen.

Oracle bietet seit Version 8i die Möglichkeit, bestimmte System- und Datenbankressourcen einzelnen Benutzern oder Benutzergruppen zuzuteilen. Hierbei ist bspw. eine Zuteilung maximaler CPU-Zeiten oder die Begrenzung des Parallelisierungsgrades bzw. der Anzahl gleichzeitiger aktiver Session möglich. Ein Einsatz derartiger Strategien könnte sich bei Kunden, bei denen einzelne Nutzer mit den unterschiedlichsten Anforderungen eingesetzt werden, als sinnvoll erweisen. So wäre es denkbar, einem Disponenten mehr Ressourcen zuzuteilen als bspw. einem Mitarbeiter, der vorwiegend für Personal-, Fahrzeug- oder Kundenverwaltung zuständig ist. Ein umfangreiches Rechtekonzept wurde bereits in C-Logistic integriert. Auf Grundlage dieser Rechtevergabe könnte es sich bspw. als sinnvoll erweisen, Einteilungen vorzunehmen, um zusätzliche Ressourcen für andere Bereiche zu gewinnen. Im Gegenzug ist es sicherlich sinnvoll, außerhalb der Kernarbeitszeiten bspw. Analyse- oder Wartungsvorgängen mehr Ressourcen zuzugestehen.

3. Anwendung ausgewählter Optimierungsansätze

Bisher wurden verschiedenste performancerelevante Einflussgrößen untersucht bzw. dargestellt. Folgend sollen einige der Optimierungsansätze auf C-Logistic angewandt werden, um zu ermitteln, ob diese nachhaltig die Gesamtperformance beeinflussen können. Zu diesem Zweck kommen die Programme AQTime, DBMonitor und Golden32 zum Einsatz. Hierbei ermöglicht es der DBMonitor, den kompletten SQL-Verkehr zwischen C-Logistic und Oracle zu überwachen und somit zeitintensive SQL-Abfragen zu identifizieren. Folgend ist es für ein erfolgreiches SQL-Tuning notwendig, den Ressourcenverbrauch der SQL-Anweisungen bspw. über den Ausführungsplan in Golden32 zu ermitteln. Hieraus lassen sich dann folgend Tuningmaßnahmen ableiten und deren Auswirkungen messen. Neben der genannten Variante können selbstverständlich auch die verfügbaren Oraclestatistiken zu Rate gezogen werden.

Zur manuellen bzw. umfassenden SQL-Schwachstellenidentifikation können die Views V$SQLAREA, V$SQL und V$SQLTEXT genutzt werden, die die Statements des Library Cache enthalten. Erläuternd sei hier erwähnt, dass der Shared Pool in der SGA den Library Cache und den Data Dictionary Cache beinhaltet. Diese enthalten die Strukturinformationen zuletzt geladener Elemente und Anweisungen. Im View V$SQLAREA ist neben dem Anweisungstext bspw. die Anzahl der Ausführungen, Sortiervorgänge, CPU- und Wartezeiten sowie Puffer- und Plattenzugriffe ersichtlich. Der View V$SQLAREA enthält im Attribut SQL_TEXT, die auf 1.000 Zeichen beschränkte SQL-Anweisung. Zusätzlich existiert noch das Attribut SQL_FULLTEXT, das die vollständige SQL-Anweisung enthält. Mit Hilfe der SQL_ID könnte aber auch bspw. ein Verbund zwischen den Views V$SQLAREA und V$SQLTEXT aufgebaut werden, wodurch ebenfalls der komplette SQL-Text erstellt werden könnte. Im Folgenden sollen die Sendungserfassung und die Disposition näher untersucht werden. Während bei Betrachtung der Sendungserfassung ebenfalls auf Punkte eingegangen wird, die bereits erfolgreich umgesetzt wurden, werden folgend in der Disposition nur noch Optimierungsansätze erläutert.

3.1 Sendungserfassung

Bereits das Öffnen der Sendungserfassung dauert bei einigen Kunden, die über ein sehr hohes Auftragsvolumen verfügen, einige Zeit. Von C-Logistic ist vorgesehen,

dass die Anzahl der angezeigten Datensätze über einen Vorfilterdialog eingeschränkt wird. Dennoch ist bekannt, dass einige Anwender diesen Filter zu Recherchezwecken prinzipiell auf alle Datensätze stellen, sodass diesem Zustand auf kurz oder lang nur durch die Bereitstellung einer Archivierungslösung entgegenzuwirken ist. Das Öffnen der Sendungserfassung mit ca. 2.000 Sendungen dauert bei CIS nur wenige Sekunden. Fest steht hierbei jedoch auch, dass dies nicht einmal dem üblichen täglichen Auftragsvolumen einiger C-Logistic-Kunden entspricht, sodass eine Bewertung an dieser Stelle unmöglich ist.

Bei Betrachtung mit dem DBMonitor fällt auf, dass die Anzahl der ausgeführten SQL-Anweisungen beim Öffnen der Sendungserfassung als überschaubar einzustufen ist und wiederholtes Ausführen gleicher SQL-Anweisungen nicht stattfindet. Bei den SQL-Anweisungen ist insgesamt zu erkennen, dass umfangreiche Abfragen bereits optimiert wurden und standardgemäß die Zugriffe über Indizes erfolgen. Im Zusammenhang mit kleineren bis teilweise sehr kleinen Relationen wie NIEDERLASSUNGEN, GBEREICH und FORMULAR sind Table Access Full-Situationen erkennbar. Hier wäre das Anlegen eines Index denkbar, jedoch ist dies bei kleinen Relationen weniger Erfolg versprechend. Beim Öffnen der Sendungsmaske werden wie bei anderen Erfassungsmasken verschiedene Rechte geprüft. Dazu wird ein Verbund zwischen den Relationen FORMULAR und FORMULARRECHT erzeugt. Dieser Verbund erfolgt, wie empfohlen, von der kleinen Relation FORMULAR zur größeren Relation FORMULARRECHT, wobei der Zugriff auf die Relation FORMULARRECHT über das Primärschlüsselfeld umgesetzt wird. Gleiches gilt für die Relationen IOSCHEMA und IOSCHEMARECHT. Die Relation IOSCHEMA umfasst im Gegensatz jedoch bei CIS ca. 100 Datensätze und 74 Attribute, sodass das Anlegen eines funktionsbasierten Indizes auf das Attribut MENUFORMULAR empfehlenswert wäre, da ebenfalls zahlreiche weitere Erfassungsmasken diesen Verbund nutzen. Zum Vergleich des CPU- und I/O-Bedarfs beider Umsetzungsvarianten sollen die in der folgenden Abbildung dargestellten Ausführungspläne dienen. Der Zeitbedarf beträgt hierbei 62 ms im Vergleich zu 15 ms.

```
--Ohne funktionsbasierten Index
```

#	ID	PID	Operation	Name	Rows	Bytes	Cost	CPU Cost	IO Cost
1	0		SELECT STATEMENT		2	146	7	22M	6
2	1	0	SORT ORDER BY		2	146	7	22M	6
3	2	1	NESTED LOOPS		2	146	6	194068	6
4	3	2	TABLE ACCESS FULL	IOSCHEMA	2	116	6	190268	6
5	4	2	INDEX UNIQUE SCAN	PK_IOSCHEMARECHT_ID	1	15	0	1900	0

```
--Mit funktionsbasierten Index
```

#	ID	PID	Operation	Name	Rows	Bytes	Cost	CPU Cost	IO Cost
1	0		SELECT STATEMENT		1	73	3	21M	2
2	1	0	SORT ORDER BY		1	73	3	21M	2
3	2	1	NESTED LOOPS		1	73	2	23694	2
4	3	2	INLIST ITERATOR						
5	4	3	TABLE ACCESS BY INDEX ROWID	IOSCHEMA	1	58	2	21794	2
6	5	4	INDEX RANGE SCAN	IOSCHEMA_UPPER_MENU	1		1	14443	1
7	6	2	INDEX UNIQUE SCAN	PK_IOSCHEMARECHT_ID	1	15	0	1900	0

Abbildung 13: Auswirkung funktionsbasierter Index

Als einzige wirklich auffällige bzw. zeitaufwändige Anweisung kann die in der folgenden Abbildung dargestellte Hauptabfrage identifiziert werden.

```
SELECT * FROM (
  SELECT SENDUNG.*,
        (SELECT MIN(NA.AKTIONSDATUM) FROM NACHRICHT NA WHERE NA.SENDUNG_ID=SENDUNG.SENDNR AND NA.NACHRICHTTYP_ID=:LADENACHRICHTTYP)   AS LADE_ISTDATUM,
        (SELECT MAX(NA.AKTIONSDATUM) FROM NACHRICHT NA WHERE NA.SENDUNG_ID=SENDUNG.SENDNR AND NA.NACHRICHTTYP_ID=:LIEFERNACHRICHTTYP) AS LIEFER_ISTDATUM
  FROM SENDUNG
)
WHERE
      (1=1)
  AND (SENDART = 1) /*SendungsartFilter*/
```

#	ID	PID	Operation	Name	Rows	Bytes	Cost	CPU Cost	IO Cost
1	0		SELECT STATEMENT		2073	1380K	96	15M	95
2	1	0	SORT AGGREGATE		1	17			
3	2	1	TABLE ACCESS BY INDEX ROWID	NACHRICHT	1	17	2	51818	2
4	3	2	BITMAP CONVERSION TO ROWIDS						
5	4	3	BITMAP AND						
6	5	4	BITMAP CONVERSION FROM ROWIDS						
7	6	5	INDEX RANGE SCAN	IDX_NACHRICHT_SENDUNG_ID	6		1	9171	1
8	7	4	BITMAP CONVERSION FROM ROWIDS						
9	8	7	INDEX RANGE SCAN	IDX_NACHRICHTTYP_ID	6		1	25371	1
10	9	0	SORT AGGREGATE		1	17			
11	10	9	TABLE ACCESS BY INDEX ROWID	NACHRICHT	1	17	2	51818	2
12	11	10	BITMAP CONVERSION TO ROWIDS						
13	12	11	BITMAP AND						
14	13	12	BITMAP CONVERSION FROM ROWIDS						
15	14	13	INDEX RANGE SCAN	IDX_NACHRICHT_SENDUNG_ID	6		1	9171	1
16	15	12	BITMAP CONVERSION FROM ROWIDS						
17	16	15	INDEX RANGE SCAN	IDX_NACHRICHTTYP_ID	6		1	25371	1
18	17	0	TABLE ACCESS FULL	SENDUNG	2073	1380K	96	15M	95

Abbildung 14: Abfrage beim Öffnen der Sendungsmaske

In der dargestellten Anweisung ist erkennbar, dass der Zugriff zwischen den Relationen NACHRICHT und SENDUNG über verschiedene Indizes umgesetzt wird. Abschließend wird jedoch prinzipiell ein in dieser Anweisung unvermeidbarer Full Table Scan auf die ca. 280 Attribute umfassende Relation SENDUNG ausgeführt. Als Ausweg könnte hier einerseits eine Strukturanpassung und andererseits (sofern möglich) die Einschränkung der Feldliste angefügt werden. Eine versuchsweise Einschränkung der Feldliste auf 100 Attribute entspricht hierbei einem Zeitaufwand von maximal 0,4 Sek., währenddessen die Ausgangsanweisung bis zu 1,8 Sek. benötigt. Eine ähnliche Aussage lässt sich auch für die Datensatzanzahl treffen, bei der die versuchsweise Einschränkung über ROWNUM < 500 einem Zeitaufwand von nur noch 0,3 Sekunden bei vollständiger Attributanzahl entspricht. Strukturseitig sollte an dieser Stelle erwähnt werden, dass es sich bspw. bei den Relationen SENDUNG und

SENDPOS durchaus als sinnvoll erweisen kann, diese in einem separaten Tablespace[32] zu positionieren oder eine Partitionierung auf unterschiedliche Festplatten durchzuführen [ANB06]. Eine versuchsweise Verlagerung der kompletten Sendungs- und Sendungspositionsdaten führt bei einem Verbund beider Relationen aufgrund mangelnder Analysedaten anfänglich zu unterschiedlichen Ausführungsplänen. Nach Ausführung von Analyse- und Reorganisationsläufen ist trotz gleicher Ausführungspläne bei der Abfrage von ca. 11500 Datensätzen, ein Zeitaufwand von ca. 5,5 Sek. gegenüber 18,5 Sek. erkennbar. Sodass für diesen Ansatz im vorliegenden Fall von einer deutlichen Performancesteigerung gesprochen werden kann.

Bei Betrachtung des Speichervorgangs fällt auf, dass laut DBMonitor die Anweisung SELECT * FROM PREISE ORDER BY KENNUNG bis zu 10mal hintereinander ausgeführt wird, was bspw. aus Refreshvorgängen resultieren könnte. Obwohl diese Anfrage voraussichtlich keine physikalischen Zugriffe nach sich zieht, sollte an dieser Stelle nach Möglichkeiten gesucht werden, die Anzahl der logischen Zugriffe zu verringern. Auffällig ist weiterhin, dass in der Sendungserfassung bei dynamischen Anweisungen, die mehrfach ausgeführt werden müssen, bewusst ein Prepare vorangesetzt wird, wodurch die wiederholte Ausführung unterstützt werden kann. Weiterhin sind aus SQL-Sicht nur wenige Auffälligkeiten erkennbar bzw. wurden die ausgeführten Anweisungen größtenteils bereits hinreichend optimiert. Dennoch ist im Zusammenhang mit der Ermittlung von Versicherungswerten ein zusätzlicher Einsatz von Indizes in der Relation LAENDER auf das Attribut KFZ_KENNZ und in der Relation VERSGUELTIG auf das Attribut GUELTIG_VON empfehlenswert. Rechtfertigen lassen sich diese Empfehlungen hierbei mit der Tatsache, dass unter Beachtung der Größe der betreffenden Relationen durch diese Indizes häufige Full Table Scans bei Kunden mit einem hohen Auftragsvolumen umgangen werden können.

Zur Darstellung von bspw. Gefahrgutanforderungen wird die Funktion „GetSendungsymbolsuche“ in der Sendungserfassung aufgerufen. Beim Durchlaufen der Datensätze erscheinen zahllose SQL-Anforderungen im DBMonitor, die sich lediglich in der gewünschten Sendungsnummer unterscheiden. Hier ist zum einen notwendig nach einem Ausweg zur Senkung der Anfragehäufigkeit zu suchen und zum anderen könnte diese Funktion ggf. vollständig ins Oracle ausgelagert werden.

[32] Tablespace bezeichnet hier eine logische Einheit

Neben dem Öffnen der Sendungsmaske ist vor allem das Speichern einer Sendung oder einer Sendungsposition mit einem spürbaren Zeitaufwand verbunden. Im Folgenden sollen die Ursachen hierfür erörtert werden. Ein erster Anhaltspunkt, wenn es darum geht, Schwachstellen im Zusammenhang mit langsamen INSERT-, UPDATE- oder DELETE-Anweisungen aufzudecken, ist die Ermittlung der auf eine Relation angelegten Indizes. Mit Hilfe des bereits erwähnten Views V$OBJECT_USAGE und der Relation ALL_TAB_COLUMNS kann über eine SQL-Anweisung die Attributanzahl und zumindest jeder protokollierte Index ermittelt werden, wie in der folgenden Abbildung dargestellt.

```
SELECT TABLE_NAME, SUM(ANZ_COL) ATTRIBUTE, SUM(ANZ_IND) INDIZES,
       SUM(GENUTZT) GENUTZT, SUM(UNGENUTZT) UNGENUTZT FROM
(
  SELECT TABLE_NAME, COUNT(*) ANZ_COL, TO_NUMBER(0) ANZ_IND,
         TO_NUMBER(0) GENUTZT, TO_NUMBER(0) UNGENUTZT
  FROM ALL_TAB_COLUMNS
  WHERE
       (OWNER=USER)
  GROUP BY TABLE_NAME
  UNION
  SELECT TABLE_NAME, TO_NUMBER(0) ANZ_COL, COUNT(*) ANZ_IND,
         SUM(DECODE(USED, 'YES', 1, 0)) GENUTZT, SUM(DECODE(USED, 'NO', 1, 0)) UNGENUTZT
  FROM V$OBJECT_USAGE
  GROUP BY TABLE_NAME
)
GROUP BY TABLE_NAME
ORDER BY INDIZES DESC, ATTRIBUTE DESC
```

#	TABLE_NAME	ATTRIBUTE	INDIZES	GENUTZT	UNGENUTZT
1	SENDPOS	374	48	47	1
2	SENDUNG	274	15	15	0
3	VORGANG	96	13	13	0
4	LBUCHUNG	34	13	13	0
5	NACHRICHT	41	12	11	1
6	TOUREN	133	10	10	0
7	STAMMSENDUNG	293	9	7	2

Abbildung 15: Auflistung überwachter Indizes

In Abbildung 15 ist zu erkennen, dass die Relation SENDPOS ca. 380 Attribute umfasst und in dieser Relation die meisten überwachten Indizes angelegt wurden. Bei fast 50 Indizes ist davon auszugehen, dass INSERT-, UPDATE- oder DELETE-Anweisungen spürbar verlangsamt werden. Wenn man sich die konkreten Aufzeichnungsergebnisse anschaut, ist lediglich erkennbar, dass der Index IDX_SENDPOS_ESNR innerhalb der letzten 3 Monate nicht genutzt wurde. Somit könnte dieser evtl. entfernt werden, wodurch jedoch bei weitem noch keine spürbare Beschleunigung erwartet werden kann.

Das Einfügen eines neuen Datensatzes über ein INSERT-Skript aus einem externen Programm dauert für die Relation SENDUNG ca. 160 ms. Bezug nehmend auf den vorhergehend durchgeführten Komponentenvergleich entspricht dies somit vergleichsweise der Zeit, die eine ODAC-Komponente benötigte, um erste Datensätze der nach dem Ort sortierten Kundenrelation bereitzustellen. Das Einfügen eines neuen Datensatzes in die Relation SENDPOS benötigt im Vergleich ca. 300 ms bei der

3fachen Anzahl Indizes, wenn eine Vielzahl der vorhandenen Indizes vom Einfügevorgang betroffen sind. Zum besseren Vergleich soll an dieser Stelle noch die Relation STAMMPOS angeführt werden. Diese verfügt über die gleiche Attributanzahl und sehr ähnliche Attribute. Auf der Relation STAMMPOS sind jedoch im Gegensatz zur Relation SENDPOS nur 2 Indizes angelegt. Das Einfügen eines Datensatzes dauert bei dieser Relation bis zu 30 ms, sodass in diesem Fall von einer deutlichen Einsparung aus Sicht des Zeitbedarfs gesprochen werden kann. Somit ist davon auszugehen, dass in diesem Fall die Dauer für Einfügeoperationen nicht zwangsläufig von den Triggern abhängig ist, die sich auf beiden Relationen befinden, sondern vielmehr auf die Anzahl der Indizes zurückzuführen ist. Gleiches konnte testweise für beide Relationen bei Änderungs- und Löschoperationen festgestellt werden. Der Grund für den Zeitaufwand bspw. beim Anlegen einer Sendungsposition könnte weiterhin evtl. im Zusammenspiel der Relationen SENDUNG und SENDPOS auffindbar sein.

Beim Speichern einer Sendung oder Sendungsposition werden verschiedenste Prüfungen durchgeführt. So findet bspw. beim Speichern eines Datensatzes prinzipiell die Feiertagsermittlung statt, deren Verbund ebenfalls von der kleineren zur größeren Relation aufgebaut ist. Bei dieser Funktion wird ebenfalls empfehlenswerterweise mit lokalen Daten im Datenmodul gearbeitet, sodass die Serverlast erfolgreich gesenkt werden kann. Weiterhin wäre bspw. die Erzeugung eines Clusters bei häufig miteinander verknüpften Relationen denkbar. In Frage hierfür kämen ggf. die Relationen zur Feiertags-, Zonen- und Relationsermittlung, da diese fast ausschließlich im Verbund verwendet werden. Ein versuchsweiser Einsatz eines Clusters bei der Zonenermittlung ergab keine spürbaren Zeitvorteile, dennoch ist eine Senkung des I/O-Bedarfs wie folgend dargestellt erkennbar.

Operation	Name	Rows	Bytes	Cost	CPU Cost	IO Cost
SELECT STATEMENT		4992	789K	33	13M	32
CONCATENATION						
MERGE JOIN CARTESIAN		4809	760K	20	2062955	20
NESTED LOOPS		3	354	5	48699	5
TABLE ACCESS FULL	ZONEOUT	3	147	2	22354	2
TABLE ACCESS BY INDEX ROWID	ZONE	1	69	1	8781	1
INDEX UNIQUE SCAN	IDX_ZONE_KENNUNG	1		0	1050	0
BUFFER SORT		1603	68K	19	2054174	19
TABLE ACCESS FULL	ZONEINC	1603	68K	5	671419	5
HASH JOIN		183	28K	13	11M	12
NESTED LOOPS		1	118	6	122452	6
TABLE ACCESS FULL	ZONEOUT	3	147	2	22354	2
TABLE ACCESS FULL	ZONE	1	69	1	33366	1
TABLE ACCESS FULL	ZONEINC	1222	52K	6	642519	6

Operation	Name	Rows	Bytes	Cost	CPU Cost	IO Cost
SELECT STATEMENT		4810	2104K	12	12M	11
CONCATENATION						
MERGE JOIN CARTESIAN		4809	2103K	7	11M	6
HASH JOIN		3	1071	5	10M	4
TABLE ACCESS FULL	ZONEOUT	3	273	2	18381	2
TABLE ACCESS FULL	ZONE	8	2128	2	21621	2
BUFFER SORT		1603	142K	5	11M	4
TABLE ACCESS FULL	ZONEINC	1603	142K	1	338381	1
NESTED LOOPS		1	448	5	94807	5
NESTED LOOPS		1	357	4	87686	4
TABLE ACCESS FULL	ZONEOUT	3	273	2	18381	2
TABLE ACCESS FULL	ZONE	1	266	1	23101	1
TABLE ACCESS CLUSTER	ZONEINC	1	91	1	7121	1

Abbildung 16: Auswirkungen Clustering

In der Sendungserfassung existiert ein Auswertungsregister. In der zugehörigen Abfrage werden eine Vielzahl der Attribute der Relationen SENDUNG, KUNDEN,

NIEDERLASSUNGEN und FRANKATUR über Right Outer Joins miteinander verknüpft. Die Bereitstellung der Daten dauert mit den ca. 2.000 verfügbaren Sendungen fast 7 Sekunden. Die versuchsweise Ersetzung der gesamten Auswertung durch einen materialisierten ergab hierbei keine Verbesserung. Die Aktualisierung des Views war in einem vertretbaren zeitlichen Rahmen nicht möglich, sodass davon ausgegangen werden kann, dass der Einsatz eines materialisierten Views in diesem Fall wenig sinnvoll ist. Im Gegenzug konnte beim Einsatz eines materialisierten Views für die Summenzeilen eine Zeiteinsparung von bis zu 80 % ermittelt werden. Hierbei ist jedoch eindeutig abzuwägen, ob dieser mit den sich ständig ändernden Daten der Sendungserfassung vereinbar ist. C-Logistic bietet die Möglichkeit, die Anzahl der sichtbaren Felder einzuschränken. In der eingangs erwähnten Hauptabfrage der Sendungsmaske konnte eine Senkung des Zeitbedarfs durch die Einschränkung der ausgewählten Attribute erreicht werden. An diese Stelle kann im Gegensatz erfolgreich nachgewiesen werden, dass eine Einschränkung der Sichtbarkeit der Felder keine spürbare Verbesserung nach sich zieht, sondern vielmehr dem Komfort des Anwenders dient.

Nachdem vorhergehend einige Punkte in der Sendungserfassung aus SQL-Sicht untersucht wurden, sollen abschließend zu diesem Gliederungspunkt aus Sicht von AQTime die performanceseitigen Schwachpunkte aufgezeigt werden. Im Gegensatz zur bisherigen Betrachtung lässt AQTime keine unmittelbare Beurteilung der ausgeführten SQL-Anweisungen zu, wodurch sich folgend der Betrachtungsschwerpunkt zwangsläufig verlagert. Rein aus der Überwachung des Öffnungsvorgangs und dem Anlegen einer Sendung bzw. Sendungsposition lassen sich nur wenige maskenspezifische Schwachpunkte erkennen. Als auffällig ist aber die Ermittlung der berechneten Felder der Sendungspositionen zu bezeichnen. Schwerpunkt stellt hierbei wahrscheinlich weniger der Umfang der Berechnungen als vielmehr die Zugriffsart auf die Feldinhalte dar. Die Sendungserfassung beruht auf einer umfangreichen Vererbungshierarchie. In diesem Zusammenhang wurden verschiedene Funktionalitäten gesondert ausgelagert. Im vorliegenden Fall bedeutet dies, dass die Berechnung der verschiedenen Sendungspositionsfelder in eine eigene Unit ausgelagert wurde und der Zugriff zwangläufig über „FieldByName“ stattfindet muss. AQTime identifiziert in diesem Fall folglich die betreffenden Get- und Set-Operationen als aufwändigste Vorgänge. Umfangreichere Aufzeichnungen identifizieren weiterhin bspw. die Aus-

wahldialoge für Material und Auftragsart als Schwachpunkte, sodass an dieser Stelle ggf. eine erneute Prüfung der mit den Dialogen verbundenen SQL-Abfragen interessant wäre.

3.2 Disposition

Im Vergleich zur Sendungserfassung werden beim Öffnen der Disposition bedeutend mehr SQL-Anweisungen ausgeführt. Aus Sicht der Einsparung zusätzlicher Sortieranforderungen sollte bspw. die folgende Anweisung angepasst werden.

```
SELECT KENNUNG,BEZEICHNUNG,KENNZEICHEN,NUTZLAST,FZG_GRP,DISPO_GRP,FAHRZEUGART,
       SPEDITION,NL,SENDART,WERKST_NDATUM,LEERGEWICHT,LADEVOLUMEN,LADEMETER,
       STELLPLAETZE,FAHRER,BEIFAHRER,TELEFON,DGESCHW,PRIORITAET,1 AS VISIBLE
FROM FAHRZEUG
WHERE
      (AKTIVFLAG=1)
  AND (DISPO_GRP IS NOT NULL)
  AND (FAHRZEUGART IN (0,1,2,3,4,20,30,40,50))
UNION
SELECT KENNUNG,BEZEICHNUNG,KENNZEICHEN,NUTZLAST,FZG_GRP,DISPO_GRP,FAHRZEUGART,
       SPEDITION,NL,SENDART,WERKST_NDATUM,LEERGEWICHT,LADEVOLUMEN,LADEMETER,
       STELLPLAETZE,FAHRER,BEIFAHRER,TELEFON,DGESCHW,PRIORITAET,0 AS VISIBLE
FROM FAHRZEUG
WHERE
      (AKTIVFLAG=1)
  AND ((DISPO_GRP IS NULL) OR (FAHRZEUGART NOT IN (0,1,2,3,4,20,30,40,50)))
```

Abbildung 17: UNION-Anweisung in der Disposition

Die in Abbildung 17 dargestellte Anweisung wird beim Öffnen der Disposition dynamisch erzeugt und mehrfach abgewandelt eingesetzt. Aufgrund der jeweils hinreichend differenzierenden Bedingungen ist hierbei jedoch auch die Verwendung von UNION ALL denkbar, wodurch oracleseitig die Sortierung eingespart werden kann. Das komplette Ersetzen von Union ist hierbei jedoch weniger Erfolg versprechend, da bspw. über DECODE-Anweisungen weiterhin sichergestellt werden muss, dass zwischen sichtbaren und unsichtbaren Fahrzeugen unterschieden werden kann. Der Einsatz zusätzlicher Indizes ist an dieser Stelle ebenfalls wenig Erfolg versprechend.

Obwohl die Vielzahl der SQL-Anweisungen bereits auf die Verwendung mit Indizes optimiert wurde, ist weiterhin auffällig, dass verschiedene Anweisungen laut DBMonitor mit gleichen Parametern nacheinander wiederholt ausgeführt werden. Somit könnte auch hier als Ansatz zur Optimierung die Senkung logischer Zugriffe bzw. eine zunehmende Funktions- bzw. Prozedurauslagerung angesehen werden.

Laut AQTime wird vor allem die Initialisierung der Disposition als eine der aufwändigsten Operation identifiziert. Hierbei werden bspw. Zugriffe auf die umfangreichen Relationen SENDPOS, TOUREN und FAHRZEUG ausgeführt sowie Kartendaten geladen. In diesem Zusammenhang fällt vor allem die Funktion „DoMnuChoose-

TourFahrzeug“ mit AQTime auf. Weiterführend fallen innerhalb dieser die Funktionen „Add1“, „CreateTourFromSendPosNrsAndFahrzeug“ und „ShowTourDlg“ auf. Somit lassen sich vor allem in der Disposition mit Hilfe von AQTime Schwerpunkte erkennen, deren Behebung aufgrund der enormen Funktionsvielfalt eher die Aufgabe des zuständigen Mitarbeiters darstellen sollte.

Wie in der Sendungserfassung fällt auch in der Disposition die Vielzahl der Ermittlungen bzw. Prüfungen auf. Aus fachlicher Sicht könnte es sich ggf. durchaus als sinnvoll erweisen, deren Notwendigkeit bzw. Häufigkeit gezielt in Frage zu stellen.

3.3 Kosten-Nutzen-Zusammenfassung

Aus Sicht des Kosten- und Zeitaufwands stellt eine nachträgliche Optimierung immer die schlechteste Variante dar, wie es mit der folgenden Abbildung aus [ANB06] verdeutlicht werden soll.

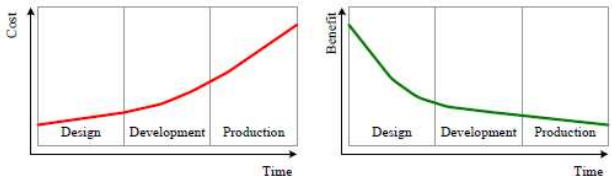


Abbildung 18: Kosten-Nutzen-Kurven bei der Optimierung

Aus Abbildung 18 lässt sich erkennen, dass es eines der Hauptziele sein sollte, den gesamten Entwicklungsprozess auf eine konsequente Optimierung auszurichten, um somit Kosten zu sparen. In diesem Zusammenhang sollte einer der wesentlichsten Grundsätze bereits in frühen Entwicklungsphasen eine möglichst ressourcenschonende und vorausschauende Entwicklung sein. Notwendige nachträgliche Anpassungen am Datenmodell stellen bspw. in diesem Zusammenhang die kostenintensivsten Tätigkeiten dar.

Insgesamt konnten keine zwangläufig gravierenden Schwachstellen ermittelt werden, die einen sofortigen Investitionsbedarf nach sich ziehen. So kann sich selbst eine bedeutend jüngere Entwicklungsumgebung nicht grundlegend von Delphi 5 absetzen. Die vorwiegend verwendeten ODAC-Komponenten erreichen im Vergleich zu anderen Entwicklungskomponenten die besten Werte. Oracle wird fast ausschließlich auf Windowsplattformen eingesetzt, was vor allem bei hoher Nutzerzahl deutliche Vorteile mit sich bringen sollte. Handlungsbedarf ist insgesamt in einigen der hier zusammengetragenen Oracleoptimierungsansätze zu finden. Hier ist es vorerst not-

wendig, in eigenen Versuchen verschiedene Ansätze umzusetzen, bevor diese beim Kunden ebenfalls realisiert werden sollten. Dennoch ist davon auszugehen, dass einige der Optimierungen spürbare Verbesserungen mit sich bringen können, wie es an einigen Stellen bereits verdeutlicht wurde. Handlungsbedarf ist ebenfalls zur Senkung der Anzahl durchgeführter logischer Zugriffe in einigen Erfassungsmasken festzustellen. Hierbei ist aber zu beachten, dass daraus nicht zwangläufig ein deutlicher Performancegewinn zu erwarten ist. Aus Kostensicht ist somit hier weniger Investitionsbedarf im Sinne einer Neuanschaffung, sondern vielmehr Kosteneinsatz aus Sicht eigener Personalressourcen anzuführen. Bezug nehmend auf die verwendeten SQL-Anweisungen hält sich der Handlungsbedarf in Grenzen, da bspw. bereits der Großteil der Anweisungen in den betrachteten Bereichen unter Verwendung von angemessenen Indizes ausgeführt wird.

Aus Kundensicht sind Probleme eher im Zusammenhang mit teils veralteten Oracleversionen oder in für das eigene Auftragsvolumen teils unzureichenden Hardwareressourcen zu finden. Besonders das Auftragsvolumen, das bei einigen Anwendern über mehrere Jahre zusammengelaufen ist, sollte auf absehbare Zeit durch die Bereitstellung einer adäquaten Archivierungslösung effektiv gesenkt werden. Die vorerst größte Verbesserung stellt die Archivierung von Sendungs- und Dispositionsdaten dar, dennoch könnten nachträglich weitere Archivierungsansätze ermittelt werden. Erste hierfür nötige Grundlagen sollen im abschließenden Teil dieser Diplomarbeit erörtert werden. Die möglichen zu erwartenden Zeiteinsparungen wurden im Kapitel 2.4 angeführt, sodass sich daraus eine Kosten-Nutzen-Betrachtung erstellen lässt. Anhand der für CIS gültigen Werte könnte die in der folgenden Tabelle abgebildete Rechnung aufgestellt werden.

Position	**Werte**	**Bemerkung**
Anschaffungskosten	5600 €	Kosten für die Entwicklung (ca. 7 MT a 800 €)
Installationskosten	400 €	Kosten für die Installation (ca. 0,5 MT a 800 €)
Jährliche Betriebskosten	600 €	Kosten Durchführung bspw. Personal (2 h pro Monat a 50 €)
Gesamtkosten	6600 €	Kosten im 1. Jahr
Folgekosten	600 €	Jährliche Betriebskosten
Jahresnutzen	1575 €	Ermittelt für 10 Mitarbeiter bei 250 Arbeitstagen im Jahr
Hochrechnung (5 Jahre)	-1125 €	5 Jahre: 7875 € (Nutzen) – 9000 € (Anschaffung und Betrieb)
Hochrechnung (7 Jahre)	825 €	7 Jahre: 11025 € (Nutzen) – 10200 € (Anschaffung und Betrieb)

Tabelle 7: Kosten-Nutzen der Archivierung aus Kundensicht bei Projektauftrag

Grundlage für die Berechnung des Nutzens in Tabelle 7 stellen folgende Überlegungen dar:

- Senkung Auftragsvolumen um 75 % entspricht einer Zeiteinsparung von 85 % (bspw.: beim Öffnen der Sendungsmaske)
- Annahme: ein Mitarbeiter gewinnt durch die Archivierung pro Tag 1:30 Min. Zeit (entspricht: bei 200 € Personalkosten täglich 0,63 €)

Die Alternative zu den Betrachtungen stellt die Beibehaltung der bisherigen Umstände dar, wodurch die zu erwartende Ersparnis als Kosten bezeichnet werden könnte. Weiterhin könnte beachtet werden, dass allgemein durch das gesenkte Auftragsvolumen Zeit für andere Prozesse gewonnen wird und die Höhe der Entwicklungskosten nicht zwangsläufig vollständig vom Kunden zu tragen ist. Erkennen lässt sich hierbei insgesamt, dass die Betriebskosten bereits im ersten Jahr vollständig für das 2. Jahr durch den Nutzen abgedeckt werden können. Bereits im 7. Jahr könnte sich die Anschaffung der Archivierung für den Kunden vollständig rentieren.

Einen ähnlich überzeugenden Eindruck, wie die mögliche Einführung einer Archivierungslösung, hinterließ der Einsatz materialisierter Views sowie die Auslagerung der Sendungsdaten in einen eigenen Tablespace. Mit Hilfe der Archivumgebung könnten folgend Untersuchungen durchgeführt werden, in denen die Performance der Sendungsdaten weiter beurteilt werden könnte. Sofern sich die, in dieser Diplomarbeit festgestellten Werte bestätigen lassen, könnte mit überschaubarem Kosteneinsatz durch Anpassung des folgend dargestellten Vorgehensmodels die Archivierung auch für die Verlagerung sämtlicher Sendungsdaten genutzt werden. Für die Einführung weiterer materialisierter Views ist es vorerst notwendig, weitere sinnvolle Ansätze zu finden. Dies könnte bspw. innerhalb einer weiteren Studentenarbeit umgesetzt werden. Der eigentliche Implementierungsaufwand ist auch hierbei als überschaubar einzustufen, da zum einen entsprechende SQL-Anweisungen bereits vorhanden sind und zum anderen Abfragen meist nur in Datenmodulen umgeleitet werden müssen.

Auf einen längeren Zeitraum bezogen könnte die Anschaffung eines Performanceprofilers für CIS eine lohnende Investition darstellen. Im Falle von AQTime variiert hierbei der Preis zwischen 425 € für eine Einzelplatzlizenz und 1.349 € für eine Mehrnutzerlizenz. Besonders wenn sich die Durchführung teilautomatisierter Softwaretests etabliert hat, könnte die gleichzeitige Anwendung eines Performanceprofilers neben den Testergebnissen auch Informationen zu aufgedeckten Schwachstellen während der Testdurchführung bereitstellen.

3.5 Ausblick

Weniger intensiv betrachtet wurden bisher die konkreten Auswirkungen einiger der zusammengetragenen Oracle-Optimierungsansätze. Im Gegensatz bspw. zur Optimierung eines SQL-Statements sind vor allem die Auswirkungen der verschiedensten Oracleparameter meist erst über einen längeren Zeitraum bewertbar. Hierbei ist auch zu beachten, dass teilweise bspw. im Falle der Pctfree- und Pctused-Parameter keine konkrete Empfehlung abgegeben werden kann. Als ähnliches Beispiel könnte der Parameter Db_file_multiblock_read_count angeführt werden. Dieser ermöglicht es, mehr Oracle-Blöcke in einem Betriebsystem-I/O bereitzustellen, jedoch werden hierdurch Full Table Scans vom Optimizer stärker favorisiert, wodurch sich die Gesamtperformance insgesamt verschlechtern kann. Somit sind meist die optimalen Einstellungen vielmehr von den jeweiligen Benutzeranforderungen abhängig.

Trotz wiederholter Betrachtung der in der Studienarbeit beurteilten Lösungen zur Erstellung teilautomatisierter Softwaretests war es nicht möglich, eine Lösung zu finden, die in der Lage ist, einen Stresstest, wie er bspw. einen Einsatz beim Kunden simulieren könnte, durchzuführen. Besonders die Kombination aus einem umfangreichen Stresstest mit der Durchführung gezielter Oracleoptimierungen könnte eine lohnenswerte Grundlage für tiefgreifendere Untersuchungen darstellen. Hierbei sind die Simulationen bei CIS mit weniger „Unannehmlichkeiten“ verbunden, als es meist eine aktive Ermittlung von Werten beim Kunden bedeutet. So ist es bspw. in Oracle üblich, dass die Erstellung von Statistiken erfahrungsgemäß zusätzliche Ressourcen verbraucht, was bei verschiedenen Kunden nicht zwangsläufig auf Verständnis stößt oder zusätzliche Probleme verursachen kann. Aus dem Stresstest ermittelte Erfahrungen könnten im Gegensatz später gezielt in die Optimierung beim Kunden einfließen.

4. Archivierung

Es ist davon auszugehen, dass die Einführung einer geeigneten Archivierungslösung für C-Logistic eine deutliche Laufzeitverbesserung aufgrund des verringerten Datenvolumens nach sich ziehen wird. Das grundlegende Ziel im Zusammenhang mit der Einführung einer Archivierungslösung ist hierbei die Schaffung eines überschaubaren Produktivdatenbestandes und eines Archivdatenbestandes, der zu Recherchezwecken weiterhin schnelle Zugriffe erlaubt. Das Zugriffsprofil einer Archivumgebung zeichnet sich meist dadurch aus, dass Zugriffe weniger zeitkritisch sind, meist seltener erfolgen und im Wesentlichen lesend stattfinden. Im abschließenden Teil dieser Diplomarbeit sollen folgend einige Grundlagen zur Einführung einer Archivierungslösung erörtert werden.

4.1 Grundlegende Faktoren

Grundlegend sind von einer Archivierung vorerst Bewegungsdaten betroffen. Stammdaten werden meist kontinuierlich genutzt, sodass eine Archivierung an diesen Stellen weniger sinnvoll erscheint. Im konkreten Fall bedeutet dies, dass bspw. Sendungsdaten archiviert werden sollen. Weiterhin ist ebenfalls eine Archivierung von Touren denkbar, wodurch die Disposition entlastet werden kann. Die Dispositionsdaten verlangen hierbei nach einer alternativen Betrachtung. In der Sendungserfassung existiert eine Vielzahl an Verweisen, die im folgenden Kapitel verdeutlicht werden sollen. Folgend soll der Betrachtungsmittelpunkt vorerst nur auf die Sendungserfassung gelegt werden. Die Überlegungen für die Disposition könnten jedoch analog dazu erfolgen.

Die Vielzahl der möglichen Archivierungsansätze sieht Archivierung als einen Prozess, in dessen Verlauf die Archivdaten bspw. auf einen Tertiärspeicher ausgelagert werden. Der Zugriff auf diese Daten ist in diesem Fall prinzipiell mit einer Wiedereinführung der Daten in die Produktivumgebung verbunden. Die Gründe einer derartigen Vorgehensweise sind meist in begrenztem Speichervolumen oder in den Kosten der Datenerhaltung auf teuren Speichermedien zu finden. Einer der grundlegendsten resultierenden Problembereiche ist hierbei die Beachtung der Problematik unterschiedlicher C-Logistic-Programmversionen und deren zugehörigen Datenbankstruktur. So ist ein Update in C-Logistic meist prinzipiell mit Anpassungen der zugrundeliegenden Tabellenstruktur verbunden. Hierbei werden bspw. Attribute angefügt,

entfernt oder diverse Verknüpfungen und Prüfungen angelegt. Die Wiedereinführung des archivierten Datenbestands würde schlimmstenfalls unzählige Strukturanpassungen erfordern bzw. könnte sich nach einigen Jahren softwareseitig als unmöglich erweisen, sodass derartige Konzepte nicht verfolgt werden sollten.

Die Notwendigkeit der Archivierung lässt sich neben der Beherrschung des ständigen Anstiegs des Datenaufkommens durch folgende weitere Faktoren rechtfertigen:

- Unternehmensspezifische Regelungen (Bsp.: Recherchen)
- Interne Vereinbarungen zwischen Auftraggeber und Auftragnehmer
- Nachträgliche Ableitung bzw. Erstellung von Beschreibungen/Vorgängen
- Rechtsverbindliche Aufbewahrungsfristen wie zum Beispiel:
    - 6 Jahre für Dokumente wie Handelbriefe
    - 10 Jahre bei Buchungen nach Handels- und Steuerrecht bzw. Inventare, Jahres- und Konzernabschlüsse usw.
    - 30 Jahre für Daten, die umweltschutzgesetzliche Richtlinien betreffen

Rechtlich gesehen zählen Daten aus C-Logistic zu den elektronischen Dokumenten, die im Streitfall als Beweismittel zugelassen werden könnten, jedoch der freien richterlichen Beweiswürdigung unterliegen. Regelungen für Beweismittel in Papierform sollten dennoch beachtet werden, da die Daten aus C-Logistic ggf. ein wichtiges Bindeglied darstellen könnten. Auf absehbare Zeit könnte durch die Sicherstellung der Datenauthentizität, bspw. mit Hilfe elektronischer Signaturen, Softwareversiegelungen oder kryptografischer Zeitstempelverfahren die Beweiskraft elektronischer Dokumente auch in Deutschland einem Urkundenbeweis gleichgesetzt werden.

Eine grundlegende Herangehensweise zur Umsetzung einer Archivierung könnte bspw. die Datenreplikation über Snapshot’s darstellen. Hierbei handelt es sich um ein vollständiges, zu einem definierten Zeitpunkt erstelltes und schreibgeschütztes Abbild einer Datenbank. Es ist jedoch davon auszugehen, dass dieses Abbild ohne weitere Anpassungen immer nur mit einer bestimmten Programmversion von C-Logistic uneingeschränkt kompatibel ist. Problematisch erscheint hierbei die Tatsache, dass Fehler in einer bestimmten Programmversion entweder ständig bei Recherchen auftauchen oder mühsam nachträglich in entsprechende Programmversionen eingepflegt werden müssen. Besonders der Pflegeaufwand erscheint im Falle der Entwicklerversionen von C-Logistic in Anbetracht der Kundenvielzahl als nicht

beherrschbar, wodurch diese Lösung nicht umgesetzt werden sollte. Ebenfalls als problematisch sind bei dieser Umsetzung Faktoren wie der Transferaufwand, das benötigte Datenvolumen und die Handhabung von sich in Transaktionen befindlichen Daten einzustufen.

Aus Sicht der vollständigen Umgehung der Problematik Programmversion und Datenbankstruktur wäre auch eine Umsetzung denkbar, in der bestimmten Kriterien entsprechende Datensätze über ein zusätzliches Kennzeichen als archiviert gekennzeichnet werden[33]. Hiermit würde der Aufwand der Auslagerung vollständig entfallen und Recherchen könnten im größtmöglichen Umfang ermöglicht werden. Programmseitig müsste sichergestellt werden, dass eine nachträgliche Anzeige der Archivdaten im Produktivbestand und die Veränderung der Datensätze in der Archivumgebung unmöglich sind. Für den Aufruf der Archivumgebung wären ggf. ebenfalls umfangreiche Programmanpassungen notwendig. Besonders unpraktikabel erscheint diese Umsetzung unter dem Gesichtspunkt, dass das Ziel der Senkung der in einem Mandanten physisch vorhandenen Datenmengen nicht erreicht werden kann.

Eine weitere spontane Überlegung ist die gezielte bzw. „händische“ Umsetzung einer Archivierung, bei der nur Datensätze archiviert werden, die bestimmte Bedingungen erfüllen. So könnte bspw. ein zusätzlicher Tablespace angelegt werden, in den die Archivdaten verschoben werden. In diesem sollte es dann möglich sein, Strukturänderungen wie im Produktivmandanten durchzuführen, sodass für die Recherche immer die aktuelle Programmversion genutzt werden kann. Bei entsprechenden Aktualisierungen ist hierbei darauf zu achten, dass bspw. beim Einfügen neuer Bedingungen automatisch sichergestellt wird, dass diese auch erfüllt werden. Denkbar ist hierbei zur Erläuterung, dass ein zusätzlicher Check-Constraint eingeführt wird, bei dem bspw. zusätzlich beim Update sichergestellt werden muss, dass dieser auch im Archivmandanten ohne Nutzeranpassungen erfüllt werden kann. Besonders geeignet erscheint diese Überlegung auch aus Sicht des Gedankens einer Langzeitarchivierung. Migrationen und damit verbundene neue Datenmodelle, Schemata oder Formate könnten bei dieser Umsetzung analog zum Produktivmandanten durchgeführt werden, wodurch eine andauernde Erneuerung des Archivdatenbestandes er-

[33] Gängiges Vorgehen des strukturell-objektorientierten Datenbankmodels PRODAT

reicht werden kann. Für einen Updatevorgang sollte sich hierbei nach Möglichkeit kein deutlicher Mehraufwand im Vergleich zum derzeitigen Zustand ergeben.

4.2 Sendungsarchivierung

Rein konzeptionell gesehen besteht eine Archivierung aus den Schritten [HER97]:

1. Auswählen der zu archivierenden Daten (SELECT) und Löschen der Daten nach der Archivierung (DELETE)
2. Einfügen der Daten in die Archivumgebung (INSERT).

Bei der Auswahl der zu archivierenden Sendungen sind jedoch weitere grundlegende Dinge zu beachten. Zum einen sollte anhand der Datenbankstruktur festgelegt werden, welche Informationen mit archiviert werden müssen. Zum anderen muss über den Bearbeitungsstatus ermittelt werden, ob die Sendung überhaupt archiviert werden kann. Der Bearbeitungsstatus kann derzeit die in der folgenden Tabelle aufgelisteten Werte annehmen.

Bearbeitungsstatus	**Wert**
Anfrage	-1
Angebot	0
Auftrag	1
Vordisponiert	2
Disponiert	3
Unterwegs	4
Ausgeliefert	5
Schaden	6
Nachbearbeitet 1	7
Nachbearbeitet 2	8
Nachbearbeitet 3	9
Erlöse bewertet	10
Kosten bewertet	20
Erlöse und Kosten bewertet	30
Erlöse fakturiert	40
Kosten fakturiert	50
Erlöse und Kosten fakturiert	60

Tabelle 8: Möglicher Bearbeitungsstatus einer Sendung

Bei einem Bearbeitungsstatus 60 (Erlöse und Kosten fakturiert) kann eine Sendung sofort archiviert werden. Durch die Wahl abgeschlossener Sendungen können Probleme bspw. durch sich in Bearbeitung befindliche Sendungen vermieden werden. Für jeden anderen Bearbeitungsstatus gilt, dass die Sendung entweder nicht archiviert werden kann oder es zusätzlich nötig ist, die Frachtzahler- und Frachtführerkennzeichen zu betrachten um zu ermitteln, ob die Sendung ebenfalls archiviert werden kann.

Die der Sendungserfassung zugrunde liegende Datenbankstruktur soll in den folgenden Betrachtungen näher erläutert werden. Eine definierte Integritätsbeziehung besteht hierbei lediglich zwischen den Relationen SENDUNG und SENDPOS. Dennoch gibt es zahlreiche Attribute, die mit Primärschlüsselwerten anderer Relationen gefüllt sind, sodass bspw. derzeit mit deren Hilfe OnGetText-Ereignisse umgesetzt werden.

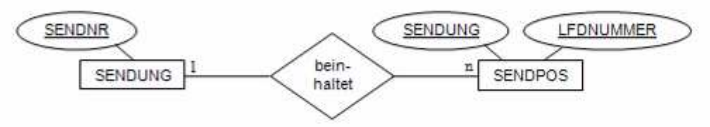


Abbildung 19: ERD[34] SENDUNG und SENDPOS

Attribut	Verweis auf Relation
AUSGBORDERO, EINGBORDERO	BORDERO
WECHSELBRUECKE1, WECHSELBRUECKE2, ANHNR, LKWNR, GERAET1...	FAHRZEUG
FZG_GRP	FZGRP
GBEREICH	GBEREICH
KOST	KOST
LADE_KENNUNG, LIEFER_KENNUNG, FZ_KENNUNG, F2_KENNUNG...	KUNDEN
WANR, WENR	LBEWEGUNG
LIEFERART	LIEFART
MATKENNUNG	MATERIAL/ LEISTUNG
NL	NIEDERLASSUNGEN
PALART1, PALART2, PALART3	PALETTEN
FAHRER, BEGLEITER	PERSONAL
FZ_FAKTPOSNR, F2_FAKTPOSNR, FF_FAKTPOSNR, ES_FAKTPOSNR...	POSITIONEN
FZ_PROJEKT, FZ_PROJEKTNR, F2_PROJEKT, F2_PROJEKTNR, FF_PROJEKT...	PROJEKTE
ABSRELATION, EMPFRELATION	RELATION
LAENDERROUTE, ROUTE, ROUTENKENNUNG	ROUTE
VP_TOUR	TOUREN
FZ_FAKTNR, F2_FAKTNR, FF_FAKTNR, ES_FAKTNR, VS_FAKTNR, L1_FAKTNR...	VORGANG
ENTFZONE	ZONE

Tabelle 9: Verweise auf andere Relationen

In Tabelle 9 wurden einige Verweise der Relationen SENDUNG und SENDPOS zusammengetragen. Auch hier sollte erwähnt werden, dass neben den Primärschlüsselwerten meist weitere Werte übernommen werden. So wird bspw. anhand der Ladekennung die vollständige Kundenadresse in die Sendungsposition übernommen. Besonders anhand dieser Zusammenhänge lässt sich ein Zwiespalt in Bezug auf die Umsetzung der Archivierung erkennen. Soll dem Kunden die Möglichkeit gegeben werden, möglichst aufschlussreich recherchieren zu können, sollten betreffende Datensätze der aufgeführten Relationen ebenfalls zur Verfügung gestellt werden. Problematisch erscheint hierbei die Behandlung nachträglich gelöschter, veränderter oder in Bearbeitung befindlicher Datensätze. Insgesamt würde unter Beachtung dieses Gesichtspunkts die Betrachtung deutlich komplexer ausfallen, sodass es sich als

[34] ERD – Entity Relationship Diagramm

sinnvoller erweisen könnte, die Archivierung etappenweise zu implementieren, um mögliche Probleme eher abfangen zu können.

4.2.1 Entwurf eines Vorgehensmodels

Grundlegende Überlegung stellt hierbei die Bereitstellung eines eigenständigen Tools oder die Integration in C-Logistic dar. Bei der Integration in C-Logistic ist hierbei die Vergabe von Archivierungsrechten und zwingend die Einrichtung von Datensatzsperren zur Vermeidung überschneidender Archivierungsvorgänge notwendig. Ein eigenständiges Tool könnte im Gegensatz gezielt nur einigen Benutzern oder dem DBA zur Verfügung gestellt werden. Ein mögliches Vorgehen bei der Umsetzung einer Archivierung soll mit dem im Anhang beigefügten Nassi-Shneiderman-Diagramm[35] verdeutlicht werden. Dieses Diagramm erhebt keinen Anspruch auf Vollständigkeit, es dient vorwiegend dazu, notwendige Schritte bei der Umsetzung einer Archivierung zu verdeutlichen. Bei der konkreten Umsetzung wären noch Ergänzungen wie bspw. eine Protokollierung oder die Einbindung eines Fortschrittsbalkens denkbar. Zur Darstellung des Ablaufes hätte auch ein Programmablaufplan oder ein UML[36]-Aktivitätsdiagramm verwendet werden können.

Bezug nehmend auf das Struktogramm ist weiterhin zu erläutern, dass über eine Master-Detail-Beziehung zwischen den ODAC-Komponenten qrSendung und qrSendpos sichergestellt werden kann, dass zusammengehörende Datensätze gemeinsam archiviert werden. Das Schreiben der Datensätze kann folgend über TOraSQL-Komponenten umgesetzt werden. Um Constraintverletzungen auszuschließen, ist es notwendig, die Sendung vor den Sendungspositionen anzulegen. Für die Übertragung der Daten ist der Einsatz bspw. eines DML-Arrays[37] durchaus sinnvoll, wie für die Sendungspositionen bereits dargestellt. Zusätzlich könnte bei umfangreichen Archivierungsvorgängen mit einer hohen Anzahl an Einfüge- bzw. Löschvorgängen eine Geschwindigkeitssteigerung durch die zeitweilige Entfernung der Indizes erreicht werden. Die ausgeführten Löschvorgänge könnten weiterhin die Effektivität der verwendeten Indizes des Produktivdatenbestandes spürbar einschränken, so dass sich die Verwendung des in Kapitel 2.4 erläuterten ANALYZE-Befehls als durchaus sinnvoll erweisen könnte. Der getrennte Zugriff auf die Produk-

[35] Auch als Struktogramm bekannt
[36] UML – Unified Modeling Language
[37] vgl.: 2.5 Entwicklungskomponenten

tiv- und Archivumgebung kann durch die Verwendung unterschiedlicher TOraSession-Komponenten erreicht werden. Der Aufruf PruefeArchivumgebung verweist auf eine Funktion, in der festgestellt werden sollte, ob bereits eine Archivumgebung erzeugt wurde. Hierfür könnte bspw. die SQL-Anweisung SELECT COUNT(*) AS ANZ FROM ALL_TABLES WHERE OWNER = 'ARCHIV' genutzt werden[38]. Sofern die ermittelte Anzahl nicht größer Null ist, muss die Archivumgebung ebenfalls noch erzeugt werden. Das Entfernen, Anlegen sowie die Rechtevergabe im Tablespace kann bspw. über die in der folgenden Abbildung dargestellten Anweisungen umgesetzt werden.

```
DROP TABLESPACE ARCHIV INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS;
CREATE TABLESPACE ARCHIV;
DATAFILE 'D:\ORACLE\ORADATA\ARCHIV.ORA' SIZE 512M REUSE AUTOEXTEND ON NEXT 32M MAXSIZE UNLIMITED
DEFAULT STORAGE (INITIAL 64K NEXT 64K MINEXTENTS 1 MAXEXTENTS UNLIMITED PCTINCREASE 0) ONLINE;
DROP USER ARCHIV CASCADE;
CREATE USER ARCHIV IDENTIFIED BY ARCHIV;
GRANT CONNECT TO ARCHIV;
GRANT RESOURCE TO ARCHIV;
GRANT CREATE TRIGGER TO ARCHIV;
GRANT CREATE PROCEDURE TO ARCHIV;
GRANT CREATE VIEW TO ARCHIV;
GRANT QUERY REWRITE TO ARCHIV;
ALTER USER ARCHIV DEFAULT TABLESPACE ARCHIV;
ALTER USER ARCHIV TEMPORARY TABLESPACE TEMP;
```

Abbildung 20: Skript Tablespace ARCHIV

Dem Anwender selbst sollte bspw. in einem Auswahldialog die Möglichkeit gegeben werden, einen Zeitraum für die zu archivierenden Sendungen anzugeben[39]. Neben der Einschränkung des Lieferbeginndatums sollten weiterhin in qrSendung nur Datensätze abgefragt werden, die den Bearbeitungsstatus 5, 40, 50 oder 60[40] aufweisen, um dem dargestellten Schema zu genügen. Nach erfolgreicher Archivierung können die gewählten Sendungen über eine weitere TOraSQL-Komponente entfernt werden. Hierfür wird, laut Struktogramm, mit Hilfe der in einer Stringliste gespeicherten Sendungsnummern dynamisch die Anweisung der Komponente OraSQLDelete-Sendung erweitert. Die Wahrung der Integrität wird beim Löschen durch die Fremdschlüsselbeziehung im Produktivmandanten sichergestellt.

Einige Gedanken der dargestellten Umsetzung, wie bspw. die Erstellung des Tablespaces mit der Abbildung der vollständigen Tabellenstruktur, sind bereits zum einen auf die möglichst problemlose spätere Pflege der Archivierungsumgebung sowie zum anderen auf die Integration weiterer Archivierungen ausgerichtet. Im vorliegenden Struktogramm wurden vorerst zur Verdeutlichung zwei TOraScript-Komponenten

[38] vgl.: Struktogramm Komponente qrVorhanden
[39] vgl.: Struktogramm Funktion SetSQLFilter und Filterbeispiel
[40] vgl.: Tabelle 8: Möglicher Bearbeitungsstatus einer Sendung

verwendet, welche die Installationsskripte der Relation SENDUNG und SENDPOS enthalten. Weiterhin muss gelten, dass Nutzer keine Veränderungen an den Archivdaten vornehmen können. Hierzu sollte nach Möglichkeit das Rechtekonzept aus Oracle genutzt werden. Oracle biete es an, über die als Rollen bezeichneten Gruppenrechte den Zugriff einzuschränken. Als Ansatz sollten hier mehrere Gruppen eingerichtet werden, von denen mindestens eine Gruppe volle Zugriffrechte besitzt. Diese Rechte können dann für die Archivierung genutzt werden, währenddessen andere Nutzer keine Veränderungen vornehmen dürfen.

4.3 Abschließende Bemerkungen zur Archivierung

Neben den bisher betrachteten bzw. angedeuteten Problemstellungen gibt es noch zahlreiche weitere Faktoren, die bei der Umsetzung betrachtet werden sollten. So wird bspw. durch die Primärschlüsselintegrität sichergestellt, dass ohne Archivierung keine doppelten Schlüsselwerte vergeben werden können. Nach Durchführung einer Archivierung ist es jedoch denkbar, dass ein neuer Datensatz mit einem bereits im Archiv vorhandenen Schlüsselwert angelegt werden könnte. Nach [HER97] gibt es zur Behandlung dieser Situation folgende denkbare Lösungsansätze:

- Übergreifende Integritätsbedingen (Archiv- und Produktivmandant)
- Künstliche Erweiterung der Primärschlüsselfeldes (Bsp.: Zeitstempel)
- Zeitweilige Tolerierung der Integritätsverletzung

In den Betrachtungen wurde bisher nicht weiter vertieft, wie die Erstellung der Archivumgebung umgesetzt werden könnte. Dies ist darin begründet, dass der Zugriff auf den Archivmandanten sowie die Abbildung der Tabellenstruktur ggf. über CIS-Standardfunktionen bzw. CIS-Standardroutinen umgesetzt werden könnte. In einer späteren Erweiterung könnte bspw. die Archivierung von Stammdaten, wie Fahrzeugendaten, in Betracht gezogen werden. Hierfür könnte das bereits derzeit häufig verwendete Attribut AKTIVFLAG, dass Stammdaten als inaktiv kennzeichnet, zur Identifikation archivierungsfähiger Datensätze genutzt werden.

Als abschließender Kommentar sollte noch erwähnt werden, dass beim Kunden dafür gesorgt werden muss, dass die hier dargestellte Archivierung keinesfalls als eine Art eines Backups verstanden wird. Vielmehr erfordert die Integration der Archivierung gleichzeitig die Einbindung in die bestehende Backupstrategie.

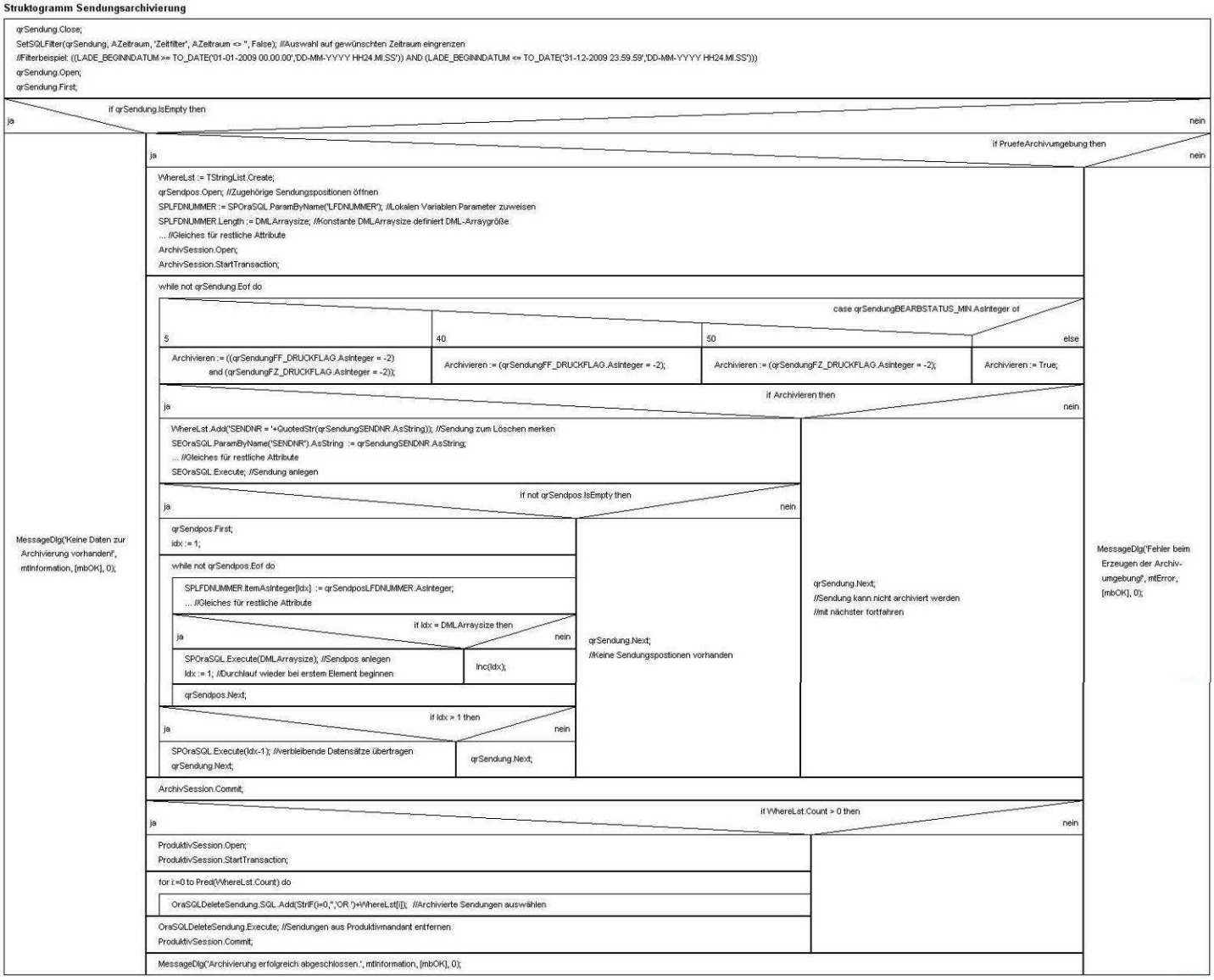
5. Abschlussbemerkung

Ein Teil möglicher Einschätzungen wurde bereits im Kapitel 3.5 als Ausblick vorweggenommen. Grundlegend kann aber gesagt werden, dass C-Logistic, entgegen der Einschätzung einiger Anwender, größtenteils in den betrachteten Problembereichen bereits auf eine schnelle Abarbeitung hin optimiert wurde. So stellen bspw. die verwendeten Komponenten nach derzeitigem Stand das Maß der Dinge dar. Zahlreiche zeitaufwändige Funktionen wurden bereits auf die Oracledatenbank ausgelagert. Komplexe Abfragen werden weitestgehend unter Verwendung von Indizes umgesetzt, um zusätzlich Ressourcen zu sparen. Dennoch konnten einige Ansätze, wie bspw. der Einsatz materialisierter Views gefunden werden, die eine Performancesteigerung nach sich ziehen können. Neben den hier betrachteten Problembereichen ist eine Betrachtung anderer Bereiche ebenfalls als empfehlenswert anzusehen. Dies lässt sich darin begründen, dass bspw. in einer Mehrnutzerumgebung die Ausführung aufwändiger Anweisungen in nicht betrachteten Erfassungsmasken auch zwangläufig Auswirkungen auf alle anderen Nutzer nach sich ziehen können. Grundlegend ist nach [AHR06] Tuning als Summe aller Maßnahmen anzusehen, welche die Optimierung der Antwortzeiten und die Verbesserung der Skalierbarkeit zum Ziele haben. Ein grundlegendes Problem ist hierbei die Tatsache, dass Tuning aus Zeitgründen meist nur als eine einmalige Aktion angesehen wird, die man im Problemfall ausführt. Besonders sich häufig ändernde Anforderungen und konzeptionelle Schwächen in neuen Modulen können einmalige Tuningbemühungen zwecklos erscheinen lassen. Somit sollte zur Kosteneinsparung und zur Vermeidung von Engpässen bzw. Ressourcenverschwendung Tuning kontinuierlich umgesetzt werden. Hierbei ist es ebenfalls wichtig, dass nach Möglichkeit zukünftige Anforderungen realistisch eingeschätzt werden, um zusätzliche Folgeprobleme zu vermeiden.

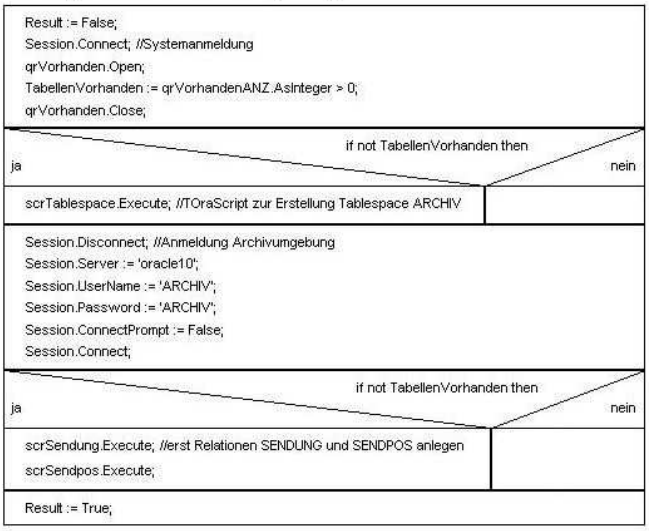
C-Logistic automatisiert eine Vielzahl der speditionsseitigen Ausgabenbereiche, sodass ebenfalls beachtet werden sollte, dass bspw. die verschiedensten umfangreichen Berechnungen grundlegend Zeit beanspruchen. Als problematisch ist insgesamt der enorme Datenbestand einiger Kunden anzusehen. Besonders im Fall der Sendungserfassung ist davon auszugehen, dass der durch die Einführung einer Archivierungslösung und dem damit verbundenen eingeschränkten Produktivbestand an Bewegungsdaten eine deutliche Beschleunigung zu erwarten ist. So wie es bereits im entsprechenden Kapitel dargestellt wurde.

Anhang A1 Struktogramme Archivierung

Struktogramm Sendungsarchivierung



Struktogramm PruefeArchivumgebung



Literaturverzeichnis

[ABO06] AutomatedQA test, debug, deliver: *Case Study: Borland Uses AQTime to Tune Delphi 2006.* Online im Internet: URL: http://www.automatedqa.com/about/case-studies/borland-case-study/ (Stand: 15.07.2009)

[AHR06] Ahrends, Johannes: *Oracle 10g für den DBA Effizient konfigurieren, optimieren und verwalten.* Addisson-Wesley Verlag, München, 2006, ISBN 978-3-8273-2171-8

[ALL09] Allround Automations: *Direct Oracle Acces*s. Online im Internet: URL: http://www.allroundautomations.com/ (Stand 08.07.2009)

[ANB06] Annette Bunz: *Datenbank-Tuning anhand fortschrittlicher Analyse-Methoden bei heutigen Datenbanksystemen.* Diplomarbeit Fachhochschule Augsburg, 2006, Online im Internet: URL: http://www.hs-augsburg.de/informatik/diplomarbeiten/langfassungen-/muellenbach-bunz-2006.pdf (Stand: 23.07.2009)

[AQT09] AutomatedQA test, debug, deliver: *Profiling and Debugging Source Code with AQTime.* Online im Internet: URL: http://www.automatedqa.com/products/aqtime/ (Stand: 15.07.2009)

[BUR98] Burleson, Don: *Oracle 8 Tuning.* SYBEX-Verlag, Düsseldorf, 1998, ISBN 3-8155-7291-6

[CWL06] Pedigo, Larry: *A Comparison of Oracle Database 10gR2 Real Application Cluster Performance on Microsoft Windows 2003 Server Enterprise Edition x64 and Red Hat Enterprise Linux x86_64.* Performance Tuning Corporation, 2006, Online im Internet: URL: http://www.perftuning.com/pdf/Comparison_Oracle_Windows_Linux.pdf (Stand: 25.06.2009)

[DEV09] Devart: *Oracle Data Access Components.* Online im Internet: URL: http://www.devart.com/odac/ (Stand: 07.07.2009)

[FRO94] Froese, Jürgen: *Effiziente Systementwicklung mit ORACLE 7 ein Handbuch für die Praxis des Anwendungsentwicklers.* Addisson-Wesley Verlag, Bonn, 1994, ISBN 3-89319-525-4

[HER97] Herbst, Axel: *Anwendungsorientiertes DB-Archivieren.* Springer, Berlin, 1997, ISBN 3-540-63209-3

[KER00] Kerninghan, Brian, Pike, Rob: *Programmierpraxis.* Addisson-Wesley Verlag, München, 2000, ISBN 3-8273-1583-2

[LON05] Loney, Kevin: *ORACLE DATABASE 10g Die umfassende Referenz.* Carl Hanser Verlag München Wien, München, 2005, ISBN 3-446-22833-0

[LUG07] Juni 2007 Lohn+Gehalt: *Archivierung - Ganz nach Vorschrift: Compliance bei der Datenarchivierung.* Online im Internet: URL: http://www.elektronische-steuerpruefung.de/loesung/lug_2007_06.pdf (Stand 02.08.2009)

[PED05] Pedigo, Larry: *Comparison of 32-bit and 64-bit Oracle Database Performance on the Dell PowerEdge 6850 Server with Microsoft Windows Server 2003.* Performance Tuning Corporation Tune Right – Run Fast®, 2005, Online im Internet: URL: http://www.perftuning.com/pdf/Compare3464.pdf (Stand: 27.06.2009)

[SCH09] Schubert, Denny: *Möglichkeiten einer teilautomatisierten Fehlerprüfung und –analyse von Softwareupdates am Beispiel von C-Logistic.* Studienarbeit Staatliche Studienakademie Glauchau, Glauchau, 2009

[SCO09] SourceCodeOnline: *NC OCI8 1.0.4.* Online im Internet: URL: http://www.sourcecodeonline.com/details/nc_oci8.html (Stand 06.07.2009)

[SQD09] SQLDirect Web: *SQLDirect.* Online im Internet: URL: http://www.sqldirect-soft.com/index.html (Stand: 10.07.2009)

[STR09] Strateg GmbH: *Berechnen Sie ihre Mitarbeiterkosten.* Online im Internet: URL: http://www.strateg.de/unternehmen/rechner.php3 (Stand 01.08.2009)

[UHA09] Uni Hannover: *Planungsmethoden: Kosten-Nutzen-Analyse.* Online im Internet: URL: http://www.laum.uni-hannover.de/ilr/lehre/Ptm/Ptm_BewKna.htm (Stand: 01.08.2009)

[VEN08] Vorlesungsunterlagen Lehrveranstaltung Datenbanken*: Kap. 5 - Entwurf relationaler Datenbanken.* Thorsten Lehnguth, BA Glauchau, Stand 21.01.2008

[VEI08] Vorlesungsunterlagen Lehrveranstaltung Datenbanken*: Kap. 6 - Einsatz von RDBMS in der Praxis.* Thorsten Lehnguth, BA Glauchau, Stand 05.03.2008

[WPA09] Wikipedia, Die freie Enzyklopedie: *Programmablaufplan*. Online im Internet: URL: http://de.wikipedia.org/wiki/Programmablaufplan (Stand 28.07.2009)

[ZEO09] Sourceforge FIND AND DEVELOP OPEN-SOURCE SOFTWARE: *ZeosLib.* Online im Internet: URL: http://sourceforge.net/projects/zeoslib/ (Stand: 09.07.2009)

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Diplomarbeit mit dem Thema

   „Lösungsansätze zur Laufzeitverbesserung kommerzieller Datenbankanwendungen am Beispiel der Softwarelösung „C-Logistic““

   ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

3. dass ich meine Diplomarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Unterschrift

Thesenblatt

- Tuning soll der Sicherstellung angemessener Antwortzeiten für geschäftsrelevante Prozesse dienen.

- Für C-Logistic können als performancerelevante Einflussgrößen Faktoren wie Hardware, Betriebssystem, Datenbank, Datensatzanzahl, verwendete Entwicklungsumgebung bzw. Entwicklungskomponenten sowie die Arbeitsweise des Anwenders gefunden werden.

- Eine Performancesteigerung lässt sich durch den Einsatz von Indizes, materialisierten Views sowie durch Clustering oder die Verwendung verschiedener Tablespaces erwarten.

- Aus Sicht des Kosten- und Zeitaufwandes stellt eine nachträgliche Optimierung immer die schlechteste Variante dar. Vielmehr sollte der gesamte Entwicklungsprozess auf eine konsequente Optimierung ausgerichtet werden, um Kosten zu sparen.

- Besonders das Auftragsvolumen, das bei einigen Anwendern über mehrere Jahre zusammengelaufen ist, sollte auf absehbare Zeit durch die Bereitstellung einer adäquaten Archivierungslösung effektiv gesenkt werden.

- Am sinnvollsten erscheint die Umsetzung einer Archivierung, bei der Datensätze, die bestimmten Kriterien entsprechen in einen separaten Tablespace verschoben werden. Der Zugriff auf die Archivdaten könnte immer mit der aktuellen C-Logistic-Programmversion umgesetzt werden.

- Grundlegend kann gesagt werden, dass C-Logistic, entgegen der Einschätzung einiger Anwender, in den betrachteten Problembereichen bereits auf eine schnelle Abarbeitung hin optimiert wurde.