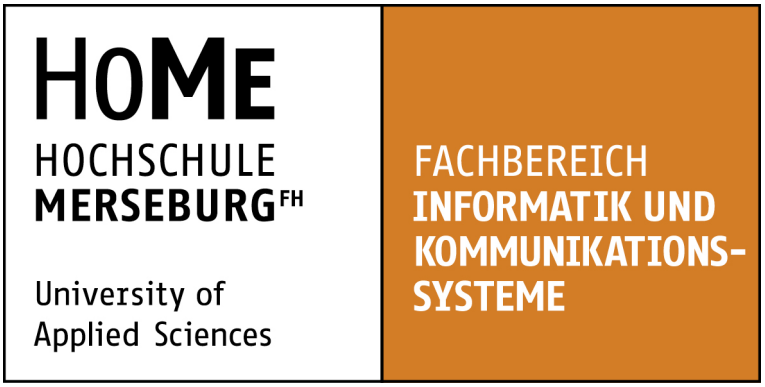
Masterarbeit



Visualisierung der Report Server Utilisation

Vorgelegt von:	Mario Schulz
Matrikelnummer:	17022
Matrikel:	MIKS12
Studiengang:	Informatik und Kommunikationssysteme

Erstbetreuer:	Prof. Dr. rer. pol. Uwe Schröter
Zweitbetreuer:	Dipl.-Informatiker (FH) Michael Werner

9. September 2014

Inhaltsverzeichnis

Nomenklatur

ANSI	American National Standards Institute
BI	Business Intelligence
DM	Data Mining
DOM	Document Object Model
DTD	Document Type Definition
DW	Data Warehouse
EDI	Electronic Data Interchange
ERM	Entity-Relationship-Modell
ERP	Enterprise Resource Planning
ISO	International Organisation for Standardization
JDBC	Java Database Connectivity
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OLE DB	Object Linking and Embedding, Database
RDL	Report Definition Language
RPC	Remote Procedure Call
SGML	Standard Generalized Markup Language
SSRS	SQL-Server Reporting Services
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	eXtensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language
XSLT	eXtensible Stylesheet Language Transformation

1. Einleitung

1.1. Aufgabenstellung

Das betriebliche Berichtswesen beschäftigt sich mit der Auswertung und Aufbereitung von unternehmensspezifischen Daten. Für ein Unternehmen besitzt dieser Bereich einen hohen Stellenwert, da es für das Management eines Unternehmens, der Geschäftsleitung und dessen anderen Führungs- und Entscheidungsebenen nötige Information zur Entscheidungsfindung bereitstellt. Angefangen von einzelnen Wertgrößen, wie dem Umsatz oder den Herstellkosten einzelner Produkte, daraus abgeleiteten weiteren Größen, wie zum Beispiel der Umsatz pro Kunde, bis hin zu detailierten Verkaufsstatistiken einzelner Filialen eines Unternehmens, können erstellte Berichte die verschiedensten Daten optisch ansprechend präsentieren. Die Abbildung 1 zeigt als Beispiel für einen Bericht eine Verkaufsübersicht einer fiktiven Firma. Abhängig von dem Unternehmen, das diese Berichte zur Einschätzung seiner betriebswirtschaftlichen Lage verwendet, kann ein ständiger, mitunter auch tagtäglicher Zugriff darauf eine besonders bedeutungsvolle Rolle spielen.

ADVENTURE WORKS

Sales Order
Order #: SO57030

Bill to:	Closeout Boutique 1050 Oak Street Seattle, Washington 98104 United States	Ship To:	Closeout Boutique 1050 Oak Street Seattle, Washington 98104 United States

Contact: Der Unterbericht 'subStoreContacts' wurde an dem angegebenen Speicherort

Date	Order Date	Sales Person	Purchase Order	Shipment Method
13.05.2014	01.11.2003	David Campbell, Sales Representative 740-555-0182	PO16414180321	CARGO TRANSPORT 5

Line	Order Qty	Product Number	Product Name	Carrier Tracking Number	Unit Price	Subtotal	Discount	Line Total
1	6	FR-M21B-42	LL Mountain Frame - Black, 42	373C-4C0F-88	$149,87	$899,24	$0,00	$899,24
2	4	FR-M21B-48	LL Mountain Frame - Black, 48	373C-4C0F-88	$149,87	$599,50	$0,00	$599,50
3	3	FR-M63S-42	ML Mountain Frame-W - Silver, 42	373C-4C0F-88	$218,45	$655,36	$0,00	$655,36
4	3	FR-M21B-44	LL Mountain Frame - Black, 44	373C-4C0F-88	$149,87	$449,62	$0,00	$449,62

Abbildung 1: Ein beispielhafter Bericht

Mittels des SQL-Server Reporting Services (SSRS) (siehe Abschnitt 2.6.3) vom Microsoft SQL-Server ist es möglich, solche Berichte für Unternehmen zu kreieren. Dabei ist der Report Server der SSRS ein zentraler Bestandteil der SSRS-Installation. Er besteht unter anderem aus unterschiedlichen Erweiterungen, die die Authentifizierung, Datenverarbei-

tung und das Rendering von Berichten übernehmen. [vgl. MicSE]

Bei der hier vorliegenden Masterarbeit geht es um die „Visualisierung der Report Server Utilisation“, also um die sinnvolle Darstellung der Verwendung des Report Servers. Dabei soll ein Werkzeug zur Überwachung eines Report Servers erstellt werden. Mit diesem soll ein Systemadministrator die Verwendung von Quelldaten und die Nutzung von Berichten überprüfen und protokollieren können.

Die Analyse der Nutzungsstatistik soll hierbei nach unterschiedlichen Gesichtspunkten erfolgen. Einer davon ist der Zeitfaktor, also wann Berichte von den Anwendern abgerufen werden und wie oft und häufig dies geschieht. Auch sollen dabei die Reports mit erfasst werden, die überhaupt nicht aufgerufen werden. Berichte können beim Aufruf unterschiedliche, zum Teil frei wählbare Parameter übergeben bekommen. Die Analyse soll auch im Hinblick auf die übergebenen Parameter geschehen. Welche Parameter werden besonders oft an einem Bericht übergeben und gibt es eventuell sogar Parameter, die immer wieder den gleichen Wert beim Abruf eines Berichtes haben.

Ein weiterer Punkt der näheren Betrachtung sind die Nutzer und Anwender der Berichte selbst, das heißt, welche Nutzer rufen welche Berichte auf und geschieht dies immer wieder zu einem gleichen Zeitpunkt. Die Erzeugung von Reports kann abhängig von den zu verarbeitenden Daten einen längeren Zeitraum in Anspruch nehmen. Dies ist ebenso eine wichtige und von dem zu entwickelnden Werkzeug auszuwertende Information. Auch die resultierende Größe eines erstellten Berichtes und genauso dessen Inhalt sind nicht unerhebliche Faktoren, deren Überprüfung eine große Rolle spielen kann. Gegebenenfalls können besonders speicherplatzhungrige Berichte komprimiert oder Berichte mit ähnlichen oder gar gleichen Ergebnismengen zu einem Bericht zusammengefasst werden. Schließlich ist der Status eines Reports gleichermaßen eine bedeutende Information. Berichte, deren Erstellung häufig fehlschlägt, können gezielt auf Fehler untersucht werden.

Damit das für diese Masterarbeit zu schaffende Werkzeug diese einzelnen Faktoren auswerten, aufbereiten und ausgeben kann, liegt dem SSRS ein relationales und umfangreiches Datenmodell zugrunde. Im Zuge dieser Masterarbeit wird dieses Datenmodell genauer untersucht, sowie die Planung und Implementierung des Werkzeuges aufgezeigt. Die anzufertigende Lösung soll dabei vorzugsweise mittels SSRS realisiert werden und darüber hinaus innerhalb der von Microsoft SQL-Server zur Verfügung gestellten Softwarelösungen verfügbar sein. Es soll demnach eine Art **Metareport** entstehen, also ein Report, der die zuvor genannten Faktoren der Verwendung des SSRS auf übersichtliche Art und Weise präsentiert.

1.2. Motivation

Der Nutzen des hier zu erstellenden Überwachungswerkzeuges ist sehr vielfältig. Dies ist nicht nur im Hinblick auf die möglichen kontrollierbaren Faktoren, sondern auch auf die unterschiedlichen von Microsoft SQL-Server bereitgestellten Funktionen gemeint. Das Über-

wachungswerkzeug kann insbesondere durch die Anzeige unterschiedlicher Statistiken den Berichtsverwalter dabei unterstützen, mit den gegebenen Möglichkeiten des Microsoft SQL-Servers verschiedene Optimierungen für die Erstellung von Berichten vorzunehmen, wodurch sich das Nutzungserlebnis des Berichtanwenders erheblich steigern kann.

Auch im Hinblick auf die benötigten Systemressourcen für die Erstellung von Berichten, kann das Werkzeug nützliche Daten bereitstellen. Mit diesen Daten kann der Ressourcenverbrauch womöglich gesenkt oder der Systemadministrator über ungewöhnlich lange blockierte Systemressourcen informiert werden. Das ausführende System kann dadurch entlastet werden.

Die Motivation für die Entwicklung und Umsetzung dieses Überwachungswerkzeuges ist somit vor allem die grundlegende Optimierung des Erstellungsprozesses von Berichten und die Reduktion benötigter Systemressourcen für die Ausführung dieses Prozesses.

1.3. Optimierungsmöglichkeiten

Basierend auf der gegebenen Aufgabenstellung soll das Überwachungswerkzeug bestimmte Funktionen bereitstellen. Dazu gehört unter anderem die frühzeitige Erkennung von Zugriffsproblemen auf einzelne Berichte und damit verbunden die Vorbeugung dieser Zugriffsprobleme. Beispielsweise kann sowohl der Status eines erstellten Reports, als auch dessen Erstellungsdauer ein Indikator für eine mögliche Komplikation beim Aufruf sein. Somit sollte der durchschnittlich benötigte Zeitaufwand jedes abgerufenen Berichtes innerhalb des Metareports angezeigt werden.

Von großem Nutzen für die Anwender der Berichte ist eine mögliche Nutzungsstatistik, die von dem Werkzeug erstellt werden soll. Mitunter kommt es vor, dass Berichte jeden Tag immer wieder zur selben Uhrzeit abgerufen werden. Dies kann beispielsweise dann der Fall sein, wenn die Geschäftsleitung sich zu Beginn jedes aktuellen Geschäftstages über die Einnahmen des entsprechenden Vortages informieren will. In diesem Fall müsste die Geschäftsleitung jedes mal die Erzeugung des Reports manuell anstoßen und die nötige Erstellungszeit des Berichtes abwarten. Eine automatisierte Berichtserzeugung nach einem bestimmten Zeitmuster wäre hier mehr als sinnvoll. Der Microsoft SQL-Server stellt dafür die **Abonnementfunktion** zur Verfügung. Mit diesem Feature kann die Bereitstellung eines Berichtes automatisiert und geplant erfolgen, da Übermittlungserweiterungen des SSRS die verarbeiteten Berichte anhand ihrer festgelegten Abonnementeinstellungen verteilen. Für jedes Abonnement kann ein Zeitplan, ein Ziel und ein Dateiformat für den Report festgelegt werden. So kann ein häufig zur selben Zeit aufgerufener Bericht zum Beispiel jeden Werktag zu einer bestimmten Uhrzeit im PDF-Format in einen freigegeben Ordner kopiert oder an eine festgelegte E-Mailadresse versendet werden. Einige dieser Einstellungen sind in Abbildung 2 auf Seite 4 zu sehen. Die für die Masterarbeit zu erstellende Lösung kann dabei helfen Reports zu erkennen, die mehrfach in gleichen Intervallen vom Berichtsnutzer abgerufen werden. Somit spart sich der Anwender die Generierungszeit des

Berichtes, da dieser bereits im Vorfeld durch die Abonnementfunktion erstellt wurde.

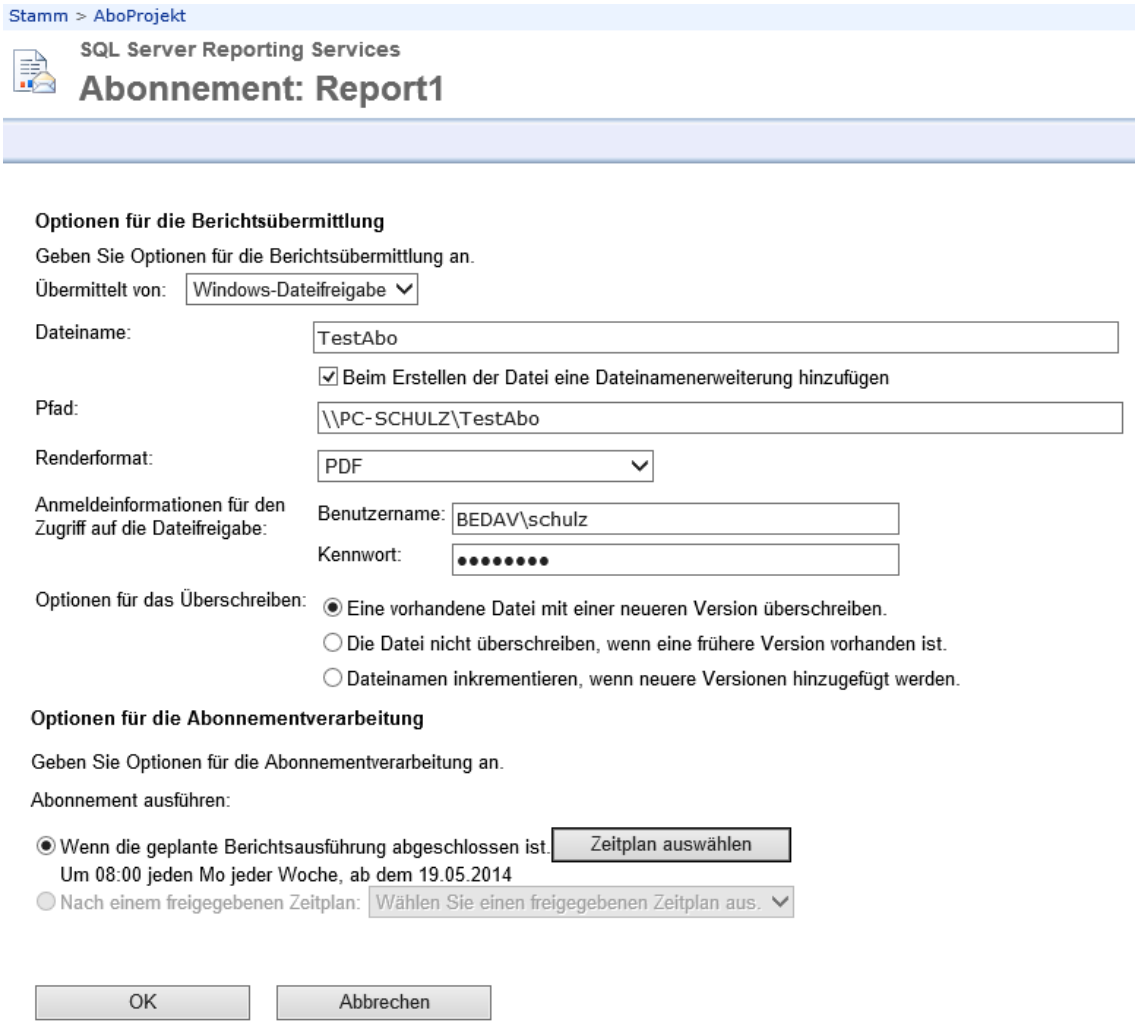


Abbildung 2: Einstellungen für ein Berichtsabonnement

Ein weiteres Feature vom Microsoft SQL-Server ist das **Cachen** bzw. **Zwischenspeichern** eines Berichtes. Dazu erstellt der SSRS eine Kopie eines bereits zuvor erzeugten Reports, die dann dem nächsten Anwender, der diesen Bericht erneut kreiert, angezeigt wird. Legt man für einen Bericht fest, dass dieser vom SSRS zwischengespeichert werden darf, so muss man im gleichen Augenblick auch festlegen, wann eine zwischengespeicherte Version eines Reports abläuft. Dazu kann der Berichtsverwalter bestimmen, dass diese Berichtskopie entweder nach einer frei wählbaren Anzahl von Minuten oder aber zu einer fest geplanten Uhrzeit gemäß eines festlegbaren Zeitplans verfällt. Dabei ist bei Berichten, denen bei ihrer Erstellung Abfrageparameter übergeben werden, zu beachten, dass mehrere verschiedene Versionen dieses Berichtes zwischengespeichert werden. Das heißt, für jede eindeutige Parameter-Wert-Kombination wird genau eine Berichtskopie erzeugt. Andere Nutzer, die denselben Bericht mit der gleichen Parameter-Wert-Kombination abrufen, be-

kommen dann die entsprechend zwischengespeicherte Kopie angezeigt. Sobald eine Kopie im Cache des SSRS verfällt, wird beim nächsten Aufruf des Berichtes dieser komplett neu erzeugt und anschließend eine neue Kopie zwischengespeichert. Der prinzipielle Ablauf des Cachens eines Reports ist in Abbildung 3 dargestellt. In der linken Hälfte der Abbildung ist dabei das Erstellen und Abrufen einer Berichtskopie dargestellt, sobald der Benutzer einen Report abruft. Die rechte Hälfte der Abbildung stellt das automatische Löschen einer Berichtskopie dar, sobald diese abgelaufen ist. Durch das Cachen wird ein Bericht deutlich zügiger abgerufen, was vor allem bei sehr umfangreichen oder auch häufig abgerufenen Berichten einen positiven Nutzen hat. Mit Hilfe des anzufertigenden Werkzeuges kann der Umfang eines Reports oder auch dessen Aufrufhäufigkeit überwacht werden und somit frühzeitig eine passende Zwischenspeicherung durch den Berichtsverwalter konfiguriert werden.

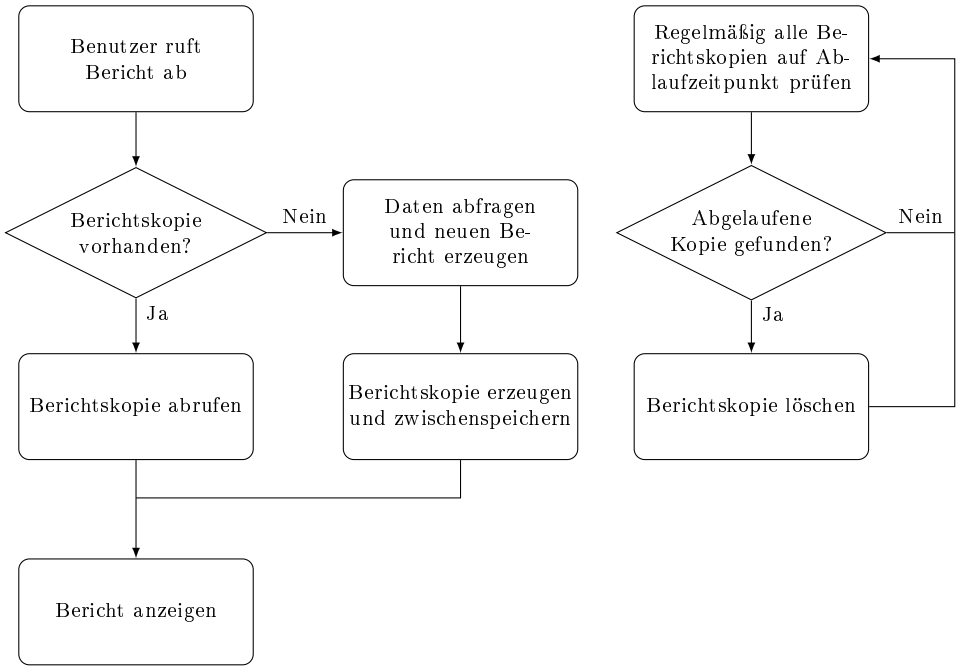


Abbildung 3: Prinzipieller Ablauf des Cachens

Das Erstellen eines Berichtes kann stark abhängig von der Anzahl und dem Umfang der zu verarbeiteten Information eine enorme Zeitspanne in Anspruch nehmen. Während dieser Zeitspanne verbraucht der Rechner, der mit der Aufgabe der Berichtsgenerierung vertraut worden ist, gegenbenenfalls eine erhebliche Menge an Ressourcen. Hierbei kann vor allem die Menge des genutzten Arbeitsspeichers stark anwachsen. Ungeduldige Berichtsnutzer, die dieses Zeitintervall nicht abwarten wollen, könnten geneigt sein, den momentanen Verarbeitungsprozess unsachgemäß zu beenden und anschließend die Erstellung des gleichen Reports erneut anstoßen. Dies hat nicht nur zur Folge, dass für die erneut gestartete Berichterstellung weitere Systemressourcen benötigt werden, sondern auch, dass die zuvor

bereits beanspruchten Ressourcen nicht mehr freigegeben werden, aufgrund des inkorrekten Abbruchs des vorangegangenen Prozesses der Berichtsgenerierung. In solch einem Fall muss der im Hintergrund weiterlaufende Prozess der ersten Berichtserzeugung durch den Berichtsverwalter oder auch dem Systemadministrator manuell gestoppt werden. Leider wissen die Anwender der Berichte oftmals nicht, dass ein Prozess im Hintergrund auf dem System unerwünschte Nebenwirkungen, nämlich das Ausbremsen des Systems, zur Folge hat. Ein merklich langsamerer Zugriff oder gar eine Zugriffsverweigerung auf die einzelnen Reports durch den möglichen Ausfall des Systems können die Folge sein. Auch in diesem Fall kann das in dieser Masterarbeit besprochene Werkzeug Abhilfe schaffen. Den Berichtsverwalter oder auch Systemadministrator kann der Metareport auf einen ungewöhnlich lang andauernden Erstellungsprozess aufmerksam machen. Dadurch kann sich der entsprechende Systemverwalter bereits frühzeitig um das Problem kümmern und dieses im besten Falle auch beheben.

Wie man sieht, sind die hier aufgezeigten Anwendungsmöglichkeiten in Verbindung mit den bereitgestellten Funktionen des Microsoft SQL-Servers zahlreich. Das Werkzeug bzw. der Metabericht kann die Arbeit von Berichtsverwaltern und Systemadministratoren unterstützen und damit die Erfahrungen der Anwender der Berichte maßgeblich verbessern.

2. Grundlagen

2.1. Business Intelligence

2.1.1. Definition

Der Begriff *Business Intelligence* hat sich seit Mitte der 90er Jahre in der betrieblichen Praxis entwickelt. Das Problem an diesem Begriff ist jedoch der unsichere Umgang mit diesem und dessen schwammige Abgrenzung. Im Großen und Ganzen kann man jedoch festhalten, dass sich die meisten Definitionen über die verwendeten Systeme abgrenzen. [vgl. KMU06, S. 2 f]

Nichtsdestotrotz existieren dennoch verschiedene Definitionen für dieses Konzept der entscheidungsorientierten Informationsverarbeitung. Eine davon lautet zum Beispiel:

„Unter Business Intelligence (BI) ist ein Gesamtansatz zu verstehen, mit dem Komponenten für die Beschaffung, Aufbereitung und Bereitstellung von Daten zur Unterstützung betrieblicher Entscheidungsprozesse zusammengeführt werden.“ [Alp11, S. 235]

Auch eine andere Quelle bescheinigt der BI ein noch verhältnismäßig junges Dasein und deshalb auch eine uneinheitliche Verwendung [vgl. BK10, S. 65]. So heißt es dort weiter:

„Allgemein umfasst der Begriff des BI die analytischen Konzepte, Prozesse und Werkzeuge, um Unternehmens- und Wettbewerbsdaten in konkretes Wissen (intelligence) für strategische Entscheidungen umzuwandeln. Es wer-

den Unternehmensinterne und -externe Daten als Quellen herangezogen.“ [BK10, S. 65]

Der bedeutungsreiche englische Begriff *Intelligence* wird auch in [KMU06] als *Information* verstanden, die es zu generieren, speichern, recherchieren, analysieren, interpretieren und zu verteilen gilt. Die dortige Definition für BI lautet wie folgt:

„Unter BI wird ein integrierter, unternehmensspezifischer, IT-basierter Gesamtansatz zur betrieblichen Entscheidungsunterstützung verstanden.“ [KMU06, S. 8]

So heißt es weiter: **„BI–Werkzeuge dienen ausschließlich der Entwicklung von BI–Anwendungen.“** [KMU06, S. 8] und **„BI–Anwendungssysteme bilden Teilaspekte des BI–Gesamtansatzes ab.“** [KMU06, S. 8]

Schaut man sich diese verschiedenen Definitionen für BI an, stellt man fest, dass immer mit Hilfe von verarbeiteten und aufbereiteten internen sowie externen Daten der Prozess der betrieblichen Entscheidungsfindung unterstützt werden soll. Das es sich hierbei um eine IT-gestützte Datenverarbeitung handelt, versteht sich von selbst. Darauf kann man auch kommen, wenn man sich die einzelnen Bestandteile einer BI anschaut.

Zusammenfassend umfasst BI **„Verfahren, Methoden und Werkzeuge, um entscheidungs- und analyserelevante Daten aus unternehmensinternen und -externen Quellen zu integrieren und für Analysezwecke optimiert aufzubereiten“** [Gro10, S. 306].

In der Praxis werden durch umfangreiche Werkzeuge für BI, wie zum Beispiel ERP-Systeme (siehe Abschnitt 2.2), schlichte Daten aussagekräftig und führen zu einer sicheren Entscheidungsgrundlage. Diese Daten können durch das Berichtswesen (siehe Abschnitt 2.3) analysiert und ausgewertet werden, wodurch im Unternehmen Risiken minimiert und der Informationsfluss verbessert werden kann. [vgl. MicBI]

2.1.2. Business Intelligence-Architektur

Grundlegend baut eine BI auf interne oder externe Datenbestände auf, die für die Entscheidungsunterstützung aufzubereiten sind. Hierbei handelt es sich um die **originäre Datenschicht** wie Abbildung 4 auf Seite 8 zeigt. [vgl. Alp11, S. 235]

Diese Datenbestände werden in einem Data Warehouse (DW)-System abgelegt, welches als zentraler Datenspeicher dient und damit die **Bereitstellungsschicht** bildet [vgl. Alp11, S. 235]. Bei einem DW handelt es sich um eine themenbezogene, integrierte und dauerhafte Datenhaltung [vgl. KMU06, S. 10 f].

Auf der noch fehlenden Schicht, der **Dialog- und Analyseschicht**, sind die Anwendungsprogramme angesiedelt, die die bereitgestellten Daten verarbeiten. Zu diesem Zweck werden unter anderem häufig Systeme für das Online Analytical Processing (OLAP), Berichts- und Data Mining-Systeme verwendet. Diese Systeme werden im Folgenden kurz beschrieben. [vgl. Alp11, S. 235]

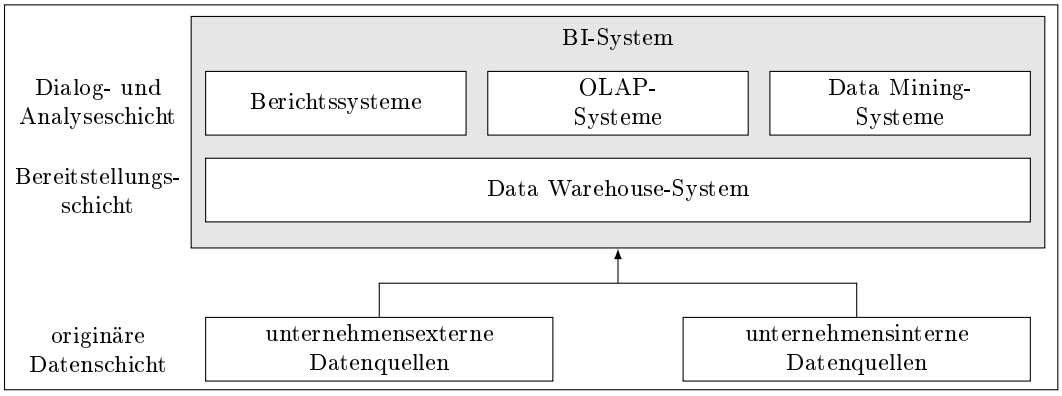


Abbildung 4: BI-Architektur [vgl. Alp11, S. 236]

Das **OLAP-Konzept** zielt darauf ab, Daten speziell für analytische multidimensionale Fragestellungen aufzubereiten. Dabei steht im Kern der Architektur der OLAP-Server, bei dem es sich um einen leistungsstarken Datenbankserver handelt. Dieser extrahiert die Daten aus Datenquellen und speichert diese anschließend in einer multidimensionalen Datenbank, der OLAP-Datenbank. Für das Frontend kommen Werkzeuge zum Einsatz, mit denen die Datenbestände des OLAP-Servers interaktiv untersucht werden können. Dazu nutzt man unter anderem eigenständige Betrachtungsprogramme (Beispielsweise Cognos von IBM, ProClarity von Microsoft oder Targit vom gleichnamigen Unternehmen), Erweiterungen für Tabellenkalkulationsprogramme wie zum Beispiel Microsoft Excel oder auch Anwendungen aus dem World Wide Web (WWW). [vgl. Alp11, S. 242]

Unter dem Begriff des **Data Mining (DM)** versteht man alle Methoden, mit denen in einer gegebenen Datenmenge bisher unbekannte Erkenntnisse aufgedeckt werden können. Als Zielsetzung versteht man die Suche nach Hypothesen, die einer weiteren Verifikation bedürfen. Im Deutschen kann der Begriff DM mit Datenmustererkennung übersetzt werden. Der Prozess des DM ist in unterschiedliche Phasen unterteilt: Die Auswahl der Daten aus geeigneten Datenquellen, die Exploration der Daten, die Stichprobenziehung ausgewählter Datensätze, die Vorverarbeitung der Daten inklusive einer eventuell benötigten Bereinigung der Daten und die Transformation der Daten in die von speziellen DM-Algorithmen benötigte Form. Am Ende dieser zeit- und kostenintensiven Vorarbeiten steht die Wissensgewinnung und damit das eigentliche DM. [vgl. Alp11, S. 33]

Mit Berichtssystemen wird das Management mit relevanten Daten versorgt. Hier steht vor allem der Systemoutput im Vordergrund, da die in diesen Systemen erzeugten Berichte der Leitungs- und Entscheidungsebene inhaltlich richtige und interessante Daten zur Verfügung stellen sollen. Weitere Einzelheiten finden sich in Abschnitt 2.6.3 auf Seite 30. [vgl. Alp11, S. 240]

2.2. ERP-Systeme

2.2.1. Definition

Unter dem Namen Enterprise Resource Planning (ERP)-Systeme sind in den letzten fünfzehn Jahren integrierte betriebswirtschaftliche Standardsoftware-Pakete auf den Markt gekommen, die nahezu alle Aufgabenbereiche und Prozesse im Unternehmen unterstützen [SK10, S. 153].

Ein ERP-System umfasst genauer gesagt die Verwaltung aller zur Durchführung der Geschäftsprozesse notwendigen Informationen über die Ressourcen Material, Personal, Kapazitäten, Finanzen und Information. Daneben ist ein weiteres wesentliches Merkmal von ERP-Systemen die Integration verschiedener Funktionalitäten, Aufgaben und Daten in ein großes Informationssystem. Damit deckt ein ERP-System die Funktionen aus mehreren Unternehmensbereichen ab. Als minimaler Integrationsumfang ist hierbei allerdings eine gemeinsame Datenhaltung anzusehen. Um ERP-Systeme von speziellen Anwendungssystemen, etwa für die Fertigung, besser abgrenzen zu können, sollte ein ERP-System die Verwaltung von mindestens drei der zuvor genannten Ressourcen integrieren.
[vgl. Gro10, S. 4 f]

Neben dem für ERP-Systeme typischen Funktionsumfang, der die Verwaltung von Material, Finanzen und Personal und darüber hinaus die Abbildung der Geschäftsprozesse in Vertrieb, Leistungserstellung und Service beinhaltet, gibt es auch Unterschiede bei der Vielzahl an unterschiedlichen Anwendungssystemen. So weisen manche Systeme eine integrierte Finanzbuchhaltung auf, während andere hingegen ihr System mit einer externen Finanzbuchhaltung koppeln. Aufgrund der Angleichung der Bedienoberflächen und der abgestimmten Schnittstelle zur Übergabe von Buchungssätzen und Kontoinformationen sind diese alternativen Ansätze fast als gleichwertig zu den integrierten Lösungen zu betrachten. Mitunter kann ebenso die Personalverwaltung in ein eigenes Personalverwaltungssystem ausgelagert sein. Auch Funktionen zur Verwaltung von Kundenbeziehungen können entweder in ein ERP-System integriert oder als externes Customer-Relationship-Management-System über entsprechende Schnittstellen an ein ERP-System angebunden sein. [vgl. Gro10, S. 17]

ERP-Systeme sind zurzeit die wichtigsten betriebswirtschaftlichen Transaktionssysteme; Systeme, die die Bearbeitung wiederkehrender Geschäftsvorgänge unterstützen. Sie bilden in Unternehmungen häufig das Rückgrat der Informationsverarbeitung. [vgl. Alp11, S. 29]

Dadurch sind ERP-Systeme heute auch meistens Standard-Softwaresysteme, das heißt, dass sie nicht vom Unternehmen selbst entwickelt werden. Sie werden stattdessen von einem ERP-Vertreiber erworben und an die Bedürfnisse des Unternehmens angepasst, was als Customizing bezeichnet wird (siehe Abschnitt 2.2.2). ERP-Systeme sind auch für verschiedene Unternehmensgrößen und Branchen ausgelegt. So sind manche dieser Systeme nur für kleine oder mittelständische Unternehmen besonders gut geeignet. Andere fokussieren auch

nur die Spezifika einer bestimmten Branche. Beispiele sehr bekannter ERP-Systeme sind: SAP mit SAP ERP, Microsoft mit Dynamics AX und Dynamics Nav und Oracle mit Oracle E-Business-Suite. Mit der Zeit wurden auch einige ERP-Systeme im Open Source-Bereich verfügbar, zum Beispiel AvERP und Compiere. [vgl. BK10, S. 52]

2.2.2. Aufbau eines ERP-Systems

Unabhängig von ihrer genauen Ausprägung sind ERP-Systeme aus mehreren Ebenen aufgebaut. [vgl. Gro10, S. 9]

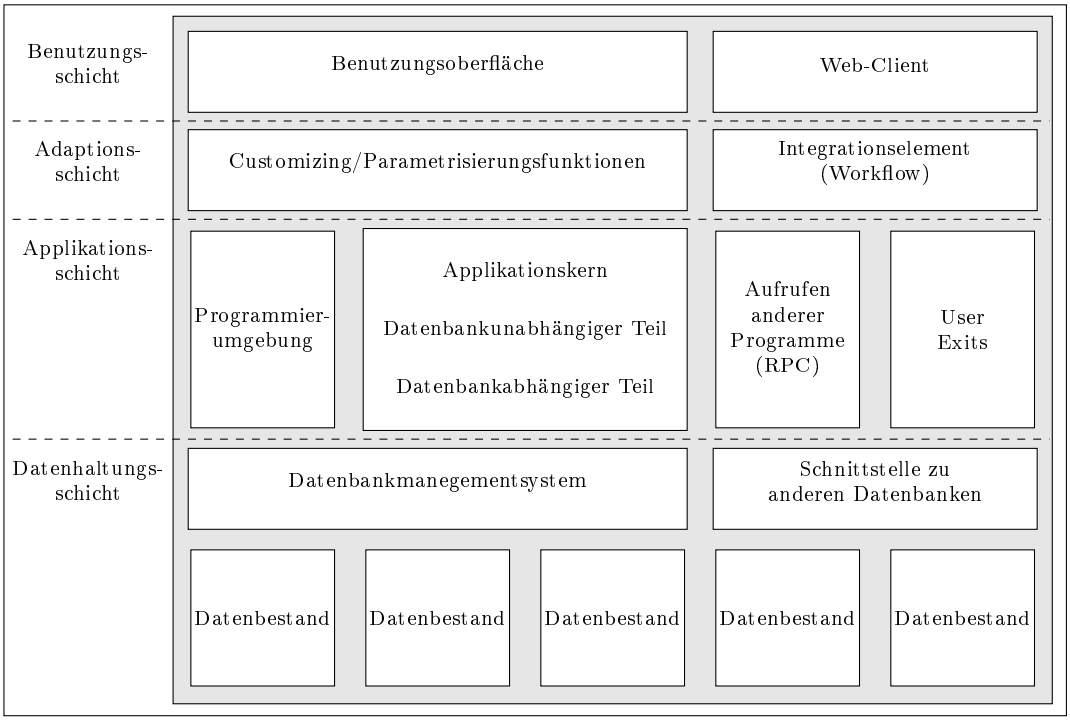


Abbildung 5: Aufbau eines ERP-Systems [vgl. Gro10, S. 9]

Aufgrund ihrer elementaren Eigenschaft bestehen ERP-Systeme auf ihrer untersten Ebene aus einzelnen Datenbeständen, welche mit Hilfe eines entsprechendem Datenbankmanagementsystems zugänglich sind. Am gebräuchlisten sind dabei die Datenbankmanagementsysteme von IBM, Microsoft oder Oracle. Diese grundlegende Ebene ist die Datenhaltungsschicht. Üblicherweise befinden sich hier auch Schnittstellen, die den Zugriff auf andere Datenbanken – auch Datenbanken anderer Informationssysteme – erlauben. [vgl. Gro10, S. 9 f]

Zur zweiten Ebene, der sogenannten Applikationsschicht, gehört unter anderem der Applikationskern. Dieser besteht aus zwei Teilen. Teil eins ist datenbankabhängig und erlaubt der Anwendung den Zugriff auf die Daten, die durch das Datenbankmanagementsystem verwaltet werden. Der zweite und datenbankunabhängige Teil reicht die Daten an den Applikationskern weiter. Mit dieser Trennung ist es möglich, auf die Optimierungsroutinen

der einzelnen Datenbankmanagementsysteme individuell eingehen zu können. Des Weiteren gehört üblicherweise eine Programmierumgebung mit zur Applikationsschicht. Mit dieser, zum ERP-System mit ausgelieferten Programmiersprache, können Anwendungen ergänzt und erweitert werden. Außerdem gehört zum Lieferumfang eines ERP-Systems auch eine Middleware, die den Aufruf anderer Programme über Remote Procedure Call (RPC) oder die Integration von Programmbausteinen, die in anderen Programmiersprachen geschrieben wurden, über so genannte User Exits erlaubt. [vgl. Gro10, S. 10]

In der nächsten Ebene, der Adaptionsschicht, ist das so genannte **Customizing** möglich. Damit kann die Funktionalität des genutzten Datenmodellauschnittes an die jeweils abgebildeten betrieblichen Prozesse und Datenstrukturen angepasst werden. Abhängig vom Umfang des genutzten ERP-Systems können diese Customizing-Funktionen mehr oder weniger stark ausgeprägt sein. Außerdem gehören in diese Schicht rudimentäre Funktionen von Workflow-Management-Systemen wie Weiterleitungs- und Benachrichtigungsmechanismen, Vertretungsregelungen oder der Aufruf von Programmmasken. Diese Funktionen werden dazu genutzt, um Prozesse, die verschiedene Informationssysteme nutzen, in einem allgemeingültigen rechnerunterstützten Modell abbilden zu können. Teilweise sind diese Funktionen auch frei konfigurierbar. [vgl. Gro10, S. 10]

Bei der obersten Ebene eines ERP-Systems handelt es sich um die Benutzungsoberfläche. Typischerweise ist diese heutzutage als Web-Client ausgeprägt, wodurch zur Bedienung des ERP-Systems nur noch die Installation eines Web-Browsers erforderlich ist. Auf der anderen Seite kann hierdurch aber auch der Funktionsumfang gegenüber der Standardbenutzungsoberfläche des ERP-Systems eingeschränkt sein. [vgl. Gro10, S. 10 f]

2.2.3. Vor- und Nachteile durch ERP-Systeme

Im Allgemeinen werden durch den Einsatz eines ERP-Systems erhebliche Vorteile erzielt [vgl. Gro10, S. 12]. Die Tabelle 1 auf Seite 12 gibt einen Überblick über diese Vorteile.

Weitere hervorzuhebenede Vorteile von ERP-Systemen liegen in der Automatisierung von Abläufen und der Standardisierung von Prozessen. Dadurch kann die Produktivität erhöht werden, indem Aktivitäten rationalisiert, Sachmittel ökonomisch eingesetzt und die Zahl der Arbeitsstationen und Transportwege vermindert werden. Außerdem kann die Koordination erleichtert werden, da Doppelbearbeitungen vermieden werden können. Ein organisatorisches Konfliktpotenzial kann durch das Festlegen eindeutiger Kompetenzen gemindert und ein lückenloses Ineinandergreifen der Verrichtungshandlungen kann gewährleistet werden. Des Weiteren entlastet die Standardisierung Führungskräfte durch eine gewissermaßen automatisierte Steuerung der Prozesse. Dies begünstigt wiederrum das Setzen von Schwerpunkten. Schließlich wird durch die Standardisierung auch die Stabilität des Systems erhöht, aufgrund der weitgehend unabhängig werdenden Aktivitätsfolgen von beteiligten Personen. [vgl. Gro10, S. 13]

	vor ERP-Einsatz	mit ERP-Einsatz
Durchlaufzeit	Kostenintensive Engpässe (z.B. Personal)	Zeit- und Kostenersparnis in Geschäftsprozessen
Auftragsbearbeitung	Bearbeitung durch mehrere Stellen benötigt Daten an mehreren Stellen (Kunden, Produkte, Aufträge)	Schnellere Bearbeitung durch gemeinsame Daten reduziert Zeitbedarf und Aufwand für mehrfache Aktualisierung
Finanzielle Situation	Steigende Kosten durch Überbestände und zu hohe Außenstände	Verbesserung der operativen Leistung durch Bestandskontrolle und automatisches Mahnwesen
Geschäftsprozesse	Verbreitung fragmentierter Abläufe mit Mehrfachaufwand	Neugestaltung basierend auf „Best Practice“-Prozessen
Produktivität	Fehlende Fähigkeit, schnell gegenüber Kunden und Lieferanten reagieren zu können	Verbesserungen beim Liquiditätsmanagement und Kundenservice
Supply Chain-Management	Fehlende Integration	Verbindungen zu Lieferanten und Kunden
E-Business	Web-Schnittstellen als isolierte Systeme bzw. Einzelkomponenten	Web-Schnittstellen sind das Front-End des ERP-Systems[1]
Information	Keine effiziente Beobachtung und Steuerung der Ressourcen des Unternehmens	Bereichsübergreifender Zugang zu den gleichen Daten zur Planung und Steuerung
Kommunikation	Keine effiziente Kommunikation mit Kunden und Lieferanten	Ermöglicht die Kommunikation des Unternehmens mit Kunden und Lieferanten[2]

Tabelle 1: Vorteile durch den Einsatz eines ERP-Systems [Gro10, S. 12]

Allerdings führt eine zu starke Standardisierung auch zu erheblichen Nachteilen. Dazu gehört die verminderte Anpassungsfähigkeit an ungeplante Einflüsse. Dieser Mangel an Flexibilität kann zu hohen Umstellungskosten führen. Auf lange Sicht kommt es darüber hinaus auch zu einem Verlust an Initiative und Bereitschaft, neue Wege der Problemlösung zu gehen und Innovationen zu tätigen. Formale Elemente der Organisation werden in den Vordergrund gerückt, was in einer Bürokratisierung gipfeln kann. Bei Mitarbeitern kann eine starke Standardisierung zu Motivations- und Identifikationsproblemen führen, da sich nur selten die Gelegenheit zum selbständigen Entscheiden und Handeln ergibt. Zusätzlich erschwert die Fremdbestimmung des Verhaltens der Mitarbeiter auch die Förderung und Ausprägung eines höheren Reifegrades der Organisationsmitglieder. Schließlich stellen die in einem ERP-System abgebildeten Abläufe ein Modell des Geschäftes dar, das

[1] In der Praxis müssen es nicht nur Web-Schnittstellen sein, so gibt es unter anderem auch die Möglichkeit, Microsoft Office Produkte als Front-End für ein ERP-System zu verwenden. [siehe Nie08]

[2] Unterstützend kommen dabei Electronic Data Interchange (EDI)-Systeme zum Einsatz. [vgl. EDI14]

auf den Vermutungen des Softwareherstellers über dieses Geschäft basiert. Weichen diese Vermutungen zu weit vom Verständnis des ERP-System anwendenden Unternehmens über dessen Geschäft ab, so führt eine ERP-Systemeinführung nicht zu dem erhofften Erfolg. [vgl. Gro10, S. 13-15]

2.3. Berichtssysteme

2.3.1. Definition und Klassifizierung

Um die Entscheidungsebene mit bedeutsamen Daten zu versorgen werden Berichtssysteme verwendet. Diese Systeme erzeugen als Output Berichte, die der Leitungsebene eines Unternehmens inhaltlich richtige und wichtige Information zur Verfügung stellen. Darüber hinaus können Berichtssysteme auch für die Massenberichtserstattung von Kunden verwendet werden, um zum Beispiel über die jährlichen Wertverläufe von Lebensversicherungen zu informieren. [vgl. Alp11, S. 240]

Bei Berichtssystemen wird zwischen aktiven und passiven Systemen unterschieden. Die Unterscheidung wird anhand der Auslösung der Berichtserstellung getroffen. [vgl. Alp11, S. 240]

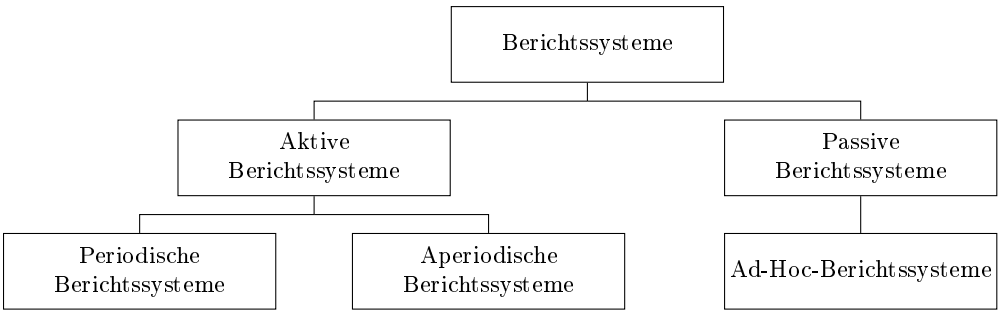


Abbildung 6: Klassifizierung der Berichtssysteme [vgl. KMU06, S. 111]

Erfolgt die Erstellung der Reports automatisch, so spricht man von **aktiven Berichtssystemen**. Die automatische Erstellung der Berichte kann dabei zu bestimmten Zeitpunkten oder aufgrund von zuvor festgelegten Datenkonstellationen erfolgen. Oft verfügen die Berichte über einen fest definierten Inhalt und sie werden außerdem für einen bestimmten Kreis an Adressaten generiert. Von einem **Standardbericht** (oder auch einem **periodischen Berichtssystem** [KMU06, S. 111]) spricht man dann, wenn die Berichtserstellung periodisch, also in festgelegten Intervallen erfolgt. Hierbei kann es sich beispielsweise um einen Monatsbericht mit den entsprechenden grundlegenden betrieblichen Kennzahlen handeln. Hingegen spricht man von einem **Ausnahmebericht** (bzw. einem **aperiodischen Berichtssystem** [KMU06, S. 111]), wenn die Berichtserstellung nicht periodisch, sondern durch bestimmte Datenkonstellationen, die vorher durch Regeln bestimmt wurden, ausgelöst wird. Als Beispiel sei hier das Unterschreiten eines zuvor definierten Mindestbetriebsergebnisses genannt. [vgl. Alp11, S. 240]

Bei **passiven Berichtssystemen** hingegen, wird die Erstellung der Berichte durch den Anwender ausgelöst (man spricht auch von **Ad-Hoc-Berichtssystemen** [KMU06, S. 112]). Hierbei kann der Nutzer durch das Interagieren mit dem Berichtssystem maßgeschneiderte Reports zu einem bestimmten Thema anfertigen. Dabei stellen solche Systeme oft bereits fertige und parametrisierbare Berichtsvorlagen für den Benutzer bereit, die innerhalb der Systeme adäquat verwaltet werden können. [vgl. Alp11, S. 240 f]

2.3.2. Reporting-Plattformen

Ältere Berichtssysteme waren zumeist Individualentwicklungen. Für die konkrete Ausgestaltung boten diese häufig keine guten allgemeingültigen Werkzeuge an. Dieser Umstand hat sich mittlerweile stark geändert und es stehen heute eine Vielzahl an Entwicklungswerkzeugen zur Verfügung. Dabei handelt es sich bei Reporting-Plattformen um die umfassendste Form, da diese alle Prozesse unterstützen und das Berichtswesen vereinheitlichen. [vgl. KMU06, S. 112]

Üblicherweise sind diese Plattformen ein integraler Bestandteil von betrieblichen Standardsoftwaresystemen (siehe Abschnitt 2.2). Die Berichte, die man ändern und betrachten will, kann man sich abhängig von der Plattform aus einer großen Anzahl an bereits vordefinierten Berichten auswählen. Daneben gibt es am Softwaremarkt aber auch eigenständige Reporting-Plattformen, die an eine Vielzahl von verschiedenen Datenquellen angebunden werden können. [vgl. Alp11, S. 241]

Um einen Bericht entwerfen und gestalten zu können, wird innerhalb der Reporting-Plattformen eine entsprechende Entwurfsumgebung zur Verfügung gestellt. Innerhalb dieser kann der Endanwender oft recht intuitiv Berichte entsprechend dessen hohen inhaltlichen und grafischen Anforderungen erstellen. Hierzu werden innerhalb der Entwurfsumgebung abstrakte Schablonen (oder auch Reporting Templates [Alp11, S. 241]) erzeugt. Dabei werden einzelne und von der Entwurfsumgebung zur Verfügung gestellte Berichtselemente per Drag-and-drop-Technik zur Schablone hinzugefügt. Außerdem kann die Schablone um Texte, Grafiken und Daten aus operativen Systemen sowie multimedialen Inhalten ergänzt werden. [vgl. KMU06, S. 112 f]

Damit die generierte Berichtsschablone mit realen Inhalten befüllt werden kann, muss der Bericht anschließend ausgeführt werden. Diese Aufgabe wird der entsprechenden Reporting Engine zuteil. Die Reporting Engine bildet die zentrale Steuerungskomponente der Berichtsplattform. [vgl. Alp11, S. 241]

Das wirkliche Berichtsergebnis bei der Berichtserstellung folgt aus dem Befüllen der Berichtsschablone aus der dispositiven Datenhaltung. So können periodische, aperiodische und Ad-Hoc-Berichte angefertigt werden. Auch das Entwerfen und Erzeugen interaktiver Berichte ist möglich. Diese Sonderform bietet die Möglichkeit, einzelne Parameter des Berichtes auszuwählen. Dadurch können sogar direkt zur Laufzeit noch die Details eines Berichtes den Wünschen des Betrachters angepasst werden. Beispielsweise kann somit ei-

ne bestimmte Filiale ausgewählt werden, wodurch anschließend die Erzeugung des dazu passenden Detailberichtes angestoßen wird. Abgesehen davon werden auch alle gängigen und strukturierten Formate (zum Beispiel XML [siehe Abschnitt 2.5]) von den einzelnen Plattformen unterstützt. [vgl. KMU06, S. 113]

Für die Verteilung, Diskussion und Verwaltung der einzelnen Berichte sind das Intranet und auch das Internet die bevorzugten Kommunikationsplattformen. Mit diesen werden die verschiedensten Funktionen umgesetzt oder eingebunden. Dazu gehören unter anderem Push-Ansätze der Berichtsverteilung per E-Mail oder auch eher führungsorientierte Möglichkeiten der Weiterverarbeitung, so zum Beispiel das Kommentieren, die Wiedervorlage und Weiterleitung. Auch portalbasierte Verwaltungssysteme können damit umgesetzt werden. [vgl. KMU06, S. 113 f]

2.4. Datenbanken

2.4.1. Definition und relationale Datenbanken

Grundlegend ist eine **Datenbank** eine Menge von zusammenhängenden Daten. Zu ihnen gehören auch papiergebundene Ansammlungen von Daten, wie zum Beispiel Telefonbücher. Diese sind allerdings eine recht sperrige Angelegenheit. Aufgrund vieler Einträge nimmt das Suchen erheblich viel Zeit in Anspruch, die Sortierung innerhalb der Datensammlung ist festgelegt und kann nachträglich nicht mehr geändert werden und außerdem beginnt eine solche papiergebundene Datensammlung bereits ab dem Moment des Druckvorganges zu altern. [vgl. Bea07, S. 1]

Um die Nachteile dieser manuellen Systeme auszumerzen, wurden **Datenbanksysteme** entwickelt. Datenbanksysteme gehören mit zu den ersten Computeranwendungen überhaupt und da sie elektronisch arbeiten, sind sie dazu in der Lage, Daten schneller abzufragen, diese mit verschiedensten Indizes zu versehen und den Benutzern immer die neuesten Daten zu liefern. [vgl. Bea07, S. 1]

Sie sind folgendermaßen definiert:

„Ein Datenbanksystem [...] ist eine Ansammlung von Daten, die allen Benutzern bzw. Anwendungen zur Verfügung steht und in der die Daten nach einheitlichen Regeln abgespeichert werden. Ein Datenbanksystem besteht aus einer Datenbasis und einem Datenbankmanagmentsystem. Der Begriff der Datenbank wird synonym verwendet.“ [FW07, S. 21]

Unter einer **Datenbasis** versteht man die eigentlichen, in der Datenbank enthaltenen Daten, die auch physikalisch in einem Dateisystem gespeichert werden. Verwaltet wird die Datenbasis von dem **Datenbankmanagementsystem**. Ein solches Softwaresystem steuert alle Prozesse der Datenbank und stellt die unterschiedlichen Dienstleistungen zur Verfügung. [vgl. FW07, S. 22]

Jedem Datenbankmanagementsystem liegt ein **Datenmodell** zugrunde, welches die Da-

tenobjekte und deren Operatoren festlegt die verwendet werden dürfen. Es besteht aus Basisdatentypen und Typkonstruktoren, mit denen aus den einfachen Datentypen komplexere Daten und darüber hinaus Typkonstruktionsregeln erstellt werden können. Diese Regeln legen die möglichen Kombinationen von Basisdatentypen und Typkonstruktoren fest. Somit beschreibt ein Datenmodell die fundamentale Organisation und die zur Verfügung stehenden Typen eines Datenbankmanagementsystems. Alle konkreten Ausprägungen daraus mit deren Daten bezeichnet man hingegen als **Datenbankschema**. Dieses gibt die konkreten Typen wieder, die zur Abbildung einer bestimmten Miniwelt verwendet werden. [vgl. FW07, S. 30]

In den ersten Jahrzenten wurden Datenbankdaten auf unterschiedliche Art und Weise und somit mit verschiedenen Datenmodellen gespeichert und dargestellt. Beispielsweise gibt es **hierarchische Datenbanksysteme**, die die Daten in sogenannten Baumstrukturen anordnen. Dabei hat jeder einzelne Baum entweder genau einen oder keinen Elternknoten und beliebig viele Kindknoten. Man spricht hierbei auch von einer Single-Parent-Hierarchie. [vgl. Bea07, S. 2-3]

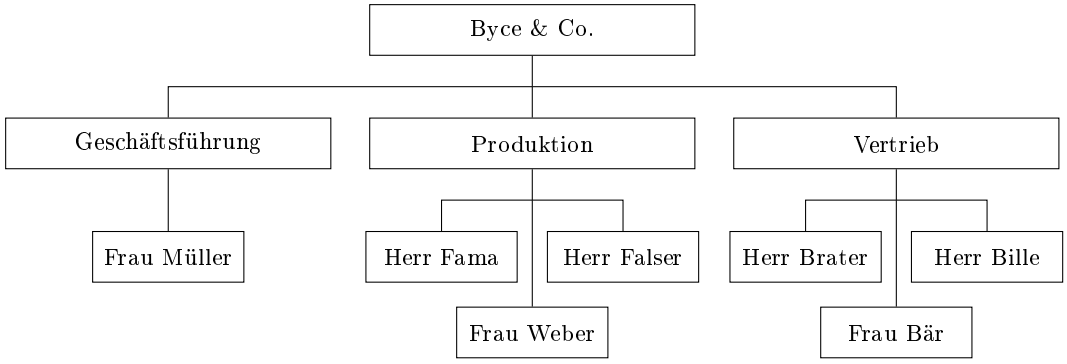


Abbildung 7: Beispiel hierarchische Datenbank [vgl. FW07, S. 31]

Ein weiteres Datenmodell ist das **Netzwerk-Datenbanksystem** (siehe Abbildung 8 auf Seite 17). Dieses stellt Datensätze und Verknüpfungsmengen dar, um damit die Beziehungen zwischen den einzelnen Datensätzen genau zu definieren. Oft kann man dabei von verschiedenen Punkten aus auf einzelne Einträge zugreifen, weshalb solch ein Datenbanksystem auch eine Multi-Parent-Hierarchie ist. Diese beiden Arten von Datenbanksystemen trifft man hauptsächlich noch im Mainframe-Umfeld an. [vgl. Bea07, S. 3 f]

Dr. E. F. Codd vom IBM Forschungslabor brachte 1970 ein Paper mit dem Titel „A Relational Model of Data for Large Shared Data Banks“[3] heraus. In diesem Paper riet er dazu an, die Daten als Mengen von Tabellen (auch Relationen genannt) abzubilden. Jede Tabelle einer **relationalen Datenbank** besteht aus mehreren Zeilen und kann eine unterschiedliche Information besitzen, mit der eine einzelne Zeile der Tabelle eindeutig identifiziert werden kann. Diese Information wird als **Primärschlüssel** bezeichnet und

[3] Zu finden in [Cod70].

kann aus mehreren Spalten der Tabelle zusammengesetzt sein, die jeweils eine einzelne Information enthalten. Weiterhin enthält eine Tabelle alle weiteren Daten, die für die vollständige Beschreibung der dargstellten Entität benötigt werden. Dazu gehören auch Daten die dazu verwendet werden, um zu einer anderen Tabelle zu navigieren. Diese Spalten, die zusammengenommen die Zeile innerhalb einer anderen Tabelle identifizieren, bezeichnet man als **Fremdschlüssel**. Sie zeigen also die Verbindungen der einzelnen Tabellen untereinander an. Das heißt, um Datensätze verschiedener Tabellen miteinander zu verbinden, werden in einem relationalem Datenbankmodell keine Zeiger zwischen zusammenhängenden Entitäten verwendet, sondern stattdessen redundante Daten. [vgl. Bea07, S. 4-6]

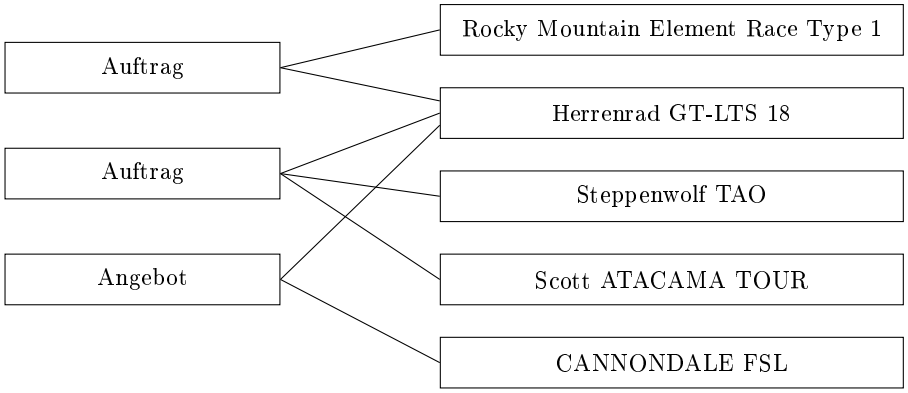


Abbildung 8: Beispiel Netzwerk-Datenbank [vgl. FW07, S. 31]

Ebenfalls von Codd stammen auch die 12 Basisregeln, die eine relationale Datenbank ausmachen und die sich auch auf andere Datenbanktypen abstrahieren lassen. [vgl. FW07, S. 35]

- **Informationsregel**
  In einer relationalen Datenbank wird jede Information einmalig durch Werte in Tabellen abgebildet und redundanzfrei und einheitlich verwaltet. Dies trifft sowohl auf die Anwendungsdaten, als auch auf die Metadaten zu, die den Aufbau einer Datenbank beschreiben.

- **Garantierter Zugriff**
  Jedes einzelne Feld der Datenbank kann durch eine entsprechende Kombination aus Tabellenname, Primärschlüssel und Spaltenname abgerufen werden.

- **Systematische Behandlung fehlender Informationen**
  Um Tabellenspalten mit einer nicht vorhandenen Information von den Standardwerten der einzelnen Datentypen unterscheiden zu können, müssen diese in einer einheitlichen Form dargestellt sein. Diese sogenannten Nullwerte sind eine systematisch fehlende Information. Jede einzelne Spalte einer Relation kann so konfiguriert werden, dass sie das Eintragen von Nullwerten verweigert.

- **Dynamischer Online-Katalog (Data Dictionary)**
  Mit einem Data Ditionary sind die Tabellen gemeint, in denen das Datenbankschema gespeichert ist. Da das Datenbankschema genauso wie die eigentlichen Daten gespeichert wird, kann ein entsprechend autorisierter Anwender auch auf dieses Data Dictionary zugreifen und somit das Datenbankschema einsehen.

- **Allumfassende Sprache**
  Das Datenbanksystem muss eine Sprache unterstützen, die die Erfüllung der folgenden Aufgaben ermöglicht:
  - Benutzerdaten definieren und manipulieren
  - definieren von virtuellen Tabellen (sogenannte Sichten)
  - Integritätsregeln prüfen
  - Rechtevergabe an Benutzer und Autorisierung
  - Kontrolle und Handling von Transaktionen (eine Folge von Datenänderungen, die entweder vollständig oder gar nicht durchgeführt wird)

- **Benutzersichten und Datenänderungen**
  Hierunter wird die Notwendigkeit verstanden, für verschiedene Benutzergruppen und Anwendungen auch unterschiedliche Sichten zu besitzen. Außerdem sollen bei einfachen Sichten auch Datenänderungen möglich sein.

- **HIGH-LEVEL INSERT, UPDATE und DELETE**
  Es muss die Möglichkeit geben, Daten in der Datenbank einfügen, ändern und löschen zu können. Für eine schnelle Durchführung soll das System dabei selber den optimalen Zugriffspfad suchen.

- **Physische Datenunabhängigkeit**
  Werden Änderungen an der Speicherstruktur oder der Zugriffsmethode in der Datenbank angewandt, müssen Anwendungsprogramme und Anwenderoberflächen davon unberührt bleiben.

- **Logische Datenunabhängigkeit**
  Auch sollen Anwendungsprogramme und Anwenderoberflächen unverändert bleiben, wenn sich die Basisrelationen ändern, von denen die Anwendungsprogramme nicht betroffen sind. Beispielsweise könnten in einer Tabelle weitere Spalten ergänzt oder gar neue Tabellen erzeugt werden.

- **Integritätsunabhängigkeit**
  Alle Integritätsbedingungen, die die Datenbank zu erfüllen hat, werden im Data Dictionary hinterlegt und gehören nicht ins Anwendungsprogramm. Die Definition erfolgt mittels der relationalen Datenbanksprache und wird vom Datenbankmanagementsystem ausgeführt.

- **Verteilungsunabhängigkeit**
  Wird eine verteilte Datenhaltung über mehrere Rechner hinweg eingeführt oder wieder zurückgenommen und werden mehrere Datenbanken zu einer Datenbank zusammengefasst, bleiben die Anwendungsprogramme davon unberührt.

- **Unterwanderungsverbot**
  Erlaubt das Datenbankmanagementsystem eine andere Programmiersprache der 3. Generation, wie zum Beispiel C oder Java, darf diese Sprache die zuvor genannten 11 Regeln weder verletzen noch außer Kraft setzen.

[vgl. FW07, S. 35-37]

Mitunter werden diese Regeln auch als die „goldenen Regeln“ von Datenbanken bezeichnet. [vgl. FW07, S. 37]

2.4.2. Die Datenbanksprache SQL

Mit der Beschreibung des relationalen Modells stellte Codd auch eine Sprache namens DSL/Alpha vor. Damit sollten die Daten innerhalb der relationalen Tabellen bearbeitet werden können. Für die Entwicklung des Prototyps dieser Sprache, der auf den Ideen von Codds Paper basieren soll, stellte IBM bereits kurze Zeit nach der Herausgabe von Codds Paper eine Arbeitsgruppe zusammen. Zuerst wurde eine vereinfachte Version der Sprache mit dem Namen SQUARE kreiert. Es folgten Verfeinerungen an dieser ersten Version der Sprache, wodurch die Sprache SEQUEL entstand. Schließlich wurde diese Sprache später in SQL umbenannt, wobei SQL kein Akronym ist und derren Name entweder als einzelne Buchstaben oder wie das englische Wort „Sequel“ ausgesprochen werden kann. [vgl. Bea07, S. 7]

Im Jahre 1981 erfolgte die Markteinführung mit SQL/Data Systems. Schon kurz darauf folgten einige weitere Datenbanksysteme wie DB2 und Oracle, allerdings setzte sich einzig der mächtige Ansatz von IBM durch. Bereits 1980 wurde an der Standardisierung gearbeitet. Es entstand der als SQL1 bezeichnete Standard, welcher in den Jahren 1986 von der American National Standards Institute (ANSI) und 1989 von der International Organisation for Standardization (ISO)-Behörde verabschiedet wurde. Mit der Zeit nahm der Umfang der Datenbanksprache SQL mehr und mehr zu und wurde in den unterschiedlichen ANSI-SQL-Standards beschrieben (siehe Tabelle 2 auf Seite 20). [vgl. FW07, S. 193]

Die Datenbanksprache SQL geht mit dem relationalen Datenbankmodell Hand in Hand. Dies zeigt sich schon dadurch, dass das Ergebnis einer SQL-Abfrage immer eine Tabelle (auch Ergebnismenge genannt) ist. Eine Ergebnismenge, die durch eine Abfrage zurückgegeben wird, kann durch einfaches Speichern bereits als eine neue permanente Tabelle innerhalb einer relationalen Datenbank genutzt werden. Außerdem kann eine Abfrage

nicht nur auf bereits in der Datenbank vorhandene Tabellen zugreifen, sondern auch auf Ergebnismengen anderer Abfragen und diese somit als Eingabe nutzen. [vgl. Bea07, S. 7]

Jahr	Bezeichnung	Besonderheiten
1989	SQL1	erster Sprachstandard
1992	SQL2 bzw. SQL-92	-
1999	SQL3 bzw. SQL1999	Trigger und rekursive Abfragen als neue Features
2003	SQL2003	erste Änderungen für bessere XML-Unterstützung
2006	SQL2006	genauere Festlegung von Zusammespiel von SQL und XML
2008	SQL2008	neuer „INSTEAD OF“-Trigger und neues „TRUNCATE“-Statement
2011	SQL2011	aktuelle Revision

Tabelle 2: Die einzelnen SQL-Standards [vgl. SQLHI]

Die für die Abfragen zur Verfügung gestellten SQL-Anweisungen lassen sich in verschiedene Kategorien unterteilen. Zum einen gibt es **SQL-Schema-Anweisungen**. Mit diesen lassen sich die Strukturen der Datenhaltung festlegen, die innerhalb der Datenbank gespeichert werden. Zu dieser Art von Anweisungen gehören unter anderem „CREATE“ zum Anlegen neuer Tabellen oder auch „DROP“ zum Entfernen bereits vorhandener Tabellen. Durch **SQL-Datenanweisungen** können die zuvor definierten Datenstrukturen dann bearbeitet werden. Zum Beispiel kann mit der „INSERT“-Anweisung eine neue Zeile in eine Tabelle eingefügt und mit der „DELETE“-Anweisung wieder entfernt werden. Mittels **SQL-Transaktionsanweisungen** können Transaktionen, also Folgen von einzelnen Änderungsanweisungen, gestartet, beendet oder auch im Fehlerfall wieder zurückgerollt werden. Typische Anweisungen dieser Kategorie sind „COMMIT“ und „ROLLBACK“. Mit beiden Anweisungen wird eine Transaktion beendet, wobei nur bei einer „COMMIT“-Anweisung alle Änderungen der Transaktion ausgeführt werden, während im Fehlerfall oder auch beim Ausführen der „ROLLBACK“-Anweisung alle Änderungen einer Transaktion wieder rückgängig gemacht werden. [vgl. Bea07, S. 7 f]

Trotz dieser Vielzahl an Anweisungen ist es allein mit SQL nicht möglich vollständige Anwendungen zu schreiben. Im Gegensatz zu Programmiersprachen wie Java, C#, C und vielen anderen, muss man bei SQL einen Teil der Kontrolle abgeben. Dies ist der Optimierungskomponente der Datenbank-Engine geschuldet, die sich die formulierten SQL-Anweisungen genau betrachtet und unter Einhaltung der Tabellenkonfiguration und der zur Verfügung stehenden Indizes versucht, den effizientesten Ausführungspfad zu ermitteln. Zwar ermöglichen viele Datenbank-Engines die Beeinflussung der Optimierungsentscheidungen mittels sogenannter Optimierungshinweise (auch optimizer hints genannt), jedoch werden diese von normalen SQL-Anwendern selten genutzt. Um also mit SQL Daten zu

bearbeiten, schreibt man entweder ein einfaches Skript oder man integriert SQL in die entsprechende Programmiersprache. Manche Datenbankhersteller haben eine solche Integrierung bereits vorgenommen und stellen diese den Benutzern bereit. Als Beispiel seien hier PL/SQL von Oracle oder auch Transact SQL von Microsoft (siehe Abschnitt 2.6.2) genannt, bei denen SQL-Datenanweisungen bereits Teil der Grammatik sind. Um bei nicht speziellen Datenbanksprachen SQL-Anweisungen direkt innerhalb des Programmcodes ausführen zu können, werden entsprechende Toolkits benötigt. Diese werden entweder von Datenbankherstellern, Drittanbietern oder Open Source-Providern zur Verfügung gestellt. Zu diesen Toolkits gehören unter anderem Java Database Connectivity (JDBC) für die Programmiersprache Java, RogueWave SourcePro DB für die Programmiersprache C++, sowie die von Microsoft bereitgestellten Toolkits Open Database Connectivity (ODBC) und Object Linking and Embedding, Database (OLE DB). Jeder Datenbankhersteller stellt aber auch zumindest eine Möglichkeit bereit, um SQL-Befehle interaktiv abzusetzen. Dabei handelt es sich um einfache Tools, die entweder über eine grafische Oberfläche verfügen oder auf Kommandozeilenebene ausgeführt werden können. [vgl. Bea07, S. 9 f]

2.4.3. ERM

Das **Entity-Relationship-Modell (ERM)** wurde 1976 von Peter Chen vorgeschlagen und wird vor allem zur Modellierung von Datenbanken und zur Abbildung von Datenmodellen verwendet. Der Vorteil bei der Verwendung eines ERM liegt in der sehr abstrakten und damit datenbankunabhängigen Darstellung von Miniwelten. Ein ERM besteht aus Entitäten (engl. **Entity**), die eindeutig voneinander unterscheidbare Objekte darstellen. Eine Person oder eine Firma wären Beispiele für ein Entity. Jede Entität kann über Attribute verfügen, denen bestimmte Werte zugeordnet werden. Typische Attribute einer Person sind beispielsweise Vorname oder Nachname, sowie Alter. Die einzelnen Entities können in bestimmten Beziehungen (engl. **Relationship**) zueinander stehen. Innerhalb eines ERM beschreiben zusätzlich angegebene 1:1-, 1:n- und n:m-Kardinalitäten die einzelnen Beziehungen näher. Als Beispiel könnten die beiden Entities Firma und Person über eine „arbeitet in“-Beziehung zusammengebracht werden. Die Kardinalität 1:n würde dabei aussagen, dass in einer Firma beliebig viele Personen arbeiten, aber jede Person in genau einer Firma arbeitet. [vgl. Che76, S. 10-12, 20]

Mittlerweile gibt es eine Vielzahl an unterschiedlichen Notationsformen für ERMs.

Die **Chen-Notation** nutzt für die Kennzeichnung von Entitäten Rechtecke und für deren Beziehungen zueinander Rauten. Einzelne Entitäten sind mittels Linien immer über eine entsprechende Beziehung miteinander verbunden. Die Kardinalitäten für die Beziehungen werden dabei direkt an den Verbindungslinien notiert. [vgl. Che76, S. 19 f]

Bei der sogenannten **Krähenfußnotation** werden insbesondere die Kardinalitäten auf andere Art und Weise dargestellt. Diese Notation verwendet für die Anzeige von „beliebig viele“ einen Krähenfuß am entsprechenden Ende der Verbindungslinie zu einer Entität.

Wird dieser Krähenfuß weggelassen, so meint man damit eine Kardinalität von „eins“. Diese Form der Darstellung wurde von Dr. Gordon Everest entwickelt, der den Krähenfuß ursprünglich als „invertierten Pfeil“ bezeichnete und ihn mittlerweile „Gabel“ nennt. [vgl. HM10, S. 309]

Weiterhin ist auch eine Darstellung mit Hilfe der **Unified Modeling Language (UML)** möglich, obwohl die UML grundlegend für die Modellierung von Software kreiert wurde. Wie zuvor erläutert, steht eine Entity im ERM für ein Objekt, von dem es mehrere Instanzen geben kann, die sich in ihren definierten Attributen voneinander unterscheiden. In der UML entspricht eine Klasse am ehesten einer Entität. Klassen werden in der UML als Rechtecke dargestellt, die aus drei Bestandteilen bestehen. Der erste Bestandteil setzt sich aus dem Stereotyp, was in dem Fall einfach Entity ist, und dem Klassennamen zusammen. Die Attribute einer Entity werden im zweiten Bestandteil einer Klasse, den Klasseneigenschaften, abgebildet. Beim dritten Bestandteil der Klasse würde das Klassenverhalten mit der Angabe von Methoden beschrieben werden. Da Entities kein Verhalten besitzen, fällt dieser Klassenbestandteil somit weg. Auch bei der UML-Darstellung eines ERM werden einfache Verbindungslinien für die Anzeige von Beziehungen verwendet. Deren Name wird dabei schlicht zusammen mit den Kardinalitäten direkt an der Verbindungslinie vermerkt. [vgl. Gor03, S. 9-11]

2.5. XML

2.5.1. Definition

„Der Ursprung von XML liegt in Überlegungen, wie Struktur, Inhalt und Layout eines Dokumentes getrennt werden können.“ [Bec09, S. 1]

Beim **Inhalt** eines Dokumentes handelt es sich um die zu vermittelnde Information. Mit der **Struktur** ist die Aufteilung und Abfolge dieser Information gemeint. Unter dem **Layout** versteht man die Visualisierung des Inhaltes und der Struktur des Dokumentes. Das Layout wird durch eine entsprechende Formatierung ausgedrückt. [vgl. Bec09, S. 4 f]

Um Struktur, Inhalt und Layout voneinander zu trennen, kann man die Inhalte mit einer Strukturinformation statt mit einer Formatierungsinformation speichern. Zum Beispiel wird ein Absatz, der eine Überschrift darstellt, auch als Überschrift gespeichert und ausgezeichnet, während die tatsächliche Formatierungsinformation für diese Überschrift getrennt davon gespeichert wird. Diese Auszeichnungen bezeichnet man als **Markup**. Ursprünglich stammt der Begriff aus dem Verlagswesen und beschreibt typografische Festlegungen in Form handschriftlicher Markierungen in Manuskripten. Bei elektronischen Dokumenten versteht man unter Markup Zeichenfolgen an bestimmten Punkten innerhalb eines Dokumentes, um die Form der Darstellung, des Drucks oder der Struktur der Dokumente zu beschreiben. [vgl. Bec09, S. 6]

Im Jahre 1986 wurde die Standard Generalized Markup Language (SGML) als ISO-

Standard veröffentlicht. Sie ist eine Metasprache zur Beschreibung von Auszeichnungssprachen. Mit ihr können also eigene Markup-Sprachen definiert werden. Die Struktur eigener Sprachen wird in einer sogenannten Document Type Definition (DTD) oder in einem ausdrucksstärkerem XML Schema beschrieben. Eine solche Markup-Sprache legt in ihrer Beschreibung die Auszeichnungselemente, **Tags** genannt, fest und bestimmt auch deren Namen. [vgl. Bec09, S. 8, 85]

Mit der Verbreitung des Internets wurden zwar die Anwendungsmöglichkeiten von SGML stark nachgefragt, nicht aber deren Komplexität, weshalb sich SGML nur in wenigen Branchen durchgesetzt hatte. Man benötigte ein Format, das sich zum universellen Datenaustausch zwischen Anwendungen eignete. Aus diesem Grund wurde vom World Wide Web Consortium (W3C) eine Arbeitsgruppe zur Entwicklung der Sprache eXtensible Markup Language (XML) beauftragt. Der Entwurf von XML verfolgte 10 Ziele: [vgl. Bec09, S. 9]

1. Die Sprache soll sich im Internet einfach nutzen lassen.
2. Sie soll ein breites Feld an Anwendungen unterstützen.
3. Es soll eine Kompatibilität zu SGML vorhanden sein.
4. Programme zu schreiben, die XML-Dokumente verarbeiten, soll simpel sein.
5. Optionale Merkmale sollen bei XML möglichst nicht vorhanden sein.
6. Erstellte Dokumente sollen menschenlesbar und angemessen verständlich sein.
7. Der XML-Entwurf soll schnell abgefasst werden.
8. Ein formaler und präziser Entwurf von XML.
9. Ein XML-Dokument soll einfach erstellbar sein.
10. Die Knappheit von XML spielt eine untergeordnete Rolle.

[vgl. Bec09, S. 9]

Die Version 1.0 der XML-Spezifikation wurde vom W3C im Jahre 1998 verabschiedet. Bereits kurz darauf wurden viele Open Source-Werkzeuge zur Verarbeitung entwickelt. [vgl. Bec09, S. 9]

Ein XML-Dokument besteht vor allem aus Nutzdaten und den Tags, die Aussagen, welche Bedeutung die Nutzdaten besitzen. Außerdem dienen die Tags zur hierarchischen Gliederung des XML-Dokumentes. Als Beispiel soll das kurze Listing 1 auf Seite 24 dienen. [vgl. Moo08, S. 1]

```
1 <Name>
2     <Vorname>Hugo</Vorname>
3     <Zuname>Mueller</Zuname>
4 </Name>
```

Listing 1: Beispielhaftes XML-Dokument [vgl. Moo08, S. 2]

Das beispielhafte XML-Dokument besteht aus den beiden Nutzdatenworten *Hugo* und *Mueller* und aus den Tags, die sich innerhalb von spitzen Klammern befinden. Das Startelement befindet sich in Zeile 1 und heißt *<Name>*. Es beschreibt den Inhalt des Dokumentes in den Zeilen 2 und 3. Den Abschluss bildet das Endeelement in Zeile 4, welches mit einem Schrägstrich vor dem Elementnamen eingeleitet wird. [vgl. Moo08, S. 2]

Aufgrund der beliebigen hierarchischen Schachtelung der Elemente, nimmt ein XML-Dokument die Form eines Datenbaumes an. Abstrakt gesehen kann dieser als Graph mit Knoten und Kanten aufgefasst werden. Ein solcher Datenbaum besteht aus exakt einem Wurzelknoten, beliebig vielen Blättern und beliebig vielen dazwischen liegenden Zwischenknoten. Alle diese Knoten werden durch einzelne XML-Elemente abgebildet.
[vgl. Moo08, S. 2]

2.5.2. XPath

Die XML Path Language (XPath) ist eine Art Hilfssprache, die von anderen XML-Sprachen, wie zum Beispiel der eXtensible Stylesheet Language Transformation (XSLT) und XML Query Language (XQuery) (siehe Abschnitt 2.5.3), für den Zugriff auf Strukturbestandteile eines XML-Dokumentes verwendet wird. Die XPath-Version 1.0 wurde 1999 und die Nachfolgerversion XPath 2.0 im Jahr 2007 vom W3C verabschiedet. [vgl. Bec09, S. 147]

Der Name XPath kommt daher, dass für die Navigation innerhalb der Hierarchie eines XML-Dokumentes ein Pfad (engl. path) verwendet wird. Dabei nutzt XPath eine kompakte und nicht an XML angelehnte Syntax, wodurch der Gebrauch von XPath innerhalb von Uniform Resource Identifiers (URIs) und XML-Attributen erleichtert wird. [vgl. Ber11]

Diese von XPath zur Adressierung von Knoten innerhalb eines XML-Dokumentes verwendeten **Lokalisierungspfade** beginnen immer bei einem bestimmten Knoten, welcher als **Kontextknoten** bezeichnet wird. Es wird zwischen absoluten und relativen Lokalisierungspfaden unterschieden. Ein absoluter Pfad startet beim Dokumentknoten, während ein relativer Pfad einen beliebigen anderen Knoten als Ursprung hat. Jeder Lokalisierungspfad besteht aus einzelnen **Lokalisierungsschritten**, die jeweils durch einen Slash voneinander getrennt sind. Deren Auswertung erfolgt sukzessive von links nach rechts, wobei jeder einzelne Schritt eine (Teil-)Sequenz von Knoten auswählt, die wiederrum als Ausgangspunkt für den nächsten Lokalisierungsschritt dient. Die einzelnen Lokalisierungsschritte haben alle die folgende Syntax: `achse::knotentest[prädikat]`. Die **Achse** gibt die Suchrichtung

der auszuwählenden Knoten an. Mit dem **Knotentest** können die selektierten Knoten ein erstes Mal und mit dem optionalen **Prädikat** ein weiteres Mal gefiltert werden. [vgl. Bec09, S. 160-162]

Als Beispiel zeigt das Listing 2 den Zugriff auf den Textinhalt des *Zuname*-Knotens aus Listing 1 auf Seite 24.

```
1 /child::Name/child::Zuname/text()
2 /Name/Zuname/text()
```

Listing 2: Beispielhafter XPath-Pfadausdruck

Beide in Listing 2 zu sehenden Pfadausdrücke sind identisch und liefern das gleiche Ergebnis zurück. Der Pfad in Zeile 1 verwendet die ausführliche Syntax, während in Zeile 2 die abgekürzte Syntax benutzt wird [vgl. Bec09, S. 167]. Mit beiden gezeigten Pfadausdrücken würde man sich vom Wurzelknoten aus, entlang der entsprechenden Achse über den Knoten *Name* bis hin zum Knoten *Zuname* bewegen. Der abschließende Knotentest *text()* prüft dann, ob es sich um einen Textknoten handelt. Ist dies der Fall, wird der Knoteninhaltstext als Ergebnis zurückgeliefert.

2.5.3. XQuery

Durch die weite Verbreitung von XML wuchs auch der Bedarf nach einer Sprache, die komplexe Anfragen an XML-Dokumente stellen kann, ganz in Anlehnung an die Datenbankabfragesprache SQL. Um die Entwicklung einer solchen Sprache anzustoßen, beauftragte das W3C eine Arbeitsgruppe. Dabei stellte das W3C auszugsweise die folgenden Anforderungen: [vgl. Bec09, S. 147]

- XML-Dokumente sollen abgefragt werden können, um anschließend wieder menschenlesbare Dokumente aus XML-Daten erstellen zu können.
- Es sollen daten- und dokumentenorientierte Dokumente verarbeitet werden.
- Konfigurationsdateien sollen analysiert werden können.
- Datenströme sollen gefiltert bzw. bereinigt werden.
- Es soll innerhalb des Document Object Model (DOM) einsetzbar sein.

[vgl. Bec09, S. 147]

Die daraus entstandene Sprache heißt XQuery und da während der Entwicklung daran auch an den neuen Versionen von XPath und XSLT gearbeitet wurde, wurden alle drei Sprachen zeitgleich im Januar 2007 vom W3C verabschiedet. XQuery 1.0 und die Weiterentwicklung XPath 2.0 besitzen als Grundlage das gleiche Datenmodell und stellen die

gleiche Menge an Funktionen und Operatoren zur Verfügung. Somit ist XPath 2.0 keine eigenständige Sprache mehr, sondern eine echte Teilmenge von XQuery. Dabei umfasst der XPath-Anteil etwa 80% der Sprache. [vgl. Bec09, S. 147 f]

Das angesprochene Basisdatenmodell von XQuery 1.0 und XPath 2.0 baut auf einer **Sequenz**, also einer Sammlung einzelner Werte, auf. Innerhalb dieser herrscht eine serielle Zugangsordnung. Demnach kann eine Sequenz auch als eine **Queue** gesehen werden und funktioniert somit nach dem **First In – First Out-Prinzip**. Das heißt, der zuerst eingestellte Eintrag wird auch als erstes ausgegeben. [vgl. Moo08, S. 105]

XQuery nutzt eine sehr kompakte Nicht-XML-Syntax und ist für die Auswahl von Daten aus einer Vielzahl von Dokumenten optimiert. Während XPath nur einzelne Knoten aus einem XML-Dokument auswählen kann, besteht mit XQuery die Möglichkeit, ein neues XML-Dokument zu erzeugen. Die finale Struktur des daraus resultierenden Dokumentes wird bereits in der XQuery-Abfrage angegeben. Ein einfaches Beispiel für XQuery ist in Listing 3 dargestellt, welches die zuvor in Listing 1 auf Seite 24 gezeigten XML-Daten verwendet. [vgl. Bec09, S. 148 f]

```
1 xquery version "1.0" encoding "utf-8";
2 declare boundary-space preserve;
3 for $tag in Name
4 let $first := $tag/Vorname, $last := $tag/Zuname
5 return
6 <Person>
7   <VollerName>{data($first)} {data($last)}</VollerName>
8 </Person>
```

Listing 3: Ein Beispiel für die Verwendung von XQuery

In Zeile 1 wird zuerst eine XQuery-Version deklariert, die eindeutig beschreibt, welche XQuery-Syntax verwendet werden kann, und anschließend folgt die Angabe der genutzten Zeichensatzcodierung. Danach ist in Zeile 2 durch die *boundary-space*-Strategie *preserve* festgelegt, dass elementbegrenzende Whitespace-Zeichen erhalten bleiben sollen. Die alternative Strategie wäre *strip* und würde bedeuten, dass bei der Ausgabe diese Zeichen entfernt werden würden. Ab Zeile 3 wird ein besonderes Feature von XQuery verwendet, nämlich der sogenannte **FLWOR-Ausdruck**. Der Name FLWOR wird wie das englische Wort „flower“ ausgesprochen und steht für die Schlüsselwörter *for*, *let*, *where*, *order by* und *return*. Die *for*-Anweisung in Zeile 3 iteriert über alle Namenselemente aus Listing 1 und bindet diese an die Variable *$tag*. Mit der anschließenden *let*-Anweisung werden die Kindelemente *Vorname* und *Zuname* des aktuellen Namenselementes an die entsprechenden *$first*- und *$last*-Variablen gebunden. Nun hätte man die Möglichkeit, mit den optionalen und in Listing 3 nicht verwendeten *where*- und *order by*-Anweisungen die einzelnen Elemente zu filtern und zu sortieren. Abschließend konstruiert die *return*-Anweisung für jedes gebundene Namenselement ein neues Element für die Ausgabe, welches die Daten

der Kindelemente nutzt. Wie zuvor bereits erwähnt, wird dabei gleich die Struktur des Ergebnisdokumentes festgelegt. [vgl. Boa11]

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Person>
3   <VollerName>Hugo Mueller</VollerName>
4 </Person>
```

Listing 4: Ergebnis des XQuery-Ausdrucks

Das resultierende XML-Dokument ist in Listing 4 zu sehen und enthält die zuvor im XQuery-Ausdruck definierten Tags *Person* und *VollerName*.

2.6. Microsoft SQL-Server

2.6.1. Definition

Im Kern ist der SQL-Server eine relationale Datenbank, welche sich an dem Standard der aktuellen SQL-Version orientiert. Entstanden ist der SQL-Server gegen Ende der 1980er-Jahre aufgrund einer Zusammenarbeit der beiden Firmen Microsoft und Sybase. Die erste Version erschien 1989 und war noch kein eigenes Produkt von Microsoft, sondern war identisch zur Sybase SQL-Server Version 4.0. Eine eigenständige Entwicklung seitens Microsoft erschien erst 1995, allerdings entsprach diese immernoch von der Codebasis her den Produkten von Sybase. Dies änderte sich mit der Version 7.0 des SQL-Servers im Jahre 1999. Hier entwickelte Microsoft eine neue, eigene Datenbank-Engine inklusive Codebasis. [vgl. CF13, S. 27]

Die aktuellste Version ist der SQL-Server 2014. Neben den bereits genannten Versionen, gehören unter anderem der SQL-Server 2012, 2008, 2008 R2 und der SQL Server 2005 zu den Vorgängerversionen. Zusätzlich gibt es mit der Einführung von SQL-Server 2012 Änderungen bezüglich der Versionen und des Lizenzmodells. Im Unternehmensumfeld stehen seit dem drei wesentliche Haupteditionen zur Verfügung: Die Standard Edition, die Enterprise Edition und die Business Intelligence Edition. [vgl. CF13, S. 27 f, S. 51]

Bei der **Standard Edition** handelt es sich um den Einstieg in den Bereich der kostenpflichtigen SQL-Server-Editionen für Unternehmen und sie kann als BI- und Datenverwaltungsplattform für kleine bis mittlere Unternehmen eingesetzt werden. Diese Edition verfügt zwar nicht über alle Features der BI- oder Enterprise Edition, aber dennoch muss man nicht auf die Funktionalitäten verzichten, die von kleinen bis mittelgroßen Datenbanken genutzt werden. Der Schwerpunkt der **BI-Edition** liegt auf Self-Service-BI[4], sowie auch auf unternehmensweite Business Intelligence. Der Einsatz dieser Edition wird als BI-

[4] Unter Self-Service-BI versteht man das Unabhängigwerden der BI-Nutzer von der IT-Organisation durch leicht verwendbare technische Hilfsmittel. Die BI-Anwender sollen in der Lage sein, viele Aufgaben in der Abteilung selbstständig zu lösen. [vgl. Mar13]

Plattform direkt im Unternehmen gesehen. Die **Enterprise Edition** stellt eine umfassende Plattform für mittelgroße und große Datenbanken, sowie Anwendungen mit geschäftskritischen Applikationen bereit. Sie umfasst verschiedenste BI-Dienste und während die anderen Editionen Begrenzungen für die genutzten Prozessorkerne und den verwendbaren Arbeitsspeicher einführen, nutzt die Enterprise Edition hingegen das Maximale des Betriebssystems. [vgl. CF13, S. 51 f]

Abhängig davon, welche SQL-Server-Version und -Edition der Nutzer verwendet, stehen diesem unterschiedliche Datenverwaltungs- und Analysetechnologien zur Verfügung. Im Folgenden werden einige der Technologien aufgezählt, die Microsofts SQL-Server 2012 umfasst. [vgl. MicTE]

Das **Datenbankmodul** ist der Hauptdienst für das Verarbeiten und Sichern von Daten und stellt kontrollierten Zugriff und schnelle Transaktionsverarbeitung bereit. Die **Data Quality Services** ermöglichen hingegen das Erstellen und die anschließende Verwendung einer Wissensdatenbank[5]. Damit können computergestützte Datenkorrekturen und Deduplizierungen der Daten durchgeführt werden. Eine weitere Technologie des SQL-Servers ist die **Replikation**, mit der Daten aus einer Datenbank in eine andere kopiert oder verteilt werden können. Zur Sicherstellung der Konsitenz der Daten, können die Daten damit auch anschließend synchronisiert werden. Die **Analysis Services** stellen eine Plattform für analytische Daten dar und ermöglichen zudem DM (siehe Abschnitt 2.1.2). Mit den **Integration Services** werden Datenintegrationslösungen bereitgestellt. Dazu gehören auch DW-Pakete zum Extrahieren, Transformieren und Laden von Daten. Abschließend seien noch die **Reporting Services** erwähnt, die im Abschnitt 2.6.3 ausführlich behandelt werden. [vgl. MicTE]

Der SQL-Server von Microsoft ist dank dieser verschiedenen Technologien eine integrierte Lösung zur Verwaltung und Analyse von Daten. Analytischen Anwendungen und Unternehmensdaten bietet der SQL-Server Sicherheit, Skalierbarkeit und Verfügbarkeit. Darüber hinaus vereinfacht er die Erstellung, Bereitstellung und Verwaltung. [vgl. MicSQ]

2.6.2. Transact SQL

Standard-SQL kennt viele Befehle und diverse Programmiersprachenkonstrukte wie zum Beispiel Variablen, Entscheidungsstrukturen oder Schleifen nicht. Deshalb implementieren die verschiedenen Hersteller von Datenbanksystemen für sie wichtige Ergänzungsbefehle auf eigene Art und Weise. Diese Weiterentwicklungen des SQL-Standardbefehlssatzes bezeichnet man als SQL-Dialekte. Der SQL-Dialekt von Microsoft nennt sich T-SQL und steht für *Transact SQL*. [vgl. CF13, S. 148 f]

Im Folgenden werden einige Erweiterungen dieses Dialektes gezeigt.

Variablen sind Platzhalter für Werte die während der Verarbeitung eines Skriptes oder

[5] Eine Wissensdatenbank ist ein organisierter Speicher (meistens eine Datenbank), in dem Konzepte, Artikel, Prozesse, Handbücher oder dergleichen abgelegt sind. [vgl. DicKB]

einer Prozedur dynamisch zugewiesen werden. Oft verwendet man sie als Zähler für die einzelnen Schlüsselwörter der Ablaufsteuerungssprache von T-SQL. Ein kurzes Beispiel zeigt das Listing 5. [vgl. MicVA]

```
1 DECLARE @anzahl_zeilen INT;
2 SELECT @anzahl_zeilen = COUNT(*) FROM TESTTABELLE;
```

Listing 5: Verwendung von Variablen

Mit der *DECLARE*-Anweisung in Zeile 1 wird die ganzzahlige Variable *@anzahl_zeilen* deklariert. Anschließend wird die Anzahl der Tabellenzeilen einer fiktiven Testtabelle ermittelt und dieser Wert der Variablen *@anzahl_zeilen* zugewiesen. Als Alternative kann man auch mit der *SET*-Anweisung einer Variablen einen Wert zuweisen.

Die **Entscheidungsstrukturen** können mit dem *IF*-Schlüsselwort und dem optionalen *ELSE*-Schlüsselwort definiert werden. Mit dem *IF*-Schlüsselwort werden dabei die Bedingungen für die Ausführung einer T-SQL-Anweisung festgelegt. Das Listing 6 stellt ein kurzes Beispiel dar. [vgl. MicIF]

```
1 IF @kosten <= @vergleichs_preis
2 BEGIN
3     PRINT 'Diese Produkte sind guenstiger als
4     $'+RTRIM(CAST(@vergleichs_preis AS varchar(20)))+'.'
5 END
6 ELSE
7     PRINT 'Diese Produkte sind teurer als
8     $'+ RTRIM(CAST(@vergleichs_preis AS varchar(20)))+'.'
```

Listing 6: Entscheidungsstrukturen in T-SQL

Zuerst wird überprüft, ob die beispielhaften Produktkosten kleiner oder gleich einem festgelegten Preis sind. Die weitere Ausführung ist abhängig vom Ergebnis dieses booleschen Ausdrucks. Sollte *TRUE* zurückgegeben werden und die Kosten damit tatsächlich kleiner oder gleich dem Vergleichspreis sein, dann wird die Anweisung des *IF*-Zweiges ausgeführt. Entsprechend wird die Anweisung im *ELSE*-Zweig nur dann ausgeführt, wenn der boolesche Ausdruck stattdessen *FALSE* zurückgibt. Im gezeigten Beispiel handelt es sich in beiden Fällen nur um einfache Ausgaben des Ergebnisses.

Will man eine SQL-Anweisung oder einen ganzen Anweisungsblock wiederholt ausführen, kann man die ***WHILE*-Anweisung** verwenden. Hierbei werden die Anweisungen solange ausgeführt, wie die angegebene Bedingung *TRUE* zurückgibt. Im Listing 7 auf Seite 30 ist dies an einem kurzen Beispiel dargestellt. [vgl. MicWH]

In der Bedingung der *WHILE*-Schleife in Zeile 1 wird mit der *SELECT*-Anweisung der durchschnittliche Listenpreis eines Produktes ermittelt. Solange dieser geringer als 300 ist, wird der Rumpf der *WHILE*-Schleife ausgeführt. Innerhalb der Schleife wird mit der *UP-*

DATE-Anweisung der Listenpreis verdoppelt. Hierdurch wird der Listenpreis kontinuierlich erhöht, solange die Bedingung der Schleife *TRUE* zurückgibt.

```
1 WHILE (SELECT AVG(ListPrice) FROM Production.Product) < 300
2 BEGIN
3    UPDATE Production.Product SET ListPrice = ListPrice * 2
4 END
```

Listing 7: Schleifen in T-SQL

Dies sind nur einige der Erweiterungen, die im SQL-Dialekt von Microsoft hinzugefügt wurden. Die übrigen und hier nicht erwähnten Erweiterungen können zum Beispiel in der offiziellen Transact-SQL-Referenz unter [MicRE] nachgeschlagen werden. Wie man aber bereits an den gezeigten Beispielen sieht, wird T-SQL aufgrund der Erweiterungen praktisch zu einer eigenen Programmiersprache.

2.6.3. Reporting Services

Die Reporting Services sind ein wichtiger Bestandteil im Bereich BI. Erst durch die Visualisierung der Unternehmensdaten können fundierte Entscheidungen getroffen werden. Sowohl die Unternehmensleitung, Controller, Power-User und die Administratoren benötigen Daten in einer für sie passend aufbereiteten Form. [vgl. CF13, S. 591]

Konkret handelt es sich bei den Reporting Services um eine serverbasierte Berichtsplattform. Sie bietet eine Vielzahl an unterschiedlichen Werkzeugen und Diensten, mit denen Berichte für das Unternehmen erstellt, bereitgestellt und verwaltet werden können. Es können dabei interaktive, tabellarische, grafische oder auch Freiformberichte erstellt werden, denen relationale, multidimensionale oder XML-basierte Datenquellen zu Grunde liegen. Die Berichte können eine umfangreiche Datenvisualisierung beispielsweise mit Diagrammen und Karten umfassen. Bei der Bereitstellung von Reports stehen die verschiedensten Anzeigeformate zur Verfügung und auch ein Export der Berichte in andere Anwendungen, wie zum Beispiel Microsoft Excel, ist möglich. Der Zugriff auf die erstellten Berichte kann über eine webbasierte Verbindung, über eine Windows-Anwendung oder auch über eine SharePoint-Webseite erfolgen. [vgl. MicRS]

Jeder kreierte Bericht wird mittels der Report Definition Language (RDL) beschrieben. Diese Sprache enthält im XML-Format die Information über das Layout des Berichtes und wie dieser sich die benötigten Daten beschafft. Die Datei mit der Beschreibung des Berichtes hat üblicherweise die Dateiendung *.rdl*. [vgl. MicMS]

Bei den meisten Werkzeugen, die die Reporting Services-Plattform mitliefert oder unterstützt, handelt es sich um diverse Programme. Beispielsweise ist der **Report Designer** eine Anwendung, mit der Berichte erstellt werden können. Der Report Designer wird zusammen mit dem SQL-Server installiert und, falls vorhanden, in eine bereits installierte

Visual Studio-Version integriert. Auch mit dem **Report Builder** lassen sich Berichte entwerfen. Im Gegensatz zum Report Designer ist das Erscheinungsbild dieser Software eher vergleichbar mit den Microsoft Office Produkten und kann als eigenständige Anwendung ohne den Report Server heruntergeladen werden. Das Werkzeug namens **Power View**[6] hilft dem Endkunden dabei seine Daten visuell zu erkunden, dazu aufkommende Fragen leichter zu beantworten und die Daten innerhalb einer SharePoint-Umgebung zu präsentieren und mit anderen gemeinsam an diesen zu arbeiten. Administrative Aufgaben können mit dem **SQL-Server Management Studio** bewältigt werden, während die Konfiguration des Reporting Services Windows-Dienstes mit dem **SQL-Server Configuration Manager** erfolgt. [vgl. Tur12, S. 48 - 50]

Bereits an dieser kurzen Auflistung von ausgewählten bereitgestellten Werkzeugen der Reporting Services lässt sich gut erkennen, wie ausgereift diese Plattform ist und das damit sehr viele Aspekte der BI abgedeckt werden. Wohl nicht zuletzt deshalb gehören die SQL-Server Reporting Serices zum Industriestandard, an dem sich andere Reporting Tools zu messen haben [vgl. Tur12, S. 4].

3. Umsetzung

3.1. Konzeption

3.1.1. Anforderungsanalyse

Die Grundlage der Anforderungsanalyse bildet die Aufgabenstellung (siehe Abschnitt 1.1), sowie die erhaltenen Anregungen im Praktikum, in dessen Rahmen die Bearbeitung der Aufgabenstellung erfolgte. Daraus ergeben sich die folgenden Anforderungen:

- **Der Metareport muss die Aufrufhäufigkeit aller Berichte schnell erfassbar anzeigen und besonders oft abgerufene Berichte hervorheben.**
  Die genaue Anzahl an Aufrufen ist im Metabericht darzustellen. Reports, die selten verwendet werden, sind vernachlässigbar und müssen nicht sofort zu sehen sein. Somit genügt es, nur die Berichte auf einen Blick anzuzeigen, die kontinuierlich und besonders häufig von den Anwendern genutzt werden.

- **Die aktivsten Berichtsnutzer müssen genau identifiziert werden können.**
  Als aktive Berichtsnutzer gelten die Anwender, die außerordentlich viele Berichte abrufen. Zur Identifikation werden deren Benutzernamen und alle von ihnen betrachteten Berichte und deren Aufrufanzahl innerhalb des Metareports angezeigt. Auch hier genügt es, die Daten der besonders aktiven Anwender innerhalb des Metaberichtes sofort darzustellen.

[6] Bisher handelte es sich bei Power View um einen Bestandteil der Microsoft SQL-Server 2012 Reporting Services-Add-Ins für die Enterprise Edition von Microsoft SharePoint Server 2010 und 2013. Aktuell ist Power View jedoch eine Funktion von Microsoft Excel 2013. [vgl. MicPV]

- **Um die Berichtsgenerierung optimieren zu können, müssen die zur Berichtserstellung genutzten Ressourcen ersichtlich sein.**
  Bis ein Bericht erstellt ist und vom Benutzer betrachtet werden kann vergeht eine gewisse Verarbeitungszeit. Während dieser Zeit verbraucht das System, das die Generierung übernimmt, Ressourcen. Zu wissen wie viel Zeit aufgewendet und wie viel Speicher währenddessen genutzt wird, kann dabei helfen die Berichte zu erkennen, die besonders ressourcenhungrig sind.

- **Alle übergebenen Parameter an einen aufgerufenen Bericht müssen im Metabericht angezeigt werden.**
  Abhängig vom Aufbau eines Berichts kann dieser parametrisierbar sein und verschiedene Werte für jeden einzelnen seiner Parameter entgegennehmen. Um erkennen zu können, ob sich für einen Report ein neues Abonnement lohnt, muss bekannt sein, ob dieser oft die gleichen Werte für alle seine Parameter übergeben bekommt. Dazu muss für jeden abgerufenen Bericht die genaue Parameter-Wert-Kombination ersichtlich sein.

- **Diagramme sollen stark genutzte Zeitpunkte zur Berichtsgenerierung aufzeigen.**
  Um im Metabericht die Tageszeiten anzuzeigen, an denen sehr oft Berichte abgerufen werden, sollen Diagramme verwendet werden. Mit diesen können die Daten der Tageszeiten schnell erfasst und sofort ausgewertet werden, wodurch die Stoßzeiten der Erstellung von Reports immer erkennbar sind. Dieses Wissen kann zum Beispiel dazu genutzt werden, um passende Zeitpunkte für Abonnementzeitpläne zu ermitteln.

- **Die Daten über bereits bestehende Abonnements zu einem Bericht müssen abrufbar sein.**
  Dabei ist es wichtig zu wissen, wie viele aktive Abonnements bereits für einen Bericht eingerichtet sind, welche Parameter an den Bericht übergeben werden und welchen Zeitplan das Abonnement nutzt. Mit dieser Information kann erkannt werden, ob weitere Abonnements für diesen Report benötigt werden und welche neuen und bisher nicht berücksichtigten Daten diese Abonnements abzudecken haben.

- **Alle im Metabericht dargestellten Metadaten müssen für unterschiedliche Zeiträume abrufbar sein.**
  Nicht nur die Daten der letzten 24 Stunden, sondern auch die Daten der letzten Woche oder des letzten Monats müssen abrufbar sein, denn aus diesen älteren Daten können Verläufe und Annahmen über zukünftige Berichtsnutzungen getroffen werden. Beispielsweise kann aus den zurückliegenden Aufrufen eines Berichtes ein absehbarer Anstieg oder Rückgang der Berichtsverwendung abgelesen werden.

Diese Anforderungen sind soweit möglich in den Metareport aufzunehmen und sie definieren seine endgültig für den Berichtsverwalter bereitgestellten Funktionen.

3.1.2. Analyse des Datenmodells

3.1.2.1. Zugriff auf das Datenmodell

Um zu ermitteln, welche Daten überhaupt zur Verfügung stehen und damit abgefragt werden können, muss zuerst das grundlegende Datenmodell analysiert werden. Für den dafür nötigen Zugriff auf die Systemtabellen wird die Software *SQL Server 2012 Management Studio* von Microsoft verwendet. Damit die interessanten Tabellen betrachtet und untersucht werden können, muss anfangs mit der Software eine Verbindung zu einem SQL-Server hergestellt werden. Zu diesem Zweck wird vorher lokal eine entsprechende SQL-Serverinstanz installiert, zu der sich anschließend mit dem Management Studio verbunden werden kann. Dazu öffnet sich bereits beim Start der Software ein entsprechender Verbindungsdialog, in dem die passenden Daten eingetragen werden können. Die Abbildung 9 zeigt diesen Dialog mit den beispielhaften Verbindungsdaten.

Abbildung 9: Verbindung zu einem SQL-Server herstellen

Das Management Studio bietet dem Nutzer verschiedene Ansichten, von denen zu Beginn vor allem der Objekt-Explorer hervorzuheben ist. In diesem werden die auf dem verbundenen SQL-Server verfügbaren Objekte angezeigt (siehe Abbildung 10 auf Seite 34). Dazu gehören auch die Datenbanken, die auf diesem Server liegen. Zu jeder einzelnen Datenbank kann man sich innerhalb des Explorers unter anderem alle Tabellen, sowie alle Sichten anzeigen lassen. Die Darstellung erfolgt innerhalb des Explorers in Form einer Baumansicht

und ist sehr detailliert aufschlüsselbar, so dass man beispielsweise bei den Tabellen jede einzelne Spalte mit dem festgelegten Datentyp oder auch die definierten Primär- und Fremdschlüssel betrachten kann.

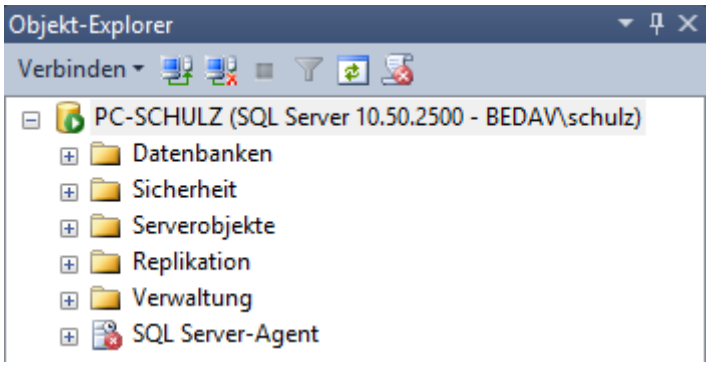


Abbildung 10: Der Objekt-Explorer

Zwar genügt diese Ansicht bereits, um sich die nötigen Informationen für einen Überblick über das Datenmodell zu beschaffen, allerdings stellt man mit dem Objekt-Explorer bereits schnell fest, dass das hier zu betrachtende Modell aus über 30 verschiedenen Tabellen besteht, die sich zum Teil außerdem, gemäß dem Schema einer relationalen Datenbank (siehe Abschnitt 2.4.1), über entsprechende Fremdschlüssel gegenseitig referenzieren. Praktischerweise bietet der Objekt-Explorer eine Funktion an, mit der man sich aus der Struktur der Datenbank ein Diagramm erzeugen lassen kann. Die Abbildung 11 zeigt Auszugsweise ein aus den Systemtabellen des SQL-Servers erzeugtes Datenbankdiagramm.

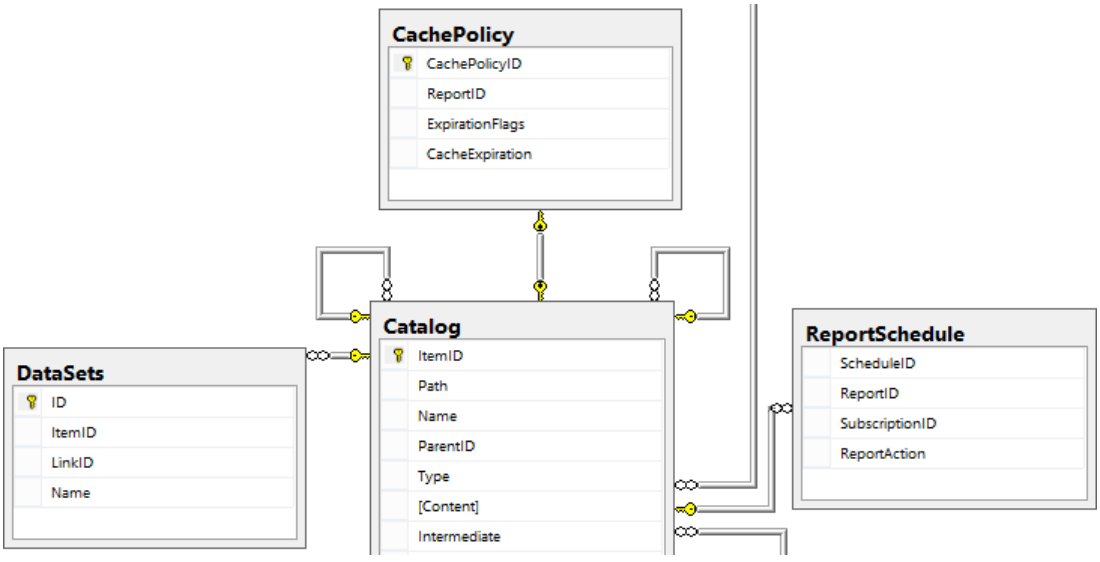


Abbildung 11: Auszug aus dem automatisch erzeugten Datenbankdiagramm

Aufgrund der Darstellung in Form eines ERM (siehe Abschnitt 2.4.3 auf Seite 21) lassen sich mit diesem die einzelnen Beziehungen der Tabellen untereinander bedeutend schnel-

ler erfassen, als wenn man dies allein über den Explorer machen würde. Die folgenden Abschnitte betrachten das Datenmodell näher.

3.1.2.2. Ist-ERM

Um die Tabellen zu ermitteln, die besonders wichtige und grundlegende Daten enthalten könnten, werden bei der ersten Betrachtung des vorliegenden Datenmodells vor allem die untereinander vorherrschenden Beziehungen untersucht. Es wird hierbei die Hypothese aufgestellt, dass diese elementaren Tabellen über besonders viele Beziehungen zu anderen Tabellen verfügen sollten. Somit fallen sofort alle beziehungslosen Tabellen aus der näheren Betrachtung heraus. Diese sind in Abbildung 12 zu sehen, wobei in dieser Darstellung aus Platzgründen nur die Namen der betreffenden Tabellen und nicht deren Attribute angezeigt werden.

Batch	ChunkData	ChunkSegmentMapping
ConfigurationInfo	DBUpgradeHistory	Event
ExecutionLogStorage	History	Keys
RunningJobs	Segment	SegmentedChunk
ServerParametersInstance	ServerUpgradeHistory	SnapshotData
SubscriptionsBeingDeleted	UpgradeInfo	

Abbildung 12: Tabellen des Datenmodells die keine Beziehungen zu anderen Tabellen besitzen

Damit alle Beziehungen in kurzer Zeit erfasst werden können, wird ein vereinfachtes ERM mit den noch übrigen Tabellen erstellt. Auch bei dieser Darstellung in Abbildung 13 auf Seite 36 sind nur die Tabellennamen und nicht deren Attribute sichtbar. Die Beziehungen werden mit einfachen Verbindungslinien angezeigt, wobei aus Gründen der Übersichtlichkeit auf die Angabe der Kardinalitäten verzichtet wird.

Schnell fällt bei dem simplen ERM die *Catalog*-Tabelle mit ihren außerordentlich vielen Beziehungen zu anderen Tabellen auf. An dieser Stelle wäre eine offizielle Dokumentation über das vorliegende Datenmodell für die nähere Analyse der *Catalog*-Tabelle von Vorteil. Da eine solche Dokumentation jedoch nicht existiert, werden für die detailliertere Untersuchung die gewonnenen Erkenntnisse aus der Verarbeitung von Testdaten, sowie die Internetquelle [Mut11] verwendet. Laut dieser Quelle ist es tatsächlich so, dass die *Catalog*-Tabelle Metadaten über alle Objekte speichert, die mit den Reporting Services des SQL-Servers zu tun haben. Zu diesen Objekten gehören unter anderem die erstellten Berichte, verwendete Datenquellen, sowie freigegebene Datasets.

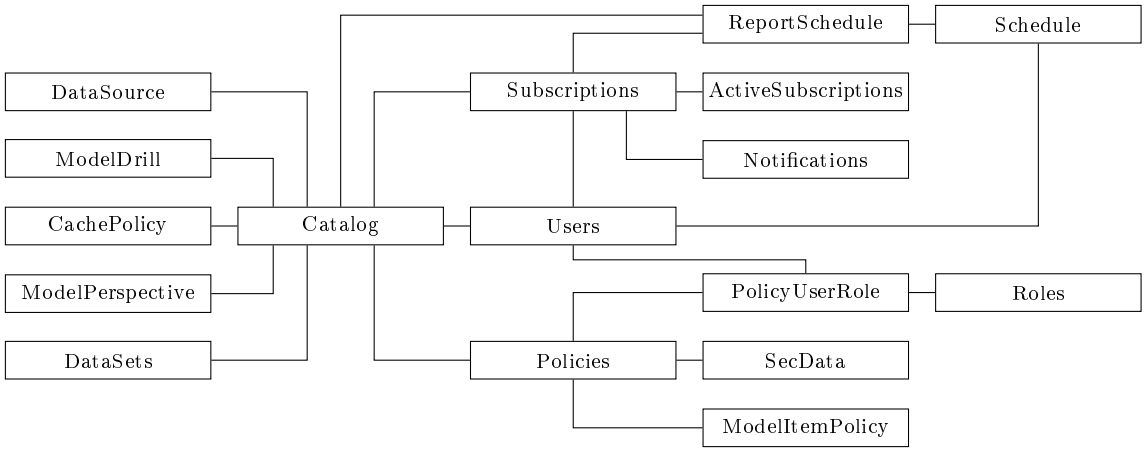


Abbildung 13: Tabellen des Datenmodells die in Beziehung zueinander stehen

Weiterhin besitzen auch die Tabellen *Subscriptions*, *Users*, *Policies* und *ReportSchedule* mindestens drei Beziehungen zu anderen Tabellen. Gemäß [Mut11] und den Rückschlüssen, die man aus den Tabellennamen ziehen kann, enthält die *Subscriptions*-Tabelle Datensätze bezüglich der erstellen Abonnements seitens der Anwender und die *Users*-Tabelle eine Auflistung interessanter Daten über die Berichtsbenutzer. Schaut man sich dazu die gestellten Anforderungen aus Abschnitt 3.1.1 an, könnten diese beiden Tabellen für die Umsetzung des Metareports von besonderer Bedeutung sein. Bleibt man bei den Anforderungen, so spielt die *Policies*-Tabelle mit ihren wenigen Daten über die Berechtigungen von erstellten Objekten eine eher untergeordnete Rolle. Die Tabelle *ReportSchedule* hingegen verknüpft die Abonnements bestimmter Berichte mit ihren zugehörigen Zeitplänen. Mit dieser Information könnte die Anforderung nach der Abfrage von Abonnementdaten womöglich gezielt erfüllt werden. Wie aber später in Abschnitt 3.2.1.3 auf Seite 45 erläutert wird, existiert auch eine Sicht, die diese Daten bereits zusammenbringt, weshalb hier nicht mehr näher auf die *ReportSchedule*-Tabelle eingegangen wird.

Die Abbildung 14 auf Seite 37 stellt die noch übrigen und im Folgenden intensiver untersuchten Tabellen dar. Hierbei wird eine an UML angelehnte Notation verwendet (siehe Abschnitt 2.4.3). Der Tabellenkopf enthält jeweils den Tabellennamen und die weiteren Zeilen eine auszugsweise Auflistung der Attribute. Die linke Tabellenspalte gibt an, ob es sich um einen Primärschlüssel (*primary key* abgekürzt mit PK) oder um einen Fremdschlüssel (*foreign key* abgekürzt mit FK) handelt, während die rechte Spalte den Attributsnamen ausgibt. Direkt an den Verbindungslinien sind die entsprechenden Kardinalitäten angegeben.

Bei der *Catalog*-Tabelle erhält jedes der für die SSRS relevanten Objekte eine eindeutige *ItemID*, die auch als Primärschlüssel verwendet wird, sowie einen Namen und eine Pfadangabe. Interessanterweise verweist die *ParentID* wieder zurück auf den Primärschlüssel der

Catalog-Tabelle. Dadurch wird eine Baumstruktur geschaffen, bei der über die *ParentID* und *ItemID* jedes einzelne Objekt bis zu einem bestimmten Wurzelobjekt eindeutig zurückverfolgt werden kann. Das *Type*-Attribut gibt Auskunft darüber, um welche Art von Objekt es sich genau handelt. Die möglichen Werte für dieses Attribut können der Quelle [Mut11] entnommen werden. Bei den übrigen Attributen dieser Tabelle handelt es sich um Verweise zu den Benutzern, die das Objekt erstellt und zuletzt modifiziert haben.

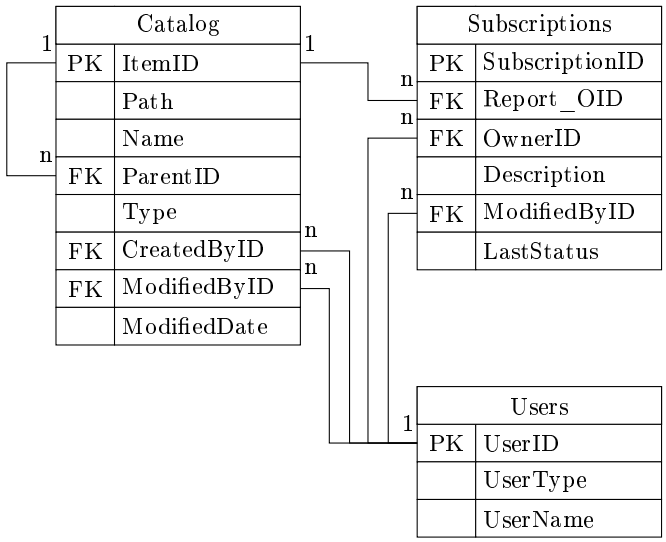


Abbildung 14: Beziehungen der näher analysierten Tabellen

Auch bei der *Subscriptions*-Tabelle verfügt jedes Abonnement über eine eindeutige *SubscriptionID*, welche ebenso als Primärschlüssel der Tabelle dient. Die einzelnen Abonnements verweisen zudem auf die entsprechenden abonnierten Berichte aus der *Catalog*-Tabelle. Da die Abonnements von Anwendern eingerichtet werden, referenzieren die Datensätze über die *OwnerID*- und *ModifiedByID*-Spalten auch die jeweiligen Ersteller und letzten Bearbeiter. Abschließend gibt die Tabelle den Status der letzten Abonnementausführung in Form einer kurzen automatisierten Meldung über das Attribut *LastStatus* aus.

Innerhalb der *Users*-Tabelle findet sich für jeden Benutzer des Berichtsservers eine eindeutige Kennung, die wiederrum als Primärschlüssel genutzt wird und auf die die *Catalog*- und *Subscriptions*-Tabellen entsprechend verweisen. Der *UserType* gibt an, ob der Benutzer manuell oder vom System automatisch angelegt wurde. Über das Attribut *UserName* kann der zugewiesene Benutzername in Erfahrung gebracht werden.

Natürlich kann eine umfassende Analyse des vorliegenden Datenmodells nicht alleine auf die Unterstellung gestützt werden, dass die Tabellen mit essenziellen Daten für den Metareport möglichst viele Beziehungen zu anderen Tabellen besitzen. Tatsächlich ergab die Untersuchung der Daten, dass ausgerechnet eine beziehungslose Tabelle die zur Erfüllung der gestellten Anforderungen bedeutsamsten Daten enthält, womit die zuvor aufgestellte Hypothese widerlegt ist. Es handelt sich hierbei um die *ExecutionLogStorage*-Tabelle. Diese

ist in Abbildung 15 zusammen mit einer Auswahl an Attributen dargestellt.

ExecutionLogStorage	
PK	LogEntryID
	InstanceName
	ReportID
	UserName
	Parameters
	TimeStart
	TimeEnd
	Status
	TimeDataRetrieval
	TimeProcessing
	TimeRendering

Abbildung 15: Auszug aus der ExecutionLogStorage-Tabelle

Jeder einzelne Datensatz innerhalb der *ExecutionLogStorage*-Tabelle steht für einen Aufruf eines bestimmten Berichtes und enthält äußerst wichtige Daten für den Metareport. Auch die Quelle [Mut11] gibt an, dass die dort hinterlegten Daten für die leistungsmäßige Optimierung der Berichtsausführung hilfreich sein können.

Den einzelnen Berichtsausführungen sind in der Spalte *LogEntryID* eindeutige Identifikationsnummern zugeordnet, die gleichzeitig den Primärschlüssel der Tabelle darstellen. Die Spalte *InstanceName* gibt den Namen der SQL-Serverinstanz an, auf der der Bericht ausgeführt wurde. Über die *ReportID*-Spalte kann das entsprechende Berichtsobjekt der *Catalog*-Tabelle direkt referenziert werden. Obwohl der Verweis klar ersichtlich ist, ist diese Tabellenspalte dennoch nicht als Fremdschlüssel definiert worden (siehe dazu Abschnitt 3.1.2.3). Der *UserName* gibt den Benutzernamen des Anwenders an, der den Bericht abgerufen hat und die Spalte *Parameters* listet die Werte der dabei übergebenen Berichtsparameter auf. Den Start- und Endzeitpunkt vom Ausführungsprozess geben die *TimeStart*- bzw. *TimeEnd*-Spalten an. Eine Aussage über die erfolgreiche Berichtsausführung trifft die *Status*-Spalte über eine entsprechende Konstante. Die übrigen in Abbildung 15 zu sehenden Tabellenspalten geben an, wie viel Zeit die einzelnen Schritte der Ausführung in Anspruch genommen haben.

Mit den abfragbaren Daten der eben vorgestellten Tabellen können einige der an den Metareport gestellten Anforderungen bereits gezielt erfüllt werden. Dazu gehört unter anderem die Berechnung der Anzahl an einzelnen Berichtsaufrufen, die Abfrage der für die Berichtskreation aufgewendeten Systemressourcen oder auch der Zugriff auf die Metadaten bestehender Abonnements. Wie dies im Detail mit den dazugehörigen SQL-Abfragen umgesetzt wird, beschreibt der Abschnitt 3.2.1 auf Seite 42.

Nützlicherweise können die gewonnenen Erkenntnisse aus der Analyse für die Microsoft SQL-Server Versionen 2008 R2 und 2012 genutzt werden, da ein Vergleich ergab, das diese

beiden Versionen das gleiche Datenmodell verwenden. Weitere Versionen von Microsofts SQL-Server wurden nicht auf Gemeinsamkeiten in ihrem Datenmodell hin untersucht.

3.1.2.3. Soll-ERM

Die Tabelle *ExecutionLogStorage*, die bereits in Abbildung 15 gezeigt wurde, verweist über die *ReportID*s der gleichnamigen Tabellenspalte auf konkrete Datensätze der *Catalog*-Tabelle, jedoch ist diese Spalte nicht als Fremdschlüssel definiert. Innerhalb des Datenmodells gibt es noch weitere Tabellen deren Spalten zwar auf Datensätze aus anderen Tabellen verweisen, die aber ebenfalls nicht als Fremdschlüssel ausgewiesen sind.

In Abschnitt 2.4.1 wurde bereits erwähnt, dass Fremdschlüssel dazu genutzt werden, die Verbindungen der Tabellen untereinander anzuzeigen. Darüber hinaus stellen sie in relationalen Datenbanksystemen die referentielle Integrität sicher. Damit ist gemeint, dass eine Tabelle nur auf Daten verweisen kann, die auch tatsächlich vorhanden sind. Dies geht soweit, dass gesetzte Fremdschlüssel das Entfernen von referenzierten Datensätzen beschränken können. So kann ein Datensatz beispielsweise nur dann gelöscht werden, wenn keine Fremdschlüsselspalte mehr auf den konkreten Primärschlüssel dieses Datensatzes verweist. Es besteht aber auch die Möglichkeit die Beschränkung so zu definieren, dass beim Löschen eines Datensatzes alle ihn referenzierenden Datensätze automatisch mit entfernt werden. Bei beiden Beispielen würde die referenzielle Integrität der Datenbank gewahrt bleiben. [vgl. MicFK]

Die folgende kurze Schilderung soll das Problem der im vorliegenden Datenmodell fehlenden Fremdschlüssel deutlich machen:

Wenn man einen neuen Bericht auf einer SQL-Serverinstanz bereitstellt, so wird der *Catalog*-Tabelle ein neuer Datensatz mit einer einzigartigen *ItemID* für das neue Berichtsobjekt hinzugefügt. Jeder einzelne Aufruf dieses Berichtes erzeugt in der *ExecutionLogStorage*-Tabelle ebenfalls einen neuen Datensatz. Der Wert der *ReportID*-Spalte entspricht dabei der *ItemID* des entsprechenden Berichtes aus der *Catalog*-Tabelle. Löscht man nun den bereitgestellten Bericht wieder, so wird auch der Datensatz mit der dazugehörigen *ItemID* aus der *Catalog*-Tabelle entfernt. Die *ExecutionLogStorage*-Tabelle bleibt davon jedoch unberührt, da sie über keine Fremdschlüssel verfügt. Einzelne Datensätze dieser Tabelle verweisen nun mit ihrer *ReportID* auf nicht mehr vorhandene Objekte, wodurch die referentielle Integrität zerstört ist.

Wieso das Datenmodell nicht nur an dieser Stelle, sondern auch bei anderen Tabellen keine Fremdschlüssel definiert und damit die referentielle Integrität nicht gewährleistet, kann aufgrund der nicht vorhandenen offiziellen Dokumentation nur vage vermutet werden. Womöglich liegt es daran, dass die *ExecutionLogStorage*-Tabelle ausschließlich Ereignisprotokolle, also Daten über vergangene Aktionen, enthält. Somit wäre es nicht sinnvoll, die Logdaten zu einem Objekt zu entfernen, sobald das Objekt nicht mehr vorhanden ist. Eventuell ist auch eine gewünschte Leistungssteigerung der Grund für das Weglassen der

Fremdschlüssel. Die ständigen Überprüfungen zur Sicherstellung der Datenintegrität können einen Leistungseinbruch zur Folge haben. Deshalb wird mitunter diese Aufgabe an die Anwendung delegiert, um auf die Nutzung von Fremdschlüsseln verzichten zu können[7].

Würde man die referentielle Integrität auf Datenbankebene aufrecht halten wollen, müsste das vorliegende Datenmodell um weitere Tabellenbeziehungen erweitert werden. Die Abbildung 16 zeigt ein entsprechendes Beispiel. Alle zuvor beziehungslosen Tabellen die dadurch Fremdschlüsselbeziehungen zu anderen Tabellen erhalten, sind dabei mit einem grauen Hintergrund versehen. Die neuen Beziehungen sind mittels gestrichelten Verbindungslinien dargestellt.

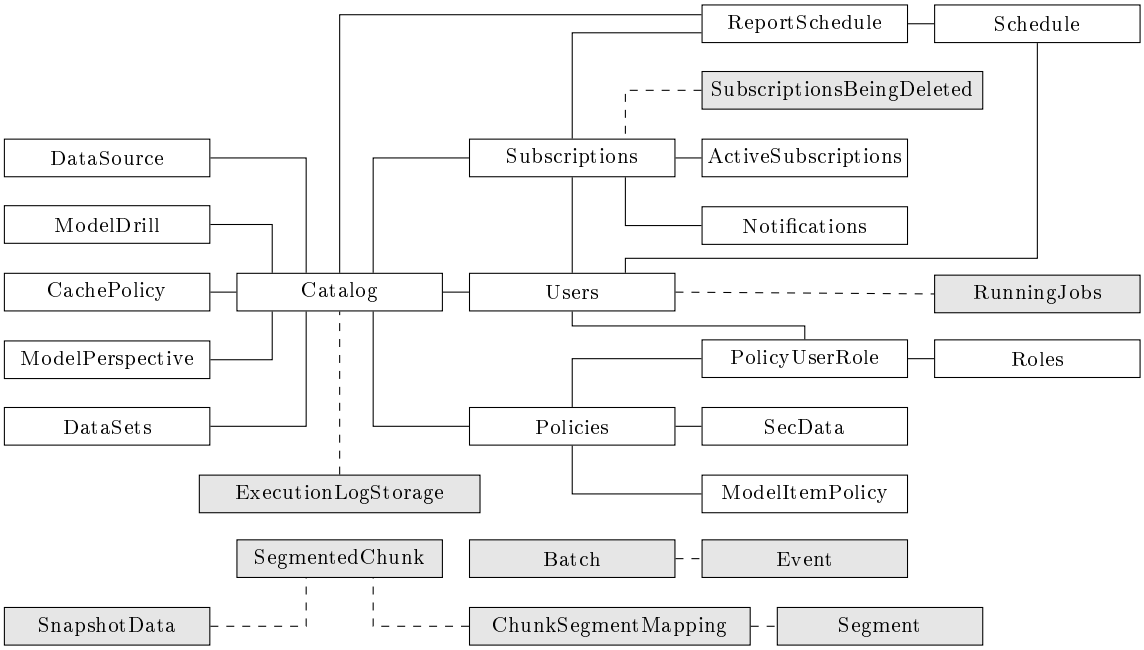


Abbildung 16: Die Tabellen des Soll-ERMs mit ihren neu definierten Beziehungen

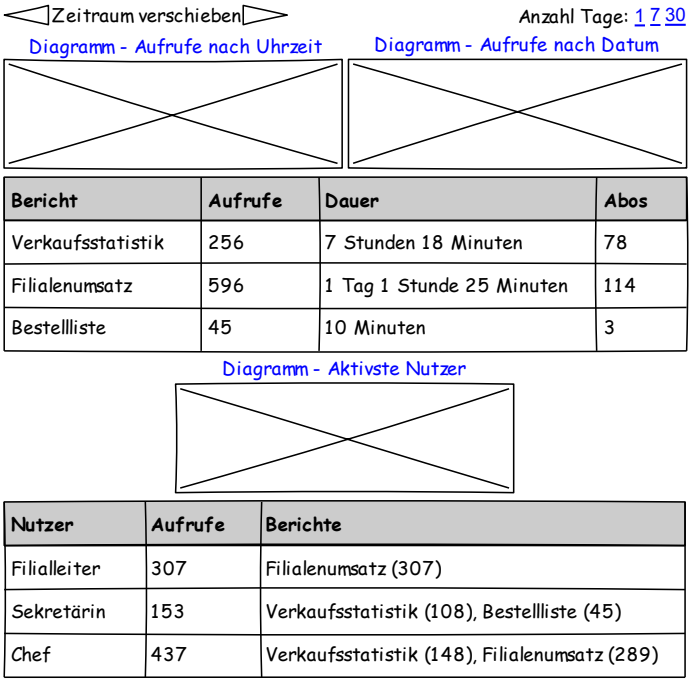
Durch die neuen Verbindungen wäre das Datenmodell leichter zu analysieren, da schneller erfasst werden kann, welche Tabellen zueinander gehören und zusätzliche Daten zu einem Objekt bereitstellen. Über den Nutzen der neuen Tabellenbeziehungen für den SQL-Server kann an dieser Stelle keine klare Aussage getroffen werden, da die Intention der Entwickler bezüglich des tatsächlichen Datenmodells aufgrund der nicht verfügbaren Dokumentation unbekannt ist.

[7] Diese und weitere Möglichkeiten zur Verbesserung der Leistung verschiedener Datenbanksysteme können unter [SynPe] nachgelesen werden

3.1.3. Mockup

Der Begriff **Mockup** stammt aus dem Englischen und lässt sich mit Attrappe, Modell oder auch Nachbildung übersetzen [vgl. MocDe]. Im Bereich der Informatik nutzt man Mockups vor allem beim Designprozess von grafischen Benutzeroberflächen. Um sich Zeit und Arbeit zu sparen, skizziert man dabei bereits vor der eigentlichen Implementierung die Oberfläche und hat somit einen einfachen „Wegwerfprototypen“ als Diskussionsgrundlage. Das dadurch erhaltene Feedback kann für die nächste Version des Prototypen genutzt werden. Diese iterative Vorgehensweise, bei der ein Prototyp mehrfach getestet und anschließend verbessert wird, wird auch als **Rapid Prototyping** bezeichnet. Um für die einzelnen Versionsskizzen nicht immer wieder Stift und Zettel nutzen zu müssen, werden Mockup-Tools verwendet, mit denen die Entwürfe digital erstellt werden können. [vgl. Moc13]

Für den Metareport wurde im Vorfeld der in Abbildung 17 dargestellte Prototyp mit Hilfe des Open Source Mockup-Tools *Pencil*[8] entworfen.



Bericht	Aufrufe	Dauer	Abos
Verkaufsstatistik	256	7 Stunden 18 Minuten	78
Filialenumsatz	596	1 Tag 1 Stunde 25 Minuten	114
Bestellliste	45	10 Minuten	3

Nutzer	Aufrufe	Berichte
Filialleiter	307	Filialenumsatz (307)
Sekretärin	153	Verkaufsstatistik (108), Bestellliste (45)
Chef	437	Verkaufsstatistik (148), Filialenumsatz (289)

Abbildung 17: Mockup des Metareports

Der Prototyp zeigt im oberen Bereich des Metareports anklickbare Elemente, mit denen der auszuwertende Zeitraum konfiguriert werden kann. Mit diesen kann die Anzahl der Tage bestimmt und das Zeitintervall in die Vergangenheit oder Zukunft verschoben werden. Gleich darauf folgen zwei Diagrammelemente, die die Anzahl der Aufrufe einmal nach der Uhrzeit und einmal nach dem Datum darstellen. Das nächste Berichtselement ist eine Tabelle, die die Namen, Aufrufe, Gesamtdauer der Erstellung und abgeschlossenen Abon-

[8] Webseite des Mockup-Tools Pencil – http://pencil.evolus.vn/

nements der Berichte auflistet. Die aktivsten Berichtsnutzer zeigt das danach zu sehende Diagramm. Der Metareport wird von einer Tabelle abgeschlossen, in der die Benutzernamen, die Anzahl der abgerufenen Berichte dieses Benutzers und die Aufschlüsselung dieser Gesamtanzahl der Aufrufe ausgegeben wird.

Im Endergebnis weicht zwar das Aussehen des Metareports vom Mockup ab (siehe Abbildung 45 im Anhang auf Seite 80), dennoch diente der Prototyp während der Umsetzungsphase als gute Orientierungshilfe.

3.2. Implementierung

3.2.1. Abfrage der Metadaten

3.2.1.1. Ermittlung der Aufrufhäufigkeit von Berichten

Gemäß der Anforderungsanalyse (siehe Abschnitt 3.1.1), soll die genaue Anzahl der Aufrufe aller Berichte angezeigt werden, wobei der momentan betrachtete Zeitraum berücksichtigt werden muss. Um diese Anforderungen erfüllen zu können, wird eine Auflistung aller angeforderten Berichte mit deren Aufrufzeitpunkt benötigt.

Laut dem Datenmodell, welches in Abschnitt 3.1.2 gezeigt wird, wird jeder abgerufene Bericht als Datensatz der *ExecutionLogStorage*-Tabelle geführt. Zu jedem dieser Datensätze gehören unter anderem die eindeutige Identifikationszeichenfolge des Berichtes und der genaue Zeitpunkt, an dem der Aufruf des Berichtes erfolgte. Filtert man diese Datensätze nach deren Aufrufzeitpunkten, genügt es, anschließend nachzuzählen, wie oft die gleiche ID eines Berichtes noch auftaucht. Genau dieses Filtern und Nachzählen führt die in Listing 8 dargstellte SQL-Abfrage durch. Aufgrund der Länge der Abfrage wird dieses Listing allerdings auf die relevantesten Zeilen gekürzt.

```
1  SELECT
2      c.path,
3      ex.ReportID,
4      COUNT(ex.ReportID) AS ReportCount,
5      (
6          SELECT COUNT(DISTINCT s.SubscriptionID)
7          FROM Subscriptions s
8          WHERE (s.Report_OID = ex.ReportID)
9      ) AS Subs
10 FROM ExecutionLogStorage ex
11 JOIN Catalog c ON ex.ReportID = c.ItemID
12 WHERE CAST(ex.TimeStart AS DATE) BETWEEN CAST(@DateFrom AS DATE
       ) AND CAST(@DateTo AS DATE)
13 GROUP BY c.path, ex.ReportID
14 ORDER BY ReportCount DESC
```

Listing 8: Ermittlung der Aufrufhäufigkeit von Berichten

Innerhalb dieser SQL-Abfrage tauchen in Zeile 12 die beiden Parameter *@DateFrom* und

@DateTo auf. Diese enthalten, wie später in Abschnitt 3.2.2.2 ab Seite 55 noch näher erläutert wird, das Start- und Enddatum des zu berücksichtigenden Zeitraumes und werden beim Aufruf des Berichtes gesetzt. In der angesprochenen Zeile werden die Datensätze gemäß der beiden gesetzten Daten gefiltert, es bleiben also nur die Datensätze übrig, welche die benötigte Information der Reports beinhalten, die innerhalb des gesetzten Zeitraumes aufgerufen wurden. Gezählt wird die Anzahl der noch übrigen aufgerufenen Berichte in Zeile 4 mit Hilfe der *COUNT*-Aggregatfunktion von SQL. Als zusätzliche Daten beinhaltet die resultierende Ergebnismenge außerdem auch die Pfade der einzelnen Reports auf dem Server. Da diese Information nur in der Catalog-Tabelle zu finden ist, müssen die Datensätze über ihre Berichts-ID verknüpft werden. Dies geschieht in Zeile 11 mit dem *JOIN*-Befehl. Darüber hinaus wird auch die Anzahl der eingerichteten Abonnements für jeden unterschiedlichen Bericht mitgezählt. Dazu erfolgt eine Unterabfrage in den Zeilen 5 bis 9, die die Anzahl der Abonnements über die bereits zuvor erwähnte *COUNT*-Aggregatfunktion aufsummiert. In Abbildung 18 ist eine mögliche Ergebnismenge abgebildet.

Ergebnisse | Meldungen

	path	ReportID	ReportCount	Subs
1	/Flaechencontrolling/Flaechencontrolling	ECFC7847-61AC-4C7A-8FB6-1E4AE5386625	3959	0
2	/Artikel-Renner-Penner inkl. Areamanager/ArtikelREnn...	3A7BB0F8-B626-4CF5-9ED4-51BD6B2942E1	2422	0
3	/Liste 803 Umsätze pro Tag (neu)/Liste803	E2ED13DA-A129-45A8-B135-AEAE359AD449	465	31
4	/Artikel-Renner-Penner inkl. Areamanager (Abonnemen...	4DA475C1-E54C-4A85-AB0E-2865B3B3C89C	315	105
5	/DynamischeKER (Abonnements)/DynamischeKER	62E87307-271C-47E7-AF85-809C8EAB3159	248	64
6	/NetSalesDaily Version3/Net Sales Daily Version3	6A3C9355-980F-45E7-AC25-EEBBE0686598	127	3
7	/DynamischeKER/DynamischeKER	F0060A9C-6D71-46C3-B5A4-E48C7C496EEC	39	0
8	/NetSalesDaily (Abonnements)/Net Sales Daily	ED69FE25-DE97-4542-A077-81B980EC13DF	39	5
9	/NetSalesDaily/Net Sales Daily	7A63471F-BAB0-4CC1-AA3E-4BEC836B0266	27	2
10	/Shop-Erfolg/ShopErfolg	6D67AAB1-CACA-4BA1-B412-45FAFBF781AB	24	0
11	/DynamischeKER FRS (Abonnements)/DynamischeKER	B1EE6B03-3387-47A8-B170-4E9A63E81402	24	10

Abbildung 18: Beispielhafte Ergebnismenge für die Anzahl der Aufrufe von Berichten

In der Tabelle mit dem Resultat zeigen die Spalten von links nach rechts die Serverpfade, Berichts-IDs, sowie die Anzahl der Aufrufe und der erstellten Abonnements.

3.2.1.2. Analyse der aktivsten Berichtsnutzer

Um zu ermitteln, welche Anwender die meisten Berichte abgerufen haben, würde es genügen, die Anzahl der Aufrufe aller Reports eines bestimmten Nutzers aufzusummieren. Allerdings verlangt die entsprechende Anforderung aus Abschnitt 3.1.1 zusätzlich noch die genaue Aufschlüsselung der einzelnen Berichte. Somit ist es nötig, die Ergebnismenge nicht nur nach Nutzern, sondern auch nach Berichten zu gruppieren und entsprechende Zwischensummen zu bilden. Die *GROUP BY*-Anweisung von SQL, die zur Gruppierung von Daten verwendet wird, verfügt nützlicherweise über entsprechende Operatoren, die das Bilden von zusätzlichen Zwischensummen von bereits gruppierten Ergebnismengen ermöglichen. Einer

dieser Operatoren ist der *ROLLUP*-Operator, der auch in der SQL-Abfrage in Listing 9 zur Ermittlung der aktivsten Berichtsnutzer eingesetzt und im Folgenden erläutert wird.

```
1  SELECT
2      ex.UserName,
3      ex.ReportID,
4      COUNT(ex.ReportID) AS ReportCount,
5      c.Path
6  FROM ExecutionLogStorage ex
7  JOIN Catalog c ON ex.ReportID = c.ItemID
8  WHERE CAST(ex.TimeStart AS DATE) BETWEEN CAST(@DateFrom AS DATE
       ) AND CAST(@DateTo AS DATE)
9  GROUP BY ex.UserName, ROLLUP(c.Path, ex.ReportID)
10 ORDER BY ReportCount DESC, ex.UserName
```

Listing 9: Abfrage der aktivsten Berichtsnutzer

Über die *WHERE*-Anweisung in Zeile 8 der SQL-Abfrage werden die Datensätze wieder nach dem betrachteten Zeitraum gefiltert. Auch das Zählen der Berichtsaufrufe, sowie die Verknüpfung der Datensätze der *ExecutionLogStorage*-Tabelle mit den Datensätzen der Catalog-Tabelle, um die Verzeichnispfade der Berichte herauszufinden, ist bereits aus dem zuvor gezeigtem Listing 8 auf Seite 42 bekannt. Neu an der SQL-Abfrage in Listing 9 ist der verwendete *ROLLUP*-Operator innerhalb der *GROUP BY*-Anweisung in Zeile 9. Für jede übergebene Spalte fügt dieser Operator der Ergebnismenge eine zusätzliche Zeile mit einer entsprechenden Zwischensumme hinzu. Innerhalb dieser Zeilen werden nicht gruppierte Spalten mit dem Wert *NULL* belegt. Somit ist es dank des *ROLLUP*-Operators möglich, nicht nur die Gesamtanzahl an von diesem Benutzer aufgerufenen Berichten zu ermitteln, sondern in der gleichen SQL-Abfrage zusätzlich noch die Aufrufanzahl der einzelnen Berichte über die Zwischensummen abzufragen. Die Abbildung 19 zeigt eine entsprechende Ergebnismenge mit den angesprochenen zusätzlichen Zeilen. Von links nach rechts sind dabei in den Spalten die Benutzernamen, Berichts-IDs, Anzahl der Aufrufe und Verzeichnispfade der Berichte dargestellt.

Ergebnisse | Meldungen

	UserName	ReportID	ReportCount	Path
1	Oola (1)	NULL	354	NULL
2	Oola (1)	4DA475C1-E54C-4A85-AB0E-2865B3B3C89C	132	/Artikel-Renner-Penner inkl. Areamanager (Abonne...
3	Oola (1)	NULL	132	/Artikel-Renner-Penner inkl. Areamanager (Abonne...
4	Oola (1)	62E87307-271C-47E7-AF85-809C8EAB3159	124	/DynamischeKER (Abonnements)/DynamischeKER
5	Oola (1)	NULL	124	/DynamischeKER (Abonnements)/DynamischeKER
6	Winter	NULL	104	NULL
7	Tahiri Veila	NULL	98	NULL
8	Exar Kun	ECFC7847-61AC-4C7A-8FB6-1E4AE5386625	91	/Flaechencontrolling/Flaechencontrolling
9	Exar Kun	NULL	91	/Flaechencontrolling/Flaechencontrolling

Abbildung 19: Beispielhafte Ergebnismenge für die aktivsten Berichtsnutzer

Die resultierende Tabelle in Abbildung 19 auf Seite 44 wirkt auf den ersten Blick chaotisch und wenig aussagekräftig, allerdings bieten die SSRS verschiedene Möglichkeiten, um diese Daten nachträglich noch aufzubereiten. So kann die Ergebnismenge beispielsweise nach einzelnen Spalten gruppiert oder bestimmte Zeilen aus der Resultatstabelle gefiltert werden. Deshalb genügt das unausgereift wirkende Abfrageergebnis als Grundlage, um mit den gegebenen Mitteln der SSRS die gestellten Anforderungen dennoch zu erfüllen. Wie diese zusätzliche Gruppierung und Filterung der Daten erfolgt, wird in Abschnitt 3.2.2.4 auf Seite 57 gezeigt.

3.2.1.3. Beanspruchte Ressourcen für die Berichtserstellung abrufen

Wird ein Bericht neu erstellt, so werden für diesen Report Daten aus zuvor festgelegten Datenquellen gesammelt, gefiltert und sortiert. Alle diese Daten werden in den entsprechenden Anzeigeelementen des Berichtes eingefügt und je nach Umfang der erhaltenen Daten über mehrere Seiten hinweg für ein bestimmtes Anzeigeformat gerendert. Diese Schritte der Berichtsgenerierung benötigen vor allem Zeit. Innerhalb der *ExecutionLogStorage*-Tabelle existieren die Spalten *TimeDateRetrieval*, *TimeProcessing* und *TimeRendering* die jeweils eine Zeitinformation für einen bestimmten Generierungsschritt des Berichtes angeben. Die SQL-Abfrage in Listing 10 summiert diese Spalten in den Codezeilen 5 bis 7 jeweils auf und bildet außerdem die durchschnittlich (Zeile 3) und die insgesamt (Zeile 4) benötigte Zeit der Berichtsgenerierung. Auch bei dieser Abfrage wird wieder der aktuell betrachtete Zeitraum mit Hilfe der beiden Parameter *@DateFrom* und *@DateTo* berücksichtigt und die Ergebniszeilen, die sich außerhalb dieses Zeitraumes befinden, durch die *WHERE*-Anweisung in Zeile 9 aus der Ergebnismenge gefiltert.

```
1  SELECT
2      ex.ReportID,
3      ( SUM(ex.TimeDataRetrieval) + SUM(ex.TimeProcessing) + SUM(
           ex.TimeRendering) ) / COUNT(ex.ReportID) AS AverageTime,
4      SUM(ex.TimeDataRetrieval) + SUM(ex.TimeProcessing) + SUM(ex
           .TimeRendering) AS TotalTime,
5      SUM(ex.TimeDataRetrieval) AS SumDataTime,
6      SUM(ex.TimeProcessing) AS SumProcessingTime,
7      SUM(ex.TimeRendering) AS SumRenderingTime
8  FROM ExecutionLogStorage ex
9  WHERE CAST(ex.TimeStart AS DATE) BETWEEN CAST(@DateFrom AS DATE
       ) AND CAST(@DateTo AS DATE)
10 GROUP BY ex.ReportID
```

Listing 10: Ermittlung der benötigten Generierungszeit

Eine andere Systemressource, die während der Berichterstellung in Beschlag genommen wird, ist der Speicher. Nach eigenen Recherchen (siehe Abschnitt 3.1.2.2) findet man innerhalb der Tabellen mit den Metadaten keine Information über den Speicherverbrauch

während der Berichtsgenerierung, allerdings stellt Microsoft selber dokumentierte Sichten mit zusätzlichen Metadaten bereit (siehe [ResMi]). Hierbei handelt es sich unter anderem um die Sicht namens *ExecutionLog3*, die in der Spalte *AdditionalInfo* verschiedene Daten im XML-Format beinhaltet, darunter auch Angaben über den geschätzten Speicherverbrauch. Beispieldaten, die innerhalb dieser Spalte stehen können, zeigt das Listing 11.

```
1  <AdditionalInfo>
2    <ProcessingEngine>2</ProcessingEngine>
3    <ScalabilityTime>
4      <Pagination>0</Pagination>
5      <Processing>0</Processing>
6    </ScalabilityTime>
7    <EstimatedMemoryUsageKB>
8      <Pagination>4</Pagination>
9      <Processing>18</Processing>
10   </EstimatedMemoryUsageKB>
11   <DataExtension>
12     <SQL>1</SQL>
13   </DataExtension>
14 </AdditionalInfo>
```

Listing 11: Beispieldaten der Spalte AdditionalInfo [ResMi]

Die für den Metareport interessante Information enthalten die Kindknoten des *EstimatedMemoryUsageKB*-Tags zu sehen in Listing 11 in den Zeilen 7 bis 10. Diese geben laut [ResMi] den Spitzenspeicherverbrauch in Kilobyte an, welcher von allen Komponenten während einer bestimmten Anfrage verbraucht wurden. Da die Daten im XML-Format vorliegen, genügt es nicht, einfach nur auf die Spalte zuzugreifen. Man muss sich zusätzlich entlang der Achsen der einzelnen XML-Knoten bewegen, um an die interessanten Werte zu gelangen. Das Listing 12 zeigt die Extraktion der XML-Daten aus der Spalte *AdditionalInfo*.

```
1  SELECT
2      c.ItemID,
3      AVG(exl.AdditionalInfo.value('(AdditionalInfo/
           EstimatedMemoryUsageKB/Pagination)[1]', 'INT')) AS
           AvgMemoryUsagePaginationKB,
4      AVG(exl.AdditionalInfo.value('(AdditionalInfo/
           EstimatedMemoryUsageKB/Processing)[1]', 'INT')) AS
           AvgMemoryUsageProcessingKB
5  FROM ExecutionLog3 exl
6  JOIN Catalog c ON c.Path = exl.ItemPath
7  WHERE CAST(exl.TimeStart AS DATE) BETWEEN CAST(@DateFrom AS
       DATE) AND CAST(@DateTo AS DATE)
8  GROUP BY c.ItemID
```

Listing 12: Zugriff auf die Daten des Speicherverbrauchs

Innerhalb der *AdditionalInfo*-Spalte liegen die Daten im XML-Datenformat vor, weshalb darauf direkt die *value*-Methode aufgerufen werden kann. Diese Methode erwartet zwei Argumente, zuerst einen XQuery-Ausdruck, der mindestens einen Wert zurückgeben muss und als zweites einen SQL-Typ, der von der Methode zurückgegeben werden soll. In den Zeilen 3 und 4 von Listing 12 auf Seite 46 werden als Argumente einfache Pfadausdrücke zu den Elementknoten mit der gewünschten Information und ein ganzzahliger SQL-Typ angegeben. Die zurückgegebenen Werte werden direkt an die *AVG*-Aggregatfunktion übergeben, die den Mittelwert gruppiert nach den Bericht-IDs (siehe Zeile 8) zurückliefert. Mit der *WHERE*-Anweisung in Zeile 7 wird die Ergebnismenge nach einem bestimmten Zeitraum gefiltert.

3.2.1.4. Ermitteln der übergebenen Berichtsparameter

Berichte können parametrisiert werden, wodurch es unter anderem möglich ist, die angezeigten Daten nach bestimmten Kriterien, beispielsweise dem Geschäftsjahr oder der Produktkategorie, zu filtern. Verfügt ein Report über Parameter, müssen diese vor seiner Erstellung zuerst gesetzt werden. Alle Parameter, die einem Bericht beim Aufruf übergeben wurden, sind in der Tabelle *ExecutionLogStorage* in der Spalte *Parameters* zu finden. Das Format, in dem die Parameter dort hinterlegt sind, entspricht dem gleichen Format, in dem auch GET-Parameter an einen URI übergeben werden. Das heißt, Parametername und -wert sind durch ein Gleichheitszeichen voneinander getrennt und die einzelnen Kombinationen aus Parametername und -wert separiert jeweils ein Ampersandzeichen (siehe [RFC05]).

```
1 Vorgabe=1&Datum=03/06/2014 00:00:00&DatumStr=2014-03-06&↩
    DatumVJStr=2013-03-06
```

Listing 13: Beispiel von gespeicherten Berichtsparametern

In Listing 13 sind beispielhaft vier verschiedene Berichtsparameter zu sehen, deren gesetzte Werte jeweils auf ein Gleichheitszeichen folgen. Da diese Form der Darstellung bei vielen Parametern sehr umfangreich werden kann und darüber hinaus Berichte erfahrungsgemäß oft mit vielen unterschiedlichen Parameterkombinationen abgerufen werden, erfolgt die Auswertung der Parameter berichtsweise.

Die SQL-Abfrage in Listing 14 auf Seite 48 liefert als Resultat alle verschiedenen Parameterkombinationen und die Häufigkeit des Auftretens jeder Kombination zurück. Gefiltert wird die Ergebnismenge über den *@ReportUniqueID*-Parameter nach einem bestimmten Bericht und über die *@DateTo*- und *@DateFrom*-Parameter nach einem festgelegten Zeitraum, was in Zeile 10 zu sehen ist. Dadurch kann genau geprüft werden, ob bei einem konkreten Report eine bestimmte Kombination an Parametern besonders häufig vorkommt.

```
1  SELECT
2      ParameterVarchar,
3      COUNT(c.Path) AS ReportCount
4  FROM ExecutionLogStorage ex
5  JOIN Catalog c ON @ReportUniqueID = c.ItemID
6  CROSS APPLY (
7      SELECT
8          ParameterVarchar = CAST(ex.Parameters AS varchar(MAX))
9  ) Vars
10 WHERE @ReportUniqueID = ex.ReportID AND CAST(ex.TimeStart AS
       DATE) BETWEEN CAST(@DateFrom AS DATE) AND CAST(@DateTo AS
       DATE)
11 GROUP BY c.Path, ex.ReportID, ParameterVarchar
12 ORDER BY ReportCount DESC, ParameterVarchar
```

Listing 14: Ermitteln der Berichtsparameter

Mit diesem Wissen kann abgewägt werden, ob sich die Einrichtung entsprechender Abonnements oder auch Caching-Strategien für diesen Bericht lohnen würden oder ob die Parameterkombinationen zu vielfältig sind, um den Report automatisiert vorzubereiten bzw. zwischenzuspeichern.

3.2.1.5. Beschaffung der Diagrammdaten

Innerhalb des Metareports sollen mehrere Diagramme angezeigt werden, die besonders häufig genutzte Zeitpunkte der Berichtserstellung aufzeigen. Um dabei den Anforderungen aus Abschnitt 3.1.1 zu genügen, werden zwei Diagramme erstellt, die die Generierungszeitpunkte einmal nach der Uhrzeit und einmal nach dem Datum darstellen. Darüber hinaus wird ein weiteres Diagramm kreiert, welches besonders aktive Benutzer aufgeschlüsselt nach Tagen anzeigt. In diesem Abschnitt geht es nur um die entsprechenden Abfragen der zugrunde liegenden Diagrammdaten. Wie aus diesen Daten die Diagrammelemente des Berichtes erstellt werden beschreibt Abschnitt 3.2.2.7 auf Seite 62.

Für das erste Diagramm, das die Anzahl der erstellten Berichte auf die Uhrzeit umlegen soll, wird die Zeiteinteilung in Stunden erfolgen. Das heißt, das für jedes stündliche Intervall die erstellten Berichte ungeachtet vom Datum gezählt werden. Dazu erfolgt die Datenbeschaffung in zwei Schritten. Im ersten Schritt wird mit der SQL-Abfrage aus Listing 15 aus Seite 49 eine Zeittabelle generiert, die alle stündlichen Intervalle eines Tages beinhaltet.

Mit der *WITH*-Anweisung in Zeile 1 wird ein allgemeiner Tabellenausdruck (engl. Common Table Expression) angegeben, welche durch eine entsprechende SQL-Anweisung definiert wird [vgl. MicWI]. Die erste *SELECT*-Anweisung ab Zeile 2 erzeugt zwei Spalten und legt das erste Intervall von 0 bis 1 Uhr fest. In der zweiten *SELECT*-Anweisung, die ab Zeile 6 zu sehen ist, wird den zuvor erstellten beiden Spalten *starttime* und *endtime* durch die *DATEADD*-Methode jeweils eine Stunde dazu addiert. Dies geschieht so lange, wie der

Wert der Spalte *starttime* vor 23 Uhr liegt (angegeben durch die WHERE-Anweisung in Zeile 10). Die *UNION*-Anweisung in Zeile 5 vereint die Ergebnisse von mindestens zwei Abfragen zu einer einzelnen Ergebnismenge, welche alle Zeilen beinhaltet, die zu den Abfragen der Vereinigung gehören [vgl. MicUN]. Dadurch entsteht schließlich die Zeittabelle mit 24 Ergebniszeilen, wobei jede Zeile ein Zeitintervall von einer Stunde beschreibt.

```
1  WITH timetable AS (
2      SELECT
3          CONVERT(TIME, '00:00:00:000', 114) AS starttime,
4          CONVERT(TIME, '01:00:00:000', 114) AS endtime
5      UNION ALL
6      SELECT
7          DATEADD(hour, 1, starttime) AS starttime,
8          DATEADD(hour, 1, endtime) AS endtime
9      FROM timetable
10     WHERE starttime < CONVERT(TIME, '23:00:00:000', 114)
11 )
```

Listing 15: Erstellen der Ergebnismenge mit den Zeitintervallen

Im zweiten Schritt wird mit Hilfe der SQL-Abfrage in Listing 16 die Anzahl der innerhalb der Zeitintervalle generierten Berichte ermittelt.

```
1  SELECT
2      t.starttime,
3      t.endtime,
4      (
5          SELECT COUNT(ex.LogEntryId)
6          FROM ExecutionLogStorage ex
7          WHERE
8              CAST(ex.TimeStart AS TIME) BETWEEN t.starttime AND
                   t.endtime AND
9              CAST(ex.TimeStart AS DATE) BETWEEN CAST(@DateFrom
                   AS DATE) AND CAST(@DateTo AS DATE)
10     ) AS ReportCount
11 FROM timetable t
12 ORDER BY t.starttime
13 OPTION (MAXRECURSION 0)
```

Listing 16: Zählen der innerhalb der Zeitintervalle erstellten Berichte

Mit der äußeren *SELECT*-Anweisung, beginnend ab Zeile 1, werden alle 24 Zeitintervalle der zuvor mit der *WITH*-Anweisung erzeugten Ergebnismenge abgefragt. Die verwendete *COUNT*-Aggregatfunktion in der Unterabfrage von Zeile 4 bis 10 zählt die Log-IDs jedes erstellten Berichtes. Dabei werden aufgrund der *WHERE*-Anweisung in Zeile 8 immer nur die Reports berücksichtigt, deren Erstellungszeit innerhalb des momentan abgefragten Zeitintervalls der äußeren *SELECT*-Anweisung liegt. Mit der *WHERE*-Anweisung in Zeile

9 wird über die beiden Parameter *@DateFrom* und *@DateTo* ausschließlich der momentan ausgewählte Zeitraum in Betracht gezogen.

Die in Abbildung 20 auszugsweise dargestellte Ergebnismenge zeigt in den ersten beiden Tabellenspalten *starttime* und *endtime* die Intervallgrenzen der einzelnen Zeitintervalle der erstellten Zeittabelle. In der letzten Spalte ganz rechts ist die entsprechende Anzahl der abgefragten Berichte innerhalb des jeweiligen Zeitintervalles zu sehen. Die Auswertung der Daten erfolgt in Abschnitt 3.4.

Ergebnisse | Meldungen

	starttime	endtime	ReportCount
1	00:00:00.0000000	01:00:00.0000000	0
2	01:00:00.0000000	02:00:00.0000000	0
3	02:00:00.0000000	03:00:00.0000000	0
4	03:00:00.0000000	04:00:00.0000000	0
5	04:00:00.0000000	05:00:00.0000000	0
6	05:00:00.0000000	06:00:00.0000000	0
7	06:00:00.0000000	07:00:00.0000000	11
8	07:00:00.0000000	08:00:00.0000000	168
9	08:00:00.0000000	09:00:00.0000000	441
10	09:00:00.0000000	10:00:00.0000000	590
11	10:00:00.0000000	11:00:00.0000000	451

Abbildung 20: Beispielhafte Ergebnismenge der erzeugten Berichte innerhalb bestimmter Zeitintervalle

Auch für das zweite Diagramm, in dem die Anzahl der erstellten Reports nach dem Datum und nicht nach der Uhrzeit aufgeschlüsselt werden soll, erfolgt die Abfrage der benötigten Daten in zwei Schritten. Wieder wird im ersten Schritt zuerst eine Ergebnismenge mit der *WITH*-Anweisung erstellt, die dieses Mal allerdings pro Zeile ein Datum und keine Uhrzeit enthält. Deren Erzeugung ist in Listing 17 zu sehen.

```
1 WITH datetable AS (
2     SELECT
3         @DateFrom AS currentdate
4     UNION ALL
5     SELECT
6         DATEADD(day, 1, currentdate) AS currentdate
7     FROM datetable
8     WHERE currentdate < @DateTo
9 )
```

Listing 17: Erzeugen der abzufragenden Datumswerte

Die generierte Tabelle verfügt nur über eine Spalte, die einen Datumswert enthält. Das Startdatum wird in Zeile 3 durch den *@DateFrom*-Parameter festgelegt. Danach wird in

einer weiteren *SELECT*-Anweisung durch den Aufruf der *DATEADD*-Methode in Zeile 6 solange ein weiterer Tag zum Startdatum hinzuaddiert, bis das Enddatum, welches durch den *@DateTo*-Parameter in Zeile 8 definiert ist, erreicht wurde. Alle dadurch erzeugten Datumswerte werden mit dem zurückgegeben Startdatum der ersten *SELECT*-Anweisung durch die *UNION*-Anweisung in Zeile 4 zu einer Ergebnismenge zusammengefügt.

Das eigentliche Zählen der abgerufenen Berichte an den einzelnen Daten erfolgt durch eine weitere SQL-Abfrage, die in Listing 18 gezeigt wird.

```
1  SELECT
2      d.currentdate,
3      (
4          SELECT COUNT(ex.LogEntryId)
5          FROM ExecutionLogStorage ex
6          WHERE CAST(ex.TimeStart AS DATE) = CAST(d.currentdate
               AS DATE)
7      ) AS ReportCount
8  FROM datetable d
9  ORDER BY d.currentdate
10 OPTION (MAXRECURSION 0)
```

Listing 18: Anzahl aller an bestimmten Daten erstellten Berichten ermitteln

Analog zu den zuvor beschriebenen Zeitintervallen ruft auch hier die zweite SQL-Abfrage über die äußere *SELECT*-Anweisung beginnend ab Zeile 1 alle Datumswerte der eben erstellten Tabelle ab und nutzt diese innerhalb der Unterabfrage in den Zeilen 3 bis 7, um die mit der *COUNT*-Aggregatfunktion gezählten Berichte zu filtern. Die Abbildung 21 präsentiert eine mögliche Ergebnismenge.

Ergebnisse | Meldungen

	currentdate	ReportCount
1	2014-04-25	339
2	2014-04-26	9
3	2014-04-27	17
4	2014-04-28	982
5	2014-04-29	592
6	2014-04-30	607
7	2014-05-01	33
8	2014-05-02	500
9	2014-05-03	21
10	2014-05-04	10

Abbildung 21: Beispielhafte Ergebnismenge der an bestimmten Daten kreierten Berichte

In der abgebildeten Tabelle sieht man in der Spalte namens *currentdate* die zuvor erzeugten Daten der über die *WITH*-Anweisung generierten Ergebnismenge, während die

zweite Spalte die Anzahl der an dem entsprechenden Datum erstellten Berichte angibt. Auch hier erfolgt die Auswertung der gewonnenen Daten in Abschnitt 3.4.

Das letzte Diagramm soll für jedes einzelne Datum eines vorgegebenen Zeitintervalls immer nur die drei Benutzer anzeigen, die an dem entsprechenden Tag die meisten Berichte abgerufen haben. Dazu werden zuerst mit allgemeinen Tabellenausdrücken zwei Ergebnismengen erzeugt, die anschließend von einer weiteren SQL-Abfrage ausgewertet werden, um die passenden Benutzer zurückzugeben. Die erste der beiden Mengen ist absolut identisch zur Datentabelle aus Listing 17 auf Seite 50 und enthält jeden einzelnen Tag der näher betrachtet werden soll. Für die Erzeugung der zweiten Tabelle wird die SQL-Abfrage aus Listing 19 verwendet.

```
1  WITH TOPRECORDS AS (
2      SELECT
3          D.currentdate AS Date,
4          ex.UserName,
5          COUNT(ex.ReportID) AS ReportCount,
6          ROW_NUMBER() OVER (
7              PARTITION BY D.currentdate
8              ORDER BY COUNT(ex.ReportID) DESC
9          ) AS RowNo
10     FROM ExecutionLogStorage ex
11     RIGHT JOIN DATETABLE D ON CAST(D.currentdate AS DATE) =
           CAST(ex.TimeStart AS DATE)
12     GROUP BY D.currentdate, ex.UserName
13 )
```

Listing 19: Anzahl angefragter Berichte pro Nutzer an einem bestimmten Tag

Die Abfrage erstellt eine Ergebnismenge, die die Anzahl an Berichten, die von einem einzelnen Nutzer an einem bestimmten Tag erstellt wurden, beinhaltet. Dazu werden die Datensätze der *ExecutionLogStorage*-Tabelle, die unter anderem Daten über die angeforderten Berichte und den dazugehörigen Nutzerdaten enthält, mit der zuerst erstellten Ergebnismenge, die die Datumsangaben der näher betrachteten Tage beinhaltet, verknüpft. Dies geschieht über die *JOIN*-Anweisung in Zeile 11. Um später daraus nur die drei aktivsten Benutzer zu ermitteln, werden die Ergebniszeilen mit Hilfe der *ROW_NUMBER*-Funktion (für weitere Erläuterungen zu dieser Funktion siehe [MicRO]) in Zeile 6, partioniert nach dem aktuellen Datum der *SELECT*-Anweisung und absteigend sortiert nach der Anzahl der abgerufenen Berichte eines bestimmten Benutzers an diesem Datum, fortlaufend durchnummeriert. Aufgrund der Partitionierung nach dem aktuellen Datum in Zeile 7 und der Gruppierung der Ergebnismenge zuerst nach dem Datum und anschließend nach dem Benutzernamen in Zeile 12 erhält somit an jedem verschiedenen Datum der aktivste Benutzer die Zeilennummer 1, der zweitaktivste Benutzer die Zeilennummer 2, usw. Um nun aus dieser Ergebnismenge die drei aktivsten Nutzer auszuwählen, muss man nur nach

deren zugewiesener Zeilennummer in der Spalte *RowNo* filtern, was in der SQL-Abfrage in Listing 20 geschieht.

```
1 SELECT Date, UserName, ReportCount, RowNo
2 FROM TOPRECORDS
3 WHERE RowNo <= 3
```

Listing 20: Ermittlung der drei aktivsten Berichtsnutzer an einem bestimmten Tag

Die Ergebnismenge, die die SQL-Abfrage in Listing 20 zurückgibt, könnte beispielsweise wie in Abbildung 22 aussehen.

Ergebnisse | Meldungen

	Date	UserName	ReportCount	RowNo
1	2014-04-25	Exar Kun	48	1
2	2014-04-25	Commodore Zuggs	20	2
3	2014-04-25	Jan Dodonna	18	3
4	2014-04-26	Oola (1)	9	1
5	2014-04-27	Oola (1)	7	1
6	2014-04-27	Danni Quee	6	2
7	2014-04-27	Davin Felth	4	3

Abbildung 22: Beispielhafte Ergebnismenge für die Abfrage nach den drei aktivsten Benutzern an einem Tag

Dabei enthält die Spalte *Date* das betrachtete Datum, die Spalte *UserName* den Benutzernamen, die Spalte *ReportCount* die Anzahl an von diesem Benutzer an diesem Datum abgerufenen Berichten und die Spalte *RowNo* die von der *ROW_NUMBER*-Funktion zugewiesene fortlaufende Zeilennummer für dieses Datum.

3.2.1.6. Daten über bestehende Abonnements abfragen

Die Metadaten der Abonnements der einzelnen Berichte befinden sich in der *Subscriptions*-Tabelle. Auf die meisten Spalten dieser Tabelle kann wieder über einfache *SELECT*-Anweisungen zugegriffen werden. Interessanter sind die Spalten *MatchData* und *DataSettings*, in denen die Zeitpläne und zusätzlichen Optionen der einzelnen Abonnements im XML-Format hinterlegt sind. Das Listing 21 auf Seite 54 zeigt einen stark gekürzten Auszug der SQL-Abfrage, mit welcher auf die Abonnement-Metadaten zugegriffen wird.

Auf die XML-Daten der *MatchData* und *DataSettings* Spalten ist dieses Mal kein sofortiger Zugriff möglich, da diese innerhalb der *Subscriptions*-Tabelle ungewöhnlicherweise in einem einfachen Textdatenformat gespeichert werden. Deshalb müssen die gespeicherten Daten zuerst in das XML-Datenformat konvertiert werden. Diese Konvertierung erfolgt über die *CAST*-Methode, die beispielhaft für die Daten der *MatchData*-Spalte in Zeile 13

von Listing 21 aufgerufen wird. Anschließend erfolgt der Zugriff auf die umgewandelten XML-Daten, wie bereits in Abschnitt 3.2.1.3 beschrieben, über die value-Methode (siehe Zeile 10). Auch diese SQL-Abfrage liefert als Resultat nur die Abonnementdaten eines bestimmten Berichtes zurück, zu erkennen an der *WHERE*-Anweisung in Zeile 19. Im Gegensatz zu den zuvor erläuterten Abfragen der vorangegangenen Abschnitte wird die Ergebnismenge dieses mal nicht nach einem festgelegten Zeitraum gefiltert, da es bei den Abonnements viel mehr darauf ankommt, ob ein Bericht überhaupt über Abonnements verfügt und um wieviele es sich dabei handelt.

```
1  SELECT
2      s.Description,
3      s.LastRunTime,
4      s.LastStatus,
5      s.ModifiedDate,
6      u.UserName AS ModifiedBy,
7      -- ...
8      (
9          SELECT
10             'StartDateTime: ' + ISNULL(XMLMatch.x.value('(
                   ScheduleDefinition/*:StartDateTime)[1]', '
                   VARCHAR(50)'), '-')
11             -- ...
12         FROM Subscriptions sub1
13         CROSS APPLY (SELECT CAST(sub1.MatchData AS XML)) AS
               XMLMatch(x)
14         WHERE s.SubscriptionID = sub1.SubscriptionID
15     ) AS MatchData,
16     -- ...
17 FROM Subscriptions s
18 JOIN Users u ON u.UserID = s.ModifiedByID
19 WHERE s.Report_OID = @ReportUniqueID
```

Listing 21: Abfrage der Abonnementdaten

Aufgrund der Möglichkeit direkt auf die Zeitpläne und zusätzlichen Optionen zugreifen zu können, kann man diese beliebig aufbereiten. Der Einfachheit halber werden diese Daten durch Listing 21 zeilenweise innerhalb einer einzelnen Ergebnisspalte zurückgegeben.

3.2.2. Design des Metareports

3.2.2.1. Aufbau des Metareports

Der Prototyp des Metareports, der in Abbildung 17 auf Seite 41 gezeigt wurde, stellt die gesammelten Metadaten innerhalb eines einzelnen Berichtes dar. Jedoch ist die Menge an darzustellenden Metadaten sehr groß, wodurch es schwierig ist die Daten in nur einem Bericht übersichtlich zu präsentieren. Deshalb setzt sich der fertige Metareport aus insgesamt vier verschiedenen *.rdl*-Dateien (siehe Abschnitt 2.6.3) zusammen, die über Verknüpfungen

miteinander verbunden sind. Die Namen der einzelnen Dateien und deren Verbindungen untereinander sind in der Übersicht in Abbildung 23 dargestellt.

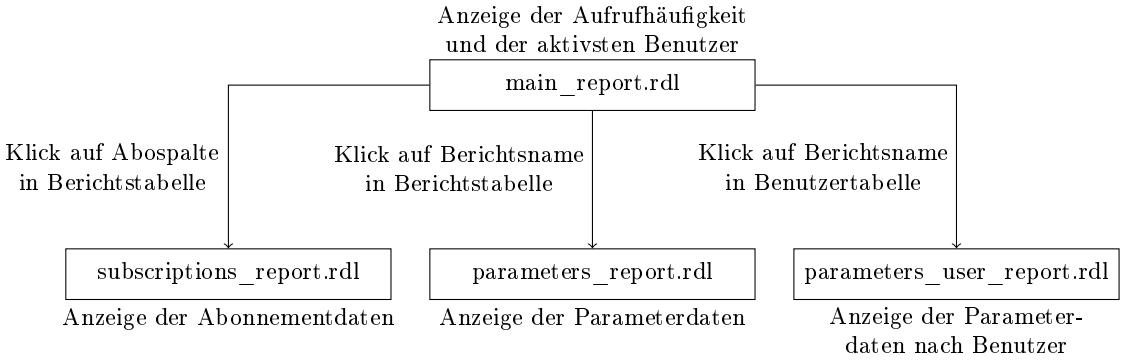


Abbildung 23: Aufbau des Metareports

Die Datei mit dem Namen *main_report.rdl* stellt die Startansicht mit den Diagrammen und Tabellen für die am häufigsten aufgerufenen Berichte und die aktivsten Benutzer dar. In ihr verweisen anklickbare Verknüpfungen auf die übrigen drei *.rdl*-Dateien. Klickt man den Namen eines Reports in der Tabelle mit den am häufigsten aufgerufenen Berichten an, dann gelangt man zur *parameters_report.rdl*-Datei, die die Parameter und dazugehörigen Werte des angeklickten Berichtes in tabellarischer Form abbildet, die dem entsprechenden Report bei seinen Aufrufen übergeben wurden. Wenn man in der Startansicht stattdessen in der Tabelle mit den aktivsten Benutzern einen Berichtsnamen auswählt, wird die *parameters_user_report.rdl*-Datei geöffnet. Diese gibt ebenfalls die Übergabeparameter des gewählten Berichtes aus, allerdings werden die Daten zusätzlich nach dem Nutzer gefiltert, welcher sich in der entsprechenden Zeile der Benutzertabelle befand. Bei der Startansicht ist abschließend noch die Anzahl der Berichtsabonnements der einzelnen Reports verlinkt. Klickt man darauf, dann gelangt man zur *subscriptions_report.rdl*-Datei, die alle Abonnementdaten des dazugehörigen Berichtes auflistet.

In den folgenden Abschnitten werden die eben erwähnten Berichtselemente, die sich innerhalb der vier Beschreibungsdateien der einzelnen Berichte befinden, ausführlich erläutert.

3.2.2.2. Filterung der Daten nach verschiedenen Zeiträumen

Der Nutzer des Metareports hat die Möglichkeit, die Daten nach einem Tag, einer Woche, einem Monat oder zwei Monaten zu filtern. Dazu erhält der Bericht insgesamt drei Parameter: *@LookBackDays* für die Länge des betrachteten Zeitraumes in Tagen, *@DateFrom* für das Startdatum und *@DateTo* für das Enddatum des Zeitraumes. Standardmäßig wird der Parameter *@LookBackDays* auf einen Tag, *@DateTo* auf das heutige Datum und *@DateFrom* auf das Resultat von *@LookBackDays* subtrahiert von *@DateTo* gesetzt.

Abbildung 24: Anpassung des betrachteten Zeitraumes

Während der Betrachtung des Metareports hat der Anwender jederzeit die Möglichkeit, die Länge des Zeitraumes anzupassen und zu verschieben. Zum Verschieben nutzt man die beiden Pfeile, die an Position 1 in Abbildung 24 zu sehen sind. Um die Anzahl der Tage des Zeitraumes zu ändern, klickt man die farbig hervorgehobenen und unterstrichenen Zahlenwerte an Position 2 der Abbildung 24 an. Immer wenn man den Zeitraum auf irgendeine dieser Arten ändert, erfolgt eine Neuberechnung der drei zuvor genannten Parameter über entsprechend hinterlegte Aktionen. Jede dieser Aktionen ruft den Metareport mit den angepassten Parameterwerten erneut auf.

3.2.2.3. Darstellung der Aufrufhäufigkeit von Berichten

Die Auflistung der Anzahl der Aufrufe einzelner Berichte erfolgt innerhalb des Metareports in einem Tabellenelement. Jeder einzelnen Spalte dieser Tabelle wird ein Element der Ergebnismenge aus der (gekürzten) SQL-Abfrage aus Listing 8 von Seite 42 zugewiesen. Ein Auszug des Tabellenelementes ist in Abbildung 25 zu sehen, wobei aus Platzgründen einige Spalten der Tabelle nicht abgebildet sind.

Am häufigsten aufgerufene Berichte:

Bericht	Aufrufe	Live-Aufrufe	Live-Anteil	Durchschnittszeit
/Flaechencontrolling/Flaechencontrolling	2120	1163	54,86%	00:00:22.677
/Artikel-Renner-Penner inkl. Areamanager/ArtikelREnnerPenner	833	448	53,78%	00:00:51.236
/Liste 803 Umsätze pro Tag (neu)/Liste803	165	114	69,09%	00:00:57.36
/Artikel-Renner-Penner inkl. Areamanager (Abonnements)/ArtikelREnnerPenner	142	139	97,89%	00:02:11.312

Abbildung 25: Aufrufhäufigkeit von Berichten

In der Abbildung sind die Spalten *Live-Aufrufe* und *Live-Anteil* zu sehen. Diese geben absolut bzw. prozentual an, wie oft die Quelle der Berichtsausführung *live* war. Damit ist gemeint, wie oft der Bericht beispielsweise nicht aus dem Cache geladen wurde. Weitere mögliche Quellen der Berichtsausführung sind der Auflistung aus [ResMi] zu entnehmen. Da sich die Bedeutung der übrigen dargestellten Tabellenspalten aus dem Namen ableiten lässt und die Zuweisung der Ergebnismengenelemente zu den Tabellenspalten und die darauf folgende Formatierung der Wertausgabe intuitiv erfolgt, wird an dieser Stelle nicht näher darauf eingegangen. Interessanter ist die letzte Tabellenspalte, die die Anzahl der Abonnements ausgibt. Diese ist auszugsweise in Abbildung 26 auf Seite 57 zu sehen.

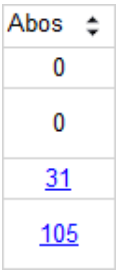


Abbildung 26: Anzahl der abgeschlossenen Berichtsabonnements

Das Besondere an dieser Spalte ist, dass ihr sowohl eine bedingte Formatierung, als auch eine bedingte Klickaktion zugewiesen sind. Dazu werden innerhalb der Textfeldeigenschaften der Tabellenspalte die folgenden Ausdrücke aus Listing 22 verwendet.

```
1 =Iif(Fields!Subs.Value > 0, "Blue", "Black")
2 =Iif(Fields!Subs.Value > 0, "Underline", "Default")
3 =Iif(Fields!Subs.Value > 0, "subscriptions_report", NOTHING)
```

Listing 22: Ausdrücke für die bedingte Formatierung und Klickaktion

Die Farbe des Textes innerhalb der Tabellenzeilen von Abbildung 26 auf Seite 57 wird durch den Ausdruck in Zeile 1 festgelegt. Gemäß der dort angegebenen *if*-Bedingung wird die Textfarbe zu blau, falls mindestens ein Abonnement vorhanden ist, ansonsten wird die Textfarbe zu schwarz. Nach dem gleichen Prinzip funktioniert auch der Ausdruck in der Zeile 2, nur das hier der Text unterstrichen wird, falls Abonnements für diesen Bericht abgeschlossen wurden und ansonsten der Inhalt der Spalte unangetastet bleibt. Weist man einem Textfeld eine „Gehe zu Bericht“-Aktion zu, muss man dieser den Namen des Berichtes übergeben, der bei einem Klick auf das Textfeld aufgerufen werden soll. Da der Aufruf des Abonnementdetailberichtes (siehe Abschnitt 3.2.2.8) bei nicht vorhandenen Abonnements keinen Sinn macht, wird für die Übergabe des Berichtnamens der Ausdruck aus Zeile 3 verwendet. Dieser weist der Klickaktion nur dann einen gültigen Berichtsnamen zu, wenn überhaupt Abonnements abgeschlossen wurden für den Bericht. Ist dies nicht der Fall, wird als Name stattdessen *NOTHING* übergeben, was dazu führt, dass das Textfeld keine Klickaktion mehr besitzt.

3.2.2.4. Anzeige der aktivsten Berichtsnutzer

Für die Darstellung der aktivsten Anwender innerhalb des Metareports ist es aufgrund des verwendeten *ROLLUP*-Operators (siehe Abschnitt 3.2.1.2) nötig, die Ergebnismenge der SQL-Abfrage aus Listing 9 auf Seite 44 nachzubearbeiten. Um die erhaltenen Daten übersichtlich zu präsentieren, wird ein Tabellenelement in den Metareport eingebettet, welchem einzelne Spalten vom Abfrageergebnis zugewiesen werden. Da die Ergebnismenge

aufgrund der durch den *ROLLUP*-Operator ergänzten Zwischenergebnisse einige Benutzernamen mehrfach enthält, muss für eine angemessene Darstellung der Daten eine zusätzliche Gruppierung nach Benutzernamen erfolgen. Dazu wird dem Tabellenelement zur standardmäßig vorhandenen Detailgruppe eine übergeordnete Zeilengruppe hinzugefügt, in deren Eigenschaften diese Gruppierung konfiguriert wird (siehe Abbildung 27).

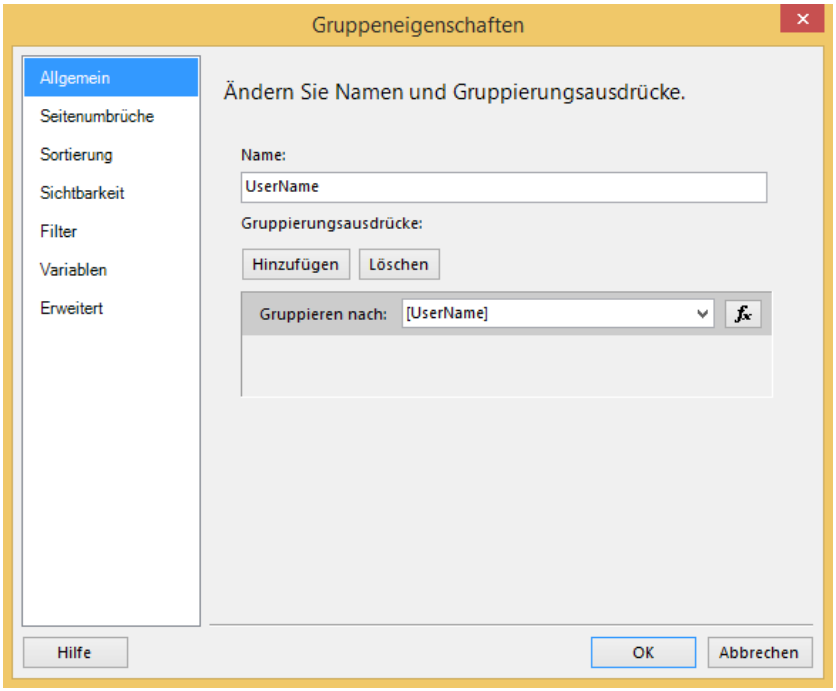


Abbildung 27: Konfiguration der Gruppeneigenschaften der Zeilengruppe

Ein weiteres Problem, das durch die Verwendung des *ROLLUP*-Operators hinzukommt, sind die Spalten mit *NULL*-Werten, die durch die Zwischenergebniszeilen entstehen. Damit diese *NULL*-Werte bei der geforderten Auflistung der einzelnen, durch den Anwender aufgerufenen Berichte nicht mit angezeigt werden, müssen diese manuell gefiltert werden. Dazu wird der zuvor bereits erwähnten Detailgruppe des Tabellenelementes eine Filterung hinzugefügt. Dies geschieht, genau wie zuvor bei der übergeordneten Zeilengruppe, über die Gruppeneigenschaften der Detailgruppe (siehe Abbildung 28 auf Seite 59).

Im Ausdrucksfeld der in Abbildung 28 auf Seite 59 dargestellten Gruppeneigenschaften ist der Code aus dem Listing 23 hinterlegt, der die Verzeichnispfade und die Berichts-IDs auf *NULL*-Werte überprüft, und einen entsprechenden booleschen Wert zurückliefert.

```
1 =(Fields!Path.Value IS NOTHING) OR (Fields!ReportID.Value IS
      NOTHING)
```

Listing 23: Prüfen der Daten auf *NULL*-Werte

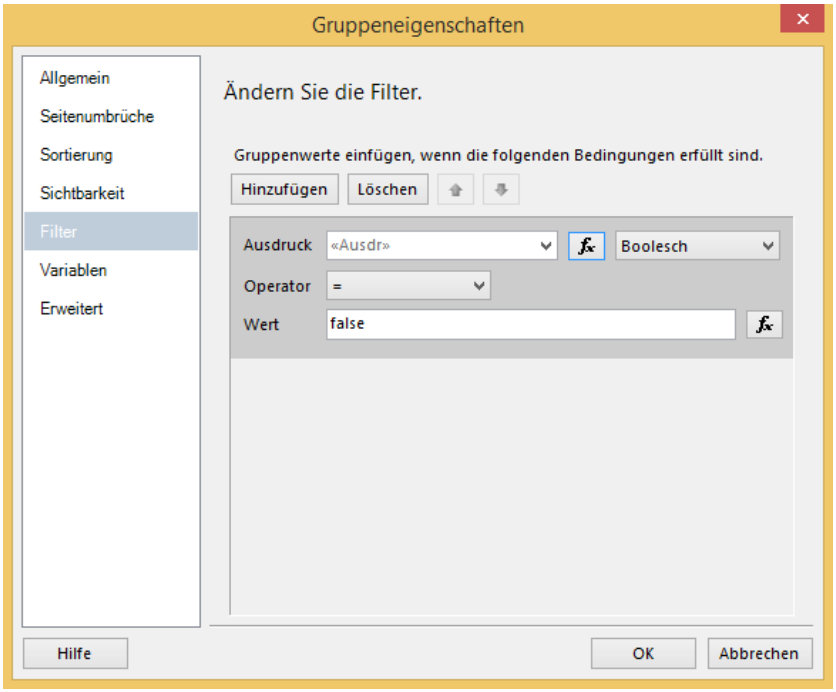


Abbildung 28: Konfiguration der Gruppeneigenschaften der Detailgruppe

Der Filter der Detailgruppe sagt kurz zusammengefasst also aus: „Wenn entweder der Verzeichnispfad oder die Berichts-ID oder beide Felder einen *NULL*-Wert haben, dann filtere diese Ergebniszeile heraus und zeige sie nicht an.“ Dadurch wird eine lückenlose Auflistung aller abgerufenen Berichte eines Benutzers möglich, so wie sie beispielhaft in der vorletzten Spalte der Tabelle in Abbildung 29 dargestellt ist.

Aktivste Benutzer:

Benutzer	Durchschnittszeit	Gesamtzeit	Aufgerufene Berichte	Aufrufe
Oola (1)	00:01:28.939	05:02:23.717	⊞	204
Oppo Rancisis (1)	00:00:34.782	01:27:32.138	⊞	151
Exar Kun	00:00:43.055	01:06:44.167	⊞	93
Jan Dodonna	00:00:39.871	00:45:51.129	⊟ /Flaechencontrolling/Flaechencontrolling	50
			/Liste 803 Umsätze pro Tag (neu)/Liste803	19
Tahiri Veila	00:00:25.746	00:28:45.045	⊞	67
Commodore Zuggs	00:00:26.158	00:24:51.047	⊞	57

Abbildung 29: Anzeige der aktivsten Benutzer im Metareport

Konkret zeigt die beispielhafte Ergebnistabelle in Abbildung 29 den Benutzernamen des Aufrufers, die durchschnittliche und insgesamt benötigte Erstellungszeit aller aufgerufenen Berichte, die bereits erwähnte Auflistung der abgerufenen Berichte und in der letzten Spalte die Anzahl der Aufrufe insgesamt bzw. für jeden einzelnen Bericht.

3.2.2.5. Ausgabe der beanspruchten Ressourcen

Die Spalten, die die während der Berichtserstellung beanspruchten Zeit- und Speicherressourcen angeben, sind mit Bestandteil der bereits in Abschnitt 3.2.2.3 erläuterten Tabelle. In Abbildung 30 ist jede Spalte, die den Ressourcenverbrauch betrifft, mit Beispieldaten abgebildet.

Durchschnittszeit	Gesamtzeit	Datenzeit	Rechenzeit	Renderzeit	Avg-Pagination	Avg-Processing
00:00:22.677	13:21:16.379	12:28:14.88	00:14:41.096	00:38:20.403	423 KB	6323 KB
00:00:51.236	11:51:20.077	11:39:40.273	00:02:07.297	00:09:32.507	180 KB	175 KB
00:00:57.36	02:37:44.496	02:32:04.033	00:04:12.072	00:01:28.391	1132 KB	716 KB
00:02:11.312	05:10:46.341	05:06:36.113	00:00:41.801	00:03:28.427	463 KB	280 KB

Abbildung 30: Darstellung der aufgewendeten Systemressourcen

Alle innerhalb der Tabelle eingebundenen Ressourcenangaben weisen entweder einen Gesamt- oder Durchschnittsverbrauch aus. Dies liegt daran, dass jede Tabellenzeile die gesamten Aufrufe eines Berichts repräsentiert. Somit gibt beispielsweise die Spalte *Gesamtzeit* die Summe der einzelnen verbrauchten Zeitressourcen aller Aufrufe des zugehörigen Berichtes an.

3.2.2.6. Auflistung übergebener Berichtsparameter

Die Anzeige aller Parameter, die an einen bestimmten Bericht während des Aufrufes übergeben wurden, erfolgt in einem gesonderten Report. Um diesen abzurufen, sind die Werte der ersten Spalte, der bereits in Abbildung 25 auf Seite 56 gezeigten Tabelle, mit einer entsprechenden „Gehe zu Bericht“-Aktion belegt. Klickt man also auf einen der dort blau und unterstrichen hervorgehobenen Berichtspfade, so wird daraufhin der entsprechende Detailreport für diesen Bericht geladen.

Der Detailreport enthält ein Tabellenelement, in dessen Zeilen alle Berichtsparameter eines Aufrufes gesammelt sind. Da Berichte erfahrungsgemäß häufig mit vielen verschiedenen Parametern aufgerufen werden, sind die einzelnen Berichtsparameter in den Zeilen der Tabelle nicht sofort sichtbar. Um die Aufrufparameter anzuzeigen, klappt man einfach die entsprechende Zeile der Tabelle auf, indem man auf das +-Symbol zu Beginn der Zeile klickt. In Abbildung 31 auf Seite 61 ist ein Teil dieser Tabelle mit einer aufgeklappten Zeile zu sehen.

Wie man an dem Beispiel in Abbildung 31 auf Seite 61 gut erkennen kann, ist die Parametervielfalt üppig und die Ausgabe dementsprechend recht unübersichtlich, da die Parameterzeichenkette genauso ausgegeben wird, wie sie in der *ExecutionLogStorage*-Tabelle zu finden ist.

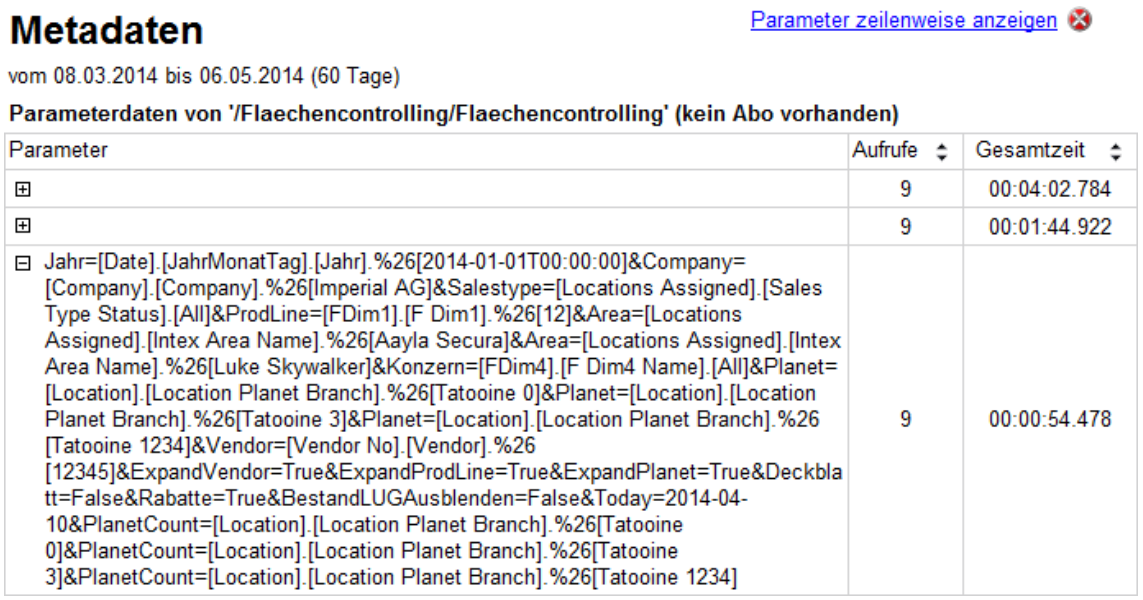


Abbildung 31: Tabelle mit den Berichtsparametern

Aus diesem Grund befindet sich über der Tabelle ein anklickbares Textfeld, mit dem die Darstellung der Parameterauflistung zwischen einer zeilenbasierten und einer nicht zeilenbasierten Ausgabe umgeschaltet werden kann. Die Abbildung 32 zeigt die Darstellungen der beiden möglichen Zustände innerhalb des Textfeldes an, an denen der Nutzer auch erkennen kann, welche Ansicht momentan verwendet wird.

(a) Aktiviert (b) Deaktiviert

Abbildung 32: Mögliche Zustände des Textfeldes zur Ansichtsumschaltung

Die folgende Abbildung 33 auf Seite 62 zeigt das zuvor bereits präsentierte Beispiel noch einmal, dieses Mal allerdings in der zeilenbasierten Ansicht. Das bedeutet also, das nun jedes einzelne Name-Wert-Parameterpaar in einer eigenen Textzeile steht. Erreicht wird diese Anzeige der Daten, indem die zuvor in Abbildung 31 zu sehenden Ampersandzeichen, die die einzelnen Kombinationen aus Parametername und -wert trennen, durch Zeilenumbrüche ersetzt werden. Diese Ersetzung erfolgt direkt beim Aufruf des Berichtes mit der Berichtsparametertabelle. Dazu wird ein Berichtsparameter ausgewertet, der beim Klicken auf das in Abbildung 32 gezeigte Textfeld auf einen entsprechenden booleschen Wert gesetzt wird. Beim ersten Aufruf des Reports ist dieser Paramater standardmäßig mit *FALSE* vorbelegt, wodurch keine Ersetzung der Ampersandzeichen durch Zeilenumbrüche erfolgt. Erst durch einen Klick auf das zuvor angesprochene Textfeld wird der Parameter auf *TRUE* gesetzt, der Report neu aufgerufen und dadurch die Ersetzung der Zeichen angestoßen.

Metadaten

Parameter zeilenweise anzeigen

vom 08.03.2014 bis 06.05.2014 (60 Tage)

Parameterdaten von '/Flaechencontrolling/Flaechencontrolling' (kein Abo vorhanden)

Parameter	Aufrufe	Gesamtzeit
⊞	9	00:04:02.784
⊞	9	00:01:44.922
⊟ Jahr=[Date].[JahrMonatTag].[Jahr].%26[2014-01-01T00:00:00] Company=[Company].[Company].%26[Imperial AG] Salestype=[Locations Assigned].[Sales Type Status].[All] ProdLine=[FDim1].[F Dim1].%26[12] Area=[Locations Assigned].[Intex Area Name].%26[Aayla Secura] Area=[Locations Assigned].[Intex Area Name].%26[Luke Skywalker] Konzern=[FDim4].[F Dim4 Name].[All] Planet=[Location].[Location Planet Branch].%26[Tatooine 0] Planet=[Location].[Location Planet Branch].%26[Tatooine 3] Planet=[Location].[Location Planet Branch].%26[Tatooine 1234] Vendor=[Vendor No].[Vendor].%26[12345] ExpandVendor=True ExpandProdLine=True ExpandPlanet=True Deckblatt=False Rabatte=True BestandLUGAusblenden=False Today=2014-04-10 PlanetCount=[Location].[Location Planet Branch].%26[Tatooine 0] PlanetCount=[Location].[Location Planet Branch].%26[Tatooine 3] PlanetCount=[Location].[Location Planet Branch].%26[Tatooine 1234]	9	00:00:54.478

Abbildung 33: Tabelle mit den Berichtsparametern in der zeilenbasierten Ansicht

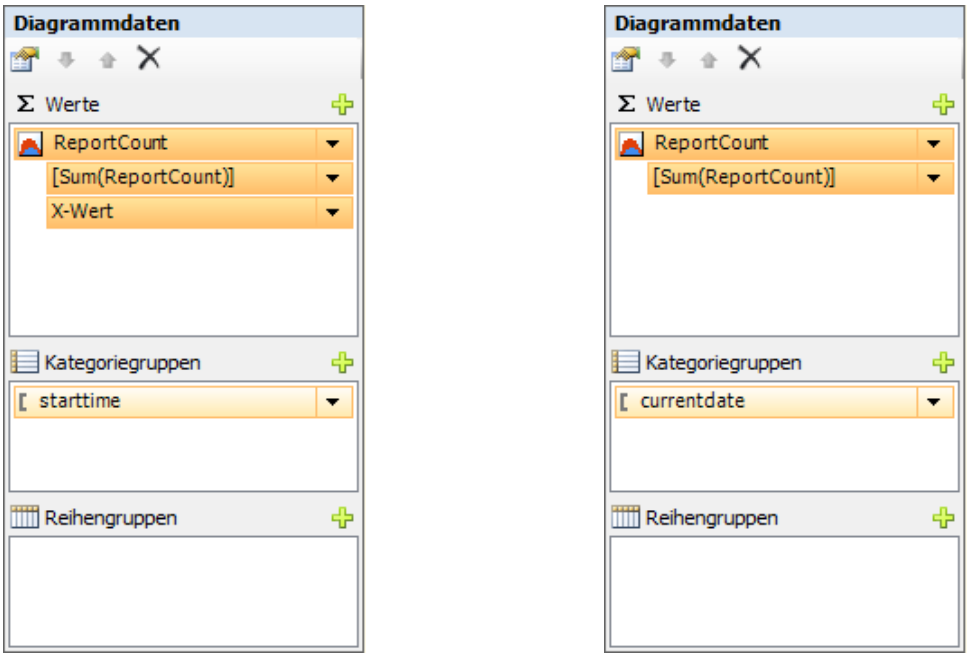
Mit dieser alternativen Darstellung wird eine manuelle Auswertung der Parameterdaten aufgrund der erhöhten Übersichtlichkeit deutlich erleichtert.

3.2.2.7. Erstellen der Diagrammelemente

Innerhalb des Metareports werden die Daten sowohl über die Aufrufhäufigkeit der Berichte nach Uhrzeit und nach Datum, als auch die Daten über die aktivsten Benutzer, neben der tabellarischen Auflistung, auch in Form von Diagrammen wiedergegeben. Für die Befüllung der dafür verwendeten Diagrammelemente braucht es nur wenige Daten, deren Beschaffung bereits in Abschnitt 3.2.1.6 ausführlich beschrieben wurde. Alle anzuzeigenden Daten werden den einzelnen Diagrammen über einen dazugehörigen Konfigurationsdialog intuitiv zugewiesen. Die beiden Einstellungsdialoge für die Aufrufhäufigkeitsdiagramme sind in Abbildung 34 auf Seite 63 zu sehen.

Für die auf der X-Achse abzutragenden Werte werden im Dialogfenster unter *Kategoriegruppen* die entsprechenden Elemente der Ergebnismenge ausgewählt, wobei in diesen beiden Fällen *starttime* für die Uhrzeit bzw. *currentdate* für das Datum genommen wird. Unter Σ *Werte* werden die auf der Y-Achse abzutragenden Werte angegeben. Hierbei handelt es sich bei beiden Diagrammkategorien um die Anzahl der aufgerufenen Berichte. Nur bei dem Diagramm, das sich nach der Uhrzeit richtet, wird zusätzlich angegeben, dass es

sich bei den darzustellenden Aufrufzahlen um skalare Werte handelt.



(a) Nach Uhrzeit (b) Nach Datum

Abbildung 34: Konfigurationseinstellungen der Diagramme nach entsprechender Kategorie

Werden die Diagramme korrekt konfiguriert und als Diagrammtyp „Geglättete Fläche“ ausgewählt, erhält man eine Ansicht, die der in Abbildung 35 ähnelt.

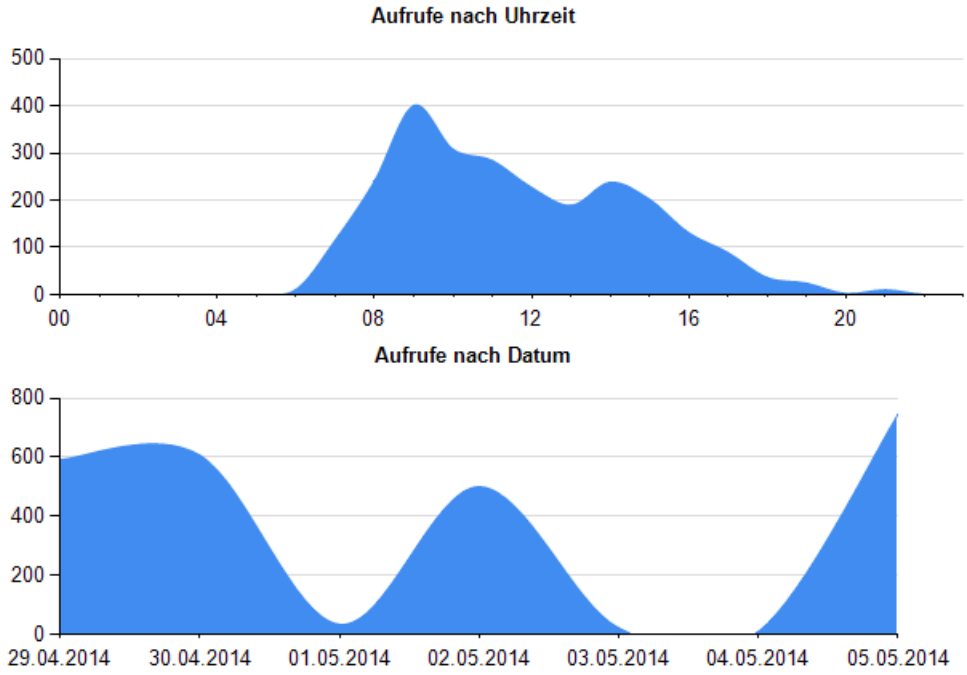


Abbildung 35: Diagramme für die Aufrufe nach Uhrzeit und Datum

Auch für das Diagramm, das die gesammelten Daten über die aktivsten Benutzer pro Tag präsentiert, müssen die anzuzeigenden Werte im entsprechenden Dialogfenster eingerichtet werden. Abbildung 36 zeigt dieses Dialogfenster mit den getätigten Einstellungen.

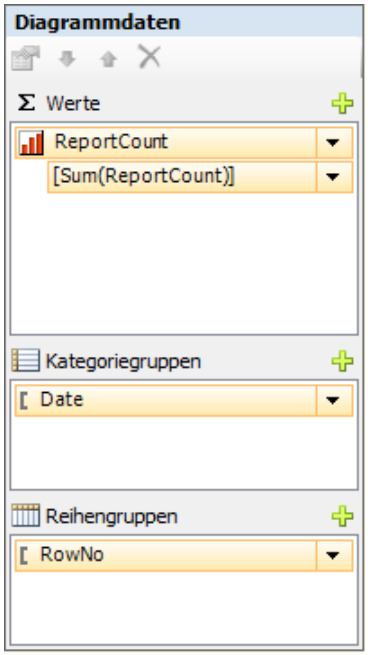


Abbildung 36: Konfigurationsdialog für das Diagramm zur Anzeige der aktivsten Benutzer

Dort findet sich unter *Kategoriegruppen* die Ergebnismengenspalte *Date*, die dafür sorgt, dass entlang der X-Achse des Diagrammes die betrachten Daten abgetragen werden. Für das Feld Σ *Werte* wird die Anzahl an abgefragten Berichten pro Nutzer an einem bestimmten Datum angegeben, womit dieser Wert im Diagrammelement auf der Y-Achse abgetragen wird. Damit pro Datum immer die drei aktivsten Anwender angezeigt werden, müssen innerhalb der Diagrammfläche Kategorien eingeführt werden. Für diese Kategorien wird die den Benutzern zugewiesene fortlaufende Zeilennummer pro Datum genutzt. Diese befindet sich in der Ergebnismengenspalte *RowNo* und wird im Konfigurationsdialog unter *Reihengruppen* eingetragen. Mit dieser Einstellung spiegelt Zeilennummer 1 die Kategorie für den aktivsten, Zeilennummer 2 die Kategorie für den zweitaktivsten und Zeilennummer 3 die Kategorie für den drittaktivsten Anwender wieder.

Aufgrund der verschiedenen Kategorien, die die Aktivität der einzelnen Nutzer abbilden, eignet sich an dieser Stelle der Diagrammtyp „Säule“. Mit den zuvor erläuterten Diagrammeinstellungen lässt sich das in Abbildung 37 auf Seite 65 gezeigte Diagramm erzeugen. An jeder einzelnen Säule steht der dazugehörige Benutzername und die drei verschiedenen Blautöne der einzelnen Säulen spiegeln die drei unterschiedlichen und zuvor erläuterten Kategorien wider.

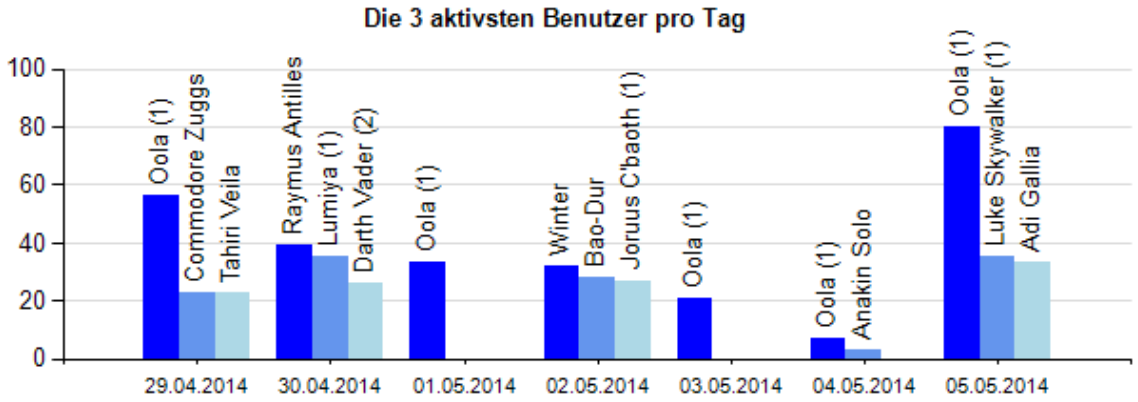


Abbildung 37: Diagramm für die Anzeige der aktivsten Berichtsbenutzer

Durch das Säulendiagramm lässt sich die Aktivität einzelner Anwender sehr schnell erfassen.

3.2.2.8. Abrufen der Abonnementdetails

Auch für die Präsentation der Abonnementmetadaten wird ein einfaches Tabellenelement verwendet. Da die in Listing 21 auf Seite 54 gezeigte SQL-Abfrage zur Beschaffung der Abonnementdaten bereits so aussagekräftige Resultate liefert, ist eine nachträgliche Bearbeitung der Daten nicht nötig, weshalb die Tabelle mit den Details zu den Abonnenemts keine außergewöhnlichen Features beinhaltet. Sie listet schlichtweg die abgefragte Information auf und wird hier deswegen nicht ausführlicher erläutert. Einen Auszug der Tabelle zeigt die Abbildung 38 auf Seite 65.

Abonnementdaten von '/Liste 803 Umsätze pro Tag (neu)/Liste803'

Beschreibung	Zuletzt ausgeführt	Letzter Status	Zuletzt bearbeitet
E-Mail senden an 'vader@todesstern.de'	05.05.2014 11:48:09	Mail sent to vader@todesstern.de	25.02.2014 09:52:10
E-Mail senden an 'vader@todesstern.de'	05.05.2014 11:36:05	Mail sent to vader@todesstern.de	24.02.2014 10:24:16
E-Mail senden an 'vader@todesstern.de'	03.05.2014 16:55:02	Mail sent to vader@todesstern.de	28.03.2014 10:57:35

Abbildung 38: Details abgeschlossener Berichtsabonnements

Von links nach rechts zeigen die Spalten hierbei eine Beschreibung des abgeschlossenen Abonnements, den letzten Ausführungszeitpunkt, Status und Bearbeitungszeitpunkt an.

3.3. Einbindung in Microsofts Softwarelösungen

Erstellte Berichte können dank der SSRS über Microsofts SQL-Server den Anwendern bereitgestellt werden. Alle Benutzer mit den entsprechenden Berechtigungen können diese Berichte anschließend abrufen. Der erstellte Metareport weist allerdings viele Daten aus,

die für den gewöhnlichen Anwender oft nicht von großem Interesse sind und eher von Systemadministratoren oder eventuell auch Berichtserstellern genutzt werden sollen. Aus diesem Grund soll der Metareport nur in Werkzeugen zur Verfügung stehen, die häufig von den Verwaltern verwendet werden. Eines dieser Tools ist das *SQL-Server 2012 Management Studio* von Microsoft, welches schon in Abschnitt 3.1.2.1 zur Analyse des Datenmodells verwendet wurde. Diese Software stellt unter anderem von Microsoft entworfene Standardberichte bereit, die typische Serverinformationen aufbereiten und ausgeben. Ab *SQL-Server 2005 Service Pack 2* hat man die Möglichkeit, selbst erstellte Berichte innerhalb des Management Studios zu verwenden. Dazu wird die Beschreibungsdatei des Berichtes (siehe Abschnitt 2.6.3) benötigt. [vgl. MicMS]

Nachdem man sich mit dem Management Studio zu einer entsprechenden SQL-Serverinstanz verbunden hat (siehe Abschnitt 3.1.2.1), wird innerhalb der Software der Objekt-Explorer zur Ausführung eines eigenen Berichtes verwendet. Dazu wird per Rechtsklick auf die Datenbank, auf die der Bericht ausgeführt werden soll, der benutzerdefinierte Bericht ausgewählt. Die Abbildung 39 zeigt dies beispielhaft an einer Testdatenbank.

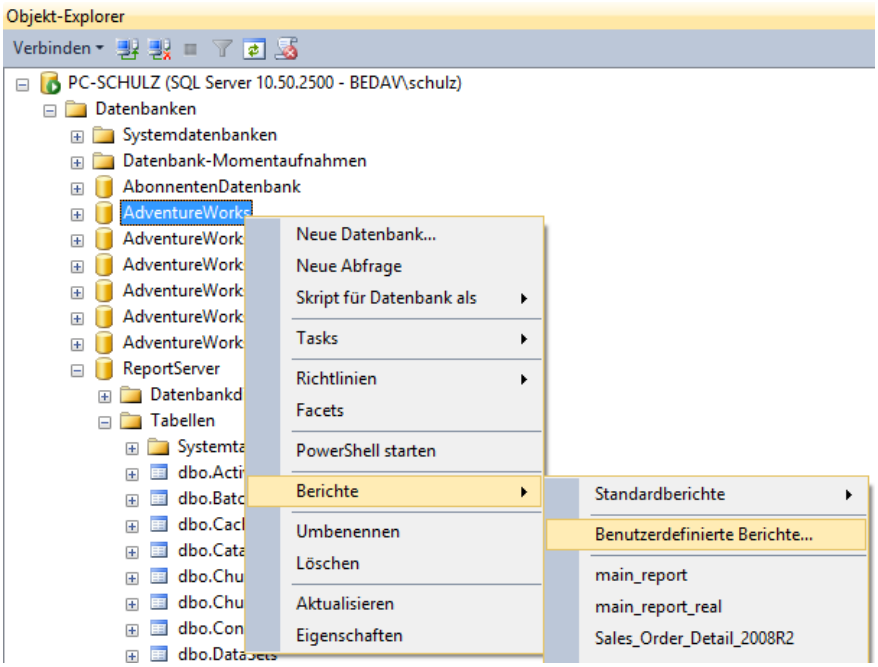


Abbildung 39: Ausführen eines benutzerdefinierten Berichtes

Als Besonderheit werden dem Bericht zu Beginn der Ausführung automatisch bestimmte Parameter übergeben, die unter anderem dazu verwendet werden können, sich direkt zu der Datenbank zu verbinden, die innerhalb des Objekt-Explorers ausgewählt wurde. Für den Metareport ist dies äußerst hilfreich, da der Bericht somit gezielt und ohne viel Aufwand die Metadaten einer bestimmten Datenbank auswerten und anzeigen kann. War die Ausführung erfolgreich, wird der Bericht direkt im Management Studio mit den passenden

Daten angezeigt (siehe Abbildung 45 im Anhang auf Seite 80).

Auch wenn sich für den Metareport die Einbindung in das Management Studio als gute Lösung angeboten hat, gibt es dennoch einige Einschränkungen für benutzerdefinierte Berichte. So werden beispielsweise keine Unterberichte innerhalb eines solchen Reports unterstützt. Auch werden die in den SQL-Abfragen des Berichtes verwendeten Parameter ungefiltert weitergereicht. Dies stellt ein Sicherheitsrisiko dar, da somit die Möglichkeit besteht, Schadcode in die Abfragen einzufügen. Eine vollständige Auflistung aller Einschränkungen von benutzerdefinierten Berichten ist der Quelle [MicMS] zu entnehmen.

3.4. Auswertung

Um mit dem fertigestellten Metareport eine erste Auswertung von Daten durchführen zu können, wurden während des getätigten Praktikums umfangreiche Testdaten bereitgestellt. Für diesen Testdurchlauf wurde ein Zeitraum von 60 Tagen eingestellt, dessen Startdatum der 08.03.2014 und Enddatum der 06.05.2014 war. Alle folgenden Abbildungen in diesem Abschnitt stellen aufgrund des zu großen Umfangs der im Metareport ausgegebenen Daten nur auszugsweise die aufbereiteten Testdaten für dieses 60-tägige Zeitintervall dar.

Als erstes ist im Metareport das Diagramm mit den Aufrufen nach der Uhrzeit zu sehen. Die Abbildung 40 zeigt dieses Diagramm mit Testdaten.

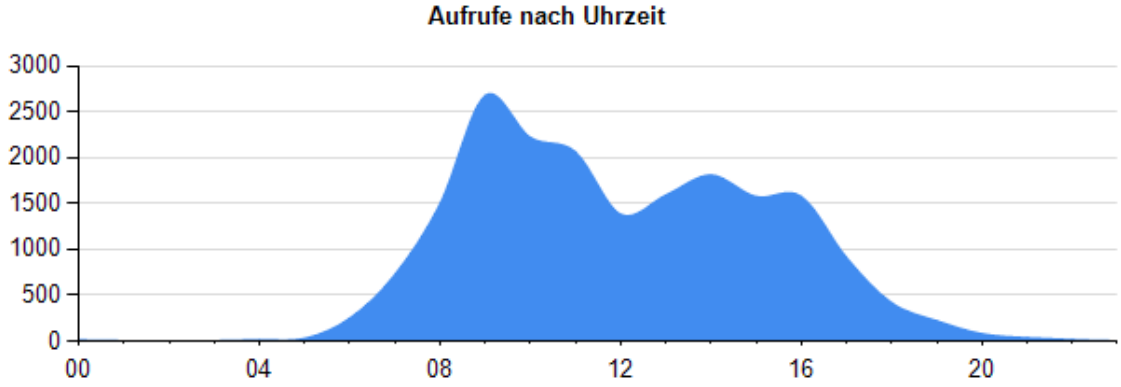


Abbildung 40: Die Aufrufe der Testdatenberichte nach der Uhrzeit

Wie in der Abbildung zu erkennen ist, werden im Zeitraum von 9 bis 10 Uhr mit großen Abstand die meisten Berichte aufgerufen. Auch in den Intervallen von 10 bis 11 Uhr und 13 bis 16 Uhr ist ein starker Abruf von Reports auszumachen. Nach 16 Uhr nehmen die Aufrufe rapide ab und zwischen 21 und 5 Uhr werden im Falle der hier betrachteten Testdaten praktisch keine Berichte mehr abgerufen. In dem Diagramm spiegeln sich somit die Arbeitszeiten des Unternehmens wieder, von dem die verwendeten Testdaten stammen.

Das Diagramm gibt den Betrachter einen schnellen Überblick über die Aufrufhäufigkeit der Berichte über den Tag verteilt und über die damit einhergehende Belastung des SQL-Servers. Die im Diagramm angezeigten Daten können beispielsweise dazu verwendet wer-

den, geeignete Zeitpunkte für zeitlich gesteuerte Abonnements zu finden. Konkret würde es sich hier anbieten, mehr Abonnements im Zeitraum von 21 bis 5 Uhr auszuführen, da dadurch der SQL-Server nicht zu seinen Stoßzeiten zusätzlich belastet werden würde.

Das zweite Diagramm im Metareport stellt die Aufrufe von Berichten an bestimmten Daten dar bzw. wie viele Berichtsaufrufe an einem konkreten Datum stattfanden. Die Abbildung 41 zeigt das aus den verwendeten Testdaten resultierende Diagramm.

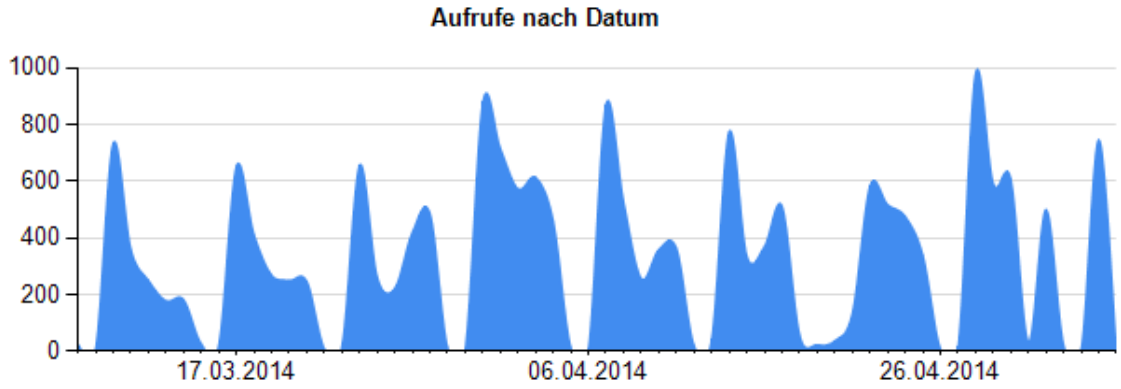


Abbildung 41: Die Aufrufe der Testdatenberichte nach dem Datum

Bereits auf den ersten Blick lässt sich in dem gezeigten Diagramm ein Muster vermuten und tatsächlich ist es so, dass ab der ersten Spitze am 10.03.2014 immer 7 Tage bis zur nächsten Spitze vergehen. Jeder dieser Tage ist ein Montag. Der einzige Montag, an dem nicht außergewöhnlich viele Berichte aufgerufen wurden, ist der 21.04.2014. An diesem Datum war der Ostermontag und damit ein Feiertag. Auch jedes andere Datum an dem besonders wenige Berichte abgerufen wurden fällt auf einen Samstag, Sonntag oder einen Feiertag. Dies zeigt auch hier, dass die Arbeitszeiten des Unternehmens einen sehr starken Einfluss auf die Anzahl der Berichtsaufrufe haben.

Die in dem Diagramm in Abbildung 41 zu sehenden Daten könnten beispielsweise zur Planung von administrativen Aufgaben genutzt werden. Es lässt sich an der Darstellung leicht ausmachen, dass an den Wochenenden nahezu keine Berichte aufgerufen werden. Diese Tage könnten vom Systemadministrator zum Beispiel zur Durchführung von größeren Wartungsarbeiten verwendet werden.

Im Metareport folgt nach dem zweiten Diagramm die Auflistung der Daten über die am häufigsten aufgerufenen Berichte. Die folgende Abbildung 42 auf Seite 69 zeigt aus Platzgründen nur einen Teil vom Ergebnis der Testdatenauswertung.

In der Abbildung sind auszugsweise einige Daten der drei am meisten abgerufenen Berichte angezeigt. Dabei fallen zwei Berichte besonders auf, da sie mit ihren mehreren Tausend Aufrufen deutlich öfter abgefragt wurden, als alle anderen Berichte. Außerdem sind in der Abbildung zwei weitere Spalten zu sehen, die angeben, wie oft die Quelle der Berichtsausführung *live* (siehe Abschnitt 3.2.2.3) war.

Aufrufe	Live-Aufrufe	Live-Anteil
9365	5318	56,79%
6329	3415	53,96%
944	731	77,44%

Abbildung 42: Testdaten der am häufigsten aufgerufenen Berichte

Mit den präsentierten Daten können ständig abgerufene Reports erkannt und dazu passende Caching-Strategien in Erwägung gezogen werden. Damit würden diese Berichte seltener komplett neu erstellt, sondern stattdessen häufiger aus zuvor zwischengespeicherten Kopien erzeugt werden. Diese verbrauchen im Allgemeinen weniger Systemressourcen und können die Serverlast reduzieren.

Nach der ersten Tabelle folgt im Metareport die Tabelle mit den Daten über die aktivsten Benutzer. In der Abbildung 43 ist ein kleiner Teil dieser Tabelle zu sehen.

Aufrufe
1879
731
390

Abbildung 43: Testdaten der aktivsten Benutzer

Die Aufrufzahlen, die in Abbildung 43 dargestellt sind, deuten an, das ein Benutzer besonders aktiv ist. Allerdings handelt es sich dabei um ein Benutzerkonto, welches automatisiert Berichte zum Beispiel für zeitlich gesteuerte Abonnements aufruft. Ansonsten lies sich aus den aufbereiteten Anwenderdaten keine weitere Besonderheit herauslesen.

In der Tabelle ist zusätzlich aufgelistet, welche Berichte ein Anwender aufgerufen hat. Durch die Auswertung dieser Daten kann man auf Benutzer stoßen, die bestimmte Berichte außergewöhnlich oft abfragen. An dieser Stelle könnte es sich für den Berichtsnutzer als komfortabel erweisen Abonnements einzurichten, die diese Berichte für ihn automatisiert zu bestimmten Zeitpunkten erzeugen würden.

Wie bereits im Abschnitt 3.2.2.1 beschrieben wurde, besteht der Metareport aus insgesamt vier unterschiedlichen Dateien. Eine dieser Dateien zeigt die übergebenen Parameter eines bestimmten Berichtes an. Um die dort aufgelisteten Daten als abschließendes Beispiel auszuwerten, wird in der Tabelle mit den Aufrufzahlen der einzelnen Reports auf den Berichtsnamen geklickt, der bei den Testdaten am meisten abgerufen wurde. Ein Teil der daraufhin angezeigten Tabelle ist in Abbildung 44 auf Seite 70 dargestellt.

Dabei handelt es sich um die genaue Anzahl an Berichtsabfragen mit allen verschiedenen Parameter und deren dazugehörigen Werten, die beim Aufruf an den Bericht übergeben worden. Vergleicht man diese Daten mit dem immens hohen Aufrufwert aus Abbildung

42 auf Seite 69, lässt sich schnell bestätigen, dass dieser Bericht nahezu immer mit einer anderen Parameter-Wert-Kombination angefragt wird.

Aufrufe
9
9
9
9
8
7

Abbildung 44: Die Berichtsparameter des am meisten abgerufenen Berichtes

Aufgrund der geringen Anzahl an gleichen Parameter-Wert-Kombinationen lässt sich für diesen Bericht beispielsweise keine sinnvolle Caching-Strategie umsetzen, da für jede unterschiedliche Kombination aus Parameter-Wert-Paaren eine neue Kopie des Berichtes zwischengespeichert werden müsste.

4. Fazit

4.1. Zusammenfassung

Die Aufgabenstellung dieser Masterarbeit war die Erstellung eines Werkzeugs zur „Visualisierung der Report Server Utilisation“. Das Werkzeug sollte also in der Lage sein, die Nutzung des Report Servers aussagekräftig darzustellen. Mit diesen Metadaten könnten Berichtsverwalter auf vorhandene Defizite bei der Verwendung der SSRS aufmerksam gemacht werden und entsprechende Optimierungsstrategien umsetzen, um die Nutzererfahrung für die Berichtsanwender zu verbessern.

Dazu wurde zu Beginn das gegebene Datenmodell auf alle enthaltenen Metadaten hin untersucht, die das Nutzungsverhalten der einzelnen Berichtsanwender Protokollieren. Die Aufrufhäufigkeit von Reports und die übergebenen Berichtsparameter mit deren dazugehörigen Werten sind nur eine kleine Auswahl an Beispieldaten, die bei der Datenmodellanalyse gefunden wurden. Während der Untersuchung wurden jedoch auch Makel am Modell aufgezeigt, deren Vorhandensein aufgrund der fehlenden Dokumentation allerdings nur vermutet werden konnte. Es wurde deshalb ein überarbeitetes Datenmodell präsentiert, dass die Zusammenhänge der erfassten Metadaten besser aufzeigen soll.

Im Anschluss an die Datenmodelluntersuchung wurde mit Hilfe des Mock-Up-Tools *Pencil* ein erster Entwurf des Reports erstellt, der am Ende die aufbereiteten Nutzungsdaten anzeigen soll. Dieser Prototyp diente als Orientierungshilfe während des späteren Erstellungsprozesses des Metareports.

Reports zeigen im Allgemeinen unterschiedliche Daten in übersichtlicher und aufbereiteter Form an. Diese werden mit Hilfe von SQL-Abfragen aus einer Datenbank entnommen und in sogenannten Datasets hinterlegt. Ein Bericht kann aus diesen Datasets Daten gezielt entnehmen und über Berichtselemente ausgeben. Nachdem der Prototyp für den Metareport erstellt war, begann die Erstellung aller benötigten SQL-Abfragen für die geplanten Berichtselemente. Mit diesen wurde unter anderem die Anzahl der Aufrufe und die benötigten Systemressourcen während der Erstellung eines Berichtes aus der Datenbank ermittelt.

Es folgte daraufhin der Designprozess des Berichtes. Hier zeigte sich jedoch, dass ein Bericht alleine zu unübersichtlich für die Ausgabe aller wissenswerten Daten werden würde. Deshalb wurden mehrere Berichte kreiert, die teilweise miteinander verknüpft wurden. Im Endergebniss entstand so ein Metareport zur Anzeige wichtiger Nutzungsdaten, welcher sich aus insgesamt vier Berichten zusammensetzt.

Damit der Metareport vor allem für Systemadministratoren komfortabel nutzbar ist, wurde der Bericht in eine entsprechende Softwarelösung integriert. Hierbei handelte es sich konkret um das *SQL Server 2012 Management Studio* von Microsoft. Der Aufruf des Berichtes innerhalb dieses Tools ist allerdings einigen Einschränkungen unterworfen. Beispielsweise ist die Verwendung von Unterberichten nicht möglich. Der Metareport wurde deshalb im Hinblick auf diese Einschränkungen designt, so dass ein Aufruf des Berichtes innerhalb der Software problemlos erfolgen konnte.

Abschließend wurde der fertige Metareport mit Hilfe einer zur Verfügung gestellten Datenbank mit einem umfangreichen Datenbestand getestet. Es zeigte sich, dass der Bericht die geforderten Metadaten ausgab und nur wenige weitere Anpassungen nötig waren.

4.2. Erweiterungsmöglichkeiten

Die Entwicklung des Metareports erfolgte gemäß der vorgegebenen Aufgabenstellung und den daraus abgeleiteten Anforderungen. Aus diesem Grund wurden manche Möglichkeiten der Implementierung weniger berücksichtigt oder gar ganz außer acht gelassen. Im Folgenden wird auf einige dieser Aspekte eingegangen, um damit Verbesserungsmöglichkeiten für eine eventuelle Weiterentwicklung aufzuzeigen.

Für die Beschaffung der Daten des Metareports wurde der Fokus nicht auf die Performance gelegt. Demnach nimmt eine Abfrage aller benötigten Metadaten auch eine gewisse Zeit in Anspruch. Es besteht also womöglich Verbesserungspotential bei den einzelnen SQL-Abfragen, mit denen die Daten für die Ausgabe im Metareport gesammelt werden. Zu untersuchen wäre, inwieweit die Metadatenabfragen optimiert werden können, um somit die Berichtsanzeige zu beschleunigen.

Die Ausführung des Metareports erfolgt immer manuell. Eine automatisierte Ausführung des Berichtes wurde während der Entwicklung nicht in Betracht gezogen. Es wäre zu untersuchen, welche Vorteile die Möglichkeit der automatischen Ausführung bringen könnte. Vorstellbar wäre ein machineller Nachrichtenversand, wenn bestimmte Werte der

Metadaten überschritten werden. Dazu könnten die Abfragen der Metadaten des Vortages zu festen Zeitpunkten durchgeführt werden und die entsprechenden Systemadministratoren beispielsweise per E-Mail über das Auftreten von ungewöhnlichen Werten informiert werden. Somit würde der Metareport nicht nur ein hilfreiches Tool zur Fehlerfindung sein, sondern könnte auch als eine Art Frühwarnsystem dienen, was die Suche nach möglichen Fehlerursachen und Optimierungsmöglichkeiten beschleunigen könnte.

Alle Metadaten des Reports präsentieren die Nutzung der Reporting Services aus verschiedenen Blickwinkeln, wozu unter anderem die Aufrufhäufigkeiten, Erstellungszeiten und die Daten der übergebenen Parameter der einzelnen Berichte gehören. Zusätzlich zu diesen Daten verfügt auch der SQL-Server selber über eine Vielzahl an abfragbaren Metadaten. Diese beinhalten unter anderem Angaben über die Größen einzelner Log-Dateien oder auch die Wartezeiten auf einzelne Threads. Hierbei könnte man untersuchen, inwiefern diese für den SQL-Server spezifischen Daten für die Systemverwalter von Nutzen wären und wie diese sinnvoll im Metareport integriert werden könnten. Dadurch würde das Spektrum der im Bericht angezeigten Daten vergrößert werden, wobei allerdings darauf geachtet werden sollte, die Ausgabe nicht zu überladen. Womöglich könnte es sich hier anbieten, einen zusätzlichen Bericht zu entwickeln.

Im Hinblick auf den Berichtsentwurf wurde ein eher zweckmäßiges Design angestrebt, weshalb vor allem Tabellen und Diagramme eingesetzt wurden. Diese beiden Berichtselemente können bei einer einer großen Datenmenge die Übersichtlichkeit bewahren und erlauben einen schnellen Blick auf interessante Werte. Die Vorteile einer Umstrukturierung von Berichtselementen oder die Änderung des Berichtdesigns könnten für eine weiterentwickelte Version des Metareports analysiert werden. Hierbei wären nicht nur die möglichen softwareergonomischen Verbesserungen des Designs, wie zum Beispiel die Usability, interessant, sondern auch die Möglichkeiten einer optisch ansprechenderen Präsentation.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingsverzeichnis

Quellenverzeichnis

[Alp11] Paul Alpar, Rainer Alt, Frank Bensberg, Heinz Lothar Grob, Peter Weimann und Robert Winter. *Anwendungsorientierte Wirtschaftsinformatik – Strategische Planung, Entwicklung und Nutzung von Informationssystemen.* 6. Aufl. Vieweg + Teubner, 2011.

[BK10] Michael Bächle und Arthur Kolb. *Einführung in die Wirtschaftsinformatik.* 2. Aufl. Oldenbourg Verlag München, 2010.

[Bea07] Alan Beaulieu. *Einführung in SQL.* 1. Auflage, korr. Nachdruck. O'Reilly, 2007.

[Bec09] Margit Becher. *XML – DTD, XML-Schema, XPath, XQuery, XSLT, XSL-FO, SAX, DOM.* 1. Aufl. W3L GmbH, 2009.

[Ber11] Anders Berglund u. a. *XML Path Language (XPath) 2.0 (Second Edition).* Jan. 2011. URL: `http://www.w3.org/TR/xpath20/` (besucht am 22. 06. 2014).

[Boa11] Scott Boag u. a. *XQuery 1.0: An XML Query Language (Second Edition).* Jan. 2011. URL: `http://www.w3.org/TR/xquery/` (besucht am 30. 06. 2014).

[CF13] Daniel Caesar und Michael R. Friebel. *Schnelleinstieg SQL Server 2012 – Für Administratoren und Entwickler.* 2. Aufl. Galileo Press, 2013.

[Che76] Peter Pin-Shan Chen. *The Entity-Relationship Model – Toward a Unified View of Data.* März 1976. URL: `http://www.comp.nus.edu.sg/~lingtw/papers/tods76.chen.pdf` (besucht am 08. 07. 2014).

[Cod70] Edgar Frank Codd. „A relational model of data for large shared data banks". In: *Communications of the ACM* (Juni 1970), S. 377–387.

[DicKB] Business Dictionary. *What is knowledge base?* URL: `http://www.businessdictionary.com/definition/knowledge-base.html` (besucht am 27. 07. 2014).

[EDI14] GXS GmbH. *Was ist EDI? | EDI Leitfaden.* 2014. URL: `http://www.edileitfaden.de/what-is-edi/` (besucht am 10. 06. 2014).

[FW07] Heide Faeskorn-Woyke, Birgit Bertelsmeier, Petra Riemer und Elena Bauer. *Datenbanksysteme – Theorie und Praxis mit SQL2003, Oracle und MySQL.* 1. Aufl. Pearson Studium, 2007.

[Gor03] Davor Gornik. *Entity Relationship Modeling with UML.* Nov. 2003. URL: `http://www.ibm.com/developerworks/rational/library/content/03July/2500/2785/2785_uml.pdf` (besucht am 08. 07. 2014).

[Gro10] Norbert Gronau. *Enterprise Resource Planning – Architektur, Funktionen und Management von ERP-Systemen.* 2. Aufl. Oldenbourg Verlag München, 2010.

[HM10] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2010.

[KMU06] Hans-Georg Kemper, Walid Mehanna und Carsten Unger. *Business Intelligence – Grundlagen und praktische Anwendungen*. 2. Aufl. Vieweg, 2006.

[Mar13] Wolfgang Martin. *Über Self-Service Business Intelligence*. Okt. 2013. URL: http://blogs.sas.com/content/sasdach/2013/10/18/uber-self-service-business-intelligence-ein-beitrag-von-dr-wolfgang-martin/ (besucht am 27. 07. 2014).

[MicBI] Microsoft. *Business Intelligence und Berichtserstellung: Microsoft Dynamics ERP*. URL: http://www.microsoft.com/de-de/dynamics/business-intelligence.aspx (besucht am 20. 05. 2014).

[MicFK] Microsoft. *Primary and Foreign Key Constraints*. URL: http://msdn.microsoft.com/en-us/library/ms179610(v=sql.110).aspx#fkeys (besucht am 15. 07. 2014).

[MicIF] Microsoft. *IF...ELSE (Transact-SQL)*. URL: http://msdn.microsoft.com/de-de/library/ms182717(v=sql.110).aspx (besucht am 03. 08. 2014).

[MicMS] Microsoft. *Custom Reports in Management Studio*. URL: http://msdn.microsoft.com/en-us/library/bb153684(v=sql.110).aspx (besucht am 21. 07. 2014).

[MicPV] Microsoft. *Power View (SSRS)*. URL: http://technet.microsoft.com/de-de/library/hh213579(v=sql.110).aspx (besucht am 04. 09. 2014).

[MicRE] Microsoft. *Transact-SQL-Referenz (Datenbankmodul)*. URL: http://msdn.microsoft.com/de-de/library/bb510741(v=sql.120).aspx (besucht am 03. 08. 2014).

[MicRO] Microsoft. *ROW_NUMBER (Transact-SQL)*. URL: http://msdn.microsoft.com/de-de/library/ms186734(v=sql.105).aspx (besucht am 23. 06. 2014).

[MicRS] Microsoft. *Reporting Services (SSRS)*. URL: http://technet.microsoft.com/de-de/library/ms159106(v=sql.120).aspx (besucht am 08. 08. 2014).

[MicSE] Microsoft. *Berichtsserver*. URL: http://msdn.microsoft.com/de-de/library/ms157231(v=sql.105).aspx (besucht am 12. 05. 2014).

[MicSQ] Microsoft. *SQL Server*. URL: http://msdn.microsoft.com/de-de/sqlserver/aa336270.aspx (besucht am 27. 07. 2014).

[MicTE] Microsoft. *Onlinedokumentation für SQL Server 2012*. URL: http://technet.microsoft.com/de-de/library/ms130214(v=sql.110).aspx (besucht am 27. 07. 2014).

[MicUN] Microsoft. *UNION (Transact-SQL)*. URL: http://msdn.microsoft.com/de-de/library/ms180026(v=sql.105).aspx (besucht am 23. 06. 2014).

[MicVA] Microsoft. *DECLARE local_variable (Transact-SQL)*. URL: http://msdn.microsoft.com/de-de/library/ms188927(v=sql.110).aspx (besucht am 03. 08. 2014).

[MicWH] Microsoft. *WHILE (Transact-SQL)*. URL: http://msdn.microsoft.com/de-de/library/ms178642(v=sql.110).aspx (besucht am 03. 08. 2014).

[MicWI] Microsoft. *WITH common_table_expression (Transact-SQL)*. URL: http://msdn.microsoft.com/de-de/library/ms175972(v=sql.105).aspx (besucht am 23. 06. 2014).

[Moc13] Johannes Müller. *Rapid Prototyping: Fünf Mockup Tools*. Aug. 2013. URL: http://www.uxcite.de/prototyping/rapid-prototyping-fuenf-mockup-tools/ (besucht am 14. 07. 2014).

[MocDe] *dict.cc Wörterbuch*. URL: http://www.dict.cc/englisch-deutsch/mockup.html (besucht am 14. 07. 2014).

[Moo08] Alfred Moos. *XQuery und SQL/XML in DB2-Datenbanken*. 1. Aufl. Vieweg + Teubner, 2008.

[Mut11] Sorna Kumar Muthuraj. *SSRS – Reportserver Database Tables Explored*. Juli 2011. URL: http://sornanara.blogspot.de/search/label/Query%20ReportServer%20Database%20Tables (besucht am 03. 06. 2014).

[Nie08] Frank Niemann. *Office als Frontend-Ersatz für die ERP-Applikation*. Sep. 2008. URL: http://www.computerwoche.de/a/grenze-zwischen-office-und-erp-faellt,1870957,3 (besucht am 10. 06. 2014).

[RFC05] Tim Berners-Lee u. a. *Uniform Resource Identifier (URI): Generic Syntax*. Jan. 2005. URL: http://tools.ietf.org/html/rfc3986#section-3.3 (besucht am 17. 06. 2014).

[ResMi] Microsoft. *Report Server Execution Log and the ExecutionLog3 view*. URL: http://msdn.microsoft.com/en-us/library/ms159110(v=sql.105).aspx (besucht am 11. 06. 2014).

[SK10] Bettina Schwarzer und Helmut Krcmar. *Wirtschaftsinformatik – Grundlagen betrieblicher Informationssysteme*. 4. Aufl. Schäffer-Poeschel Verlag Stuttgart, 2010.

[SQLHI] w3computing SQL Server. *A brief history of SQL: SQL Server*. URL: http://www.w3computing.com/sqlserver/brief-history-sql/ (besucht am 10. 06. 2014).

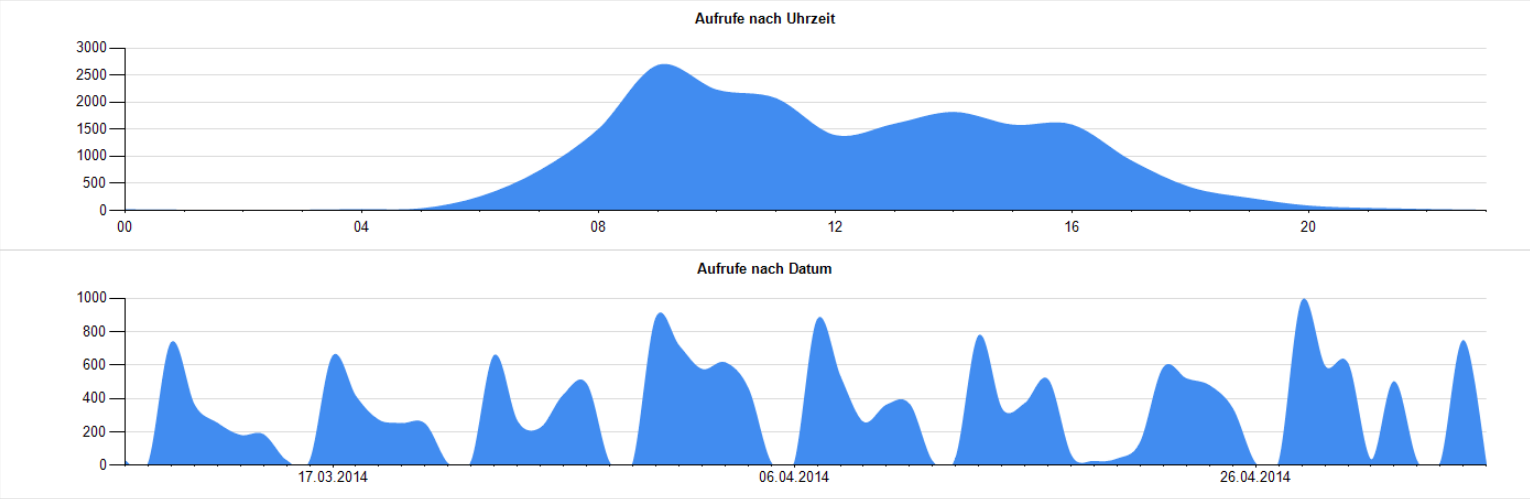
[SynPe] Synametrics. *Top 10 performance tuning tips for relational databases.* URL: `http://web.synametrics.com/top10performancetips.htm` (besucht am 21.07.2014).

[Tur12] Paul Turley, Robert M. Bruckner, Thiago Silva, Ken Withee und Grant Paisley. *Professional Microsoft SQL Server 2012 Reporting Services.* 1. Aufl. John Wiley & Sons, Inc., 2012.

Metadaten

vom 08.03.2014 bis 06.05.2014 (60 Tage) Zeitraum ändern:

Zeitraum in Tagen: 1 | 7 | 30



Am häufigsten aufgerufene Berichte:

Bericht	Aufrufe	Live-Aufrufe	Live-Anteil	Durchschnittszeit	Gesamtzeit	Datenzeit	Rechenzeit	Renderzeit	Avg-Pagination	Avg-Processing	Abos
/Flaechencontrolling/Flaechencontrolling	9365	5318	56,79%	00:00:22.471	2.10:27:26.234	2.06:19:24.789	01:11:34.21	02:56:27.235	418 KB	6895 KB	0
/Artikel-Renner-Penner inkl. Areamanager/ArtikelREnnerPenner	6329	3415	53,96%	00:00:42.033	3.01:53:51.552	3.00:28:24.718	00:16:06.81	01:09:20.024	175 KB	167 KB	0
/Liste 803 Umsätze pro Tag (neu)/Liste803	944	731	77,44%	00:00:54.383	14:15:38.264	13:38:34.363	00:27:11.861	00:09:52.04	1277 KB	841 KB	31
/Artikel-Renner-Penner inkl. Areamanager (Abonnements)/ArtikelREnnerPenner	788	767	97,34%	00:01:34.527	20:41:27.545	20:20:06.842	00:03:50.734	00:17:29.969	447 KB	291 KB	105
/DynamischeKER (Abonnements)/DynamischeKER	627	625	99,68%	00:00:24.001	04:10:49.224	02:52:59.325	00:04:37.465	01:13:12.434	1761 KB	1216 KB	64
/NetSalesDaily Version3/Net Sales Daily Version3	339	238	70,21%	00:00:59.742	05:37:32.574	05:09:16.599	00:10:30.86	00:17:45.115	994 KB	1940 KB	3
/DynamischeKER/DynamischeKER	158	98	62,03%	00:00:08.809	00:23:11.843	00:20:59.532	00:00:34.233	00:01:38.078	265 KB	329 KB	0
/Artikel-Farbe-Renner-Penner inkl. Areamanager/ArtikelFarbeRennerPenner	133	92	69,17%	00:00:25.406	00:56:19.039	00:55:36.127	00:00:08.476	00:00:34.436	40 KB	150 KB	1
/Farbe-Renner-Penner/Farbe-Renner-Penner	114	59	51,75%	00:00:02.972	00:05:38.872	00:05:24.901	00:00:03.305	00:00:10.666	60 KB	64 KB	0
/NetSalesDaily (Abonnements)/Net Sales Daily	100	100	100,00%	00:00:09.265	00:15:26.585	00:11:43.864	00:00:50.26	00:02:52.461	573 KB	202 KB	5

⊞ Die restlichen Berichte ein- und ausblenden

Aktivste Benutzer:

Benutzer	Durchschnittszeit	Gesamtzeit	Aufgerufene Berichte	Aufrufe
Oola (1)	00:01:08.805	1.11:54:46.472	⊞	1879
Tahiri Veila	00:00:16.994	03:27:02.828	⊞	731
Beru Lars (1)	00:00:11.358	01:13:49.634	⊞	390
Gilad Pellaeon	00:00:12.628	01:14:30.639	⊞	354

Abbildung 45: Ausschnitt des im Management Studio ausgeführten Metareports

B. Inhalt der CD-ROM

Die Abbildung 46 gibt einen Überblick über die Verzeichnisstruktur die sich auf der beigefügten CD-ROM befindet.

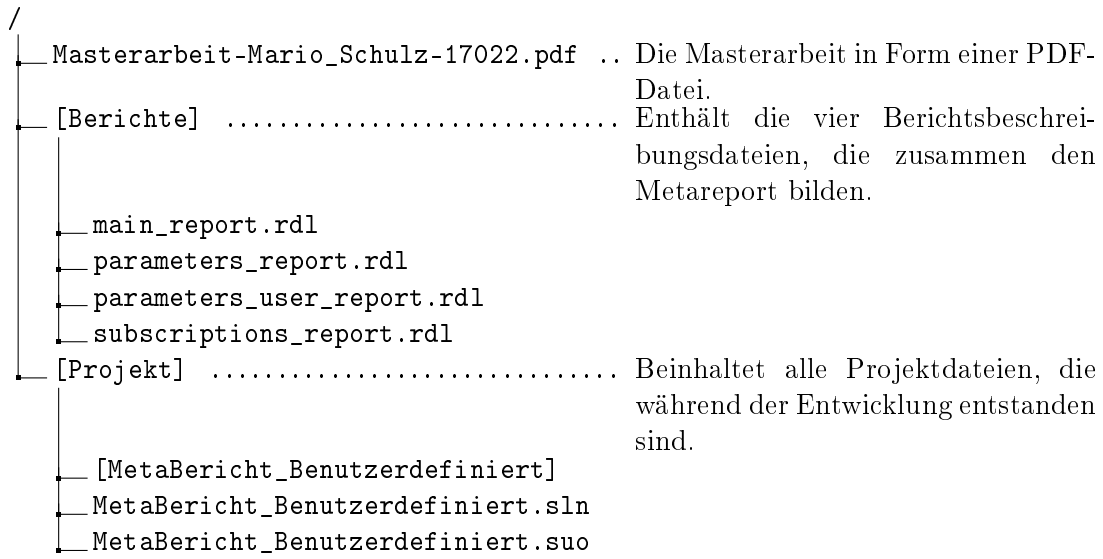


Abbildung 46: Verzeichnisstruktur der Daten auf der CD-ROM

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Ort, Datum Unterschrift

Einverständniserklärung

Ich erkläre mein Einverständnis zu einer Veröffentlichung der vorliegenden Arbeit im Internet.

☐ ja ☐ nein

Ort, Datum Unterschrift