HOME
HOCHSCHULE
MERSEBURG FH
University of
Applied Sciences

Master Theses

Prototypische Implementierung eines C/AL - Programms zur Performance-Analyse von Microsoft Dynamics NAV 2015

Zur Erlangung des akademischen Grades eines
Master of Engineering
- Informatik und Kommunikationssysteme -

Fakultät Informatik
Referent: Prof. Dr. Ronny Weinkauf
Korreferent: Dipl. Inf. Mathias Meissner

eingereicht von:
Richard Kaupa
Matr.-Nr. 20556
Gartenstraße 4
37339 Haynrode

Merseburg, den 30.09.2015

Eidesstattliche Erklärung

Hiermit erkläre ich, dass die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt wurden. Insbesondere alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen wurden, sind durch Zitate gekennzeichnet. Weiterhin versichere ich, dass die eingereichte schriftliche Version bisher nicht in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorliegt.

Merseburg, den 30.09.2015

_____
(Richard Kaupa)

Danksagung

Ich möchte mich bei allen Wegbegleitern, Mitarbeitern der msu solutions GmbH und meiner Familie, die mir zu jeder Tageszeit mit gutem Rat und Motivation zur Seite standen, bedanken.

Mein besonderer Dank gilt Herrn Prof. Dr. Ronny Weinkauf für die Betreuung dieser Arbeit und die stets gute Zusammenarbeit.

Herrn Dipl. Inf. Mathias Meissner danke ich ebenfalls sehr für die Betreuung der Arbeit und die immer währende konstruktive Kritik sowie guten Einfälle zum Voranbringen der Arbeit.

Zusammenfassung

Für die Nutzbarkeit von Computerprogrammen gibt es viele Kriterien und Vorstellungen. Gerade bei größeren Datenmengen tritt die Geschwindigkeit der Visualisierung und Verarbeitung in den Vordergrund. Zur Analyse von Leistungsengpässen gibt es diverse Softwarelösungen. Allerdings sind diese entweder nur von Spezialisten bedienbar, mit der zu betrachtenden Datenmenge überfordert oder inkompatibel zu dem in der msu solutions GmbH verwendeten Navision-System.

Die vorliegende Arbeit beschäftigt sich mit dem Problem der Performance-Analyse im Kontext des Programms Microsoft Dynamics NAV 2015. Hierzu wird ein Programm entwickelt, dass dem entsprechenden Nutzer ermöglicht, Leistungsengpässe in NAV 2015 ausfindig zu machen und zu beheben. Der erstellte Prototyp dient anschließend zur Durchführung verschiedener Tests.

So wird der Frage nachgegangen, welches NAV-System, NAV 2009 oder NAV 2015, eine Rahmenvertragsabrechnung schneller abarbeitet, welche Leistungsprobleme bei dem Ablauf einer Jahresabrechnung für Gaskunden auftreten und abschließend wie sich verschiedene Einstellungen des Cachings bei einem Validierungstest für die Richtigkeit einer UTILMD Nachricht auswirken.

Die gestellten Aufgaben werden anhand von Daten, die die Prototyp-Software erfasst hat, beantwortet. Im Ergebnis zeigt sich in Test eins, dass NAV 2015 bei einfachen INSERT-Operationen schneller ist als NAV 2009, aber bei der Verarbeitung von UPDATE-Befehlen hinter sein Vorgängersystem zurückfällt. Test zwei macht deutlich, dass die Dauer der Jahresabrechnung im extremen Maße von der Anzahl, auf einer Tabelle angelegter, Indizes abhängt und Test Nummer drei deutet auf ein in NAV 2015 integriertes Caching hin, welches das unnötige Senden von SQL Befehlen an den SQL Server verhindert.

Nach Absolvierung der genannten Tests kann festgehalten werden, dass die im Zuge dieser Arbeit erstellte Software einen Einblick in die Datenverarbeitung von NAV 2015 bietet. Aber alle Rückschlüsse auf auftretenden Leistungsprobleme müssen durch den Programmnutzer selbst gezogen werden, da das Programm keine Lösungsmöglichkeiten anbietet oder ausführen kann.

Inhaltsverzeichnis

Eidesstattliche Erklärung **II**

Danksagung **III**

Zusammenfassung **IV**

Abbildungsverzeichnis **VII**

Tabellenverzeichnis **VIII**

Listings **IX**

Abkürzungsverzeichnis **X**

1 Einführung **1**
1.1 Einleitung . . . 1
1.2 Ziel der Arbeit . . . 2
1.3 Aufbau der Arbeit . . . 2

2 Grundlagen **3**
2.1 Microsoft Dynamics Navision 2015 . . . 3
2.1.1 Grundlagen . . . 3
2.1.2 Client/Server-Architektur . . . 4
2.1.3 C/AL . . . 6
2.1.4 ADO.NET . . . 7
2.2 Microsoft SQL Server . . . 9
2.2.1 Grundlagen . . . 9
2.2.2 Ablauf einer SQL-Abfrage . . . 11
2.2.3 Der physikalische Ausführungsplan . . . 12
2.2.4 Werkzeuge zur Analyse . . . 14
2.3 XML . . . 17

3 Konzeption **18**
3.1 Anforderungen an die Software . . . 18
3.2 Anwendungsfälle . . . 19
3.3 SQL-Trace . . . 24
3.4 Datenmodell . . . 25
3.5 SQL-Script . . . 28
3.6 C/AL Programm . . . 28

4 Implementierung **29**
4.1 SQL-Prozeduren 29
4.2 C/AL Programm 33

5 Test der Anwendung **38**
5.1 Abrechnung eines Rahmenvertrages 38
5.2 Jahresabrechnung von Rahmenverträgen 43
5.3 Validierung einer UTILMD 47

6 Ausblick und Fazit **49**
6.1 Ausblick 49
6.2 Fazit 52

Literaturverzeichnis **i**

A Abbildungen des C/AL Programms **iv**

Abbildungsverzeichnis

2.1 Anzahl Unternehmen die Microsoft ERP-Systeme nutzen 3
2.2 Zwei Schichten Architektur 5
2.3 Drei Schichten Architektur 5
2.4 Schematische Architektur eines DBMS 9
2.5 Abfragereihenfolge eines SQL-Befehls 11

3.1 Anwendungsfälle des C/AL Progarmms 19
3.2 Datenmodell der Navision Tabellen 27

4.1 PA Trace List 33
4.2 PA Statement List 35

5.1 Tracelaufzeiten 1 39
5.2 Tracelaufzeiten 2 40
5.3 Vergleich von NAV 2015 und NAV 2009 41
5.4 Vergleich von NAV 2015 und NAV 2009 42
5.5 Ist-Zustand nach der Abrechnung von 30 Rahmenverträgen in GMH-System 44
5.6 Vorher/Nachher Vergleich der Jahresabrechnung 45
5.7 Zeitlicher Vergleich von UTILMD Validierungstests 48

6.1 Schema des Hirschberg-Algorithmus 51

A.1 PA Trace Overview List v
A.2 PA Statement Card vi
A.3 PA Plan Card vii
A.4 Plan viii
A.5 PA Statistics List ix
A.6 PA Trace Div Wizzard x
A.7 Vergleich zweier unterschiedlicher SQL Traces xi

Tabellenverzeichnis

2.1 ADO.NET Variablendeklaration 7

3.1 Tabelle 1 mit mitgetraceten Daten 24
3.2 Tabelle 2 mit mitgetraceten Daten 24
3.3 Navison Tabellennamen und ID's 25
3.4 Navison Pagenamen und ID's 28

5.1 Vorher/Nachher Vergleich der Rahmenvertragsabrechnung 45

Listings

2.1 C/AL Beispiel, Quelle: Eigenes Listing 7
2.2 ADO.NET Beispiel, Quelle: Eigenes Listing 8
2.3 SQL Beispiel, Quelle: Eigenes Listing 11
2.4 XML Standardbeispiel, Quelle: Eigenes Listing 17

4.1 SQL Befehl zur Definition des Zielpfades, Quelle: Eigenes Listing . . 29
4.2 SQL Befehl zur Definition einer Traceoption, Quelle: Eigenes Listing 29
4.3 SQL Befehl zur Filterung eines Traces, Quelle: Eigenes Listing 30
4.4 SQL Befehl zur Befüllung einer temporären Tabelle, Quelle: Eigenes Listing 30
4.5 SQL Befehl zur Befüllung Plan-Tabelle, Quelle: Eigenes Listing . . . 31
4.6 Insert Befehl in der Differenzerstellung, Quelle: Eigenes Listing . . . 31
4.7 SQL DELETE FROM Befehl, Quelle: Eigenes Listing 32
4.8 SQL TRUNCATE Befehl, Quelle: Eigenes Listing 32
4.9 C/AL Befehl zum Starten der Ablaufverfolgung durch Navision, Quelle: Eigenes Listing 34

6.1 LCS-Problem in C/AL, Quelle: Eigenes Listing 50

Abkürzungsverzeichnis

ERP Enterprise-Resource-Planning
SQL Structured Query Language
C/AL Client/Application Language
XML Extensible Markup Language
SSPI Security Support Provider Interface
LCS Longest Common Subsequence
API Application Programming Interface
DBMS Database Management System

1 Einführung

1.1 Einleitung

Laut des statistischen Bundesamtes wurden im Jahr 2014 mehr als 1.200 Energieversorgungsbetriebe und über 400 Wasserversorgungsunternehmen in Deutschland mit zusammen über 200.000 angestellten Personen erfasst. [1] All diese Unternehmen müssen eine immer stärker steigende Zahl von Personal, Kunden oder Lieferanten verwalten. Um dies zu tun, werden unter anderem die unterschiedlichsten ERP-Softwarelösungen eingesetzt.

Die msu Solutions GmbH, im Folgenden msu genannt, aus Halle entwickelt eine solche Softwarelösung für Energie- und Wasserversorgungsunternehmen auf Basis des von der Firma Microsoft bereitgestellten ERP-Systems Microsoft Dynamics Nav. Zum jetzigen Zeitpunkt kann auf 20 Jahre Branchenerfahrung im Bereich der Energie- und Wasserwirtschaft zurück geblickt werden.[2]

Alle Mitarbeiter der msu arbeiten kontinuierlich an neuen innovativen Lösungen, um den sich stets verändernden Anforderungen ihrer Kunden gerecht zu werden.

Diese Anforderungen an die von Energie- und Wasserversorgungsunternehmen eingesetzte Software reichen von der einfachen Bedienbarkeit und hohen Ausfallsicherheit der Programme bis hin zu einer möglichst hohen Performance. Gerade die Performance ist ein immer wichtiger werdender Faktor in Bezug auf die eingesetzte Software.

Der Bedarf an Performance schlägt sich im Wunsch der Unternehmensführungen nieder, die Arbeitszeit ihrer Mitarbeiter so wenig wie möglich mit dem Warten auf das Laden von Übersichtslisten beziehungsweise die Ausführung von Rechnungsläufen zu belasten und sie somit effizienter zu gestalten.

Hieraus ergibt sich, für die msu, die Frage nach den Möglichkeiten ihre entwickelte ERP-Softwarelösung zu verbessern und damit die oben genannten Arbeitsabläufe ihrer Kunden zu beschleunigen.

[1] [Bun15]
[2] [msu06]

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist ein, in Microsoft Dynamics Nav 2015 implementiertes, Programm zur Performanceanalyse. Die erstellte Software soll den Entwicklern der msu Solutions GmbH erlauben, eine Aussage über die Schwachstellen von, durch Navision 2015 generierten, SQL-Befehlen zu treffen und diese Probleme schnellstmöglich zu beheben.

Eine schnelle Problemanalyse hilft nicht nur den Mitarbeitern der msu bei der Entwicklung und im Test, sondern führt auch zu einer Zeitersparnis und damit zu einer Effizienzsteigerung der Mitarbeiter in den Kundenbetrieben. Was im Endeffekt zu einer Kostenersparnis aller beteiligter Firmen führt.

1.3 Aufbau der Arbeit

Die hier vorliegende Arbeit gliedert sich in sechs Teile. In der Einführung (Kapitel 1) wird der Leser in das im späteren Verlauf bearbeitete Thema eingeleitet und es wird das Ziel der Ausarbeitung definiert.

Eine Klärung der Grundlagen findet sich in Kapitel 2. So wird unter anderem auf Basiswissen zu Microsoft Dynamics NAV 2015, speziell Client/Server-Architektur, C/AL und ADO.Net als auch auf Microsoft SQL Server, mit Augenmerk auf eine SQL-Abfrage, Statement-Ausführungspläne und die verschiedensten Werkzeuge zur Leistungsanalyse, eingegangen.

Kapitel 3 beschäftigt sich mit der Konzeption der Prototyp-Software. Die Planung des Programms reicht von der Definition der Anforderungen über die Spezifikation des genutzten Datenmodells bis hin zur Beschreibung grundlegender Aufgaben und Bezeichnungen des zu erstellenden Endproduktes.

Zur Erläuterung der Implementierung im Verlauf der Arbeit generierter SQL-Prozeduren und entsprechender Navision eigener Programmbausteine dient Kapitel 4. Eine Verifikation der Funktionstüchtigkeit des Prototyps wird in Kapitel 5 durchgeführt. Hierzu werden in zwei Tests zum Einen die Abrechnung eines Rahmenvertrages in Dynamics NAV 2015 mit der Abrechnung im abzulösenden NAV 2009 verglichen und zum Anderen erfolgen Tests zur Optimierung der Rahmenvertrags-Jahresabrechnung in einem Testsystem mit Produktivdaten und entsprechend großen Datenmengen.

Zusammenfassend zieht das Kapitel 6 ein Fazit über die Erfüllung des gestellten Ziels und gibt einen Ausblick auf weitere Programmkomponenten zur Verbesserung und Weiterentwicklung des Prototyps.

2 Grundlagen

2.1 Microsoft Dynamics Navision 2015

2.1.1 Grundlagen

Microsoft Dynamics Nav ist ein, von der Firma Microsoft entwickeltes und vertriebenes, Enterprice-Resource-Planning-System. Diese ERP-Software ermöglicht es kleinen und mittelständischen Unternehmen ihre Geschäftsprozesse zu verwalten. Zu diesen Prozessen gehört zum Beispiel Personalverwaltung, Logistik oder auch die Lagerverwaltung und Buchhaltung.[1]

Die nachfolgende Abbildung 2.1 zeigt die Anzahl aller Kunden, die ERP-Systeme von Microsoft in ihren Firmen einsetzen. So ist zu sehen, dass Dynamics Nav das, mit 110.000 Nutzern, am weitesten verbreitete Microsoftprodukt im Bereich des Enterprice-Resource-Planning ist.

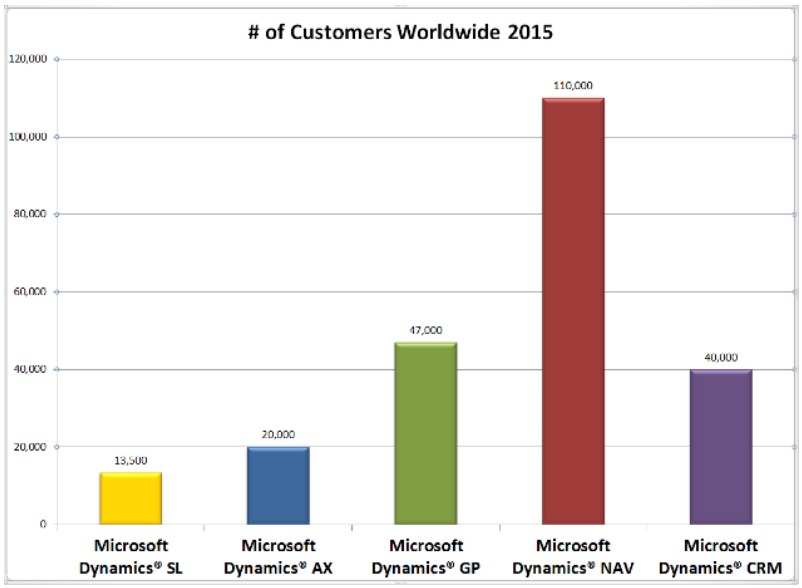


Abbildung 2.1: Anzahl Unternehmen die Microsoft ERP-Systeme nutzen, Quelle: [Edi15]

Im Vergleich zu SAP mit 26% und ORACLE mit 17% Gesamtmarktanteil liegt Microsoft mit 11% auf Platz 3 der Anbieter von ERP-Systemlösungen weltweit.[2]

[1] [Mic15b]
[2] [Com13]

2.1.2 Client/Server-Architektur

Die Client/Server-Architektur entstand aus dem Wunsch heraus die Kooperation unter Mitarbeitern einer Firma zu unterstützen, indem es möglich wurde Dateien untereinander auszutauschen oder auch sehr leistungsstarke Computerhardware mehreren Nutzern zur Verfügung zu stellen. Die einfachste Form einer solchen Architektur sind Datei-Server in einem Netzwerk, die Ihre Festplatten und die darauf lagernden Dateien allen Netzwerknutzern zur Verfügung stellen.

Mit den Jahren entwickelte sich die Client/Server-Architektur von der einfachen Dateibereitstellung zu einer Plattform, auf die gemeinsam genutzte Dienste ausgelagert und dort zentral zusammengefasst wurden. So sind heutige Datenbankmanagementsysteme oder Mailserver aktuelle Beispiele.[3]

Der Client/Server-Architektur liegen folgende Grundideen zugrunde:[4]

- Oft benötigte Dienste ausfindig machen
- Herauslösen der identifizierten Dienste aus ihren Anwendungssystemen
- Realisieren der herausgelösten Dienste in allgemein gültiger Form
- Bereitstellen der Dienste in einem Netzwerk über ein Application Programming Interface (API)

2-Schichten-Architektur

Diese Architektur wird in der Regel in zwei Varianten umgesetzt. In Variante eins wird ein sogenanntes *Thin-Client-Modell* verwendet. Ein *Thin-Client* dient ausschließlich der Darstellung der Inhalte und die Datenhaltung und Anwendungslogik werden auf den Server ausgelagert. Die daraufhin starke Belastung des Servers und des Netzwerkes ist der gravierendste Nachteil dieser Ausprägung.

Variante zwei ist das *Fat-Client-Modell*. Hierbei wird die Anwendungslogik vom Client beherbergt und der Server dient ausschließlich der Datenhaltung. Heutige Geldautomaten sind Beispiel für den Einsatz dieses Modells. Problematisch werden hier Änderungen in der Anwendungssoftware, da in diesem Fall eine Neuinstallation der Clients erforderlich wird. Bei großen Netzwerken mit vielen Client-Rechnern können somit hohe Kosten entstehen. [5]

Der älteste und einfachste Architekturansatz wird in Abbildung 2.2 schematisch mit einem Server und einer beliebigen Anzahl Client-Rechnern dargestellt.

[3] [Nie96, S.9]
[4] [Nie96, S.10]
[5] [Som07, S.305/306]

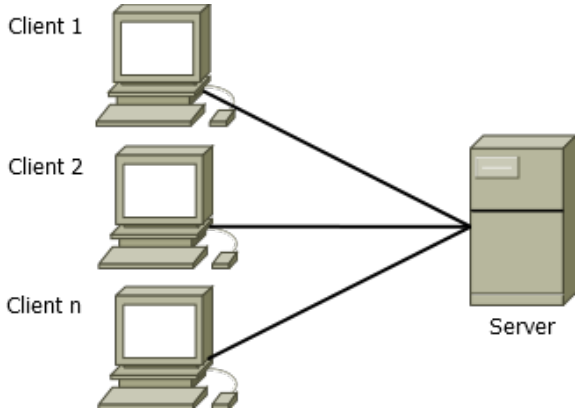


Abbildung 2.2: Zwei Schichten Architektur, Quelle: Eigene Darstellung

3-Schichten-Architektur

Netzwerkarchitekturen mit drei Schichten sind heute am weitesten verbreitet. So ist unter anderem das von der msu entwickelte ERP-System eine solche Drei-Schichten-Architektur. Sie besteht aus einem Role Tailored Client, der das Präsentationslevel übernimmt, einem Microsoft Dynamics Nav Server, der die Geschäftslogik und Kommunikation übernimmt sowie einem Microsoft SQL Server für die Datenhaltung.[6]

Abbildung 2.3 veranschaulicht den Sachverhalt.

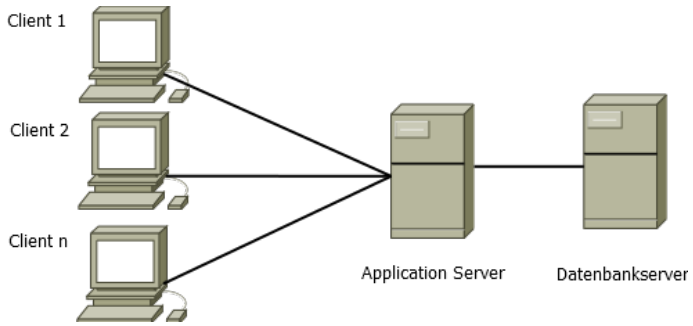


Abbildung 2.3: Drei Schichten Architektur, Quelle: Eigene Darstellung

n-Schichten-Architektur

Hauptunterschied von einer Drei-Schichten-Architektur zu einer Architektur mit n-Schichten ist die Aufteilung der Anwendungsprozesse auf mehrere Server. Außerdem ist die Skalierbarkeit von Netzwerken mit drei und mehr Schichten normalerweise besser als bei einer Zwei-Schichten-Architektur. [7]

[6] [Mic15d]
[7] [Som07, S.307]

2.1.3 C/AL

Die Programmiersprache C/AL (Client/Application Language) wurde speziell zur Anpassung von Navision entwickelt. Das verantwortliche Entwicklerteam verwendete die Programmiersprache C++ zur Entwicklung von C/AL.

C/AL ist ein Werkzeug zum Definieren der Prozesse, welche Daten manipulieren, zum Festlegen der Regeln, die die verschiedensten Anwendungen kontrollieren und zum Kontrollieren der Abfolge aller logischen Verarbeitungssequenzen. Weiterhin können mit C/AL Objekte bearbeitet, neue Funktionen erstellt und Daten auf verschiedensten Wegen beeinflusst werden. [8]

Die folgenden Objekte können mit C/AL beeinflusst werden:[9]

- Tabellen
- Tabellenfelder
- Pages
- Reports
- Datenelemente
- XML Ports
- Queries

Zur Entwicklung von C/AL Programmen wird das Client/Server Integrated Development Environment, kurz C/SIDE, eingesetzt. Es umfasst den Editor, Debugger, Reports, Page Generatoren und Code Management Werkzeuge.[10]

Ein Beispiel für C/AL Quellcode ist in Listing 2.1 dargestellt. Alle im Beispiel verwendeten Variablen sind lokale Funktionsvariablen und durch ein vorangestelltes *l_* markiert. Der Quellcodeausschnitt beginnt mit einer Wertinitialisierung der Variable l_i. Anschließend wird der Tabellenschlüssel und die Sortierreihenfolge mittels *SETCURRENTKEY* auf der Tabelle Customer, in Navision *Record* genannt, definiert. Um das beeinflusste Ergebnis einzugrenzen, dient ein Filter auf der Spalte No. im Bereich von C003 bis C005. Der dazu verwendete Befehl *SETFILTER* ist die Repräsentation einer WHERE-Klausel eines SQL Befehls.

Die eigentliche Änderung in der Datenbank erfolgt im Rahmen des *REPEAT .. UNTIL* Codebox nach erfüllter IF-Bedingung. Wenn der Befehl *FINDSET* Daten enthält, wird an jeden in der Ergebnismenge enthaltenen Namen der Wert der Zählervariable angehängt und der Datensatz mittels *MODIFY* bearbeitet. Der Parameter TRUE gibt an, dass das gewählte Ergebnis bearbeitet werden soll.

[8] [Stu07, S. 2]
[9] [Mic15i]
[10] [Stu07, S. 2]

Bei genauerer Betrachtung des Beispiels fällt eine starke Ähnlichkeit zur Programmiersprache PASCAL auf. So ist die Wertzuweisung der Variablen, das *IF..THEN* sowie das *REPEAT..UNTIL* und weitere C/AL Codekonstrukte in PASCAL ebenfalls in Gebrauch.

```
1 l_i := 0;
2 l_CustomerREC.SETCURRENTKEY("No.");
3 l_CustomerREC.SETFILTER("No.", 'C003..C005');
4 IF l_CustomerREC.FINDSET(TRUE) THEN
5   REPEAT
6     l_i := l_i + 1;
7     l_CustomerREC.Name += FORMAT(l_i);
8     l_CustomerREC.MODIFY;
9   UNTIL l_CustomerREC.NEXT = 0;
```

Listing 2.1: C/AL Beispiel, Quelle: Eigenes Listing

2.1.4 ADO.NET

ADO.NET ist ein wesentlicher Bestandteil des von der Firma Microsoft publizierten .NET Frameworks. Es besteht aus einer Menge von Klassen, die ihren Nutzern verschiedene Datenzugriffs- und Datenverarbeitungsmöglichkeiten bieten. Zu den nutzbaren Datenquellen gehören verschiedenste Datenbanksysteme und XML Daten.[11]

Die ADO.NET Funktionalitäten können von diversen Programmiersprachen genutzt werden. So existiert auch die Möglichkeit, benötigte Klassen in Form von Variablen in Navision einzubinden und dann im C/AL Code zu verwenden.

Eine solche Variablendeklaration erfolgt durch das Vergeben eines eindeutigen Variablennamens, das Definieren des Datentyps und gegebenenfalls einer Länge und Unterdatentyps. Tabelle 2.1 zeigt das Deklarieren der globalen Variable ConnectionString. Der Subtype ist in diesem Beispielfall leer, da er nicht benötigt wird. Er wird beispielsweise eingesetzt, wenn der Datentyp *Record* gewählt ist und nimmt den Namen der benötigten Datenbanktabelle auf.

Name	DataType	Subtype	Length
g_SQLString	Text		1024

Tabelle 2.1: ADO.NET Variablendeklaration, Quelle: Eigene Tabelle

[11] [Mic15a]

In den 13 Zeilen des Listings 2.2 wird gezeigt wie mit Hilfe von ADO.NET Klassen eine Datenbankverbindung aufgebaut, Befehle übergeben und ausgeführt werden. Durch das Übergeben von leeren Texten als Benutzernamen und Passwort, zusehen in Zeile vier des Listings, wird das Security Support Provider Interface (SSPI) zum Anmelden des Nutzers an die Datenbank genutzt.

Das SSPI ist eine, nicht nur bei Microsoft vertretene, Möglichkeit einen Nutzer ohne das Wechseln des Kontextes gegen ein geschütztes System, in diesem Fall die Datenbank, zu authentifizieren.[12] Im hier vorliegenden Beispiel wird die Windowsauthentifizierung des gerade angemeldeten Nutzers an die Datenbank weitergeleitet und genutzt.[13] Weiterhin soll die Verbindung sofort aufgebaut und ein Fehler bei Problemen zurück gegeben werden. Repräsentation dafür, ist die Zuweisung des Wertes 0 zur Variablen *CommandTimeout* in Zeile 10 des Listings.[14]

```
1  g_SQLConnection :=
2  g_SQLConnection.SqlConnection(
3  l_GetConnectionString(FORMAT(g_DBServer),
4            FORMAT(g_DBName), '', ''));
5  g_SQLConnection.Open;
6  g_SQLString.ADDTEXT('SELECT * FROM Personal');
7  g_SQLCommand := g_SQLCommand.SqlCommand();
8  g_SQLCommand.Connection := g_SQLConnection;
9  g_SQLCommand.CommandText := g_SQLString;
10 g_SQLCommand.CommandTimeout := 0;
11 g_SQLCommand.ExecuteNonQuery;
12 g_SQLConnection.Close;
13 g_SQLConnection.Dispose;
```

Listing 2.2: ADO.NET Beispiel, Quelle: Eigenes Listing

[12] [Mic15h]
[13] [Mic15f]
[14] [Mic15g]

2.2 Microsoft SQL Server

2.2.1 Grundlagen

Ein SQL Server ist ein Verwaltungs- und Analysesystem für Datenbanken. Er wird heutzutage in den verschiedensten Bereichen eingesetzt. E-Commerce-, Branchen- und Data Warehousing-Lösungen sind nur einige Beispiele.[15]

In der nachfolgenden Abbildung 2.4 ist die Architektur eines Datenbankmanagementsystems wie dem Microsoft SQL Server schematisch dargestellt. Das gezeigte Schaubild kann in eine interne Sicht, eine konzeptionelle Sicht und eine externe Sicht dreigeteilt werden. Zentrales Element ist das Data Dictionary. Dieser Katalog beinhaltet alle für das DBMS wichtigen Informationen die Datenhaltung betreffend. So kann man die Komponenten der Sichtendefinition, Datendefinition und Datenorganisation als Definitionskomponenten zusammenfassen. Ihnen gegenüber liegen die Transformationsbausteine für den Optimierer, die Auswertung und den Festplattenzugriff. Für Endnutzer zugänglich sind die Abfragen und Updates in der externen Sicht. Sie sind abgetrennt von den Masken, Einbettungen und Datenbankoperationen für Entwickler und Programmierer.[16]

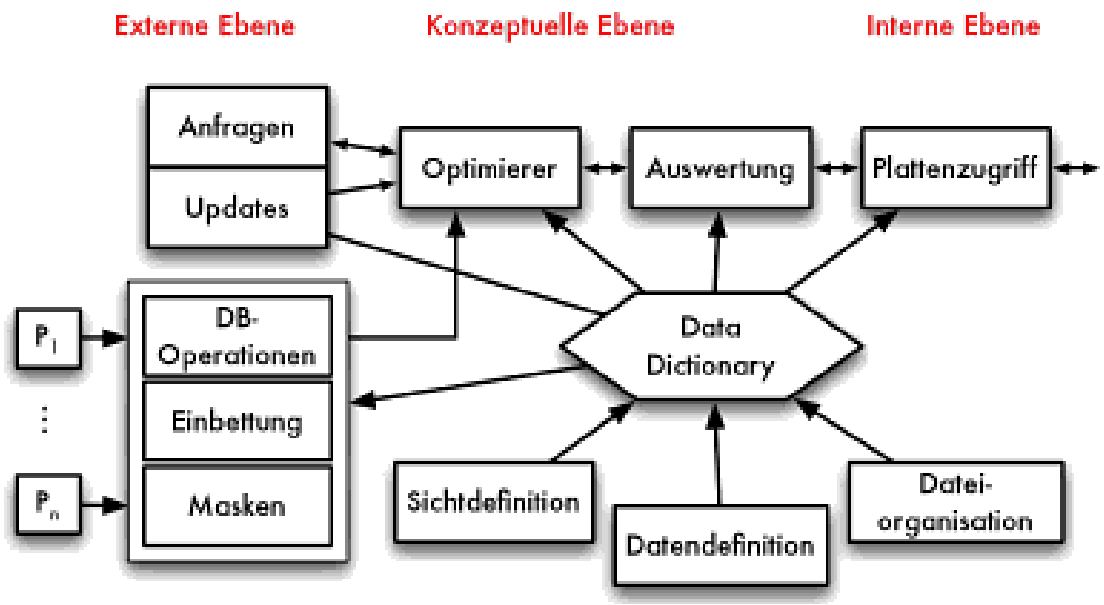


Abbildung 2.4: Schematische Architektur eines DBMS, Quelle: [SSH11, S. 3]

[15] [Mic15c]
[16] [SSH11, S. 3/4]

Alle aktuellen Datenbanksysteme haben nachfolgende Verantwortlichkeiten:

- Bereitstellung eines zuverlässigen Datenspeichers für alle Daten einer Anwendung.
- Bereitstellen einer schnellen Methode, um auf die gespeicherten Daten zu zugreifen.
- Bereitstellen eines einheitlichen Zugriffsweges auf die gespeicherten Daten.
- Kontrollieren des Zugriffes auf die Daten durch Sicherheitseinstellungen und Richtlinien.
- Sicherstellen der Datenintegritätsregeln zur Sicherung der Richtigkeit und Konsistenz der zu verwaltenden Daten. [17]

2014 veröffentlichte das Unternehmen Gartner Inc., mit Hauptsitz in Stamford im US Bundesstaat Connecticut, eine Übersicht des Marktes für Anbieter von Datenbankmanagementsystemen. Inhalt dieser Erhebung war der Vergleich der angebotenen Produkte von ihrer Leistungsfähigkeit bis hin zur Güte des Supports, welchen die betreffende Firma liefert. Im Ergebnis der Erhebung kristallisierte sich eine aus vier Unternehmen bestehende Gruppe als Marktführer heraus. So wird die Firma ORACLE auf Platz eins eingestuft und MICROSOFT an zweiter Stelle. SAP und IBM vervollständigen die Gruppe mit Platz drei und vier.[18]

[17] [RBJ03, S. 15]
[18] [Gar14]

2.2.2 Ablauf einer SQL-Abfrage

Jedes SQL-Statement, das ein Datenbankserver abarbeitet, wird nach einer immer wiederkehrenden Reihenfolge verarbeitet. Abbildung 3.1 zeigt die logische Reihenfolge der Befehlsbearbeitung für ein SELECT-Befehl auf einem SQL-Server.

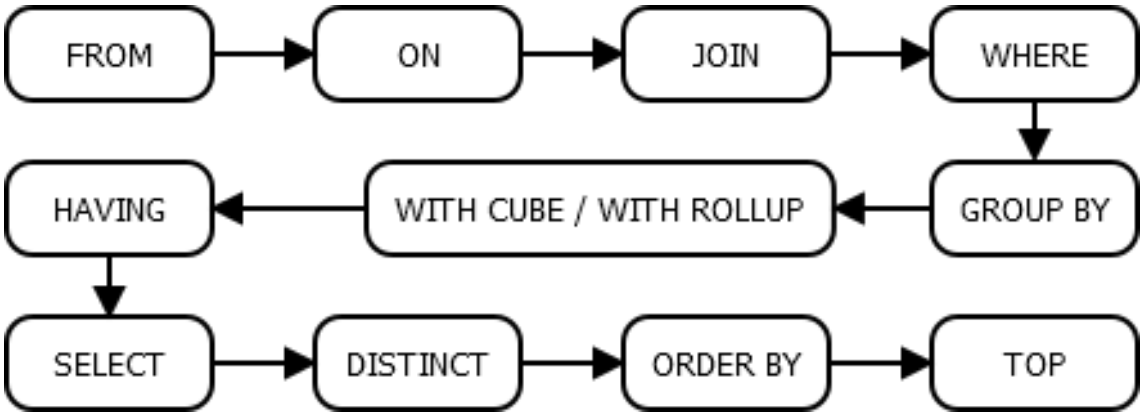


Abbildung 2.5: Abfragereihenfolge eines SQL-Befehls, Quelle: Eigene Darstellung

Gut zu erkennen ist, dass erst die gewünschten Daten zusammengetragen werden und anschließend die entstandene Datenmenge gemäß der verwendeten Einschränkungen soweit verkleinert wird bis das gewünschte Ergebnis selektiert werden kann. Erst nach der Selektierung erfolgt eine Ordnung oder eine Beschränkung eine bestimmte Anzahl von Ergebnissen.[19]

Wichtig ist hierbei die Beachtung der Priorität der Einschränkungsklauseln. Die *WHERE* Bedingung wird beispielsweise deutlich früher abgearbeitet als die *HAVING* Klausel. Somit schränken Elemente, die sich im WHERE-Teil befinden, das Statement deutlich stärker und früher ein, als nachfolgende Befehlsteile.[20]

Listing 2.3 veranschaulicht ein Beispiel-Statement für einen SELECT-Befehl zur Abfrage aller deutsch sprechenden Kunden mit ihren Namen, Adressen und ihrem Wohnsitz in Italien.

```
1 SELECT Name, Address, City
2  FROM Customers
3  WHERE Language Code LIKE 'de'
4  HAVING Country/Region Code LIKE 'I'
5  ORDER BY Name;
```

Listing 2.3: SQL Beispiel, Quelle: Eigenes Listing

[19] [Mic15e]
[20] [Sch09, S. 32]

2.2.3 Der physikalische Ausführungsplan

Der Abfrageprozessor des SQL-Servers verarbeitet SQL-Befehle gemäß eines vorher erstellten Ablaufplans. Zur Erzeugung dieses Plans sind drei Schritte von Nöten. Diese Schritte werden von drei separaten Modulen übernommen. Im Folgenden werden diese Komponenten näher erläutert.[21]

Parser

Der Parser ist der erste Schritt bei der Erstellung eines Statementausführungsplans. Er führt Syntaxüberprüfungen durch und kontrolliert beispielsweise auch die richtige Schreibweise der angegebenen Tabellen und Tabellenspalten. Das Ergebnis des Parsvorgangs ist eine Baumstruktur der Ausführungslogik des gerade auszuführenden SQL-Befehls.[22]

Algebrizer

Der Algebrizer verarbeitet den im ersten Schritt vom Parser erzeugten Ablaufbaum. Im Verlauf dieses Schrittes werden die Tabellen und Tabellenspalten geprüft und zugeordnet und die angegebenen Datentypen kontrolliert. Der Ausgangsbaum wird im Verlauf weiter vereinfacht und normalisiert. Hierbei werden unter anderem redundante Operationen entfernt. Mit Abschluss aller Tätigkeiten des Algebrizers liegt ein vereinfachter Baum vor, der die zu bearbeitende Abfrage logisch widerspiegelt. Dieser Baum wird abschließend an den Optimierer übergeben[23]

Optimizer

Den letzten Bearbeitungsschritt übernimmt der Optimizer. Durch den Optimizer wird ein physikalischer Ausführungsplan generiert. Der erstellte Plan besteht aus diversen physikalischen Operatoren. So zum Beispiel *Clustered Index Seek*, *Compute Scalar* oder auch *Select*.

Während des Erstellungsprozesses werden unter anderem die zu verwendenden Indizes, die Reihenfolge in der auf die beteiligten Datenbanktabellen zugegriffen wird und auch die Joins zwischen den Tabellen festgelegt.

Bei den meisten SQL Abfragen erstellt der Optimizer mehrere Pläne, die alle zum selben Ergebnis führen, und ermittelt beziehungsweise vergibt Kosten für die einzelnen Operatoren. Er erstellt aus zeitlichen Gründen nie alle möglichen Ausführungspläne. Im Endeffekt wird dann der Plan mit den geringsten Gesamtkosten ausgewählt und durchgeführt. Der Wert der Zeit, die benötigt wird ein Abfrageergebnis zu ermitteln, ist hierbei der einzige berücksichtigte Kostenfaktor.

[21] [Sch09, S. 34]
[22] [Sch09, S. 34]
[23] [Sch09, S. 35]

Die durch den Optimizer ermittelten und dem Benutzer ausgegeben Kosten sind allesamt vom Server geschätzt und entsprechen keiner dem Menschen bekannten zeitlichen Einheit. Es wird eine dem Optimizer eigene Uhr mit einer für ihn speziellen Zeiteinheit verwendet. Die somit bekanntgegebene Zahl ist als eine Kennzahl zu verstehen, welche einen direkten Bezug zur Ausführungsdauer hat, aber nicht linear ist. Letzt endlich kann man in Bezug auf die Plankosten festhalten, dass diese Kosten bei einer Abfrage so gering wie nur irgend möglich sein sollten. [24]

Ablauf einer Ablaufplanoptimierung

1. Suchen eines trivialen Plans

Zu Beginn eines Optimierungsvorgangs wird geprüft, ob ein trivialer Plan existiert. Ist eine SQL Abfrage einfach genug, so existiert unter Umständen nur ein einziger möglicher Durchführungsplan. Kann ein trivialer Plan gefunden werden, so ist die Optimierung bereits im ersten Schritt beendet. Bei der Suche nach einem trivialen Plan nutzt der Optimierer auch die passenden Meta-Daten zu den betroffenen Objekten, um das Ergebnis zu verbessern.[25]

2. Vereinfachung der Abfrage

Existiert kein trivialer Plan, so wird im zweiten Schritt versucht, die Abfrage zu vereinfachen. Hierzu werden in der Regel syntaktische Umwandlungen oder Neuordnungen der Operationen durchgeführt. Das gängigste Beispiel ist die Umwandlung kostenintensiver OUTER JOIN Befehle in kosteneffizientere INNER JOIN Befehle. [26]

3. Erstellen mehrerer Pläne und deren Kostenvergleich

Im Anschluss an die Statement-Vereinfachung werden weitere Pläne erzeugt und deren Kosten verglichen. Ziel ist es, einen Plan zu finden dessen Kosten kleiner als 0,2 sind. Bei Auffinden dieses Plans ist die Optimierung beendet. Die Optimierung wird ebenfalls beendet, wenn alle möglichen Pläne kontrolliert wurden.[27]

4. Versuche mit größeren Kostenschwellen

Sind die vorhergehenden Schritte ohne Ergebnis durchlaufen worden, so werden stärkere Optimierungen vorgenommen. Dies schlägt sich in einer größeren Zahl von Vertauschungen und Ersetzungen nieder. Ziel hierbei ist, dass Auffinden eines Ablaufplans mit Kosten kleiner 1,0. Bei Finden dieses entsprechenden Plans wird die Optimierung an dieser Stelle beendet.[28]

[24] [Sch09, S. 35]
[25] [Sch09, S. 37]
[26] [Sch09, S. 37]
[27] [Sch09, S. 38]
[28] [Sch09, S. 38]

5. Testen von parallelen Ausführungsplänen

Waren alle Bemühungen der Optimierung bisher erfolglos, so werden jetzt Pläne erstellt, die mehr als eine CPU des betroffenen Servers nutzen können. Dazu müssen zwei Kriterien erfüllt sein:

- Der Server muss mehr als eine CPU besitzen.
- Die bisherigen Plankosten müssen über dem Wert des Parameters *cost treshhold* liegen. Der Standardwert beträgt 5.

Ist dies der Fall, wird die vorherige Optimierungsphase erneut durchlaufen.

Abschließend werden der günstigste der einfachen Pläne und der neu erstellte parallele Ausführungsplan verglichen. Der entsprechend billigere wird abschließend ein weiteres Mal optimiert und dann ausgeführt.

Zusammenfassend kann man die durch den Optimierungsprozess erstellten Pläne in drei Gruppen teilen:

- Triviale Pläne: Der Plan ist optimal. Eine weitere Optimierung ist nicht möglich.
- Optimale Pläne: Alle Varianten sind geprüft worden. Es wird der Kosteneffizienteste Plan genutzt.
- Fast optimale Pläne: Das auszuführende Statement ist zu komplex, um alle Pläne kontrollieren zu können. Der genutzte Ausführungsplan ist der beste Plan, der gefunden wurde. [29]

2.2.4 Werkzeuge zur Analyse

Zur Analyse der Leistung von abgearbeiteten SQL-Befehlen stehen die verschiedensten Werkzeuge zur Auswahl und Verfügung. Im Folgenden wir sich auf eine kleine Auswahl beschränkt.

Einfache Werkzeuge

Das einfachste „Werkzeug“ zur Leistungsmessung ist die Möglichkeit des Servers, die Statement-Ausführungszeit zu messen. Die Zeit und die Anzahl der betroffenen Zeilen wird an der rechten unteren Ecke des Ausführungsfensters angezeigt oder kann mittels SQL-Script ermittelt werden. Zu Protokollierungszwecken kann dieser Wert auch in einer Datenbanktabelle gespeichert werden. Eine tiefere Analyse ist nicht möglich, aber es wird ein schneller Überblick über die Laufzeiten gegeben.[30]

Weiterhin können statistische Größen zu Ausführungszeiten, Übersetzungszeiten und Eingabe/Ausgabe-Vorgängen erhoben werden. So kann mittels *SET STATISTICS*

[29] [Sch09, S. 38]
[30] [Sch09, S. 48]

IO ON bzw. *SET STATISTICS IO OFF* das Erfassen der von einem SQL-Befehl benötigten I/O Operationen ein- oder ausgeschaltet werden. Mit Hilfe dieser Statistik können unter anderem Rückschlüsse über die benötigte Anzahl von *READ-AHEAD-Lesevorgängen* oder auch *logische Lesevorgängen* gezogen werden.

Durch das Erfassen von *STATISTICS TIME* wird die Abfrage- bzw. Übersetzungszeit eines Befehls protokolliert. Diese Zeiten werden zweigeteilt. So wird die Gesamtzeit der Ausführungsplanerstellung bzw. Befehlsausführung angegeben und durch den Anteil der von der CPU benötigten Zeit ergänzt.[31]

Aktivitätsmonitor

Der Aktivitätsmonitor des SQL Servers wird schon seit der Version 6.5 mitgeliefert und wurde in den vergangenen Jahren immer weiter verändert und weiterentwickelt. Er umfasst fünf Bereiche in denen die überwachten Informationen unterteilt sind.

Im Bereich eins mit der Bezeichnung „Übersicht“ werden Prozessorzeit, wartende Tasks, Datenbank E/A und die Anzahl der Batchanforderungen grafisch aufbereitet.

Zweitens werden unter „Prozesse“ alle aktuell verbundenen Benutzer mit verschiedensten Informationen gelistet. Hier sind zum Beispiel der Anmeldename, der genutzte Datenbankname oder Wartezeit und Wartetyp aufgelistet.[32]

Es folgt ein Segment mit der Bezeichnung „Ressourcenwartevorgänge“ dessen Inhalt eine Auflistung der Wartekategorien, beispielhaft seien Memory, Network I/O und Buffer I/O genannt, und die statistische Wartezeit von Nutzeranfragen ist.

Der vorletzte Bereich „Datendatei-E/A“beherbergt eine Momentaufnahme von E/A Vorgängen für jede Datenbank des Servers. Ermittelt werden die gelesenen MB/Sekunde, geschriebene MB/Sekunde und die Antwortzeit des Servers.

Abgerundet wird der Aktivitätsmonitor durch die Auflistung „Aktuelle wertvolle Abfragen“. Hierin ist eine Tabelle der *wichtigen* SQL-Abfragen dargestellt. Der Server gibt unter anderem die Anzahl der Befehlsausführungen pro Minute, die dazugehörige CPU Zeit oder auch die Zahl der logischen Schreib- und Lesevorgänge aus.[33]

Da die vom Aktivitätsmonitor publizierten Daten durchaus von Interesse sind, aber meist nur zeitlich begrenzte Werte beinhalten, nutzt er nur für einen guten Überblick aber nicht für Langzeitanalysen.

[31] [Sch09, S. 49]
[32] [Sch09, S. 51]
[33] [Sch09, S. 52]

Ablaufverfolgung

Unter einer Ablaufverfolgung versteht man das Aufzeichnen, im Englischen *tracen* genannt, aller Aktivitäten auf einem bestimmten SQL Server über einen beliebig langen Zeitraum. Diese Aktivitäten spiegeln sich in vielen verschiedenen Ereignissen, Events genannt, wider und werden in mehrere Bereiche unterteilt. Jeder Event hat zudem zahlreiche Daten die erfasst werden können. Nach Abschluss des Vorgangs kann der Nutzer den Trace in Form einer Datei, im *.trc* oder *.xml* Format, auf dem Server selbst oder einem anderen Netzlaufwerk speichern. Somit ist eine Analyse zu jedem beliebigen Zeitpunkt und an jedem Ort möglich, an dem ein SQL Server installiert ist.

Um eine Ablaufverfolgung auf einem Server zu erstellen, gibt es zwei geeignete Methoden. Die erste und einfachste Art ist die Erstellung durch den SQL Server Profiler. Er bietet eine grafische Oberfläche mit deren Hilfe der betreffende Nutzer leicht die von ihm gewünschten Ereignisse und passenden Daten auswählen kann. Die Daten können manuell, Spalte für Spalte und Event für Event, ausgewählt werden oder es wird eine der mitgelieferten oder selbst erstellten Vorlagen eingesetzt.

Vorteilhaft für den Profiler ist die einfache Bedienbarkeit und die Übersichtlichkeit. Allerdings kann sich die grafische Oberfläche negativ auf die Leistung des Servers auswirken, da er jeden auftretenden Event visualisieren muss.[34]

Die zweite Möglichkeit der Ablaufverfolgung ist ein sogenanntes Server-Side-Trace. Hierbei wird der Trace auf Serverebene mittels T-SQL erstellt und gestartet bzw. angehalten.

Vorteilhaft sind die höhere Leistung aufgrund des Fehlens einer Visualisierungsebene und die Möglichkeit, die Einstellungen des Mitschnitts in einer *Stored Procedure* dauerhaft zu speichern um diese gegebenenfalls auch auf anderen Servern nutzen zu können. Nachteilig ist der Umstand, dass eine GUI fehlt, dahingehend, weil der betroffene Nutzer jeden gewünschten Event und dazugehörige Datenspalte aus dem Kopf wissen muss oder sich anderweitig über sie informieren muss. [35]

[34] [Sch09, S. 53]
[35] [Sch09, S. 54]

2.3 XML

Die Extensible Markup Language, abgekürzt XML, wurde vom der XML Working Group entwickelt und vom World Wide Web Consortium, kurz W3C, als Standard festgehalten. XML ist ein einfaches textbasiertes Dateiformat, um strukturierte Informationen abzubilden. Es wurde vom älteren Standardformat SGML, ISO 8879, abgeleitet, um den modernen Anforderungen des Web zu entsprechen.[36]

Die Entwurfsziele von XML sind:[37]

1. XML soll unkompliziert im ganzen Internet nutzbar sein.
2. XML soll eine große Bandbreite an Anwendungen unterstützen.
3. XML soll kompatibel zum SMGL sein.
4. Die Anzahl optionaler Eigenschaften von XML soll am absoluten Minimum bleiben.
5. XML Dokumente sollen lesbar und möglichst klar strukturiert sein.
6. Das XML Design sollte schnell vorbereitet sein.
7. Das Design von XML soll formal und präzise sein.
8. XML Dokumente sollen einfach zu erstellen sein.
9. Die Kürze von XML Dokumenten minimaler Wichtigkeit.

Das unten stehende Listing 2.4 zeigt ein Standardbeispiel eines XML Dokumentes. So ist zu erkennen, dass ein XML Dokument einen Wurzelknoten, in diesem Fall *personal*, aufweist und weitere Informationen in sogenannten Kind-Knoten, hier *name*, gespeichert werden. Jeder Knoten kann Attribute besitzen. So ist *Firma* mit dem Wert *Muster GmbH* das Attribut des Wurzelknotens. Der eigentliche Inhalt eines Knotens befindet sich zwischen dem Start- und End-Tag des Knotens. Im gezeigten Beispiel durch *Max Mustermann* im ersten Knoten *name* zu sehen.

```
1 <personal Firma="Muster GmbH">
2   <name id="1" seit="2013">Max Mustermann</name>
3   <name id="2" seit="2014">Martina Mustermann</name>
4 </personal>
```

Listing 2.4: XML Standardbeispiel, Quelle: Eigenes Listing

[36] [Qui]
[37] [W3C06]

3 Konzeption

3.1 Anforderungen an die Software

An jedes neu entwickelte Produkt werden von den Nutzern oder Auftraggebern die verschiedensten Anforderungen gestellt. Diese Anforderungen werden zum Einen in Anforderungen über den Funktionsumfang beziehungsweise das Verhalten des Produktes und zum Anderen in allgemeine nicht die Funktionen betreffende Anforderungen unterteilt. Die für das in dieser Arbeit erstellte Programm relevanten Anforderungen wurden im Rahmen eines Gespräches mit dem verantwortlichen Betreuer und dem Entwicklungsleiter der msu festgelegt.

Funktionale Anforderungen

Die fertige Software soll dem Nutzer ermöglichen:

- Ablaufverfolgungen anzulegen,
- beliebige Trace-Dateien in die Datenbank zu laden,
- die durch die Ablaufverfolgung generierten SQL Befehle einzusehen,
- die erfassten SQL Befehle zu den aufrufenden Codebestandteilen im C/AL Quellcode zurückzuverfolgen,
- die Navision-Nutzer zu identifizieren, welche die SQL Befehle erzeugt haben,
- zwei unterschiedliche Traces miteinander zu vergleichen,
- statistische Daten über die Traces und SQL Statements zu erheben.

Nichtfunktionale Anforderungen

Als Rahmenbedingungen wurden die folgenden Punkte erfasst:

- Erreichen einer möglichst hohen Wartbarkeit, durch das Einhalten des Navision Style guides.
- Eine möglichst hohe Reaktionszeit auch bei großen Datenmengen ist anzustreben.
- Die Software soll möglichst in Microsoft Dynamics Nav mit Bordmitteln umgesetzt werden.
- Die Benutzerführung hat in deutscher Sprache zu erfolgen.

3.2 Anwendungsfälle

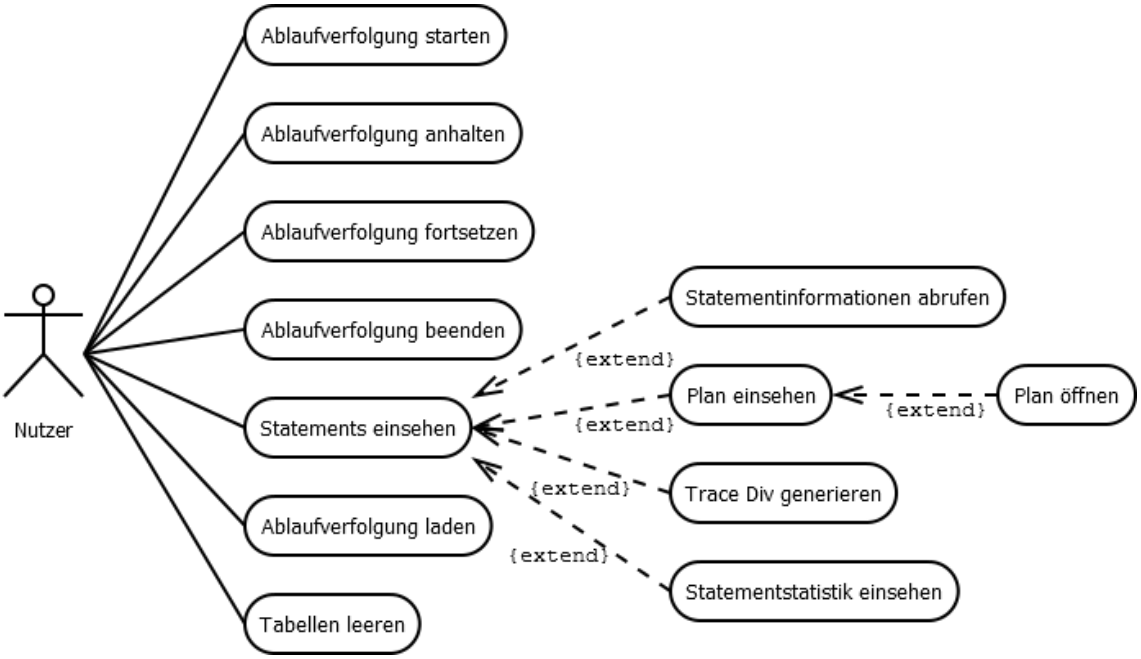


Abbildung 3.1: Anwendungsfälle des C/AL Programms, Quelle: Eigene Darstellung

Name: Ablaufverfolgung starten
Kurzbeschreibung: Der Nutzer betätigt den Button *Ablaufverfolgung starten*
Vorbedingung: Nutzer muss die Anwendung gestartet haben
Nachbedingung Erfolg: Der Trace arbeitet
Nachbedingung Fehlschlag: Es erscheint eine Fehlermeldung
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte eine Ablaufverfolgung erstellen
Standardablauf:
1. Nutzer klickt auf den Knopf *Ablaufverfolgung starten*
2. System startet den Namensvergabedialog
3. Nutzer vergibt gewünschten Tracenamen
4. System startet den Server-Side-Trace
5. System generiert eine Datei auf dem Server
6. System legt den Namen sowie die ID des Traces in der Datenbank ab

Erweiterung: -

Name: Ablaufverfolgung anhalten
Kurzbeschreibung: Der Nutzer betätigt den Button *Ablaufverfolgung anhalten*
Vorbedingung: Nutzer muss eine Ablaufverfolgung gestartet haben
Nachbedingung Erfolg: Der Trace ist unterbrochen
Nachbedingung Fehlschlag: Es erscheint eine Fehlermeldung
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte eine Ablaufverfolgung unterbrechen
Standardablauf:

1. Nutzer klickt auf den Knopf *Ablaufverfolgung anhalten*
2. System liest den Namen und die Trace ID aus der Datenbank
3. System unterbricht den Trace

Erweiterung: -

Name: Ablaufverfolgung fortsetzen
Kurzbeschreibung: Der Nutzer betätigt den Button *Ablaufverfolgung fortsetzen*
Vorbedingung: Nutzer muss die Ablaufverfolgung angehalten haben
Nachbedingung Erfolg: Der Trace arbeitet
Nachbedingung Fehlschlag: Es erscheint eine Fehlermeldung
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte eine Ablaufverfolgung fortsetzen
Standardablauf:
1. Nutzer klickt auf den Knopf *Ablaufverfolgung fortsetzen*
2. System liest den Namen und die Trace ID aus der Datenbank
3. System setzt den Trace fort

Erweiterung: -

Name: Ablaufverfolgung beenden
Kurzbeschreibung: Der Nutzer betätigt den Button *Ablaufverfolgung beenden*
Vorbedingung: Nutzer muss die Ablaufverfolgung angehalten haben
Nachbedingung Erfolg: Der Trace ist beendet
Nachbedingung Fehlschlag: Es erscheint eine Fehlermeldung
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte eine Ablaufverfolgung beenden
Standardablauf:
1. Nutzer klickt auf den Knopf *Ablaufverfolgung beenden*
2. System liest den Namen und die Trace ID aus der Datenbank
3. System beendet den Trace

Erweiterung: -

Name: Ablaufverfolgung laden
Kurzbeschreibung: Der Nutzer betätigt den Button *Ablaufverfolgung laden*
Vorbedingung: Nutzer muss die zu ladende Ablaufverfolgung in der Liste markiert haben
Nachbedingung Erfolg: Alle nachgelagerten Tabellen sind befüllt
Nachbedingung Fehlschlag: Es erscheint eine Fehlermeldung
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte eine Ablaufverfolgung in das System laden
Standardablauf:
1. Nutzer klickt auf den Knopf *Ablaufverfolgung laden*
2. System liest den Namen und die Trace ID aus der Datenbank
3. System startet einen Dialog mit einer Information für den Nutzer
4. System importiert die gewünschten Tracedaten in die Datenbank
5. System beendet den Infodialog für den Benutzer und wartet auf weitere Eingaben

Erweiterung: -

Name: Tabellen leeren
Kurzbeschreibung: Der Nutzer betätigt den Button *Tabellen leeren*
Vorbedingung: System muss Datensätze enthalten
Nachbedingung Erfolg: Alle für die Ablaufverfolgung relevanten Tabellen sind geleert
Nachbedingung Fehlschlag: Es geschieht nichts
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte alle Ablaufverfolgungen aus dem System löschen
Standardablauf:
1. Nutzer klickt auf den Knopf *Tabellen leeren*
2. System löscht alle betroffenen Tabellen
Erweiterung: -

Name: Statements einsehen
Kurzbeschreibung: Der Nutzer betätigt den Button *Statements einsehen*
Vorbedingung: Die entsprechende Ablaufverfolgung muss selektiert sein
Nachbedingung Erfolg: Eine Liste aller gewünschten Statements ist geöffnet
Nachbedingung Fehlschlag: Es geschieht nichts
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte Einsicht in die SQL Befehle einer speziellen Ablaufverfolgung
Standardablauf:
1. Nutzer klickt auf den Knopf *Statements einsehen*
2. System öffnet die Liste der betroffenen Statements
Erweiterung: Plan einsehen, Trace Div generieren, Statementstatistik einsehen

Name: Statementinformationen abrufen
Kurzbeschreibung: Der Nutzer macht einen Doppelklick auf das ihn interessierende SQl Statement
Vorbedingung: Die Statementliste muss geöffnet sein
Nachbedingung Erfolg: Die Detailinformationen zum gewählten Statement sind geöffnet
Nachbedingung Fehlschlag: Es geschieht nichts
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte Einsicht in die Details eines SQL Befehls
Standardablauf:
1. Nutzer doppelklickt auf ein Statement
2. System öffnet die Detailinformationen des gewählten Statements
Erweiterung: -

Name: Plan einsehen
Kurzbeschreibung: Der Nutzer betätigt den Button *Plan einsehen*

Vorbedingung: Die entsprechendes Statement muss selektiert sein
Nachbedingung Erfolg: Die *Plan Card* wird angezeigt
Nachbedingung Fehlschlag: Es geschieht nichts
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte Einsicht in den Plan eines SQL Befehls
Standardablauf:
1. Nutzer klickt auf den Knopf *Plan einsehen*
2. System öffnet eine Ansicht der Informationen zu einem Plan
Erweiterung: Plan öffnen

Name: Plan öffnen
Kurzbeschreibung: Der Nutzer betätigt den Button *Plan öffnen*
Vorbedingung: Die entsprechende *Plan Card* muss geöffnet sein
Nachbedingung Erfolg: Der gewünschte Plan wird mittels SQL Server Management Studio angezeigt
Nachbedingung Fehlschlag: Es geschieht nichts
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte den betreffenden Plan einsehen
Standardablauf:
1. Nutzer klickt auf den Knopf *Plan öffnen*
2. System erstellt eine SQLPlan Datei
3. System startet das SQL Server Management Studio mit der vorher erstellten Datei als Parameter
Erweiterung: -

Name: Trace Div generieren
Kurzbeschreibung: Der Nutzer betätigt den Button *Trace Div generieren*
Vorbedingung: Die Statementliste muss geöffnet sein
Nachbedingung Erfolg: Das Programm WinMerge zeigt die Unterschiede zweier gewählter Ablaufverfolgungen
Nachbedingung Fehlschlag: WinMerge öffnet sich mit einer Fehlermeldung
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte zwei Traces vergleichen
Standardablauf:
1. Nutzer klickt auf den Knopf *Trace Div generieren*
2. System öffnet ein Abfragefenster für die Nummern der zu vergleichenden Ablaufverfolgungen
3. Nutzer bestätigt die Auswahl
4. System generiert zwei Textdateien
5. System startet das Programm WinMerge mit den beiden vorher erstellten Dateien
Erweiterung: -

Name: Statementstatistiken einsehen
Kurzbeschreibung: Der Nutzer betätigt den Button *Statementstatistiken einsehen*
Vorbedingung: Die Statementliste muss geöffnet sein
Nachbedingung Erfolg: Die Statistiken der im gewählten Trace enthaltenen SQL Befehle wird angezeigt
Nachbedingung Fehlschlag: Es geschieht nichts
Akteur: Nutzer
Auslösendes Ereignis: Der Nutzer möchte erfahren wie oft bestimmte SQL Befehle auftreten und wie viel Leistung sie in Anspruch nehmen
Standardablauf:
1. Nutzer klickt auf den Knopf *Statementstatistiken einsehen*
2. System öffnet die Statistikübersicht
Erweiterung: -

3.3 SQL-Trace

Um eine Aussage in Bezug auf die Laufzeit oder Leistungsprobleme eines Listenaufrufes oder einer Stapelverarbeitung treffen zu können, müssen möglichst passende und genaue Daten erhoben werden. Hierfür wird die, im Grundlagenkapitel vorgestellte, Möglichkeit einer Ablaufverfolgung auf dem SQL-Server genutzt.

Für die Leistungsmessung und Beurteilung wird sich auf eine Menge von vier Ereignissen beschränkt. Bei den gewählten Ereignissen wird davon ausgegangen, dass sie den besten Überblick gewähren.

Das wichtigste Ereignis ist *SP: StmtCompleted* mit der Eventklasse 45 aus dem Bereich *Stored Procedures*. Hierin sind die abgearbeiteten SQL-Befehle im Klartext dargestellt. Ergänzt wird es durch das Ereignis *Showplan XML* aus dem Bereich der Performance mit der Eventklasse 122 in dem der zum SQL-Statement passende Durchführungsplan enthalten ist. Außerdem wird der Event *RPC: Completed* aus dem Bereich *Stored Procedures* und der Klasse 10 erfasst. In ihm sind die zu den SQL-Befehlen passenden, vom Server erstellten, Stored Procedures mitgeschnitten. Der letzte der vier betrachteten Events ist *SQL: BatchCompleted* aus dem Trace-Bereich *TSQL* mit der Eventklasse 12. Dieser Event dient dazu, einen am Navision arbeitenden Nutzer anhand von mitgeschnittenen Kommentaren zu identifizieren und mit den auftretenden SQL-Statements in Verbindung bringen zu können. Voraussetzung hierfür ist das Aktivieren der Option *SQL-Ablaufverfolgung* in der Sessionübersicht des zu überprüfenden Navisionbenutzers.

Die Tabellen 3.1 und 3.2 zeigen alle Daten, welche für die einzelnen Events vom Server mitgeschnitten werden sollen.

Ereignis	**TextData**	**Duration**	**SPID**	**Eventclass**	**DatabaseID**	**DatabaseName**
Showplan XML	x		x	x	x	x
SP: StmtCompleted	x	x	x	x	x	x
RPC: Completed	x	x	x	x	x	x
SQL: BatchCompleted	x	x	x	x	x	x

Tabelle 3.1: Tabelle 1 mit mitgetraceten Daten, Quelle: Eigene Tabelle

Ereignis	**CPU**	**Reads**	**Writes**	**TransactionID**	**HostName**
Showplan XML				x	x
SP: StmtCompleted	x	x	x	x	x
RPC: Completed	x	x	x	x	x
SQL: BatchCompleted				x	x

Tabelle 3.2: Tabelle 2 mit mitgetraceten Daten, Quelle: Eigene Tabelle

3.4 Datenmodell

Die Datenbasis des in dieser Arbeit erstellten C/AL Programms bilden sieben Tabellen. Sie nehmen die mitgeschnittenen Daten der Ablaufverfolgung und ihre Statistiken auf. Bei der Erstellung von Objekten in Navision 2015 muss einer speziellen Namenskonvention gefolgt werden und die Objekte müssen in einem fest definierten ID-Bereich erstellt werden. Für neue Projekte, ohne sofortigen Einsatzzweck für eine spezielle Firma, liegt dieser ID-Bereich zwischen 50000 und 99999. Um einen freien und möglichst ungenutzten Nummernbereich zu belegen, werden alle im Zuge dieser Arbeit erstellten Tabellen im Abschnitt 60001 bis 60008 angelegt.

Tabelle 3.3 listet alle im Laufe der vorliegenden Arbeit erstellten Tabellen, mit ihren ID's und Namen, auf.

ID	**Name**
60001	PA StatementPlan
60002	PA Statement
60003	PA Comment
60004	PA Procedur
60005	PA Statement Statistics
60006	PA Trace
60007	PA Statistic Overview

Tabelle 3.3: Navison Tabellennamen und ID's, Quelle: Eigene Tabelle

Der verwendete Tabellenname ergibt sich aus der Bezeichnung der Tabelle in Abbildung 3.2 und dem Tag des Projektes in der Fima selbst. Das Tag entspricht der Kurzform von *Performance Analyse*. Die Abbildung 3.2 zeigt das gesamte Datenmodell, welches in Navision und auf dem SQL Server angelegt ist.

Wurzel des Programms ist die Tabelle *Trace*. In ihr werden der Dateiname der erstellten Ablaufverfolgung sowie die vom SQl Server vergeben Trace ID gespeichert. Primärschlüssel ist eine fortlaufende Zahl. Das Speichern der Server Trace ID ermöglicht außerdem das Starten mehrerer Traces parallel.

Von der Wurzel sind drei Tabellen direkt abhängig. Tabelle eins trägt den Namen *Statistic Overview* und nimmt allgemeine Informationen zu einem Trace auf. Hierbei werden die Laufzeiten der im Trace enthaltenen SQL Befehle und Stored-Procedures sowie die Gesamtzahl von SQL Befehlen und Prozeduren gespeichert.

In der zweiten Statistiktabelle *Statement Statistics* werden zu den Textvorschauen und Hash-Werten der in einem Trace auftretenden SQL Befehle außerdem die Anzahl ihrer Durchläufe, die kumulative Laufzeit und CPU Verbrauch sowie Lese- und Schreibzugriffe erfasst. Dies erleichtert dem Nutzer die Suche nach Statements, welche alleine eine geringe Laufzeit haben aber durch vermehrtes Abarbeiten einen hohen aufsummierten Zeitaufwand aufweisen.

Die dritte dem Trace untergeordnete Tabelle *Statement* ist die Zweitwichtigste im Programm. Sie enthält die für jeden SQL Befehl relevanten Daten und von ihr sind weitere drei Tabellen abhängig. Diese Tabelle ist die Basis für eine Identifikation von Statements mit hohen Einzellaufzeiten.

Zur Sammlung des zu einem Statement gehörigen SQL-Ausführungsplans dient die Tabelle *StatementPlan*. Für einen solchen Plan sind Abarbeitungszeiten oder Lese- und Schreibzugriffe nicht erfassbar. Deshalb wird sich auf den Plan in Textform, eine Vorschau, den zugehörigen Hash-Wert und die Transaktionsnummer beschränkt.

Eine Identifikation von verantwortlichen Nutzern oder Codeunits, die für mitgeschnittene SQL Befehle verantwortlich zeichnen, werden die Kommentare in der Tabelle *Comment* angesammelt. Von Interesse sind hierbei neben dem eigentlichen Text und dessen Hash auch die betreffende Laufzeit und Transaktionsnummer.

Als letzte erstellte Tabelle dient *Procedure* zur Aufnahme aller in der Nachverfolgung erfassten Prozeduren. Eine Prozedur kann einen oder mehrere SQL Befehle beinhalten und besitzt einen eigenen Leistungsverbrauch. Alle Laufzeiten, CPU-Verbräuche sowie Schreib- und Lesezugriffe werden zum erfassten Leistungsverbrauch der Prozedur aufsummiert. Für ein vollständiges Bild werden auch hier der Text, sein Hash-Wert als auch eine Textvorschau erfasst.

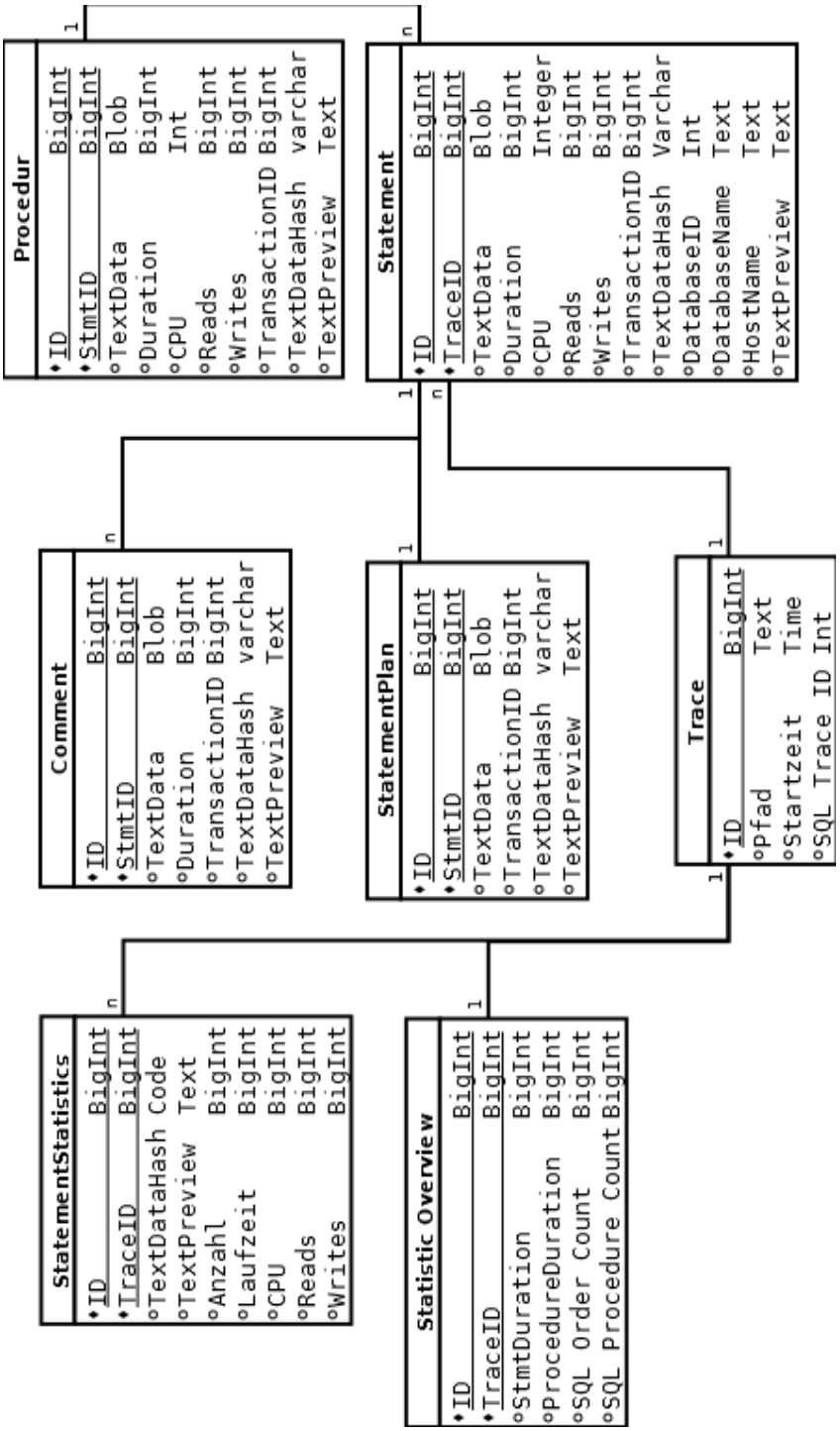


Abbildung 3.2: Datenmodell der Navision Tabellen, Quelle: Eigene Darstellung

3.5 SQL-Script

Auf SQL Server Seite sollen vier SQL-Scripte ihre Arbeit verrichten. Das erste Script dient zur Ausführung der SQL Ablaufverfolgung. Es soll durch Navision gestartet und beendet werden können. Der Output ist eine bei Start der Aktion festgelegte Datei mit der Endung *.trc*. Die genutzte Dateigröße muss ausreichend gewählt werden und darf den zur Verfügung stehenden Festplattenspeicher allerdings nicht überschreiten.

Den Aufgabenbereich des Einlesens der Datei in die Datenbank und das damit verbundene Erstellen von Statistiken über die geladenen Ablaufverfolgungen übernimmt das zweite SQL-Script. Während des Einlesens sind alle relevanten Datenbanktabellen mit passenden Daten zu bestücken und gegebenenfalls von überflüssigen beziehungsweise unbrauchbaren Datenbeständen zu säubern.

Scripte Nummer Drei und Vier sind zum Einen für das Leeren der Datenbanktabellen des zu erstellenden Performance-Analyse-Tools und zum Anderen zum Erstellen der Übersicht der Unterschiede zweier beliebiger Traces vorgesehen.

3.6 C/AL Programm

Um eine für Navisionnutzer gewohnte Arbeitsumgebung zu gewährleisten, werden alle Oberflächen mit Navision eigenen Mitteln erstellt. So müssen für jede spezifizierte Tabelle eine Listenansicht oder eine Karte, für die Ansicht eines einzelnen Datensatzes, in Form von *Pages* angelegt sein.

Auch die erstellten Pages müssen der allgemeingültigen Namenskonvention unterworfen werden. So wird unter anderem die Listenanzeige der Statements unter *PA Statement List* mit der ID 60001 und die dazu gehörige Karte unter *PA Statement Card* mit ID 60002 angelegt. Untergeordnete Ansichten, für zum Beispiel die Kommentare zu einem Statement, werden beispielhaft mit *PA Kommentar SubPage* erstellt. Eine Auflistung aller generierten Pages ist in Tabelle 3.4 zu sehen.

ID	Name
60001	PA Statement List
60002	PA Statement Card
60003	PA Plan Card
60004	PA Comment SubPage
60005	PA Procedur SubPage
60006	PA Details SubPage
60007	PA Output SubPage
60008	PA Object SubPage

Tabelle 3.4: Navison Pagenamen und ID's, Quelle: Eigene Tabelle

4 Implementierung

4.1 SQL-Prozeduren

Die Implementierung der vorliegenden Arbeit gliedert sich zum Einen in die erstellten Stored Procedures auf dem SQL Server und zum Anderen in die in Dynamics Nav 2015 erstellten Pages. Im Nachfolgenden werden die vier Prozeduren im Einzelnen genauer beschrieben.

run_trace

Zur Durchführung eines Server Side Traces bedarf es einer Stored Procedure. Für die genauere Spezifikation des Zieltraces wurde die Funktion um drei Parameter, mit den Bezeichnungen *@DBName*, *@Hostname* und *@TracefileName*, erweitert.

Unter Zuhilfenahme des Parameters *@TracefileName* wurde der Zielpfad mit dem Befehl 4.1 gesetzt.

```
1 SET @OutputFileName = 'C:\Temp\' + @TracefileName
```

Listing 4.1: SQL Befehl zur Definition des Zielpfades, Quelle: Eigenes Listing

Bei der späteren Ausführung werden an dieser Stelle maximal fünf je 1024 MB große Dateien erstellt. Standardmäßig werden 5 MB Dateien durch den Server erstellt. Auf die Nutzung des Standards wurde in diesem Fall verzichtet, da ein deutlich größeres Datenaufkommen zu erwarten ist.

Durch das wiederholte Aufrufen der Anweisung 4.1, den entsprechenden Parametern, werden die in Kapitel 3.3 festgelegten Traceoptionen festgelegt.

```
1 exec sp_trace_setevent @TraceID, 122, 7, @on
```

Listing 4.2: SQL Befehl zur Definition einer Traceoption, Quelle: Eigenes Listing

@TraceID entspricht der vom System automatisch vergebenen Trace ID, 122 dem Spezifizierten Event. Die Nummer 7 der benötigten Spalte und *@on* repräsentiert ein auf 1 gesetztes Bit.

Im Anschluss an die Auflistung der Traceevents und ihrer Spalten erfolgt die Eingrenzung des Ergebnisses durch die Befehle in Listing 4.3. Wichtig für den Filter ist der Fakt, dass der Hostname und die betreffende Datenbank von Trace zu Trace

variieren können aber der Nutzername auf der Datenbank immer gleich ist.

```
1 exec sp_trace_setfilter @TraceID, 8, 0, 6, @host
2 exec sp_trace_setfilter @TraceID, 11, 0, 6,
3                         N'MSU-SOLUTIONS\navservice'
4 exec sp_trace_setfilter @TraceID, 35, 0, 6, @db
```

Listing 4.3: SQL Befehl zur Filterung eines Traces, Quelle: Eigenes Listing

Den Abschluss der Prozedur run_trace setzt das Speichern des Dateinamens und der Trace ID in die Tabelle *PA Trace* und das Starten der Ablaufverfolgung.

load_trace

Mit Hilfe von *load_trace* werden die Datenbanktabellen des Performance-Test-Programms befüllt. Zu Beginn der Verarbeitung wird die gewünschte .trc Datei, deren Name als Parameter übergeben wird, in eine temporäre Tabelle auf dem SQL Server geladen. Das für das Laden genutzte SQL Statement zeigt das Listing 4.4. Herauszuheben ist Erstellen des Hashwertes der Spalte *TextData* durch Verwendung des Befehls *HASHBYTE* und zweier *CONVERT* Anweisungen, damit das Ergebnis in einem Datenformat bereitgestellt wird, welches der Server verarbeiten kann.

```
1 SELECT TextData, Duration, EventClass, DatabaseID,
2 DatabaseName, HostName, CPU, Reads, Writes, TransactionID
3 , CONVERT(VARCHAR(48), HASHBYTES('SHA1'
4 , CONVERT(varchar, TextData)),2) AS TextDataHash
5 INTO #TraceSammlung
6 FROM ::fn_trace_gettable('C:\Temp\'
7 + @TraceFilePath + '.trc', default);
```

Listing 4.4: SQL Befehl zur Befüllung einer temporären Tabelle, Quelle: Eigenes Listing

Die so befüllte Tabelle *#TraceSammlung* wird im Anschluss mit einem sogenannten Cursor durchlaufen und ihre Daten werden auf die Tabellen für Pläne, Statements, Kommentare und Prozeduren aufgeteilt. Als Unterscheidungsmerkmal für die durchlaufenen Daten dient die Eventklasse. Anhand derer wird entschieden in welcher Tabelle von Dynamics NAV 2015 die gerade gelesenen Daten einsortiert werden. Für ein besseres Verständnis stellt das nachfolgende Listing 4.5 den Teil des Codes dar, welcher die Plan-Tabelle füllt.

Interessant an diesem Listing sind zwei Punkte. Punkt eins ist das Fehlen einer Auto-Increment Spalte als Primärschlüssel. Dies liegt an der schlechten Umsetzbarkeit eines AUTO-Increment im Kontext des NAV 2015 Systems in dem alle genutzten Tabellen erstellt wurden.

Der Punkt zwei ist das Konvertieren der Variable *@TextData* mit dem Datentyp

nvarchar(max) in *varbinary(max)*. Ursache hierfür ist der Umstand, dass große Datenmengen in Navision nur als Datentyp *BLOB* gespeichert werden können. Dies wird nötig, da die TextData mehrere tausend Zeichen lang sein kann. Ein *BLOB* wird auf dem SQL Server als *IMAGE* Datentyp behandelt. Aus der Tracedatei werden allerdings nur Daten des Typs *nvarchar()* ausgelesen. *IMAGE* und *nvarchar()* Daten sind nicht kompatibel und eine manuelle Konvertierung wird erforderlich.

```
1  if(@Eventclass = 122)
2  begin
3   set @PlanID = (select MAX(ID)
4   from dbo.[PA StatementPlan])
5   if (@PlanID is null)
6   set @PlanID = 0
7   insert into dbo.[PA StatementPlan]
8   values(Null,@PlanID+1,
9   convert(varbinary(Max),convert(varchar(max),@TextData))
10   ,@TransactionID, @TextDataHash,
11  SUBSTRING(@TextData,0,249))
12 end
```

Listing 4.5: SQL Befehl zur Befüllung Plan-Tabelle, Quelle: Eigenes Listing

Alle Statistiken über das aktuell eingelesene Trace werden im letzten Schritt der Prozedur erstellt und in die entsprechenden Tabellen gesichert. Hierbei werden diverse *SUM* und *COUNT* Anweisungen für die betroffenen Tabellenspalten durchgeführt.

load_DIV

load_DIV tritt nur in Aktion, wenn der Nutzer des Test-Programms einen Vergleich zweier Ablaufverfolgungen erstellen will. Um dies zu ermöglichen, werden die IDs der betroffenen Prozeduren als Parameter übergeben.

Auch in dieser Prozedur wird zu Beginn eine temporäre SQL Server Tabelle, mit Namen *#TempDiv*, befüllt. Die Befüllung erfolgt in zwei Schritten. Mit Schritt Nummer eins werden die Textvorschauen und die ID des ersten zu vergleichenden Traces unter Zuhilfenahme eines Datenbankcursors für die Tabelle *PA Statement* in die oben genannte Tabelle geladen. Listing 4.6 beinhaltet die zugehörige INSERT Anweisung.

```
1 insert into #TempDiv(ID, TraceID1,TextPreview1
2                    , TextPreview2, TraceID2)
3          values(@DivID,@DivTraceID,
4          @DivTextPreview, 'n', 0 )
```

Listing 4.6: Insert Befehl in der Differenzerstellung, Quelle: Eigenes Listing

'n' und 0 sind als Angaben von Nöten, da Navision in seinen Tabellen keine *NULL* Werte zulässt. Positiv ist dieser Umstand dahingehend, dass im zweiten Schritt

die TextPreview und ID der zweiten Nachverfolgung über einen *UPDATE* Befehl eingefügt werden können. Wenn der Fall eintritt, dass der Trace 1 von der Anzahl der Befehle kürzer ist als Trace 2 werden bei den Spalten TraceID1 und TextPreview1 jeweils 0 und 'n' eingefügt.

clear_trace_tables

Nach dem Abschluss einer Problemsuche können einige Gigabyte an Daten in den Datenbanktabellen der Testsoftware enthalten sein. Um das unnütze Aufblähen des Systems zu vermeiden, ist es wichtig alle Tabellen nach getaner Arbeit zu leeren. Am effektivsten erledigt diese Aufgabe der SQL Server.

In der ersten Umsetzung dieser Prozedur wurde der in Listing 4.7 dargestellte Befehl genutzt.

```
1 DELETE FROM Tabellenname
```

Listing 4.7: SQL DELETE FROM Befehl, Quelle: Eigenes Listing

Durch die Testphase des Programms stellte sich heraus, das der DELETE Befehl mit seiner Möglichkeit des Rollback und des Auslösens alle DELETE Trigger der bearbeiteten Tabellen, zu langsam ist. Hieraus folgten minutenlange Wartezeiten beim Leeren der betroffenen Datenbanktabellen.

Zur Lösung dieses Umstandes wurde der DELETE Befehl durch den Aufruf der in Codebeispiel 4.8 gezeigten Anweisung ersetzt.

```
1 TRUNCATE TABLE Tabellenname
```

Listing 4.8: SQL TRUNCATE Befehl, Quelle: Eigenes Listing

Dies hat zur Folge, dass ein Löschvorgang sehr viel schneller vonstatten geht. Allerdings sind keine Rollbacks mehr möglich.

4.2 C/AL Programm

PA Trace List

Ausgangspunkt für die Nutzung der bereits beschriebenen Stored Procedures und das Aufrufen aller weiteren im Verlauf dieser Arbeit erstellten Oberflächen und Funktionen ist die Listenansicht „PA Trace List“. Die nachstehende Abbildung 4.1 zeigt eine beispielhafte Ansicht der genannten Listenansicht.

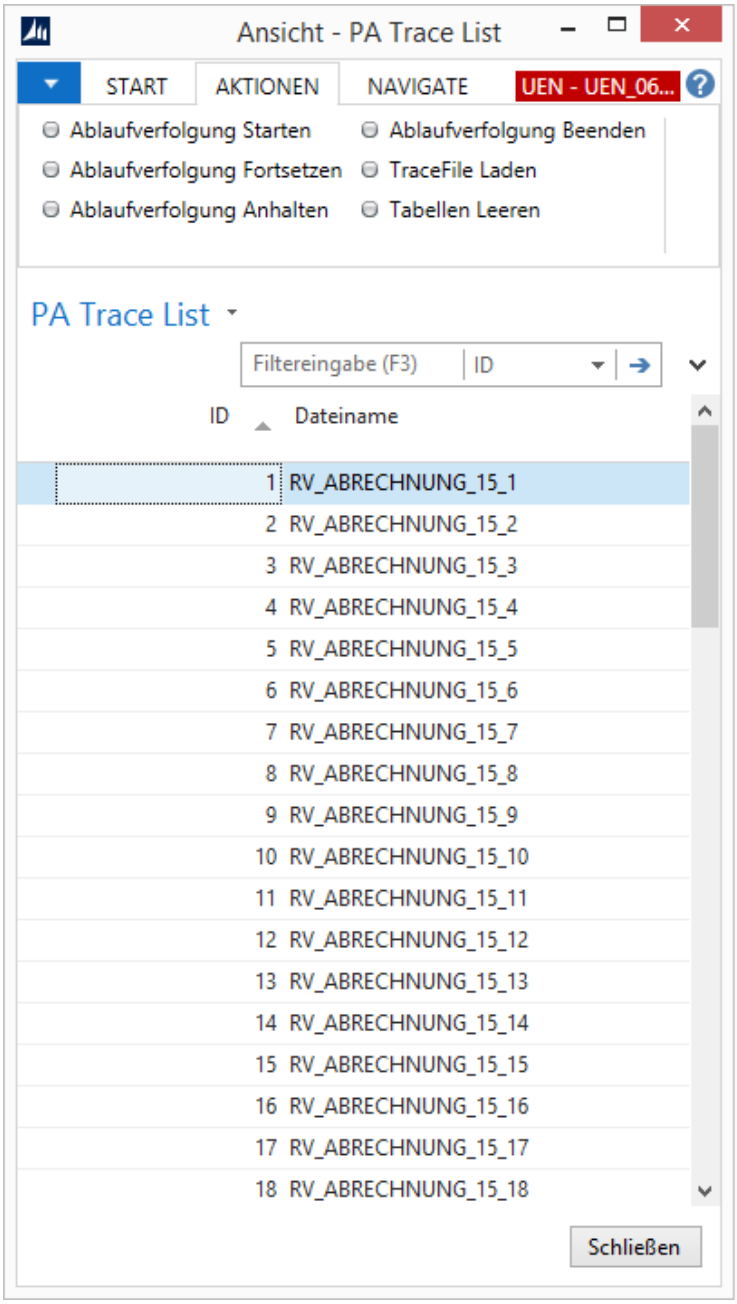


Abbildung 4.1: PA Trace List, Quelle: Eigene Darstellung

So besteht die Oberfläche aus einer Kopfzeile und einem Hauptteil. Die Kopfzeile setzt sich aus den drei Tabmenüs START, AKTIONEN und NAVIGATE zusammen. Unter dem Punkt START befinden sich Standardschaltflächen mit denen der Nutzer einen neuen Datensatz erstellen, einen bestehenden bearbeiten oder löschen kann.

Im nachfolgenden Menü AKTIONEN sind die sechs Tätigkeiten zum Starten, Anhalten, Fortsetzen, Beenden und Laden von Nachverfolgungsdateien sowie das Löschen aller für die Testsoftware relevanten Datenbanktabellen, angesiedelt. Durch sie erfolgt die Nutzung der oben beschriebenen Stored Procedures.

Für den Aufruf allgemeiner Informationen zu den einzelnen Traces und die Einsicht in die zu einem Trace gehörenden SQL Statements sind die entsprechenden Menüoptionen „Allgemeine Informationen“ und „SQL Befehle“ im Bereich NAVIGATE eingebunden. Eine deutschsprachige Beschriftung der Reiters NAVIGATE ist nicht möglich, da er von der Firma Microsoft standardmäßig vorgegeben ist.

Im Hauptteil der Traceanzeige befindet sich die Listenansicht, in welcher die vergebene ID und der vom Nutzer bestimmte Dateiname ersichtlich ist. Auf die Anzeige der durch den SQL Server generierten Trace ID wird verzichtet, da sie ausschließlich für interne Aktionen von Nöten ist.

Alle für den Nutzer bereitgestellten Aktionen arbeiten nach dem Schema aus Listing 2.2. „Ablaufverfolgung Starten“ erbittet außerdem über einen kleinen Texteingabewizard den gewünschten Dateinamen vom Programmnutzer und startet mit dem Befehl aus Codebeispiel 4.9 die vollständige SQL Ablaufverfolgung durch Navision selbst. *DEBUGGER* ist eine Navision weite quellcodemäßige Repräsentation des Navision-Debug-Tools. Dessen Funktion *ENABLESQLTRACE* benötigt die ID der Session des nachzuverfolgenden Nutzers, hier die aktuelle SESSIONID, und die Option *TRUE* zum Beginnen oder *FALSE* zum Beenden der Nachverfolgung.

```
1 DEBUGGER.ENABLESQLTRACE(SESSIONID,TRUE);
```

Listing 4.9: C/AL Befehl zum Starten der Ablaufverfolgung durch Navision, Quelle: Eigenes Listing

Als allgemeine Informationen werden die aufsummierten Laufzeiten in Sekunden von SQL Befehlen und Stored Procedures als auch deren Anzahl innerhalb eines erfassten Mitschnitts betrachtet. Zum Veranschaulichen wird auf Abbildung A.1 im Anhang verwiesen.

PA Statement List

Nach einem erfolgreich abgeschlossenen Ladevorgang einer Nachverfolgungsdatei, können die enthaltenden SQL Statements über das Betätigen der Schaltfläche „SQL Befehle“ der im vorherigen Abschnitt beschriebenen Oberfläche eingesehen werden. Schaubild 4.2 stellt die Nutzeransicht beispielhaft dar.

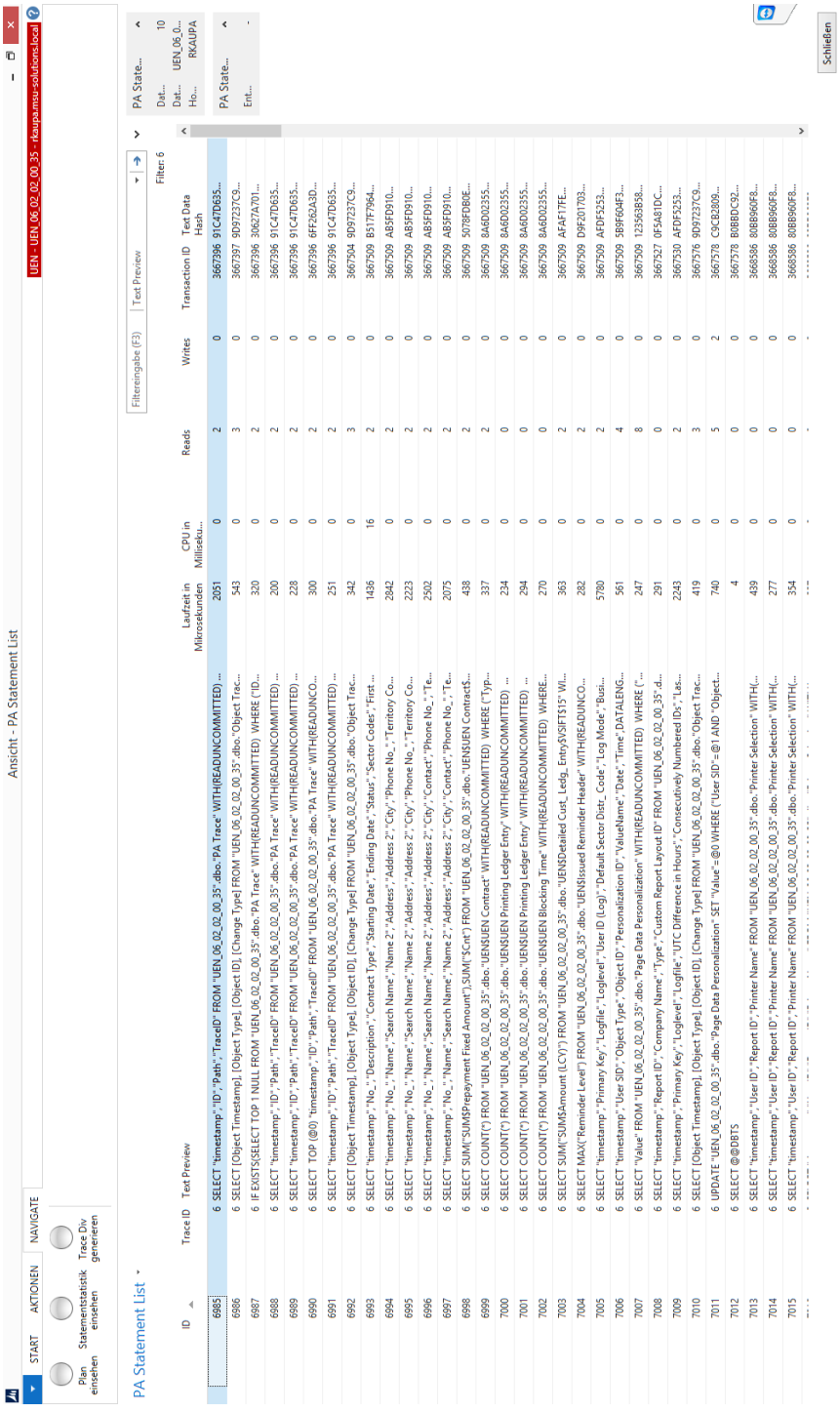


Abbildung 4.2: PA Statement List, Quelle: Eigene Darstellung

Neben den zu sehenden Standardlistenelementen Kopfzeile und Datenauflistung wurde in dieser Ansicht ein Bereich für Infoboxen bzw. Factboxen hinzugefügt. So befinden sich neben den Standardelementen im Menü START die Schaltflächen „Plan einsehen“, „Statementstatistik einsehen“ und „Trace Div generieren“ im Menüpunkt NAVIGATE. Der Listenbereich zeigt alle in der Datenbank erfassten Daten zu den Statements, beschränkt sich bei der Anzeige des SQL Befehls allerdings auf eine 250 Zeichen lange Vorschau.

Immer wiederkehrende Informationen wie der Datenbankname, die Datenbank ID und der Computername, von dem aus die Befehle gesendet wurden, werden in der oben stehenden Infobox angezeigt. In der unteren Box wird die entsprechende Codeunit angezeigt, wenn in den Kommentaren zu dem vom Nutzer selektierten SQL Befehl eine Codeunit genannt wird. Ermöglicht wird diese Anzeige durch die Nutzung von String-Operationen wie zum Beispiel STRLEN, STRPOS und DELSTR innerhalb des *OnAfterGetCurrRecord()* Triggers dieser Factbox.

Bei einem Doppelklick auf eine beliebige Zeile erscheint die durch Abbildung A.2 veranschaulichte Karte der ausgesuchten SQL Anweisung. Auf jeder Karte werden die relevanten Informationen in drei Teilbereichen für den Nutzer angezeigt. Abschnitt eins, mit der Bezeichnung „Allgemein“, weist neben den bereits in der Listenansicht zu entnehmenden Daten das vollständige SQL Statement aus. Des Weiteren werden in den Bereichen zwei und drei die Kommentare und Prozeduraufrufe inklusive zugehöriger Daten ersichtlich.

Statementinformationen

Zu jedem SQL Statement wird ein Ausführungsplan erfasst und es gibt Aussagen, die nur durch das Betrachten von kumulativen Werten oder direkte Vergleiche von Traces untereinander getroffen werden können.

Einen für ihn interessanten Ausführungsplan kann der Nutzer durch Klicken der im Vorfeld bereits genannten Schaltfläche auswählen. Dem Klick folgt die Anzeige der Abbildung A.3. In ihr sind zum Einen der Plan als XML konformer Text und zum Anderen die zugehörige Transaktionsnummer ersichtlich. Weitere Informationen zu dem betroffen Plan, speziell Informationen zu einzelnen Knoten oder Ausgabelisten, Suchprädikaten und Objektlisten, wurden aufgrund von sehr schlechten XML-Verarbeitungszeiten aus der Anzeige entfernt. Alternativ ist es dem Benutzer nun möglich, durch das Auslösen der Schaltfläche „Plan öffnen“, im Reiter AKTIONEN, den gewünschten Ablaufplan im SQL Server Management Studio aufzurufen.

Im Vergleich zu der vorhergehenden tabellenartigen Auflistung der ausgelesenen Daten im Navision steigert diese Anzeigeart die Übersichtlichkeit des Plans erheblich. Zur Förderung des Verständnisses soll hier die Abbildung A.4 dienen, welche einen kleinen Beispielplan im geöffneten SQL Management Studio zeigt.

Wenn mit Hilfe der Standardauflistung der Einzelstatements kein Problem erkannt werden kann, besteht ergänzend die Möglichkeit über den schon erwähnten Button „Statementstatistik einsehen“ die Ansicht A.5 aufzurufen. Standardmäßig ist die Ansicht nach der Spalte *Test Data Hash* aufsteigend sortiert. Für das gezeigte Beispiel, den 6. Testlauf aus dem ersten 30 Tests umfassenden Paket, erfolgte die Sortierung absteigend auf der Spalte *Duration* und ermöglicht hierdurch die unterschiedlichsten Aussagen über die Häufigkeit bestimmter Befehle deren aufsummierten Laufzeit, ihren Verbrauch an CPU Zeit und auch ihre Anzahl von Schreib- und Lesezugriffen .

Hierbei fällt der Insert Befehl auf die Tabelle *Sales Line* (Verkaufszeile) im Besonderen auf. Er wird im Verlauf der Buchung 42 mal ausgeführt und alle Aufrufe laufen zusammen 1,48 Sekunden. Dies sind 0,87 Sekunden mehr als im Fall des 39 Mal ausgeführten und 0,61 Sekunden benötigenden Insert Befehl auf die Tabelle *Sales Header* (Verkaufskopf). Auch die 395.816 Lesezugriffe auf Platz 1 im Vergleich zu den 1915 auf Platz 2 sind bemerkenswert. Abschließend kann davon ausgegangen werden, dass die Unterschiede in den Werten durchaus gravierend sind, sie aber im normalen Rahmen liegen und keine Optimierung nötig ist.

Mit dem letzten der drei Knöpfe auf der Listenansicht 4.2 ist der Nutzer im Stande, einen Vorgang zu starten, an dessen Ende zwei beliebige Ablaufverfolgungen in Form ihrer SQL Befehlsvorschauen gegenüber gestellt werden. Mit Beginn der Erstellung der Gegenüberstellung erfragt das Programm über den Wizard A.6 die beiden Nummern der gewünschten Traces. Standardwert ist in beiden Eingaben die Zahl 1. Auch eine Gegenüberstellung einer Ablaufverfolgung mit sich selbst ist möglich. Im Anschluss an die ID Auswahl kann über die Schaltfläche „Div Ansehen“ das Programm WinMerge mit den entsprechenden Daten als Parameter aufgerufen werden. Das Ergebnis des Vergleichs von Trace 6 und 7 wird in Abbildung A.7 dargestellt.

5 Test der Anwendung

Zur Verifizierung der Funktionstüchtigkeit des im Zuge dieser Arbeit erstellten Tools wurden diverse Tests durchgeführt. Nachfolgend werden drei dieser Tests genauer beschrieben.

5.1 Abrechnung eines Rahmenvertrages

Navision 2015

Aus Gründen der höheren Sicherheit, größeren Stabilität und einer einfacheren Produktpflege, durch die Mitarbeiter der msu, werden alle Kunden mit Navision 2009 zunehmend auf das aktuelle Navision 2015 umgestellt. Um diesen Umstellungsprozess mit statistischen Werten bestmöglich unterstützen zu können, wurde eine Gegenüberstellungen von Navision 2009 und Navsion 2015 durchgeführt.

Um ein möglichst gutes Testergebnis zu erzielen, wurde ein typischer Rahmenvertrag gewählt und der Abrechnungslauf über alle Positionen 120 mal durchgeführt. Diese 120 Durchführungen teilen sich in vier 30 Tests umfassende Pakete in zwei Testszenarien. Testpaket eins repräsentiert die Abrechnungen ohne Ablaufverfolgung mit einer leeren Abrechnungstabelle. Paket zwei beinhaltet diejenigen Abrechnungen mit Ablaufverfolgung auf die leere Abrechnungstabelle. Beide ergeben gemeinsam Testszenario eins. Testszenario zwei umfasst die Testpakete drei und vier. In Paket drei sind die Abrechnungen ohne Ablaufverfolgung und Nutzung einer bereits gefüllten Abrechnungstabelle enthalten. Die 30 Tests mit den nachverfolgten Rahmenvertragsabrechnungen auf eine bereits Werte enthaltende Abrechnungstabelle finden sich in der letzten Testansammlung.

Zur Veranschaulichung soll die im Anschluss gezeigte Abbildung 5.1 dienen. Sie trägt den Titel „Laufzeiten von Rahmenvertragsabrechnungen“. Unterstützend weist der Untertitel „Testszenario 1“ auf den gezeigten Sachverhalt hin. An der Y-Achse werden die Laufzeiten in der Einheit Sekunden mit einem Intervall von eins dargestellt. Die Testnummern sind an der X-Achse abzulesen.

Der in Gelb dargestellte Graph mit der Bezeichnung „Standardlaufzeiten“ zeigt die 30 Tests ohne mitlaufende Ablaufverfolgung und einer leeren Abrechnungstabelle. Die Graphen „SQL Befehlslaufzeit“ (Blau) und „Prozedurlaufzeit“ (Orange) entstehen aus den Laufzeiten der 30 Mitschnitte mit durchgeführter Ablaufverfolgung und ebenfalls leerer Abrechnungstabelle.

Wie zu erwarten war verlangsamt die Ablaufverfolgung den Abrechnungsprozess erheblich. Eine Abrechnung mit mitlaufendem Trace nimmt durchschnittlich 7,04 Sekunden in Anspruch und ist im Mittel 5,27 Sekunden langsamer als eine Abrechnung die nicht nachverfolgt wird. Der zweite gut erkennbare Umstand ist der zeitliche Mehraufwand des Servers bei der Abarbeitung neu auftretender SQL Befehle. Dies spiegelt sich in der Diskrepanz von „Prozedurlaufzeit“ und „SQL Befehlslaufzeit“ wider. Je mehr neue SQL Befehle in Erscheinung treten desto mehr neue Stored-Procedures muss der Server erstellen und kompilieren.

Weiterhin sind Schwankungen der Ausführungszeiten zu erkennen. Die Ausführung der SQL Befehle bewegt sich zwischen 4,78 und 8,48 Sekunden und die davon abhängige Prozedurlaufzeit fluktuiert von 5,18 bis zu 9,29 Sekunden Abarbeitungszeit. Bei der Kontrolle des Testrechners wurde festgestellt, dass der für die Tests verwendete Computer während der laufenden Tests an seine Leistungsgrenzen gelangte. In Folge dessen ist davon auszugehen, dass die aufgezeichneten Schwankungen kein Anzeichen eines Problems mit der Testsoftware sondern Indikator für eine schwache Hardware sind.

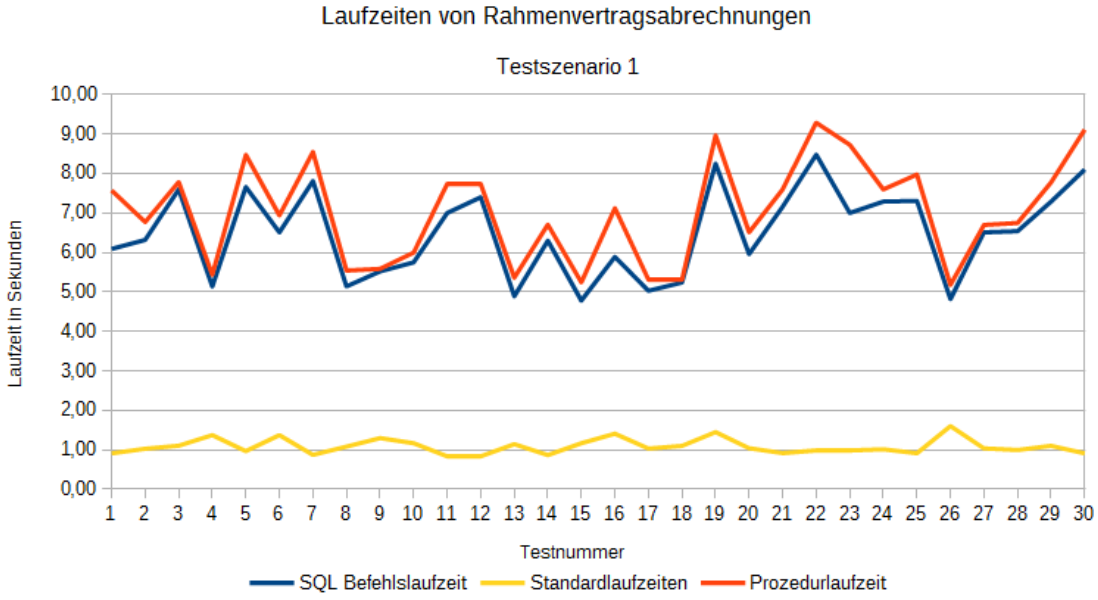


Abbildung 5.1: Tracelaufzeiten 1, Quelle: Eigene Darstellung

Das nachstehende Schaubild 5.2 stellt die Rahmenvertragsabrechnungen mit Updateoperationen auf die Datenbanktabelle der Abrechnungen dar. Als Titel wurde hier ebenso wie bei der Vorgängergraphik „Laufzeiten von Rahmenvertragsabrechnungen“ verwendet und zur Unterstützung mit „Testszenario 2“ ergänzt. Die Achsenbezeichnungen sind identisch zu denen der Vorgänger Graphik mit „Laufzeit in Sekunden“ sowie „Testnummer“. Aus Gründen der besseren Lesbarkeit verändert sich der Intervall in der Y-Achse von eins auf zwei Sekunden.

Abrechnungen mit Tabellenupdates benötigen im Mittelwert 10,52 Sekunden zur Durchführung des Prozesses und sind rund 3,6 Sekunden langsamer als die Prozesse ohne Überschreiben bestehender Abrechnungen. Auch in dieser Darstellung sind die bereits erwähnten Schwankungen gut sichtbar. So bewegt sich die Zeit der Prozedurausführungen beispielsweise zwischen 8,86 und 13,54 Sekunden. Dies entspricht einem Unterschied von 4,67 Sekunden zwischen dem erfassten Minimum und Maximum.

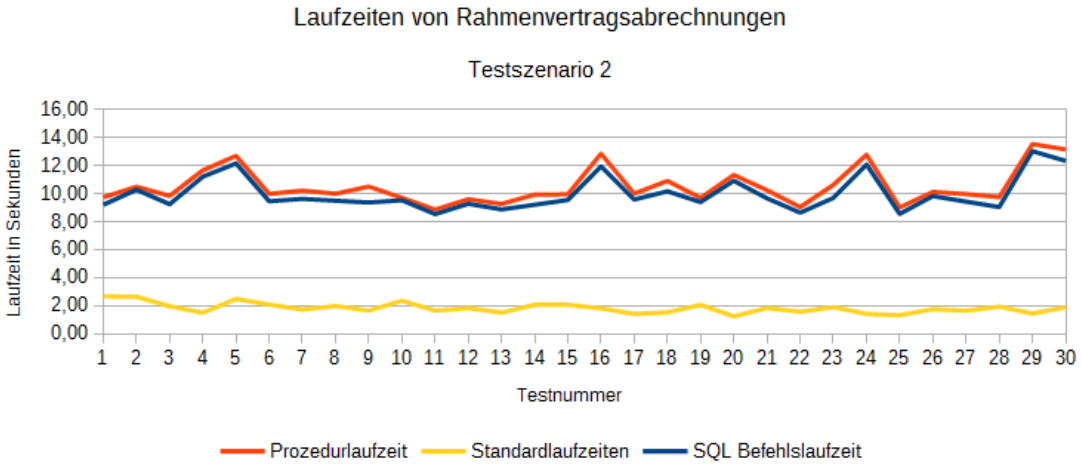


Abbildung 5.2: Tracelaufzeiten 2, Quelle: Eigene Darstellung

Vergleich von 2009 zu 2015

Im Anschluss an die Tests im Navision 2015 und deren Auswertung wurden weitere Testläufe im Navision 2009 durchgeführt. Ergebnis der ausgeführten Tests sind weitere zwei aus 30 Tests bestehende Pakete. Diese Tests ermöglichen den Vergleich der Tracelaufzeiten zwischen Navision 2009 und Navision 2015. Hierzu werden die Prozedurlaufzeiten der beiden Systeme in den nachstehenden Abbildungen gegenübergestellt. Die Gegenüberstellung der Prozedurlaufzeiten reicht aus, da in ihnen alle anderen auftretenden Laufzeiten enthalten sind.

In Abbildung 5.3, mit dem Titel „Vergleich von NAV 2009/2015“ sowie Untertitel: „Testszenario 1“, sind die Laufzeiten der Rahmenvertragsabrechnung beider Systeme in eine ungefüllte Abrechnungstabelle zu sehen. An der X-Achse sind die Nummern der Tests angetragen und an der Y-Achse befinden sich die Laufzeiten in der Einheit Sekunden mit einem Intervall von Zwei.

Dem ersten Eindruck zufolge wurden die Operationen in Navision 2015 schneller ausgeführt als in Navision 2009. Zur Untermauerung dieses Eindruckes wurden dem gezeigten Diagramm die arithmetischen Mittel der betreffenden Tests hinzugefügt. Hieraus ist ersichtlich, dass die Verarbeitung in Navision 2015 0,5 Sekunden schneller abläuft als im Navsion 2009. Weiterhin ist gut zu erkennen, dass auch die Ausführungszeiten im System von 2009 einer Schwankung von mehreren Sekunden unterliegen.

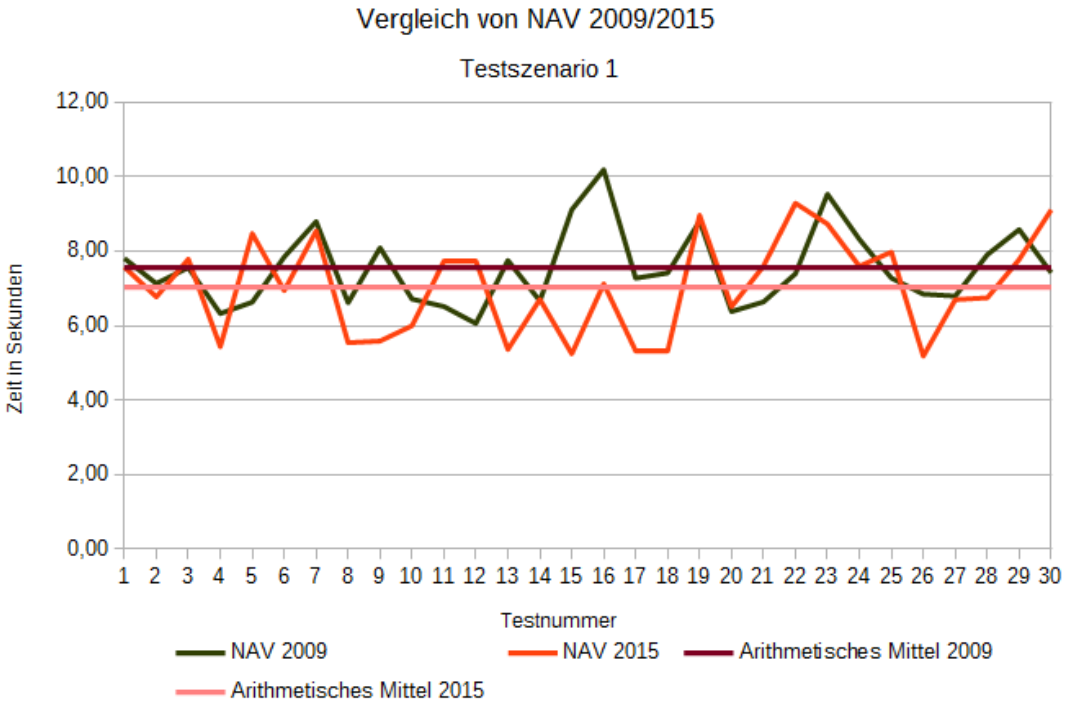


Abbildung 5.3: Vergleich von NAV 2015 und NAV 2009, Quelle: Eigene Darstellung

Das zweite Schaubild zum Vergleich von NAV 2009 und NAV 2015 5.4, mit dem Titel „Laufzeiten von Rahmenvertragsabrechnungen“ und Untertitel „Testszenario 2“, zeigt die Laufzeiten der Nachverfolgungen mit Updateoperationen auf die Abrechnungstabelle. Die Achsenbezeichnungen sind in diesem Diagramm identisch zu denen seines Vorgängers. Auch in diesem Bild sind die arithmetischen Mittel der Laufzeiten eingezeichnet.

Neben den für alle Tests typischen Schwankungen in den Ausführungszeiten ist im Vergleich zu Diagramm 5.3 auffallend, dass NAV 2009 im Durchschnitt 0,4 Sekunden schneller arbeitet als NAV 2015.

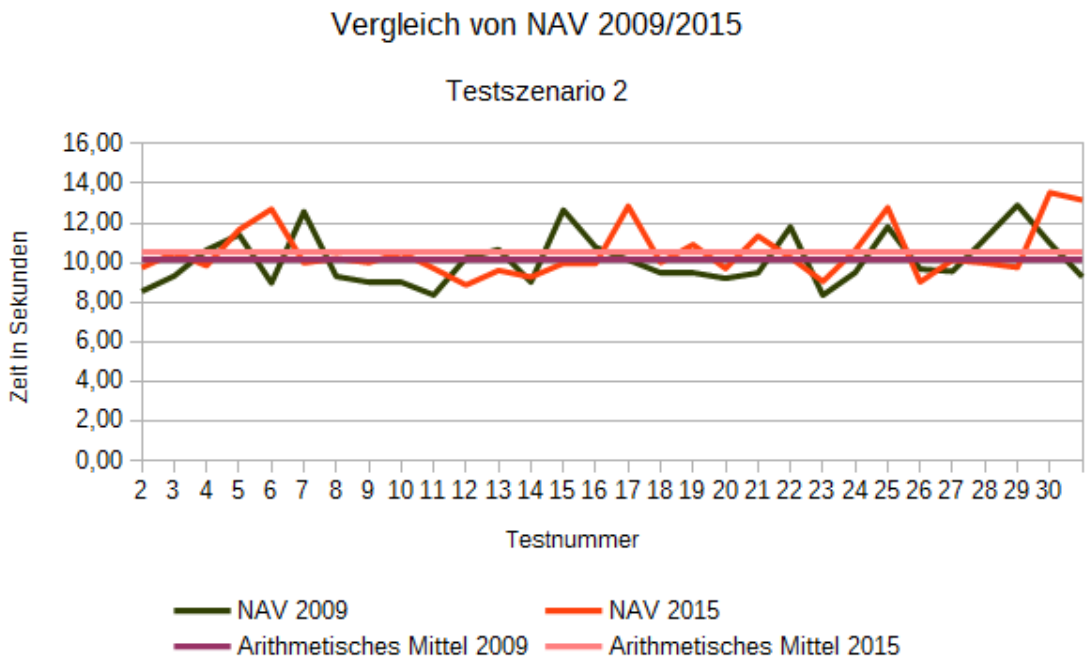


Abbildung 5.4: Vergleich von NAV 2015 und NAV 2009, Quelle: Eigene Darstellung

Bei genauerer Untersuchung der einzelnen Ablaufverfolgungen sind unterschiedliche Dinge festzustellen. Zum Einen arbeitet Navision 2009 bei der Ersterstellung einer Vertragsabrechnung, in Verbindung mit einer ungefüllten Abrechnungstabelle, rund 3058 Einzelprozeduren ab und ist langsamer als Navision 2015, welches im Schnitt 1216 Prozeduren für die selbe Aufgabe benötigt.

Zum Anderen führt Navision 2009 im Verlauf der Tests mit UPDATE-Operationen durchschnittlich 3921 Prozeduren aus und sein Nachfolger 2015 nur fast 1958 Prozeduren. Dennoch ist das ältere System in der Lage, die in diesen Beispielen an es gestellten, Ziele durchschnittlich schneller zu erreichen. Dieser Umstand ist mit der im Navision 2015 gestiegenen Komplexität der Befehle und der gestiegenen Anzahl von Spalten und Tabellen-Indizes zu begründen.

Mit der Lösung von Problemen, welche durch Einfüge- und Aktualisierungsoperationen erzeugt werden, befasst sich der nachfolgende Test.

5.2 Jahresabrechnung von Rahmenverträgen

Nach der Gegenüberstellung von Dynamics NAV 2009 und 2015 bei der Abrechnung eines Rahmenvertrages im oben gezeigten Testfall wird mit dem folgenden Test untersucht, wie sich Dynamics NAV 2015 bei der Abrechnung von mehreren Rahmenverträgen im Kontext eines Kundensystems verhält.

Basis für den vorliegenden Test ist ein Datenbankabbild der Stadtwerke Georgsmarienhütte. Jedes Jahr werden die Kundenverträge für die Sparten Strom, Gas und Wasser abgerechnet. Exemplarisch wird in diesem Beispiel die Abrechnung der Sparte Gas untersucht.

Ein solcher Abrechnungslauf umfasst rund 18700 Verträge und benötigt im Durchschnitt 24 Stunden. Die Verkürzung dieser Zeitspanne ist essenziell, da dieser Abrechnungsvorgang jährlich in jedem Stadtwerk durchgeführt werden muss.

Da immer wiederkehrende Ausführungen der selben Arbeitsschritte und damit verbundene Probleme schon früh erkannt werden können, wird auf einen kompletten 24 Stunden Test verzichtet und wie auch im voran gegangenen Test, 30 Verträge abgerechnet und interpretiert.

Darstellung 5.5 zeigt auszugsweise das Ergebnis des Mitschnittes der Vertragsabrechnungen. Die Abbildung ist zweigeteilt. Abschnitt *a* zeigt die 10 SQL Befehle mit den längsten Abarbeitungszeiten absteigend sortiert. Unter den gezeigten Statements befinden sich sechs *INSERT*-, drei *SELECT*- und ein *UPDATE*- Befehl. Obwohl ein *SELECT*-Befehl mit 1,2 Sekunden Laufzeit die gezeigte Liste anführt, legt die Anzahl an gelisteten *INSERT*-Befehlen nahe, dass hier eine größere Leistungssteigerung möglich ist. Zur Unterstützung dieser Annahme zeigt der Teil *b* die aufsummierten Daten der, bei der Nachverfolgung auftretenden, SQL Befehle.

Wie erwartet, wird für abgearbeitete *INSERT*-Befehle viel Zeit verbraucht. Besonderes Augenmerk zieht hierbei das Einfügen von Daten in die Tabelle *G/L Entry* bzw. *Sachposten* auf sich. Dieser Befehl tritt 649 auf und benötigt rund 42,8 Sekunden. In diesen 42,8 Sekunden werden 6,9 Millionen Leseoperationen, sowie 69578 Schreibzugriffe ausgeführt und verbrauchen 16,8 Sekunden CPU-Zeit. Der Unterschied zum nachfolgenden Befehl, welcher 371 mal auftritt und mit 11,5 Sekunden zu Buche schlägt, ist gravierend.

Für die hohen Abarbeitungszeiten der *INSERT*-Befehle kann eine hohe Anzahl Spalten und für die Tabelle hinterlegten Indizes verantwortlich sein. Im speziellen Fall der Sachposten-Tabelle umfasst die Tabelle selbst 58 Spalten und 18 Indizes. Zu Testzwecken werden innerhalb des Testsystems alle Indizes, mit Ausnahme des Primärschlüssels, abgeschaltet.

ID	Trace ID	Text Preview	Laufzeit in Mikrosekunden	CPU in Millisekunden	Reads	Writes	Transaction ID	Text Data Hash
509511	21	SELECT ISNULL("UEN Contract Journal Line"."timestamp",@5) AS "timestamp",ISNULL("UEN Contract Journal Li...	1200777	15	241	0	23399933	8B404BEFF...
509949	21	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$G_L Entry" ("Entry No_","G_L Account No_","Posting Dat...	610771	31	11100	111	23399933	BF5CE1C2...
518149	21	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$Integration Record" ("Integration ID","Table ID","Page I...	456238	0	38	10	23399933	BF5CE1C2...
517874	21	SELECT TOP 1 NULL FROM "UEN_701_GMH".dbo."Stadtwerke GmbH$UEN Sales Partial Prepayment" WITH(UPD...	431319	0	8	0	23399933	37445F750...
513273	21	SELECT TOP (@0) "timestamp","No_","Sell-to Customer No_","Bill-to Name","Bill-to Na...	430711	0	28	0	23399933	6FF262A3D...
497662	21	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$G_L Entry" ("Entry No_","G_L Account No_","Posting Dat...	430279	47	11082	110	23399933	BF5CE1C2...
503171	21	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$VAT Entry" ("Entry No_","Gen_ Bus_ Posting Group","Ge...	428471	0	252	3	23399933	BF5CE1C2...
517584	21	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$G_L Entry" ("Entry No_","G_L Account No_","Posting Dat...	421732	31	11193	108	23399933	BF5CE1C2...
522248	21	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$Detailed Cust_ Ledg_ Entry" ("Entry No_","Cust_ Ledger ...	420537	16	216	0	23399933	BF5CE1C2...
506071	21	UPDATE "UEN_701_GMH".dbo."Stadtwerke GmbH$UEN Statistics Entry" SET "Posting Type"=@0,"Source No_"=...	416956	15	54	0	23399933	A47DDC87...

(a) 10 langsamsten Einzelstatements

Text Data Hash	Text Preview	Count	Duration in Mikrosekunden	CPU in Milliseconds	Reads	Writes	Trace ID
BF5CE1C2B50634...	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$G_L Entry" ("Entry No_","G_L Account No_","Posting Date","Document Type","Document No_"...	649	42829456	16882	6993100	69578	21
BF5CE1C2B50634...	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$Detailed Cust_ Ledg_ Entry" ("Entry No_","Cust_ Ledger Entry No_","Entry Type","Posting Date","...	371	11518350	4300	127061	1437	21
BF5CE1C2B50634...	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$VAT Entry" ("Entry No_","Gen_ Bus_ Posting Group","Gen_ Prod_ Posting Group","Posting Date","...	310	8881063	3118	60128	601	21
EF1B69C2E0E2C82...	SELECT "timestamp","Entry No_","Customer No_","Posting Date","Document Type","Document No_","Description","Currency Code","Sales (LCY)","Profi...	1755	8431863	3597	13230	0	21
6FF262A3DE72770...	SELECT TOP (@0) "timestamp","No_","Sell-to Customer No_","Bill-to Name","Bill-to Name 2","Bill-to Address ...	573	8189306	2735	23874	0	21
BF5CE1C2B50634...	INSERT INTO "UEN_701_GMH".dbo."Stadtwerke GmbH$UEN Sales Partial Invoice Line" ("Document No_","Line No_","Sell-to Customer No_","Type","No...	477	7123709	2466	54005	618	21
6FF262A3DE72770...	SELECT TOP (@0) "timestamp","Entry No_","Posting Type","Posting Date","Source Type","Source No_","Source Line No_","Technical Sector","Balance S...	918	6936981	2732	35072	0	21
AC9CA71D4085D...	SELECT "timestamp","No_","Sell-to Customer No_","Bill-to Customer No_","Bill-to Name","Bill-to Name 2","Bill-to Address","Bill-to Address 2","Bill-to C...	991	6854689	2750	6082	0	21
A47DDC873C4DA...	UPDATE "UEN_701_GMH".dbo."Stadtwerke GmbH$UEN Statistics Entry" SET "Posting Type"=@0,"Source No_"=@1 WHERE ("Entry No_"=@2 AND "time...	928	5678245	2122	49894	797	21
CE5029E6DCC0D8...	DELETE FROM "UEN_701_GMH".dbo."Stadtwerke GmbH$Sales Line" WHERE ("Document Type"=@0 AND "Document No_"=@1)	106	3486640	1256	93913	2548	21

(b) 10 langsamsten Statements nach Laufzeiten

Abbildung 5.5: Ist-Zustand nach der Abrechnung von 30 Rahmenverträgen in GMH-System

Nach der erfolgten Schlüsselabschaltung erfolgt ein weiterer Test des Abrechnungsprozesses. Der Vergleichstest ergibt eine signifikante Verbesserung der *INSERT*-Operationen auf die Sachposten-Tabelle. So zu sehen in der nachstehenden Tabelle 5.1.

Gut zu erkennen ist, dass Abarbeitung der Statements nach Deaktivierung der Indizes rund 90% schneller ist als vorher. Ähnlich stark sinkt auch der Bedarf an CPU-Rechenzeit. Im Fall der Lese- und Schreibzugriffe auf den SQL Server sinkt die Belastung im Vorher/Nachher Vergleich der Jahresabrechnung um fast 99%.

Test	Duration in s	CPU Verbrauch in s	Reads	Writes
mit Indizes	42,83	16,88	6.993.100	69.578
ohne Indizes	4,39	1,33	8976	127

Tabelle 5.1: Vorher/Nachher Vergleich der Rahmenvertragsabrechnung, Quelle: Eigene Tabelle

Auf Basis der oben stehenden Tabelle wurde das Balkendiagramm 5.6 mit dem Titel „Jahresabrechnung Vorher/Nachher“ generiert. Es soll zur Veranschaulichung des Sachverhaltes und der Dimension der erreichten Veränderung dienen.

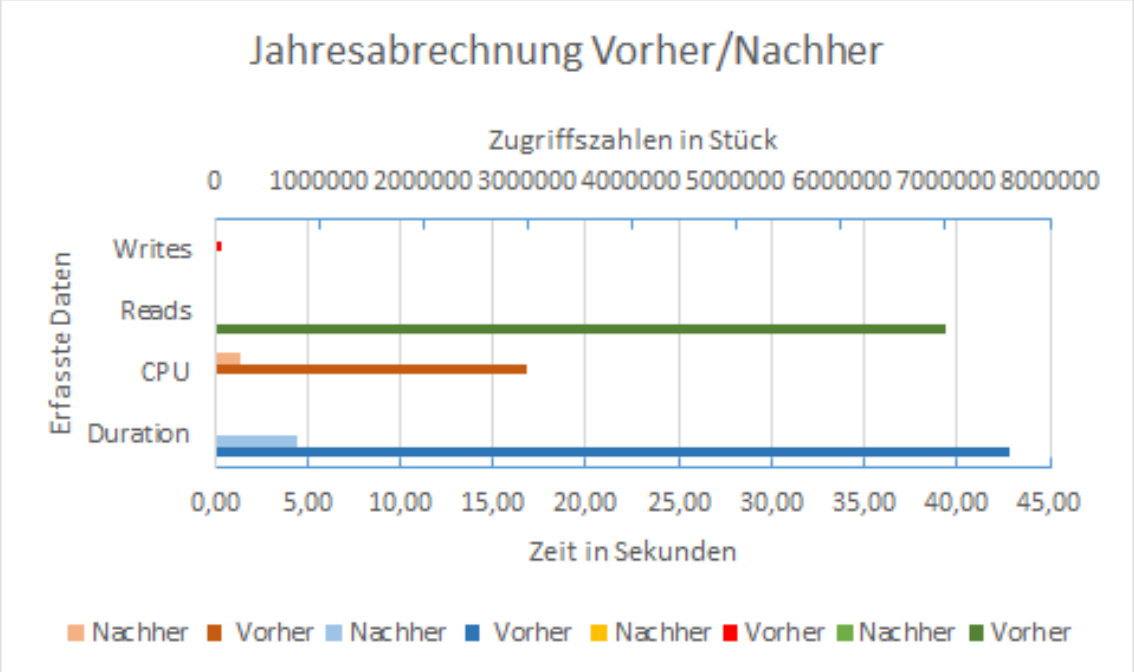


Abbildung 5.6: Vorher/Nachher Vergleich der Jahresabrechnung, Quelle: Eigene Darstellung

An der Y-Achse sind die Rubriken Writes (in Rot), Reads (in Grün), CPU (in Orange) und Duration (in Blau) aufgeführt. Die X-Achse wurde aufgrund von sehr unterschiedlichen, erfassten Wertebereichen durch eine sekundäre X-Achse ergänzt. Die obere X-Achse reicht von 0 bis 8 Millionen. Von den gezeigten acht Balken beziehen sich die ersten vier auf die genannte Skala. Dies umfasst die Vergleichswerte für die Schreib- und die Leseoperationen. Für die CPU-Verbrauchswerte und die allgemeine Laufzeit ist die primäre X-Achse mit den Werten von 0 bis 45,00 der Bezugspunkt. Auch in diesem Diagramm werden die bereits erwähnten gravierenden Leistungseinsparungen gut sichtbar.

Die aufgezeichneten Leistungsverbesserungen sind für Kundensysteme im Produktiveinsatz nicht ohne Weiteres erreichbar, da durch das Ausschalten vieler Indizes zwar *INSERT*- und *UPDATE*-Befehle stark an Leistung gewinnen, *SELECT*-Befehle aufgrund von fehlenden Indexen hingegen an Leistung einbüßen. Den Lösungsansatz hierfür stellt das Inspizieren der Nutzungsstatistiken aller in Frage kommenden Indexe dar. Wobei die Indizes mit der geringsten Nutzung zu deaktivieren sind. Dieses Vorgehen erfordert Zeit und eine durch den SQL Server geführte Statistik. Auf dem Testsystem kann diese Arbeit nicht durchgeführt werden, da eine entsprechend gefüllte Statistik fehlt.

5.3 Validierung einer UTILMD

Unter einer UTILMD versteht man eine *Utilities Master Data Message*. Diese Message ist ein auf dem Standard namens *EDIFACT* basierendes Nachrichtenformat. Der Datentyp UTILMD wird von Unternehmen in der Energiewirtschaft genutzt, um Kundenstammdaten untereinander zu versenden.

Der hier untersuchte Test überprüft den Validierungsprozess für UTILMD Nachrichten. Als Testbasis dient eine generierte, vollständig korrekte UTILMD. Sie besteht aus einer Kopfzeile sowie zwei Nachrichtenzeilen. Alle drei Komponenten weisen zusammen 625 einzelne Felder mit unterschiedlichen Daten auf.

Die Testbasis durchläuft den Test 100 mal. So muss das System ohne Möglichkeit des Zwischenspeicherns 62500 Anfragen an die Datenbank senden, um jedes Datenfeld zu überprüfen. Damit die Testzeit so gering wie möglich ausfällt, wird ein Cache von drei Zeilen Größe genutzt. Wenn das System nun im Verlauf des Tests auf eine neue Zeile trifft, werden die enthaltenen Felder einmal an der Datenbank abgefragt und das Ergebnis im Cache gespeichert. Da zu Testzwecken der Zwischenspeicher drei Reihen aufnehmen kann, muss nach dem ersten vollständigen Testlauf kein weiterer Datenbankzugriff erfolgen und die erforderlichen Daten werden nachfolgend nur noch aus dem Cache entnommen.

Der Test gilt als bestanden, wenn alle 625 Felder 100 mal richtig geprüft wurden, jede der drei Cache-Zeilen einmal befüllt wurden und nach jedem Testlauf die Meldung „Validation correct“ im Test-Log erscheint.

Um nun das Performance-Analyse-Tool selbst zu testen, werden drei Testszenarien untersucht. In Testszenario eins wird der normale Test der UTILMD Validierung mitgeschnitten. Anschließend wird der Cache in Szenario zwei von drei auf zwei Zeilen verkleinert und und die Auswirkungen auf das Ergebnis untersucht. Als dritter und letzter Tool-Test wird die Zwischenspeicherung komplett auskommentiert und mit seinen beiden Vorgängertestszenarien verglichen.

Verlaufen die Validierungstests wie erwartet, dann schlägt sich das Verkleinern des Caches beziehungsweise dessen komplettes Ausschalten in der Anzahl von ausgeführten SQL Befehlen auf der Datenbank nieder.

Wie auch bei allen vorhergehenden Tests mit dem Performance-Analyse-Tool wurden pro Testszenario 30 Test durchlaufen und anschließend die Mittelwerte verglichen. Die Ergebnisse veranschaulicht die nachstehende Abbildung 5.7. Betitelt ist die Graphik mit „Testlaufzeiten in Sekunden“. An der Y-Achse sind die Werte von 0,00 bis 50,00 angetragen und spiegeln die Zeit in der Einheit Sekunden wider. An der X-Achse befinden sich die Szenarionummern.

Wie zu sehen ist, sind die in Blau dargestellten Säulen die vom SQL Server erfassten durchschnittlichen Zeiten für die ausgeführten SQL Befehle. Ergänzt wird

dies durch die in Rot gehaltenen Zeiten der Stored Procedur Ausführung. Die letzte der drei pro Szenario gezeigten Säulen repräsentiert die Zeitspanne, welche das Navision 2015 System nach Abschluss eines Testlaufes ausgibt.

In Testgruppe eins werden die SQL Befehle im Schnitt mit 3,54 Sekunden ausgeführt und die zugehörigen Prozeduren in 4,27 Sekunden. Dies deckt sich mit der Anzeige von 4,78 Sekunden Testzeit in Navision 2015. Auch Testszenario zwei ist unter Beachtung einer gewissen Messungenauigkeit im erwarteten Zeitrahmen. Extreme Abweichungen von der durch den SQL Server erfassten Zeit und der in Microsoft Dynamics NAV 2015 angezeigten Testzeit sind nur in Testgruppe drei zu erkennen.

Trotz veränderter Cachegröße, Testgruppe zwei, beziehungsweise abgeschalteter Cachingfunktion, Testgruppe drei, werden keine signifikant größere Anzahlen von SQL Befehlen von Navision an den SQL Server gesendet. Dies legt den Verdacht nahe, dass das zwischen Navision 2015 und dem SQL Server befindliche Servicetier oder auch Navision selbst systeminterne Speicherfunktionen nutzt.

An dieser Stelle entfällt eine genauere Untersuchung des Sachverhaltes, da sie nicht Teil dieser Arbeit ist und den Rahmen überschreiten würde.

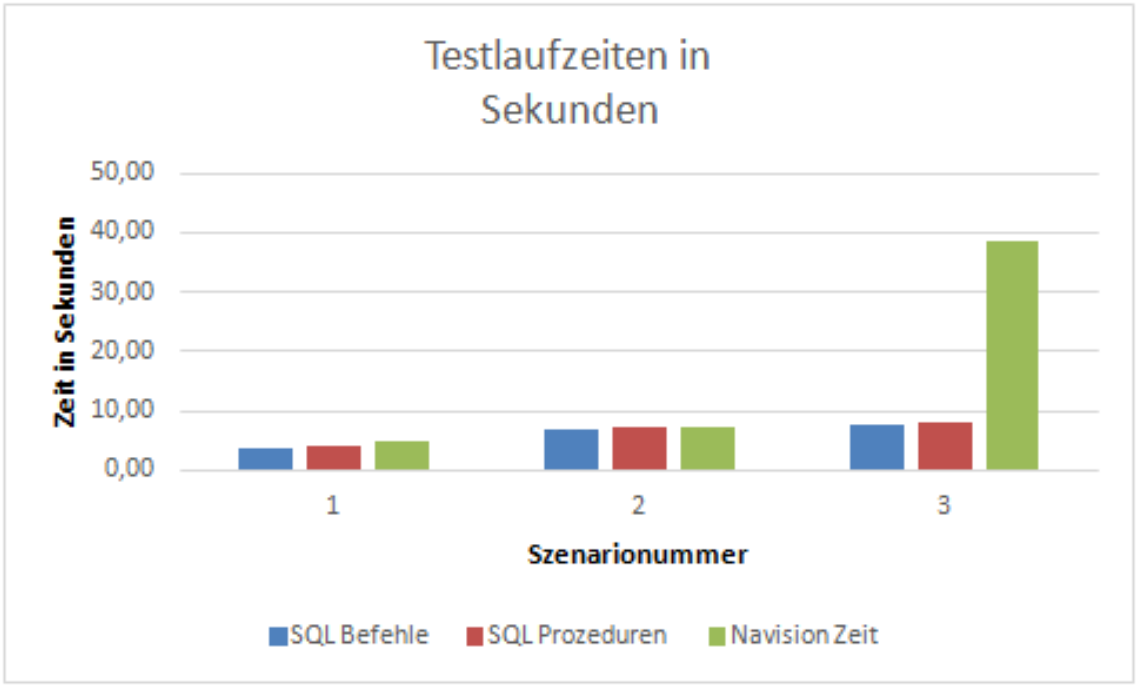


Abbildung 5.7: Zeitlicher Vergleich von UTILMD Validierungstests, Quelle: Eigene Darstellung

6 Ausblick und Fazit

6.1 Ausblick

Für eine Weiterentwicklung der vorliegenden Prototyp-Software ist die Implementierung von String-Vergleichsalgorithmen denkbar. Genutzt werden können diese Algorithmen zum Vergleich zweier beliebiger Traces mit Navision eigenen Mitteln.

Ausgangspunkt für die Gegenüberstellung und den Vergleich zweier Zeichenketten ist die sogenannte Levenshtein-Distanz. Diese Kennzahl entspricht der Anzahl aller nötigen Einfügen-, Löschen- oder auch Ersetzen-Operationen die nötig sind, den gegebenen Text a in Text b umzuwandeln. Ist die Levenshtein-Distanz gleich 0, dann sind die miteinander verglichenen Texte identisch und es müssen keine weiteren Schritte im Programm durchgeführt werden. Die Spezifikation dieser Kennziffer erfolgte 1965 durch Wladimir Lewenstein.[1]

Neben der Feststellung der Levenshtein-Distanz ist es ebenfalls von Interesse, die deckungsgleichen Teile zweier Texte zu bestimmen. Dies kann genutzt werden, um die entsprechend gleichen Textteile einzufärben. Eine solche Einfärbung bietet einen schnellen Überblick, wenn zwei SQL-Befehle nicht 100% übereinstimmen, aber eine gewisse Ähnlichkeit aufweisen.

Zu Testzwecken wird das oben genannte Problem als Problem der „Longst Common Subsequence“[2] behandelt und in Microsoft Dynamics NAV 2015 unter Zuhilfenahme der Programmiersprache C/AL umgesetzt. Das folgende Listing 6.1 beinhaltet die erwähnte Testimplementierung.

Zu sehen ist die lokale Funktion LCS mit zwei Test Parametern und einem Rückgabewert vom Typ Text. Im Verlauf der gezeigten Funktion werden die als Parameter übergebenen Texte immer wieder am Textende um ein Zeichen verkürzt und der selben Funktion erneut übergeben. Da der rekursive Aufruf zu Beginn der Funktion erfolgt, handelt es sich um eine kopfrekursive Funktion.

[1] [Lew65]
[2] [Hir75]

```
1  LOCAL LCS(l_Text1 : Text;l_Text2 : Text)
2  l_return : Text
3  BEGIN
4      l_laenge1 := STRLEN(l_Text1);
5      l_laenge2 := STRLEN(l_Text2);
6      l_return := '';
7
8          IF (l_laenge1 >  0) AND (l_laenge2 >  0)
9          THEN BEGIN
10
11         IF l_Text1[l_laenge1] =  l_Text2[l_laenge2]
12         THEN
13           l_return := LCS(COPYSTR(l_Text1, 1,
14           l_laenge1-1),
15           COPYSTR(l_Text2, 1, l_laenge2-1))
16           + FORMAT(l_Text1[l_laenge1])
17         ELSE
18         BEGIN
19           l_x := LCS(l_Text1, COPYSTR(l_Text2, 1,
20           l_laenge2-1));
21           l_y := LCS(COPYSTR(l_Text1, 1,
22           l_laenge1-1), l_Text2);
23           IF STRLEN(l_x) > STRLEN(l_y) THEN
24             l_return := l_x
25           ELSE
26             l_return := l_y;
27
28         END;
29     END;
30 END;
```

Listing 6.1: LCS-Problem in C/AL, Quelle: Eigenes Listing

Der oben dargestellte Quellcode arbeitet in Tests mit kurzen Texten, bis zu 11 Zeichen, zuverlässig. Bei Tests mit längeren Texten, 12 Zeichen aufwärts, sind lange Verarbeitungszeiten und Abstürze von NAV 2015 zu beobachten. Dieses Verhalten resultiert aus einer, für Navision, zu tiefen Rekursion. Ursache hierfür ist der zirka 16 Befehlsaufrufe fassende interne Aufrufstapel.

Außerdem disqualifiziert das beobachtete Verhalten den oben aufgeführten Code von einer Nutzung in der Weiterentwicklung des Analyse-Tools. Für eine Nutzung in der Weiterentwicklung muss nach anderen Wegen der Implementierung gesucht werden.

Für die Bestimmung der genannten Übereinstimmung und als Lösung für das festgestellte Speicherplatzproblem, kann der Hirschberg-Algorithmus genutzt werden. Dieser Algorithmus wurde von Dan Hirschberg 1975 publiziert.[3]

Der Hirschberg-Algorithmus hat einen linearen Speicherbedarf $\theta(min\{M, N\})$ und eine Laufzeit von $\theta(MN)$ in Verbindung mit einer Rekursionstiefe von $\theta(log_n)$. Ziel des Algorithmus ist es, ein optimales Alignment zwischen zwei unterschiedlichen Texten zu finden.

Die Abarbeitung erfolgt auf einer NxM Matrix. Diese Matrix wird zeilenweise, gleichzeitig von links Oben und rechts Unten bearbeitet. Wenn sich die beiden Teilberechnungen in der Mitte der Matrix treffen, ist der erste Punkt des optimalen Alignments gefunden. An diesem Punkt wird die Matrix anschließend in zwei „Unterprobleme“ geteilt. Diese Teilbereiche entsprechen dem oberen linken Quadranten und dem unteren rechten Quadranten.

In diesen neu entstandenen kleineren Matrizen erfolgt die Bearbeitung, von links Oben und rechts Unten aus, erneut bis zur Mitte des Teilbereiches. An der neuen mittleren Schnittstelle erfolgt eine weitere Quadrantenteilung und es existiert ein weiterer Punkt des optimalen Ziel-Alignments.

Dieser Ansatz von „Teilen und Herrschen“ wird solange angewandt, bis ein durchgehender optimaler Pfad in der Ausgangsmatrix entstanden ist. Die im Anschluss dargestellte Abbildung 6.1 zeigt eine solche Matrix mit grau schattierten Unterproblemen und dem, in schwarz eingezeichneten, optimalen Pfad.

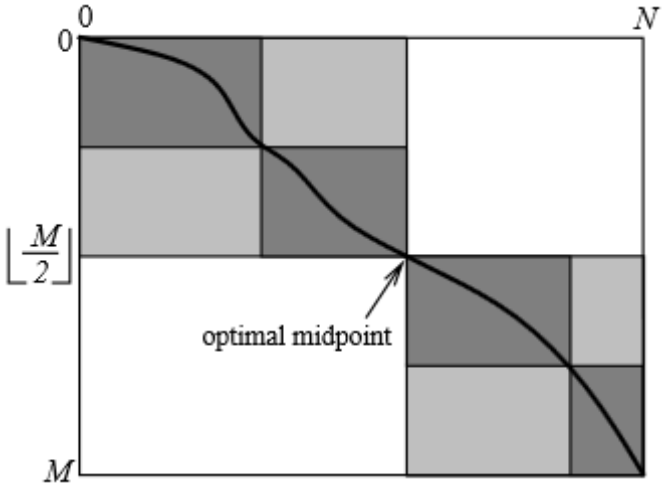


Abbildung 6.1: Schema des Hirschberg-Algorithmus, Quelle:[Cha75]

Der Speicherplatzbedarf und die Rekursionstiefe sind von der Länge der Texte m und n abhängig. Ist der längere Text in der Kopfzeile der Matrix, dann ist die Rekursionstiefe minimal. Ist hingegen der kürzere Text in der Matrixkopfzeile, so ist folglich der Speicherbedarf minimal.

[3] [Hir75]

Von einer Implementierung des Hirschberg-Algorithmus wird an dieser Stelle abgesehen, da eine Umsetzung komplex ausfällt und damit den Rahmen dieser Arbeit übersteigt.

6.2 Fazit

Das Ziel der vorliegenden Arbeit, die Erstellung eines Performance-Analyse-Tools für Microsoft Dynamics NAV 2015, kann auf Grund der anschließenden Tests zur Laufzeit von Rahmenvertragsabrechnungen und Analyse von Optimierungsmöglichkeiten als erfüllt angesehen werden.

So ist der Nutzer im Stande, Ablaufverfolgungen anzulegen, beliebige Trace-Dateien in das System zu laden und die erfassten SQL-Befehle zu inspizieren. Mittels der aufgezeichneten Kommentarzeilen können die Navision-Nutzer und eventuell ausgeführte C/AL Codeunits mit allen, durch sie erzeugten, SQL-Befehle in Verbindung gebracht werden. Ein Vergleich von SQL-Befehlen aus zwei beliebigen Ablaufverfolgungen erfolgt über das Tool WinMerge, welches aus Navision heraus mit entsprechenden Parametern gestartet wird. Ergänzend zu den genannten Funktionalitäten werden während des Datenimports durch die genutzten SQL-Scripte nützliche Statistiken über kumulatives Auftreten der mitgeschnittenen SQL-Anweisungen erstellt. Die genannten Funktionen entsprechen den in Kapitel 3 definierten funktionalen Anforderungen.

Die ebenfalls im dritten Kapitel festgehaltenen nichtfunktionalen Anforderungen werden bestmöglich erfüllt. Hierbei sind bei der Umsetzung des Prototyps unter Berücksichtigung des Navision Style guides und der Verwendung einer deutschsprachigen Benutzerführung keine nennenswerten Hindernisse aufgetreten. Die Verarbeitung der auftretenden Datenmengen erfolgt in rund 20 Sekunden für .trc Dateien ,mit 150 MB Größe, aus Testfall eins und in rund 3 Minuten mit Trace-Dateien aus Testfall zwei mit fast 2,3 GB Dateigröße.

Während der Implementierungsphase des Prototyps stellte sich heraus, dass die geplante Zerlegung des SQL Statement Plans und anschließende Anzeige auf der Detailansicht des Plans in Navision nicht praktikabel war. Zur Lösung dieses Umstandes wurde auf die Nutzung des externen Microsoft SQL Server Management Studios zurückgegriffen.

Die in Kapitel 5 aufgeführten Beispiele haben gezeigt, dass das erstellte Programm eine schnelle und übersichtliche Auflistung der erfassten Daten zeigt und dem Nutzer so Rückschlüsse auf die auftretenden Probleme erlaubt. Insbesondere auf den Leistungsgewinn um den Faktor 10 im zweiten Testbeispiel mit Produktivdaten sei an dieser Stelle noch einmal verwiesen.

Abschließend bleibt zu erwähnen, dass zum aktuellen Zeitpunkt gefundene Problemlösungen nur aus dem Wissen und den Erfahrungen des Nutzers auf Basis der erfassten Daten entstehen. Das Programm selbst ist nicht im Stande Lösungsoptionen anzubieten oder selbstständig Lösungen auszuführen.

Literaturverzeichnis

[Bun15] Bundesamt, Statistisches: Betriebe, Tätige Personen, Geleistete Arbeitsstunden, Entgelte. (2015), Juni. – https://www-genesis.destatis.de/genesis/online/data;jsessionid=7946E56761E17278543AE6930F9277AF.tomcat_GO_2_2?operation=abruftabelleAbrufen&selectionname=43111-0001&levelindex=1&levelid=1433938350327&index=1 letzer Aufruf 10. Juni 2015

[Cha75] Chao, Miller Hardison: Recent Developments in Linear-Space Alignment Methods: A Survey. (1975), Juni. – http://www.csie.ntu.edu.tw/~kmchao/papers/1994_JCB.pdf letzer Aufruf 07. September 2015

[Com13] Computerwoche: Kampf der ERP-Titanen: SAP - Oracle - Microsoft. (2013). – http://www.computerwoche.de/i/detail/artikel/2552346/1/946607/d2e434-media/ letzer Aufruf 15. Juni 2015

[Edi15] Editor, ERP Software B.: How Many Companies Use Microsoft Dynamics ERP? Updated 2015. (2015), März. – http://www.erpsoftwareblog.com/2015/03/how-many-companies-use-microsoft-dynamics-erp/ letzer Aufruf 19. Mai 2015

[Gar14] Gartner: Magic Quadrant for Operational Database Management Systems. (2014), Oktober. – http://www.gartner.com/technology/reprints.do?id=1-237UHKQ&ct=141016&st=sb letzer Aufruf 22. Juni 2015

[Hir75] Hirschberg, D. S.: A Linear Space Algorithm for Computing Maximal Common Subsequences. (1975), Juni. – https://www.ics.uci.edu/~dan/pubs/p341-hirschberg.pdf letzer Aufruf 07. September 2015

[Lew65] Lewenstein, Wladimir I.: Binary codes capable of correcting deletions, insertions, and reversals. (1965), Januar. – https://gitlab.doc.ic.ac.uk/wm613/inidividual-project/raw/4fbe43f863b229b50ca13bf5f59eb029ad71f6b6/reading/litreview/levenshtein66.pdf letzer Aufruf 07. September 2015

[Mic15a] Microsoft: ADO.NET Overview. (2015), Juli. – https://msdn.microsoft.com/de-de/library/e80y5yhx%28v=vs.110%29.aspx letzer Aufruf 30. Juli 2015

[Mic15b] Microsoft: Microsoft Dynamics Nav. (2015). – http://www.microsoft.com/de-de/dynamics/erp-nav-overview.aspx letzer Aufruf 15. Juni 2015

[Mic15c] Microsoft: Microsoft SQL Server. (2015), Juni. – https://msdn.microsoft.com/de-de/library/bb545450.aspx letzer Aufruf 15. Juni 2015

[Mic15d] Microsoft: RoleTailored Architecture. (2015). – https://msdn.microsoft.com/en-us/library/dd355215.aspx letzer Aufruf 15. Juni 2015

[Mic15e] Microsoft: SELECT (Transact-SQL). (2015). – https://msdn.microsoft.com/de-de/library/ms189499.aspx letzer Aufruf 15. Juni 2015

[Mic15f] Microsoft: SqlConnection.ConnectionString Property. (2015), September. – https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.connectionstring%28v=VS.100%29.aspx letzer Aufruf 07. September 2015

[Mic15g] Microsoft: SqlConnection.ConnectionTimeout Property. (2015), September. – https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.connectiontimeout%28v=vs.100%29.aspx letzer Aufruf 07. September 2015

[Mic15h] Microsoft: SSPI. (2015), September. – https://msdn.microsoft.com/en-us/library/windows/desktop/aa380493%28v=vs.85%29.aspx letzer Aufruf 07. September 2015

[Mic15i] Microsoft: Using C/AL. (2015), Juni. – https://msdn.microsoft.com/en-us/library/dd355277.aspx letzer Aufruf 15. Juni 2015

[msu06] msu: Das Unternehmen. (2006). – http://www.msu-solutions.de/unternehmen letzer Aufruf 15. Juni 2015

[Nie96] Niemann, Klaus D.: *Client/server-Architektur: Organisation und Methodik der Anwendungsentwicklung*. Vieweg+Teubner Verlag, 1996

[Qui] Quin, LIAM R. E.: XML ESSENTIALS. . – http://www.w3.org/standards/xml/core letzer Aufruf 15. Juni 2015

[RBJ03] Rankins, Ray ; Bertucci, Paul ; Jensen, Paul: *Microsoft SQL Server 2000 Unleashed*. Sams, 2003

[Sch09] Schmeling, Holger: *SQL Server 2008 Performance-Optimierung*. Addison-Wesley Verlag, 2009

[Som07] Sommerville, Ian: *Software Engineering*. 8. Auflage. Pearson Studium, 2007

[SSH11] Saake, Gunter ; Sattler, Kai-Uwe ; Heuer, Andreas: *Datenbanken: Implementierungstechniken*. mitp, 2011

[Stu07] Studebaker, David: *Programming Microsoft Dynamics NAV*. PACKT Publishing, 2007

[W3C06] W3C: Extensible Markup Language (XML). (2006), August. – `http://www.w3pdf.com/W3cSpec/XML/2/REC-xml11-20060816.pdf` letzer Aufruf 15. Juni 2015

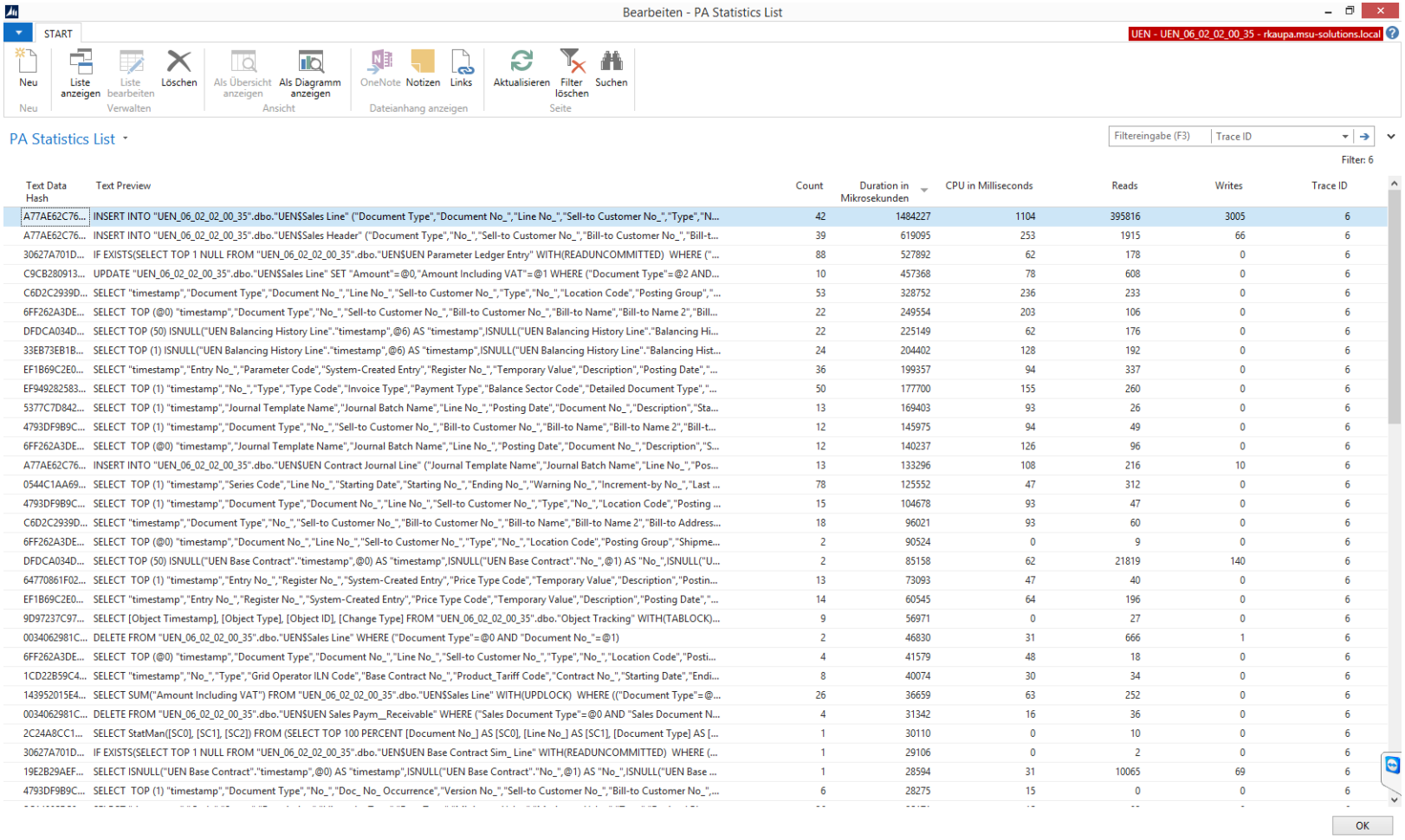
A Abbildungen des C/AL Programms

Abbildung A.1: PA Trace Overview List, Quelle: Eigene Darstellung

Abbildung A.2: PA Statement Card, Quelle: Eigene Darstellung

Abbildung A.3: PA Plan Card, Quelle: Eigene Darstellung

Abbildung A.4: Plan, Quelle: Eigene Darstellung



Text Data Hash	Text Preview	Count	Duration in Mikrosekunden	CPU in Milliseconds	Reads	Writes	Trace ID
A77AE62C76...	INSERT INTO "UEN_06_02_02_00_35".dbo."UEN$Sales Line" ("Document Type","Document No_","Line No_","Sell-to Customer No_","Type","N...	42	1484227	1104	395816	3005	6
A77AE62C76...	INSERT INTO "UEN_06_02_02_00_35".dbo."UEN$Sales Header" ("Document Type","No_","Sell-to Customer No_","Bill-to Customer No_","Bill-t...	39	619095	253	1915	66	6
30627A701D...	IF EXISTS(SELECT TOP 1 NULL FROM "UEN_06_02_02_00_35".dbo."UEN$UEN Parameter Ledger Entry" WITH(READUNCOMMITTED) WHERE ("...	88	527892	62	178	0	6
C9CB280913...	UPDATE "UEN_06_02_02_00_35".dbo."UEN$Sales Line" SET "Amount"=@0,"Amount Including VAT"=@1 WHERE ("Document Type"=@2 AND...	10	457368	78	608	0	6
C6D2C2939D...	SELECT "timestamp","Document Type","Document No_","Line No_","Sell-to Customer No_","Type","No_","Location Code","Posting Group","...	53	328752	236	233	0	6
6FF262A3DE...	SELECT TOP (@0) "timestamp","Document Type","No_","Sell-to Customer No_","Bill-to Customer No_","Bill-to Name","Bill-to Name 2","Bill...	22	249554	203	106	0	6
DFDCA034D...	SELECT TOP (50) ISNULL("UEN Balancing History Line"."timestamp",@6) AS "timestamp",ISNULL("UEN Balancing History Line"."Balancing Hi...	22	225149	62	176	0	6
33EB73EB1B...	SELECT TOP (1) ISNULL("UEN Balancing History Line"."timestamp",@6) AS "timestamp",ISNULL("UEN Balancing History Line"."Balancing Hist...	24	204402	128	192	0	6
EF1B69C2E0...	SELECT "timestamp","Entry No_","Parameter Code","System-Created Entry","Register No_","Temporary Value","Description","Posting Date","...	36	199357	94	337	0	6
EF94928258 3...	SELECT TOP (1) "timestamp","No_","Type","Type Code","Invoice Type","Payment Type","Balance Sector Code","Detailed Document Type","...	50	177700	155	260	0	6
5377C7D842...	SELECT TOP (1) "timestamp","Journal Template Name","Journal Batch Name","Line No_","Posting Date","Document No_","Description","Sta...	13	169403	93	26	0	6
4793DF9B9C...	SELECT TOP (1) "timestamp","Document Type","No_","Sell-to Customer No_","Bill-to Customer No_","Bill-to Name","Bill-to Name 2","Bill-t...	12	145975	94	49	0	6
6FF262A3DE...	SELECT TOP (@0) "timestamp","Journal Template Name","Journal Batch Name","Line No_","Posting Date","Document No_","Description","S...	12	140237	126	96	0	6
A77AE62C76...	INSERT INTO "UEN_06_02_02_00_35".dbo."UEN$UEN Contract Journal Line" ("Journal Template Name","Journal Batch Name","Line No_","Pos...	13	133296	108	216	10	6
0544C1AA69...	SELECT TOP (1) "timestamp","Series Code","Line No_","Starting Date","Starting No_","Ending No_","Warning No_","Increment-by No_","Last ...	78	125552	47	312	0	6
4793DF9B9C...	SELECT TOP (1) "timestamp","Document Type","Document No_","Line No_","Sell-to Customer No_","Type","No_","Location Code","Posting ...	15	104678	93	47	0	6
C6D2C2939D...	SELECT "timestamp","Document Type","No_","Sell-to Customer No_","Bill-to Customer No_","Bill-to Name","Bill-to Name 2","Bill-to Address...	18	96021	93	60	0	6
6FF262A3DE...	SELECT TOP (@0) "timestamp","Document No_","Line No_","Sell-to Customer No_","Type","No_","Location Code","Posting Group","Shipme...	2	90524	0	9	0	6
DFDCA034D...	SELECT TOP (50) ISNULL("UEN Base Contract"."timestamp",@0) AS "timestamp",ISNULL("UEN Base Contract"."No_",@1) AS "No_",ISNULL("U...	2	85158	62	21819	140	6
64770861F02...	SELECT TOP (1) "timestamp","Entry No_","Register No_","System-Created Entry","Price Type Code","Temporary Value","Description","Postin...	13	73093	47	40	0	6
EF1B69C2E0...	SELECT "timestamp","Entry No_","Register No_","System-Created Entry","Price Type Code","Temporary Value","Description","Posting Date","...	14	60545	64	196	0	6
9D97237C97...	SELECT [Object Timestamp], [Object Type], [Object ID], [Change Type] FROM "UEN_06_02_02_00_35".dbo."Object Tracking" WITH(TABLOCK)...	9	56971	0	27	0	6
0034062981C...	DELETE FROM "UEN_06_02_02_00_35".dbo."UEN$Sales Line" WHERE ("Document Type"=@0 AND "Document No_"=@1)	2	46830	31	666	1	6
6FF262A3DE...	SELECT TOP (@0) "timestamp","Document Type","Document No_","Line No_","Sell-to Customer No_","Type","No_","Location Code","Posti...	4	41579	48	18	0	6
1CD22B59C4...	SELECT "timestamp","No_","Type","Grid Operator ILN Code","Base Contract No_","Product_Tariff Code","Contract No_","Starting Date","Endi...	8	40074	30	34	0	6
143952015E4...	SELECT SUM("Amount Including VAT") FROM "UEN_06_02_02_00_35".dbo."UEN$Sales Line" WITH(UPDLOCK) WHERE (("Document Type"=@...	26	36659	63	252	0	6
0034062981C...	DELETE FROM "UEN_06_02_02_00_35".dbo."UEN$UEN Sales Paym__Receivable" WHERE ("Sales Document Type"=@0 AND "Sales Document N...	4	31342	16	36	0	6
2C24A8CC1...	SELECT StatMan([SC0], [SC1], [SC2]) FROM (SELECT TOP 100 PERCENT [Document No_] AS [SC0], [Line No_] AS [SC1], [Document Type] AS [...	1	30110	0	10	0	6
30627A701D...	IF EXISTS(SELECT TOP 1 NULL FROM "UEN_06_02_02_00_35".dbo."UEN$UEN Base Contract Sim_ Line" WITH(READUNCOMMITTED) WHERE (...	1	29106	0	2	0	6
19E2B29AEF...	SELECT ISNULL("UEN Base Contract"."timestamp",@0) AS "timestamp",ISNULL("UEN Base Contract"."No_",@1) AS "No_",ISNULL("UEN Base ...	1	28594	31	10065	69	6
4793DF9B9C...	SELECT TOP (1) "timestamp","Document Type","No_","Doc_ No_ Occurrence","Version No_","Sell-to Customer No_","Bill-to Customer No_",...	6	28275	15	0	0	6

Abbildung A.5: PA Statistics List, Quelle: Eigene Darstellung

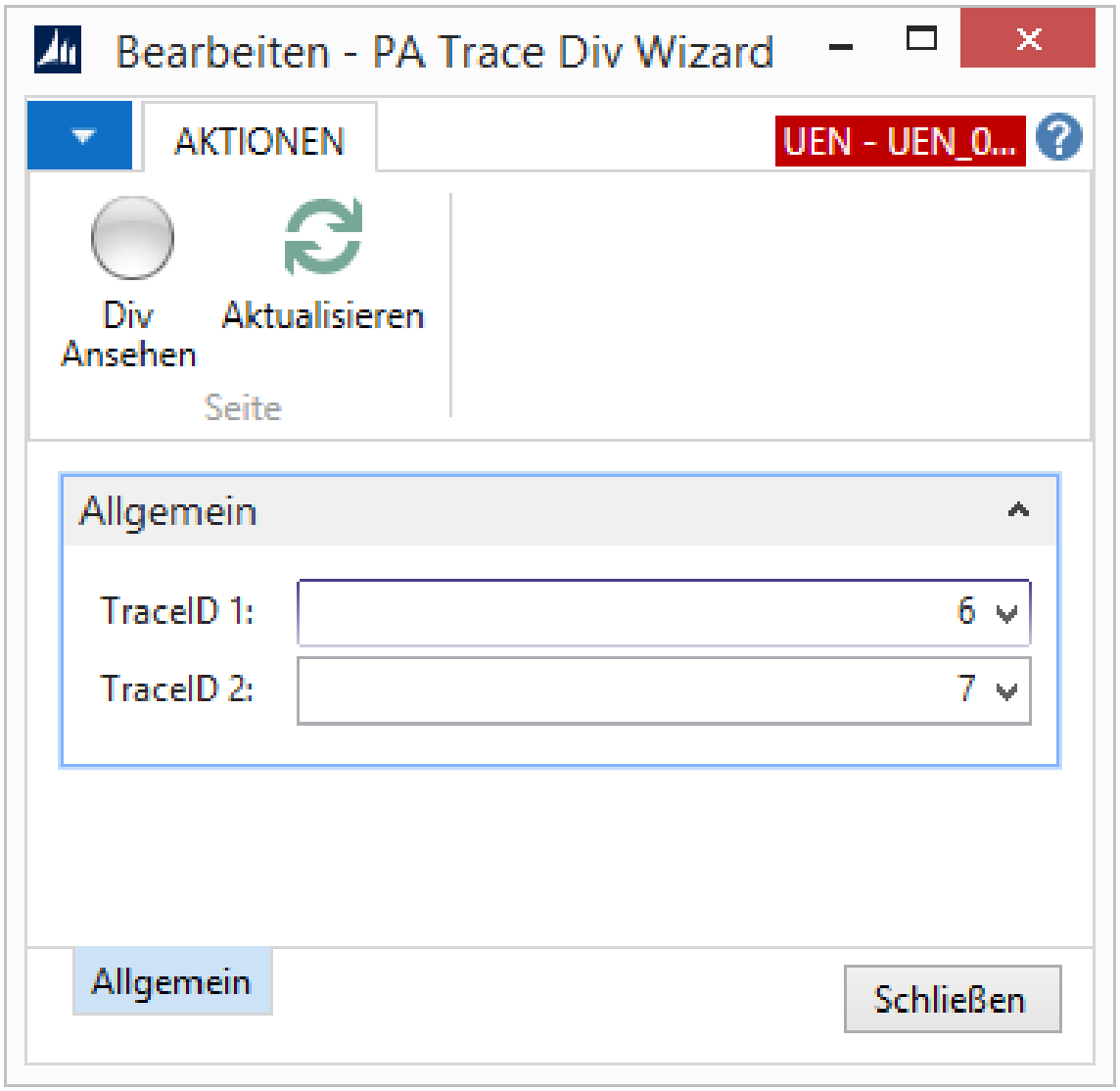


Abbildung A.6: PA Trace Div Wizzard, Quelle: Eigene Darstellung

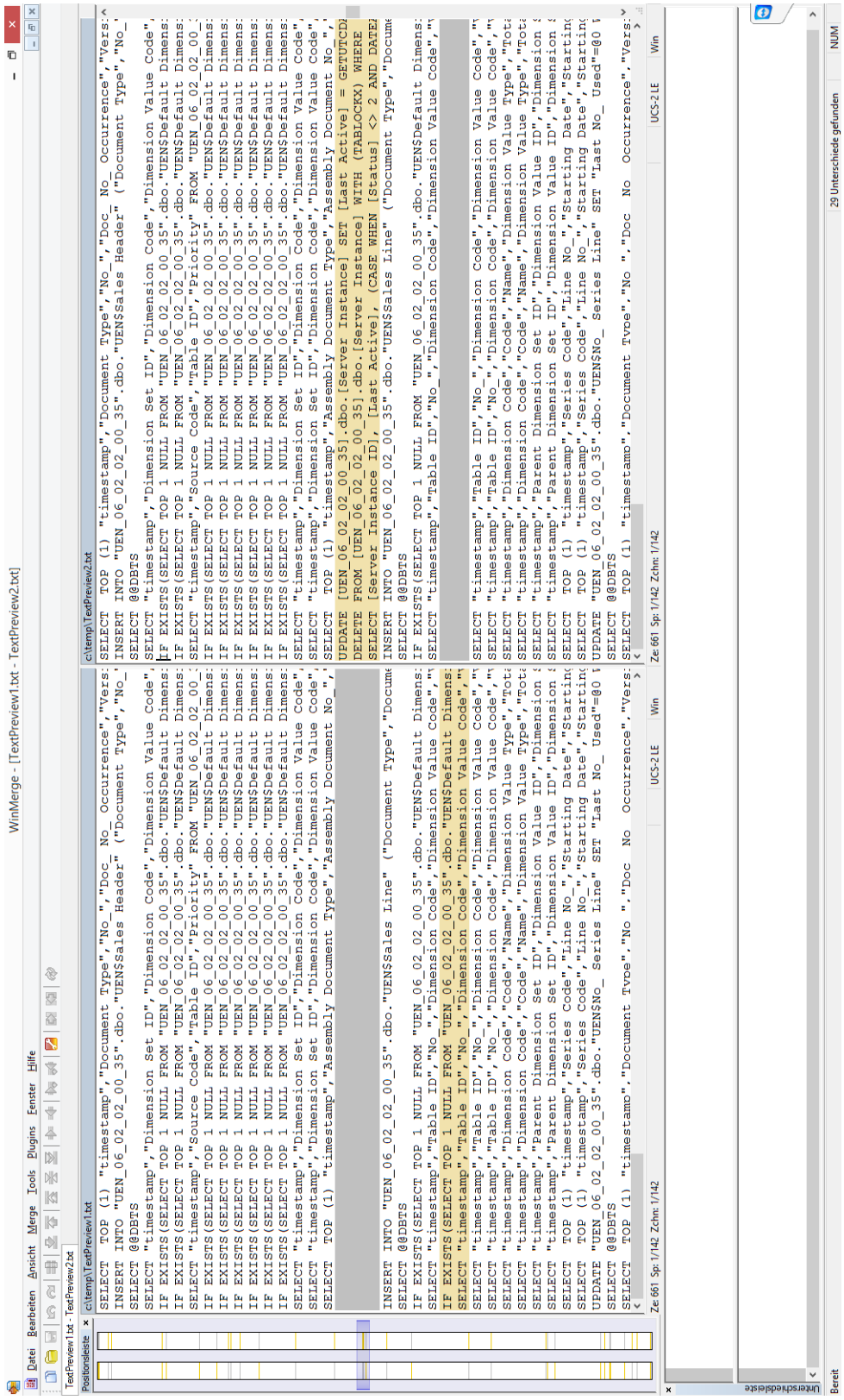


Abbildung A.7: Vergleich zweier unterschiedlicher SQL Traces, Quelle: Eigene Darstellung