

D I P L O M A R B E I T

Webbasierte Darstellung großer Datenmengen als Pivot-Tabelle mithilfe ressourcenoptimierter Aggregationsverfahren

ausgeführt am

Institut für Computergraphik und Algorithmen
der Technischen Universität Wien

unter der Anleitung von

Univ.-Prof. Dipl.-Ing. Dr.techn. Günther R. Raidl

und

Univ.-Ass. Mag.rer.soc.oec. Dipl.-Ing. Martin Gruber

durch

Anna Katarzyna Potocka

Matrikelnummer 9501886

Dr.Billrothstrasse 1/8, A-3430 Tulln

Datum

Unterschrift

Inhaltsverzeichnis

1	Einleitung	5
1.1	Einführung	5
1.2	Begriffsdefinitionen	5
1.3	Einfache Tabellen	6
1.4	Kreuz-Tabellen	7
1.4.1	Definition	7
1.4.2	Anwendung	8
1.5	Pivot-Tabellen	9
1.5.1	Aufbau einer Pivot-Tabelle	9
1.5.2	Positionierung der Summen und Randsummen	10
1.6	Tabellenkalkulationssoftware	13
1.6.1	MS Excel	13
1.6.2	MS Access	15
1.6.3	baaGrid von Barry Andrew	15
1.7	Motivation	16
1.7.1	CARE	16
1.7.2	Aufgabenstellung	17
2	Zugrundeliegende Datentabellen	20
2.1	Aufbau einer Datentabelle	20
2.2	Regeln für die Datentabellen	21
2.3	Bereiche der Pivot-Tabelle in der Datentabelle	22
2.3.1	Zeilenfelder	22
2.3.2	Spaltenfelder	22
2.3.3	Datenfelder	23
3	Datenstruktur Baum für den Zeilenbereich	24
3.1	Baum	24
3.1.1	Definition	24
3.1.2	Zeilenfelder als Baum	26
3.1.3	Begriffsbedeutung für die Pivot-Tabelle	26

3.2	Baumaufbau	28
3.2.1	Rekursiver Aufbau	28
3.2.2	Nur ein SELECT	29
3.2.3	Analyse	32
4	Rekursiver Aufbau des Spaltenbereichs	33
4.1	Herkunft der Spaltenbezeichnungen	33
4.2	colspan	34
4.3	HTML Code	35
5	Datenfelder	38
6	Implemetierung mittels einzelner SQL-Abfragen	43
6.1	Performance	43
6.2	Durchführungsmöglichkeiten	44
6.2.1	Ein SQL-Statement pro Knoten und Spaltenfeld	44
6.2.2	Ein SQL-Statement pro Knoten	47
6.3	Verbesserung der SQL Performance	50
6.3.1	MySQL-Tabellentypen	50
6.3.2	Index	51
6.3.3	Query Cache	52
7	Cachen	53
7.1	Cachen	53
7.2	Durchführungsmöglichkeiten	54
7.2.1	Einzelne Abfragen abspeichern	54
7.2.2	Zwischenspeichern der Werte aus einem einzigen SQL- Statement	60
7.2.3	Weitere Möglichkeiten	61
8	Speicheroptimierung vs. Laufzeitoptimierung - Vergleichser- gebnisse	64
8.1	Einleitung	64
8.1.1	Testen	65
8.1.2	Technische Daten	65
8.2	Formale Analyse	66
8.2.1	Methode 1: je eine SQL-Abfrage pro Knoten und pro Spalte	66
8.2.2	Methode 2: je eine SQL-Abfrage pro Knoten	68
8.2.3	Methode 3: Zwischenspeicher mit mehreren SQL-Abfragen	70

8.2.4	Methode 4: Zwischenspeicher mit einer einzigen SQL- Abfrage	71
8.3	Testergebnisse	75
9	Realisierung	83
9.1	Variablen	83
9.2	Objekt node	84
9.3	Funktionen	85
10	Schlussfolgerungen	87
10.1	Wahl der Methode	87
10.2	Future Work	88

Zusammenfassung

Pivot-Tabellen sind komplexe, mehrdimensionale Tabellen, die zur gleichzeitigen Auswertung von mehreren, zusammenhängenden Merkmalen zugrunde liegender Daten dienen. Bestehende Software stößt bei sehr großen Datenmengen nicht nur an Zeit- sondern auch an Kapazitätslimits. Auch der Trend zu webbasierten Tools und die Anforderung nach Plattformunabhängigkeit kann mit den vorhandenen Lösungen bisher nicht abgedeckt werden.

Im Zuge dieser Diplomarbeit werden vier Methoden zur Erstellung von Pivot-Tabellen aus gegebenen Datenbanktabellen entwickelt, die sich durch die Anzahl und Art der Datenbankabfragen, die Komplexität der Berechnungen und die Größe des benötigten Cache voneinander unterscheiden. Die Performanceuntersuchungen stellen laufzeitintensive SQL-Abfragen und speicherintensive PHP-basierende Lösungen gegenüber. Die endgültige Wahl der richtigen Methode hängt zusätzlich von der Zusammensetzung und Menge der darzustellenden Daten ab.

Abstract

A pivot table is a complex, multidimensional table which makes it possible to analyse data comparing more than one attribute at the same time. Creating pivot tables with huge amounts of analysed data is not only a question of time. Limits concerning the memory capacity are often reached. The growing demand for platform independence and web based applications without high license costs is also an important criterion. These requirements cannot be supplied by existing solutions.

In this thesis four methods for creating a pivot table out of a database table are presented. They differ in the amount of SQL queries needed, the complexity of the operations and how much cache is used for storing the SQL results. These factors are of main interest when discussing the computational results. For choosing the appropriate method also the kind and the amount of the analysed data to be presented in a pivot table have to be considered.

Kapitel 1

Einleitung

1.1 Einführung

Pivot-Tabellen sind eine spezielle Art von mehrdimensionalen Tabellen, die sowohl in Zeilen wie auch Spalten Merkmalsausprägungen gruppieren (beispielsweise nach Jahreszahlen, Ländern, Projekten, etc.). Sie dienen der Auswertung einer Liste oder einer Datenbank und ermöglichen eine komfortable, umfangreiche und schnelle Analyse von Daten, z.B. in Prozent oder durch Summation (Sch04). Pivottabellen sind eine Sonderform von Kreuztabellen. Im Folgenden wird zuerst auf die Unterschiede und Anwendungsgebiete dieser Tabellenarten eingegangen. Danach folgt eine kurze Betrachtung der bestehenden Softwarelösungen und deren Einschränkungen. Daraus wird ersichtlich, dass diese für bestimmte Anwendungen nicht ausreichen, wodurch sich für ein konkretes Projekt die Notwendigkeit ergab, ein neues Programm zu entwickeln.

1.2 Begriffsdefinitionen

Merkmal

Als Merkmal wird im Zusammenhang mit Pivot-Tabellen eine bestimmte Eigenschaft einer Datenmenge bezeichnet. Wenn man beispielsweise eine Anzahl von Autos analysieren will, könnten die Merkmale der Daten sein: Baujahr, Modell, Farbe, Motorleistung, etc. In einer einfachen (horizontalen) Tabelle entspricht ein Merkmal der Spaltenüberschrift.

Merkmalsausprägung

Die Merkmalsausprägung ist ein bestimmter Wert eines Merkmals. Bei den obigen Auto-Merkmalen wären das zum Beispiel „1975“, „Ente“, „rot“ und „35PS“.

Variable

Alle Merkmale, auf die Aggregationsfunktionen angewendet werden können, werden als Variablen bezeichnet. Variablen müssen reine Zahlenwerte sein. Im Autobeispiel kann das die Stückzahl sein, ebenso wie Hubraum, Leistung oder Kilometerstand.

Gruppierung nach einem Merkmal

Bei der für eine Datenanalyse wichtigen Gruppierung nach einem Merkmal werden alle Daten mit derselben Merkmalsausprägung zusammengefasst. Im einfachste Fall werden Summen berechnet. Aus obigem Fall könnte man zum Beispiel darstellen, wie viele Autos mit Baujahr 1975 in der Tabelle zu finden sind. Es sind aber auch alle anderen statistischen Aggregationsfunktionen anwendbar, wie zum Beispiel Mittelwert, Median, etc.

1.3 Einfache Tabellen

„Einfache“ Tabellen bestehen aus Spalten und Zeilen, in denen die darzustellenden Daten als Matrix angeordnet sind. Eine einfache Tabelle kann vertikal mit einer Vorspalte (=erste Spalte) oder horizontal mit einer Kopfzeile (=erste Zeile) angeordnet sein. Einfache Tabellen sind immer Ausgangspunkt für Pivot-Tabellen und werden in dieser Arbeit mit Datentabellen bezeichnet. Der Aufbau und die Regeln für Datentabellen werden in Kapitel 2 behandelt. Abb. 1.1 zeigt einen Ausschnitt aus der in den folgenden Beispielen verwendeten Datentabelle. Es handelt sich immer um horizontal angeordnete Tabellen mit einer Kopfzeile. Die Spalten beziehen sich auf Variablen (Überschriften), die Zeilen auf Beobachtungen dieser Variablen (die eigentlichen Werte).

id	Wissenschaftslehre	Wissenschaft	Bereich	Wissenschaftler	Monat	Jahr	Krankenstand	Urlaub	Ideen	Patente	Einnahmen	Ausgaben
286	Fachwissenschaft	Naturwissenschaften	Physik	Newton	Juli	2005	7	6	125	8	1886	32
121	Fachwissenschaft	Naturwissenschaften	Physik	Newton	Jänner	2006	0	5	142	8	193	36
122	Fachwissenschaft	Naturwissenschaften	Physik	Newton	April	2006	3	6	199	9	2417	30
123	Fachwissenschaft	Naturwissenschaften	Physik	Newton	September	2006	5	3	235	8	2501	42
124	Fachwissenschaft	Naturwissenschaften	Physik	Newton	Jänner	2006	4	1	2	10	1042	10
125	Fachwissenschaft	Naturwissenschaften	Physik	Newton	September	2006	3	5	141	8	793	26
126	Fachwissenschaft	Naturwissenschaften	Physik	Doppler	Juli	2006	1	0	188	3	254	38
127	Fachwissenschaft	Naturwissenschaften	Physik	Doppler	Juni	2006	4	2	259	0	988	17
128	Fachwissenschaft	Naturwissenschaften	Physik	Doppler	Jänner	2006	2	2	104	6	35	34
129	Fachwissenschaft	Naturwissenschaften	Physik	Doppler	März	2006	3	4	218	9	2122	33
130	Fachwissenschaft	Naturwissenschaften	Chemie	Avogadro	März	2006	6	6	50	3	2108	4
131	Fachwissenschaft	Naturwissenschaften	Chemie	Avogadro	Februar	2006	7	5	39	5	2316	46
132	Fachwissenschaft	Naturwissenschaften	Chemie	Avogadro	Mai	2006	8	1	210	4	367	24
133	Fachwissenschaft	Naturwissenschaften	Chemie	Avogadro	April	2006	6	4	128	5	2053	36
134	Fachwissenschaft	Naturwissenschaften	Chemie	Avogadro	September	2006	7	5	222	3	2112	18
120	Fachwissenschaft	Naturwissenschaften	Physik	Faraday	November	2006	1	2	212	6	979	49
118	Fachwissenschaft	Naturwissenschaften	Physik	Faraday	Mai	2006	1	2	81	5	794	20
117	Fachwissenschaft	Naturwissenschaften	Physik	Maxwell	September	2006	0	2	115	5	223	20
285	Fachwissenschaft	Naturwissenschaften	Physik	Newton	Februar	2005	6	4	47	8	324	43
283	Fachwissenschaft	Naturwissenschaften	Physik	Newton	Februar	2005	7	3	242	7	1530	32
282	Fachwissenschaft	Naturwissenschaften	Physik	Faraday	Dezember	2005	2	5	282	9	278	43
281	Fachwissenschaft	Naturwissenschaften	Physik	Faraday	November	2005	2	4	10	9	2078	33

Abbildung 1.1: Ausschnitt aus einer horizontal angeordneten Tabelle mit einer Kopfzeile (fett dargestellt). In der Kopfzeile stehen die Merkmale der Daten.

1.4 Kreuz-Tabellen

1.4.1 Definition

Kreuz-Tabellen sind Tabellen, die anders als einfache Tabellen sowohl eine Kopfzeile als auch eine Vorspalte besitzen. In diesen sind die Gruppierungseigenschaften (Merkmale) angeführt, nach denen die Merkmalsausprägungen aus der Datentabelle gruppiert¹ werden. Am Schnittpunkt der entsprechenden Spalten und Zeilen werden alle Datenwerte der Ausgangstabelle eingezeichnet, die exakt diesen beiden Merkmalsausprägungen entsprechen. Auf diese Datenwerte wird eine Aggregationsoperation wie Summenbildung oder Prozentberechnung angewandt. Im Beispiel Abb. 1.2 werden alle Ideen und Patente von drei Wissenschaften aufsummiert. Im links oberen Zahlen-Feld steht somit die Summe über alle Ideen der „Formalwissenschaften“ (19521), usw. Das selbe Ergebnis erhält man, wenn man die zugrundeliegende Datentabelle durchsucht und für alle Zeilen, die in der Spalte „Wissenschaft“ die Merkmalsausprägung „Formalwissenschaften“ eingetragen haben, die Wer-

¹ Gruppieren nach einem Feld X bedeutet, dass man alle Datensätze aus der Datentabelle, die in der Spalte X exakt den gleichen Inhalt haben, zu je einem einzigen Datensatz in der Ergebnistabelle, sei es eine Kreuz-Tabelle oder eine Pivot-Tabelle, zusammenfasst. Die einzelnen Werte können dann summiert werden oder z.B. der Prozentwert zum Ganzen bestimmt werden.

AUSPRÄGUNGEN	MERKMAL I			
	MERKMAL II	Erfolge		Gesamt
		Ideen	Patente	
		Wissenschaft		
		Formalwissenschaften	19521 639	20160
		Geisteswissenschaften	13813 456	14269
		Naturwissenschaften	16947 592	17539
		Gesamt	50281 1687	51968
RANDSUMMEN				

Abbildung 1.2: Beispiel einer Kreuz-Tabelle.

te aus der Spalte „Ideen“ aufsummiert. Eine Keuztabelle liefert somit also einen schnellen Überblick über Zahlen, die über eine unter Umständen riesige Tabelle verteilt sind.

1.4.2 Anwendung

Kreuz-Tabellen finden häufig in der Statistik Anwendung, z.B. zur Analyse zweier nominalskaliertter Variablen² mit jeweils mehreren Ausprägungen. Eine IxJ-Kreuz-Tabelle hat in I Zeilen I verschiedene Ausprägungen der einen Variable und in J Spalten J verschiedene Ausprägungen der anderen Variable. Die Randsummen (Spalten- oder Zeilensummen) geben jeweils die Gesamtzahl der Beobachtungen einer bestimmten Merkmalsausprägung an (KB05). In der Kreuz-Tabelle in Abb. 1.2 wird der Zusammenhang zwischen dem Merkmal Wissenschaft, mit den Merkmalsausprägungen Formalwissenschaft, Geisteswissenschaft und Naturwissenschaft, und dem Merkmal Erfolg, mit den Ausprägungen Ideen und Patente, dargestellt³. Kreuz-Tabellen dienen aber nicht nur der tabellarischen Darstellung von Ergebnissen, sie sind auch Ausgangstabellen für multivariate Analysemethoden wie die Kontingenzanalyse und Korrespondenzanalyse, vgl. (KB05).

² Bei nominalskalierten Merkmalen wird der Untersuchungseinheit für die entsprechende Ausprägung (genau) ein Name bzw. (genau) eine Kategorie zugeordnet.

³ Alle Zahlen in diesem und allen weiteren Beispielen sind frei erfunden und sollen nur der Veranschaulichung dienen.

1.5 Pivot-Tabellen

Eine Pivot-Tabelle ist eine Kreuz-Tabelle mit mehr als zwei Merkmalen sowohl in den Spalten als auch in den Zeilen, die jeweils mehrere Gruppierungsebenen aufweisen können. Diese Merkmale können, solange es für die Analyse sinnvoll ist, beliebig vertauscht werden. Auch die Spalten und Zeilenmerkmale samt Ausprägungen können ausgetauscht werden. Dadurch ist eine interaktive Datenauflistung und Datenauswertung möglich. Aus dieser Funktionalität kommt auch die Bezeichnung Pivot-Tabelle⁴.

1.5.1 Aufbau einer Pivot-Tabelle

Die Einteilung einer Pivot-Tabelle in Bereiche basiert auf (Sch04). Schwabe gliedert eine Pivot-Tabelle in vier Bereiche:

- Zeilenfelder
- Spaltenfelder
- Datenfelder
- Seitenfelder

Diese Gliederung ist die für Microsoft (MS) Excel und auch andere Tabellenkalkulationsprogramme übliche Gliederung. Die Zeilen- und Spaltenfelder enthalten jeweils Merkmalsgruppen mit Merkmalen und ihren Ausprägungen, die Datenfelder die berechneten Werte (Beobachtungen). In den Datenfeldern können auch noch zusätzliche Aufteilungen stattfinden, damit man mehrere Variablen in einer Tabelle darstellen und vergleichen kann. Die Seitenfelder sind Filter für die Ausprägungen. Auf Seitenfelder wird in dieser Arbeit jeweils nur kurz eingegangen, da sie zur eigentlichen Problematik nicht viel beitragen. Anders als bei Kreuz-Tabellen werden die Gesamtergebnisse nicht nur in den Randsummen angezeigt, sondern bereits als Spalten und Zeilen innerhalb der Tabelle.

Alle Zeilenfelder gemeinsam werden als Zeilenbereich bezeichnet. Alle Spaltenfelder gemeinsam ergeben den Spaltenbereich.

⁴ Der Wortstamm für Pivot kommt aus dem Französischen und ist pivoter, was drehen bedeutet. Der Pivot war in der Militärtechnik der Drehpunkt bei fahrbaren Geschützen. Pivot wird hier als Synonym für Drehpunkt verwendet.

ZEILENFÄLDER

				SPALTENFÄLDER								
				2006								2007
				Jänner	Februar	März	April	Jänner				
				Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen		
Wissenschaftslehre ±	Wissenschaft ±	Bereich ±	Wissenschaftler	248	24	39	5	566	13	327	14	315
Fachwissenschaft ±	Naturwissenschaften ±			248	24	39	5	566	13	327	14	315
		Chemie ±				39	5	348	4	128	5	315
			Avogadro			39	5	50	3	128	5	315
			Boyle									
			Mendelejew					298	1			
		Physik ±		248	24			218	9	199	9	
			Doppler	104	6			218	9			
			Faraday									
			Maxwell									
			Newton	144	18					199	9	
Universalwissenschaft ±				353	17	526	19	899	32	797	20	786
	Formalwissenschaften ±			169	3	526	19	251	15	763	16	490
		Informatik ±		169	3	309	6	242	9	134	0	490
			Dijkstra			272	0	129	9	134	0	
			Knuth	169	3			113	0			253
			Turing			37	6					237
		Mathematik ±				217	13	9	6	629	16	
			Cauchy							190	6	
			Fibonacci					9	6			
			Gauß							236	2	
			Pythagoras			217	13			203	8	
	Geisteswissenschaften ±			184	14			648	17	34	4	296
		Philosophie ±		184	14			648	17	34	4	296

DATENFÄLDER

Abbildung 1.3: Aufbau einer Pivot-Tabelle: Im oberen Bereich befinden sich die sogenannten Spaltenfelder (im Beispiel die Jahreszahlen und Monate), links die Zeilenfelder (alle Textfelder der Wissenschaftslehren, Wissenschaftler, etc.). Der restliche große Bereich beinhaltet die Datenfelder. Diese Pivot-Tabelle dient zur Analyse von zwei Variablen (Ideen und Patente).

1.5.2 Positionierung der Summen und Randsummen

Für die Datenauswertung sind meist nicht nur die Werte allein nach den Merkmalsgruppierungen zusammengefasst interessant sondern auch z.B. die Summen oder Mittelwerte über alle dargestellten Variablen. Es gibt verschiedene Möglichkeiten diese Berechnungswerte zu positionieren.

Spaltensummen

Immer wenn eines der Spaltenfelder komplett aufgeführt wurde, kann man horizontal daneben dessen Berechnungswert als zusätzliche Spalte anzeigen lassen. Weiters ist es auch möglich, die Gesamtsumme über alle Spalten (Randsumme) wie bei Kreuz-Tabellen als letzte Spalte anzuführen. Bei vielen Untergruppen wird dies jedoch sehr schnell unübersichtlich. Abb. 1.4 zeigt ein Beispiel einer Pivot-Tabelle mit Spaltensummen.

Zeilensummen

Ebenso gibt es Gruppierungsebenen nach den Zeilenfeldern, deren Berechnungswerte für jede Spalte angezeigt werden können. Hier besteht die Möglich-

				2006					
				Jänner			Februar		
Wissenschaftslehre ±	Wissenschaft ±	Bereich ±	Wissenschaftler	Ideen	Patente	Summe	Ideen	Patente	Summe
Fachwissenschaft ±				144	130	274	56	43	99
	Naturwissenschaften ±			144	130	274	56	43	99
		Chemie ±					56	43	99
			Avogadro				56	43	99
			Boyle						
			Mendeleejew						
		Physik ±		144	130	274			
			Doppler	48	60	108			
			Newton	96	70	166			

Abbildung 1.4: Für jede Merkmalsausprägung des Spaltenbereichs wird als letzte Spalte noch die Summe der Werte für diese Ausprägung eingefügt. In diesem Beispiel werden die Summen aus den *Ideen* und *Patenten* für die Monate *Jänner* und *Februar* in der jeweils letzten Spalte dieser Ausprägungen mit der Überschrift *Summe* angeführt.

keit, diese entweder oberhalb oder unterhalb der jeweiligen Gruppierungsebene anzuzeigen. Abb. 1.3 zeigt eine Pivot-Tabelle mit Berechnungswerten oberhalb der jeweiligen Gruppierungsebene. In der ersten Daten-Zeile („Fachwissenschaft“) stehen pro Spalte die Summen über alle darunterstehenden Zeilen bis zur nächsten Merkmalsausprägung der „Wissenschaftslehre“, in diesem Beispiel „Universalwissenschaft“. Dieses System setzt sich pro Ebene fort. Zum Beispiel stehen in der Zeile „Chemie“ die Summen über alle Werte der Chemie zugeordneten Wissenschaftler.

Wie auch bei den Spaltenfeldern kann ebenfalls eine Gesamtsumme (Randsumme) als letzte Zeile eingefügt werden.

Übersichtlichkeit vs. Notwendigkeit

Wolfgang Ludwig-Mayerhofer kritisiert in seinem Artikel „Kleine Anmerkung, die Verbesserung der Darstellung von Kreuztabellen betreffend“ (LM94) die wenig einheitlichen Standards bei der Darstellung von Kreuz-Tabellen und die unterschiedliche Positionierung von Randsummen. Bei Pivot-Tabellen ist das Problem noch viel gravierender, wenn man sehr viele Gruppierungsebenen mit vielen Merkmalsausprägungen hat. Der Betrachter sieht dann kaum auf einen Blick, welche nun die tatsächlichen Datenfelder aus der Datentabelle sind und welche bereits die Spalten/Zeilensummen. Will man nun zusätzlich nicht die Summen, Mittelwerte usw., sondern Prozentwerte, dann wird es umso schwieriger, da man zusätzlich wissen muss, ob die unabhängigen Variablen (auf die sich die Prozentuierung bezieht) in den Spalten- oder Zeilenfeldern zu suchen sind. Bereits Hans Zeisel hat in seinem Buch „Die Sprache der Zahlen“ (HZ70) versucht, Regeln für die Dar-

stellung von Daten in Kreuz-Tabellen zu formulieren. Zeisel schlug vor, die unabhängigen Variablen in den Spaltenfeldern anzuordnen und nur Berechnungswerte der Gruppierungsebenen der Zeilenfelder zu berechnen und diese als Spaltensummen anzuzeigen.

Mayerhofer übernimmt die Zeiselsche Regel für Kreuz-Tabellen, auch wenn er darauf hinweist, dass es bei sehr vielen Merkmalsausprägungen der unabhängigen Variablen oft notwendig sein könnte, die Spalten- und Zeilenfelder zu vertauschen (somit bei Pivot-Tabellen auch bei sehr vielen verschiedenen Merkmalen). Dieses Problem ist bei webbasierten Tools nur zum Teil gegeben, da durch die variable Größe und Verschiebbarkeit des Browserfensters keine prinzipielle Einschränkung bezüglich Platz für die Tabelle gegeben ist (jedoch leidet die Übersichtlichkeit). Weiters vertritt er die Ansicht, dass die Einführung eines einfachen Zeichens (z.B. Pfeil) als explizites Zeichen für die Leserichtung einer Tabelle die Lesbarkeit einer Kreuz-Tabelle vereinfachen könnte.

Der Aufbau der im Rahmen dieser Diplomarbeit implementierten Pivot-Tabelle basiert nun auf der Zeiselschen Regel und berücksichtigt Mayerhofers Anmerkungen. Das bedeutet, dass auf die gleichzeitige Bildung von Spalten- und Zeilensummen verzichtet wird und nur Spaltensummen gebildet werden. Auf ein eindeutiges Zeichen für die Summationsrichtung bzw. Prozentuierungsrichtung wird verzichtet. Für die Positionierung der Berechnungswerte für jede Gruppierungsebene bleiben somit zwei mögliche Positionierungen: am Anfang der jeweiligen Gruppierungsebene oder am Ende. Gerade bei Summenbildungen⁵ ist der Betrachter gewöhnt, die Summe unterhalb einer Untereinanderauflistung von Werten zu finden. Trotzdem ist es nicht notwendigerweise die beste Positionierung. Eine Positionierung unterhalb der jeweiligen Merkmalsausprägung würde die Pivot-Tabelle, durch das Einfügen zusätzlicher Zeilen, unnötig vergrößern und damit unübersichtlich machen. Bei einer Positionierung oberhalb der Merkmalsausprägungen der jeweiligen Gruppierungsebene kann bereits die vorhandene Zeile, in der die Merkmalsausprägung angeführt wird, verwendet werden. Rein implementierungstechnisch ist allerdings erstere Variante leichter umzusetzen, da man - wie beim händischen Rechnen - zuerst die Werte benötigt, bevor diese aufsummiert werden können⁶.

Um die Lesbarkeit zusätzlich zu erhöhen, werden den einzelnen Merkmalsausprägungen verschiedene Hintergrundfarben zugewiesen und auch die

⁵ Es wird desweiteren nicht mehr auf die unterschiedlichen Möglichkeiten der mathematischen Operationen, die hier angewendet werden können, eingegangen. Stellvertretend für alle wird die Summation behandelt.

⁶ Beim Aufbau einer HTML-Seite (mittels PHP) gilt das umso mehr, da der Seitencode zeilenweise von oben nach unten generiert wird.

jeweiligen Spalten mit den entsprechenden Hintergrundfarben versehen. Eine ähnliche Methode schlägt A.S.C Ehrenberg (Ehr86) vor, um die Lesbarkeit von Tabellen zu erhöhen. Er regt an, die Anzahl der Trennlinien möglichst zu reduzieren und diese nur jeweils unter einer Summe zu zeichnen. In seinen Beispieltabellen werden die Gruppen-Summen ebenfalls oberhalb der gesamten Gruppierungsebene angeführt. Ehrenbergs weitere Tipps beziehen sich eher auf Printpublikationen und bestehen darin, die Tabellen nach Spaltenfeldern zu trennen und in mehreren Tabellen anzuordnen, was dem eigentlichen Sinn einer Pivot-Tabelle, dem Gegenüberstellen mehrerer gruppierter Daten, widerspricht und deshalb hier nicht weiter verfolgt wird.

1.6 Tabellenkalkulationssoftware

Bei Recherchen zu dem Suchbegriff „Pivot-Tabelle“ erhält man fast ausschließlich Ergebnisse in Zusammenhang mit MSOffice®. Pivot-Tabellen sind hier in den Programmen MS Excel und MS Access integriert. Weiters bietet das Softwarepaket SPSS® (eine Statistik- und Analysesoftware) eine Darstellung mittels Pivot-Tabellen an.

Abgesehen von diesen lizenzpflichtigen, lokal zu installierenden Programmen, ist als einzige funktionsfähige Softwarelösung das baaGrid von Barry Andrew (baa08) zu finden.

Einschränkungen von lokaler Software

Lokal installierte Software hat Einschränkungen bezüglich der Erstellung von Pivot-Tabellen. Diese entstehen einerseits aus der Rechnerleistung und andererseits aus der Speicherkapazität des verwendeten PCs, der für den Aufbau der Pivot-Tabelle mit der entsprechenden Tabellenkalkulationssoftware eingesetzt wird. Letztere bestimmt die maximalen Größe der zu erstellenden Pivot-Tabelle. Die Anzahl der zu verarbeitenden Datensätze ist zwar rein theoretisch unbegrenzt, kann aber zu sehr langen Laufzeiten führen. Zusätzlich kommen softwarespezifische Beschränkungen dazu. Bei MS Excel sind dies zum Beispiel Beschränkungen bei der Anzahl der Spaltenfelder, Zeilenfelder und der Elemente pro Pivot-Feld (Mic08).

1.6.1 MS Excel

MS Excel bietet sehr viele Möglichkeiten in Bezug auf Pivot-Tabellen und ist ohne Zweifel der Vorreiter auf diesem Gebiet. Abgesehen von der Interaktivität, d.h. der Möglichkeit, jederzeit Zeilen- und Spaltenfelder per

Zeilenbeschriftungen	2006		Februar		März		April		2006 S_Ideen	2006 S_Patente	2007		Gesamt	Gesamt
	Jänner	S_Ideen	S_Patente	S_Ideen	S_Patente	S_Ideen	S_Patente	S_Ideen	S_Patente	S_Ideen	S_Patente	Jänner		
Fachwissenschaft	248	24	39	5	566	13	327	14	4152	137	315		16947	592
Naturwissenschaften	248	24	39	5	566	13	327	14	4152	137	315		16947	592
Chemie			39	5	348	4	128	5	1700	40	315		7130	234
Avogadro			39	5	50	3	128	5	649	20	315		2744	95
Boyle									451	5			2018	57
Mendelejew					298	1			600	15			2368	82
Physik	248	24			218	9	199	9	2452	97			9817	358
Doppler	104	6			218	9			769	18			3092	93
Faraday									678	26			2615	99
Maxwell									286	10			1463	31
Newton	144	18					199	9	719	43			2647	135
Universalwissenschaft	353	17	526	19	899	32	797	20	8219	290	786		33334	1095
Formalwissenschaften	169	3	526	19	251	15	763	16	4744	157	490		19521	639
Informatik	169	3	309	6	242	9	134	0	2608	64	490		10143	277
Dijkstra			272	0	129	9	134	0	1146	28			4400	121
Knuth	169	3			113	0			811	15	253		2735	76
Turing			37	6					651	21	237		3008	80
Mathematik			217	13	9	6	629	16	2136	93			9378	362
Cauchy							190	6	606	19			2073	52
Fibonacci					9	6			409	30			2678	121
Gauß							236	2	672	19			2087	86
Pythagoras			217	13			203	8	449	25			2540	103
Geisteswissenschaftler	184	14			648	17	34	4	3475	133	296		13813	456
Philosophie	184	14			648	17	34	4	3475	133	296		13813	456
Aristoteles	145	9					34	4	471	29			2941	90
Platon	39	5			351	11			1033	48			3976	158
Sokrates									1293	29	296		4541	120
Stoa					297	6			678	27			2355	88
Gesamtergebnis	601	41	565	24	1465	45	1124	34	12371	427	1101		50281	1687

Abbildung 1.5: MS Excel Pivot-Tabelle.

Drag & Drop zu vertauschen, gibt es auch eine Reihe von verschiedenen Ansichtsmöglichkeiten. In Abb. 1.5 wird der in jüngeren Office-Versionen als Standard definierte Aufbau einer Pivot-Tabelle gezeigt. Die Randsummen sind hier nicht am Ende einer Gruppierungsebene, sondern befinden sich bereits in der Zeile, in der das Gruppierungsmerkmal angeführt wird.

Spaltenfelder

Das Produkt der Anzahl der Elemente in allen Spaltenfeldern einer Pivot-Tabelle darf 32.768 nicht überschreiten⁷. Weiters sind einzelne Arbeitsblätter in MS Excel auf 256 Spalten beschränkt⁸, das heißt auch bei erfolgreicher Erstellung einer Pivot-Tabelle mit einer großen Anzahl von Spaltenfeldern kann es vorkommen, dass MS Excel nicht in der Lage ist, die gesamte erweiterte Pivot-Tabelle anzuzeigen.

Zeilenfelder

Ähnlich wie bei den Spaltenfeldern existiert auch eine Beschränkung der Zeilenfelder: das Produkt der Anzahl der Elemente in allen Zeilenfeldern einer Pivot-Tabelle darf 2^{31} (etwa 2,1 Milliarden) Elemente nicht überschreiten.

⁷ Beispielsweise hat eine Pivot-Tabelle mit fünf Spaltenfeldern, die jeweils 10, 5, 2, 40 bzw. 3 Elemente enthalten, als Produkt dieser Werte $10 \times 5 \times 2 \times 40 \times 3$ oder 12.000. Bei Hinzufügen eines weiteren Feldes, das drei Elemente enthält, wäre das Produkt schon 12.000×3 oder 36.000 und ist damit von MS Excel nicht mehr verarbeitbar.

⁸ Erst die Version MS Excel 2007 bietet 16.384 Spalten.

				2006									
				Jänner		Februar		März		April		Gesamt	
Wissenschaftslehre	Wissenschaft	Bereich	Wissenschaftler	S_Ideen	S_Patente	S_Ideen	S_Patente	S_Ideen	S_Patente	S_Ideen	S_Patente	S_Ideen	S_Patente
Fachwissenschaft	Naturwissenschaften	Chemie	Avogadro			39	5	50		3	128	5	217
			Boyle										
			Mendelejew					298		1		298	1
			Gesamt			39	5	348		4	128	5	515
		Physik	Doppler	104	6			218		9			322
			Newton	144	18					199		9	343
			Gesamt	248	24			218		9	199	9	665
		Gesamt		248	24	39	5	566	13	327	14	1180	56
				248	24	39	5	566	13	327	14	1180	56
		Gesamt		248	24	39	5	566	13	327	14	1180	56
Universalwissenschaft	Formalwissenschaften	Informatik	Dijkstra			272	0	129		9	134	0	535
			Knuth	169	3			113		0			282
			Turing			37	6						37
			Gesamt	169	3	309	6	242		9	134	0	854
		Mathematik	Cauchy								190	6	190
			Fibonacci					9		6			9
			Gauß							236	2	236	2
			Pythagoras			217	13			203	8	420	21
			Gesamt			217	13	9		6	629	16	855
				169	3	526	19	251	15	763	16	1709	53
	Geisteswissenschaften	Philosophie	Aristoteles	145	9						34	4	179
			Platon	39	5			351	11				390
			Sokrates										
			Stoa					297	6				297
			Gesamt	184	14			648	17	34		4	866
				184	14			648	17	34		4	866

Abbildung 1.6: MS Access Pivot-Tabelle.

Maximale Anzahl von Elementen pro Pivot-Feld

Weiters besteht eine Beschränkung auf 8.000 eindeutige Elemente pro Zeilen-, Spalten- oder Seitenfeld. Felder, die diese Beschränkung überschreiten, werden nicht zur Pivot-Tabelle hinzugefügt.

1.6.2 MS Access

MS Access bietet im Großen und Ganzen ähnliche Funktionen wie MS Excel. Als Standard werden die Gesamtsummen einer Gruppierungsebene jedoch immer darunter angeführt.

1.6.3 baaGrid von Barry Andrew

Barry Andrew (baa08) stellt auf seiner Homepage die PHP-Klasse baaGrid vor. Diese Implementierung erzeugt Tabellen zur Darstellung von Datenbankabfragen. Über Parameter können verschiedene Layout-Einstellungen vorgenommen werden. Für allgemeine Problemstellungen ist diese Klasse jedoch ungeeignet, da sie mehreren Einschränkungen unterliegt. Es werden keine Spaltenfeldergruppierungen und auch nur zwei Gruppierungsebenen bei den Zeilenfeldern unterstützt. Daher eignet sich diese Klasse eigentlich nur zur Darstellung von Kreuztabellen und nicht von komplexen Pivot-Tabellen. Das SQL-Statement wird nicht dynamisch generiert, vor allem fehlt jedoch auch die Möglichkeit der Interaktivität, weil jede Veränderung der Parameter über den Programmcode erfolgen muss.

1.7 Motivation

1.7.1 CARE

Die non-profit Organisation CARE Österreich hat sich im Zuge einer Überarbeitung der Softwarelandschaft ihrer Finanz- und Projektverwaltung für die Implementierung eines neuen Reportingsystems entschieden. Dieses Softwarepaket soll vor allem zur Dateneingabe, Wartung und Analyse von Projektdaten dienen.

Eine Analyse ergab folgende Mindestanforderungen an die Software:

- webbasiert/browserfähig
- plattformunabhängig
- für große Datenmengen geeignet
- flexibel
- lizenzfrei

Diese Punkte können bestehende Softwarelösungen wie MS Excel oder MS Access nicht erfüllen.

Trend zu webbasierten Tools

Die Entwicklung geht immer mehr in Richtung webbasierter Tools. Dies hat viele Vorteile. Für den Zugriff auf die Daten ist lediglich ein Browser notwendig. Dadurch ist der Nutzer vor allem ortsunabhängig (ein Punkt, der für die weltweit operierende Organisation CARE unabdingbar ist). Aber auch die Tatsache, dass an den Client-PC nur geringe technische Anforderungen bezüglich Rechnerleistung gestellt werden und auch das Betriebssystem keine Rolle spielt, ist ein nicht zu unterschätzender Vorteil. Aufwendige und kostspielige Softwareinstallationen entfallen und die Wartung wird zentral auf dem Server durchgeführt.

Ein großes Problem von webbasierten Tools ist jedoch die Sicherheit der Daten vor unerwünschten Zugriffen nichtauthorisierter Personen. Diese muss durch geeignete Maßnahmen gewährleistet werden. Der Punkt Datensicherheit ist jedoch nicht Gegenstand dieser Arbeit, sollte aber dennoch nicht unerwähnt bleiben.

Plattformunabhängigkeit

Plattformunabhängigkeit ist immer wieder ein wichtiges Thema. Dies ist zurückzuführen auf eine größer werdende Betriebssystem- und Versionsvielfalt der Serverlandschaften. Im Zusammenhang mit webbasierten Tools bedeutet dies auch Unabhängigkeit von einem bestimmten Browser, der Einhaltung von internationalen Standards (W3C) kommt dabei eine besondere Bedeutung zu.

Datenmengen

Bei Reportingtools werden mitunter sehr große Datenmengen verarbeitet. Das können zum Beispiel tägliche Stundenerfassungen von tausenden Mitarbeitern sein, oder die Rückverfolgung von einzelnen (CARE-)Paketen. Eine Analyse-Software muss diese Datenmengen mit möglichst kurzer Responsezeit verarbeiten und übersichtlich darstellen können.

Flexibilität

Eine Eigenentwicklung bringt immer ein hohes Maß an Flexibilität. Gerade im Bereich von Reportingtools kann das ein großer Vorteil sein und ist in bestimmten Fällen sogar unumgänglich (bei CARE gibt es zum Beispiel einige Ausnahmefälle, die speziell behandelt werden müssen). Ein besonderes Augenmerk ist dabei jedoch auf die Dokumentation zu legen, um das Projekt weiterentwickeln zu können.

Lizenzkosten

Softwareprodukte großer Firmen sind meistens mit hohen Lizenzkosten verbunden. Diese sind oftmals nicht nur einmalig zu entrichten, sondern bringen auch laufende Kosten für Wartung und Upgrades mit sich. Im Gegensatz dazu gibt es aber auch gute, kostenfreie Open Source Entwicklungsumgebungen, welche für die meisten Eigenentwicklungen eine gute Basis bieten.

1.7.2 Aufgabenstellung

Aus diesen Überlegungen ergibt sich nun die Aufgabenstellung dieser Arbeit: ein plattformunabhängiges, webbasiertes Tool zur Darstellung von großen Datenmengen als Pivot-Tabellen. Aber auch bei einem webbasierten Tool muss man zwei wichtige Kenngrößen im Auge behalten: Speicherkapazität und Laufzeit. Wenn die Software nicht performant genug implementiert ist, können schnell die Hardwaregrenzen erreicht sein. Durch die Wahl eines

leistungsfähigen Server-Rechners (der aber wieder hohe Kosten verursacht) kann man dem zwar stark entgegenwirken, trotzdem bleiben die prinzipiellen Schwierigkeiten erhalten. Ressourcenoptimierte Aggregationsverfahren, Cachen und andere Tuningtechniken sollen hier Abhilfe leisten.

Die Anforderungen der Browserfähigkeit, der Plattformunabhängigkeit und der Lizenzkosten werden durch die Implementierung mit den Webtechnologien PHP, MySQL und HTML erfüllt.

Kapitel 2

Zugrundeliegende Datentabellen

Während Kreuz-Tabellen oft manuell erstellt werden indem die Werte, die z.B. Ergebnisse einer Untersuchung sind, gleich vorab gruppiert und in die Kreuz-Tabelle eingetragen werden, sind Pivot-Tabellen einerseits zu umfangreich und sollen andererseits interaktiv und flexibel sein, was eine Direkteingabe von Werten unmöglich macht. Pivot-Tabellen brauchen also immer bereits eine einfache Tabelle als Ausgangsbasis, die Datentabelle.

Aufbereitung der Daten

Die hier verwendeten Daten und Werte sind frei erfunden und für den Zweck dieser Diplomarbeit aufbereitet. Die in den Beispielen getroffene Einteilung der Wissenschaft basiert auf (RE05), die Aufteilung der Tabelle ähnelt den Beispieltabellen aus (SZ03).

2.1 Aufbau einer Datentabelle

Abb. 1.1 zeigt eine Datentabelle. Für die im Rahmen dieser Diplomarbeit implementierte Software zum Aufbau von Pivot-Tabellen muss es sich bei einer zugrunde liegenden Datentabelle immer um eine horizontale Tabelle mit einer Kopfzeile handeln. Die Implementierung sieht vor, dass die Daten aus einer Datenbank gelesen werden, dessen Aufbau eben diesem Schema entspricht. Natürlich ist auch die Verwendung mehrerer Tabellen als Basis für eine Pivot-Tabelle möglich, dies ist aber hier nicht vorgesehen und wird nicht weiter behandelt.

2.2 Regeln für die Datentabellen

Um aus einer Datentabelle eine Pivot-Tabelle erstellen zu können, muss diese – abgesehen von dem zuvor besprochenen Aufbau – auch noch weitere Kriterien bezüglich der Daten erfüllen, die Ignaz Schels (Sch07) anhand von MS Excel Pivot-Tabellen zusammengefasst hat.

- Die Bezeichnungen der Gruppierungsebenen, der Spaltenfelder in der Ausgabe sowie in den dynamisch aufgebauten SQL-Abfragen werden direkt aus den Bezeichnungen in der Kopfzeile übernommen. Diese sind einzeilig und enthalten keine Leerzeichen und spezielle Sonderzeichen.
- Wenn es sich nicht um eine Datentabelle aus einer Datenbank handelt, muss der Bereich der Datentabelle abgeschlossen und als solcher erkennbar sein. Es dürfen keine Leerzeilen in der Tabelle vorkommen, sehr wohl aber als Abgrenzung des relevanten Bereichs.
- Da die Datentabelle spaltenweise behandelt wird, einerseits beim Aufbau der Spaltenfelder und andererseits bei den Gruppierungen und Berechnungen für die Datenfelder, ist es wichtig, dass Werte aus der gleichen Spalte immer gleiches Format haben. Bei einer Datentabelle, die aus einer Datenbank gelesen wird, ist das selbstverständlich, nicht jedoch, wenn die Daten aus einer Textdatei gelesen werden.
- Auch das Verbinden von Zellen einer Tabelle über mehrere Spalten hinweg ist durch eine Datenbank unterbunden, kann aber sehr wohl bei Datentabellen anderen Ursprungs vorkommen, was ein automatisiertes Einlesen der Daten aber verkompliziert.
- Während das Nichteinhalten der ersten Punkte den Aufbau einer Pivot-Tabelle unmöglich macht, gibt es auch Regeln, die eingehalten werden sollten um eine sinnvolle Datenanalyse durch Pivot-Tabellen zu ermöglichen. Da z.B. die Hauptaufgabe einer Pivot-Tabelle darin besteht, Daten zu gruppieren, muss diese Gruppierung möglich sein. Eine Tabelle, die in den für die Analyse wichtigen Spalten nur einfach vorkommende Einträge hat, ergibt wiederum eine einfache Tabelle aber keine Pivot-Tabelle.
- Backhaus (KB05) warnt – speziell in Bezug auf Kreuz-Tabellen, die später Ausgangsbasis für multivariate Analysemethoden sind (vgl. 1.4.2) – auch davor, Merkmals-Zusammenhänge aufzudecken, wo keine sind. Vor allem die Interaktivität, die Kreuz- und Pivot-Tabellen der Datenanalyse anbieten, kann dabei zu falschen Schlüssen verleiten. Man

muss sich vorher im Klaren sein, welche Hierarchie bei den Zeilenfeldern herrscht, was die abhängigen und was die unabhängigen Variablen sind und wie sich die Spaltenfelder zusammensetzen, bevor man willkürlich Spalten- und Zeilenfelder vertauscht.

Eine Tabelle, die alle diese Kriterien erfüllt, heisst pivotierbar und kann in Folge als Datentabelle verwendet werden.

2.3 Bereiche der Pivot-Tabelle in der Datentabelle

2.3.1 Zeilenfelder

In den Spalten, aus denen sich der Bereich der Zeilenfelder zusammensetzt, stehen Ausprägungen verschiedener Merkmale, wobei jede Spalte mehrere Ausprägungen zu einem Merkmal beinhaltet. Die Anzahl der verschiedenen Merkmale, deren Zusammenhang untersucht wird, entspricht der Anzahl der Gruppierungsebenen des Zeilenbereichs in der Pivot-Tabelle. In Abb. 2.1 sind es vier Merkmale:

- Wissenschaftslehre
- Wissenschaft
- Bereich
- Wissenschaftler

Die Ausprägungen des Merkmals „Bereich“ sind z.B. Chemie, Physik, usw. Die Merkmale selbst stehen in der Kopfzeile der Datentabelle und werden zu den Überschriften der einzelnen Gruppierungsebenen im Zeilenbereich der Pivot-Tabelle.

2.3.2 Spaltenfelder

Die Anzahl der Gruppierungsebenen der Spaltenfelder entspricht ebenfalls der Anzahl der Merkmale, die man untersuchen will. Die Überschriften der Spaltenfelder entsprechen allerdings bereits den Ausprägungen. In Abb. 2.1 sind es zwei Merkmale, Jahr und Monat, welche man hier untersuchen will. Davon werden nur die Ausprägungen 2006 und 2007 für das Jahr und „Jänner“ bis „April“ für „Monat“ untersucht.

id	Wissenschaftslehre	Wissenschaft	Bereich	Wissenschaftler	Monat	Jahr	Krankenstand	Urlaub	Ideen	Patente	Einnahmen	Ausgaben
206	Fachwissenschaft	Naturwissenschaften	Physik	Newton	Juli	2005	7	6	125	8	1886	30
121	Fachwissenschaft	Naturwissenschaften	Physik	Newton	Jänner	2006	0	5	142	8	193	35
122	Fachwissenschaft	Naturwissenschaften	Physik	Newton	April	2006	3	6	199	9	2417	30
123	Fachwissenschaft	Naturwissenschaften	Physik	Newton	September	2006	5	3	235	8	2501	40
124	Fachwissenschaft	Naturwissenschaften	Physik	Newton	Jänner	2006	4	1	2	10	1042	10
125	Fachwissenschaft	Naturwissenschaften	Physik	Newton	September	2006	3	6	141	8	793	25
126	Fachwissenschaft	Naturwissenschaften	Physik	Doppler	Juli	2006	1	0	188	3	254	35
127	Fachwissenschaft	Naturwissenschaften	Physik	Doppler	Juni	2006	4	2	259	0	988	10
128	Fachwissenschaft	Naturwissenschaften	Physik	Doppler	Jänner	2006	2	2	104	6	35	34
129	Fachwissenschaft	Naturwissenschaften	Physik	Doppler	März	2006	3	4	118	9	2122	30
130	Fachwissenschaft	Naturwissenschaften	Chemie	Avogadro	März	2006	6	6	50	3	2108	40
131	Fachwissenschaft	Naturwissenschaften	Chemie	Avogadro	Februar	2006	7	5	39	5	2316	40
132	Fachwissenschaft	Naturwissenschaften	Chemie	Boyle								
133	Fachwissenschaft	Naturwissenschaften	Chemie	Mendelejew								
134	Fachwissenschaft	Naturwissenschaften	Chemie									
120	Fachwissenschaft	Naturwissenschaften	Physik									
118	Fachwissenschaft	Naturwissenschaften	Physik									
117	Fachwissenschaft	Naturwissenschaften	Physik									
205	Fachwissenschaft	Naturwissenschaften	Physik									
203	Fachwissenschaft	Naturwissenschaften	Physik									
202	Fachwissenschaft	Naturwissenschaften	Physik									
201	Fachwissenschaft	Naturwissenschaften	Physik									

SPALTENFELDER											
				2006				2007			
				Jänner	Februar	März	April	Jänner	Februar	März	April
Wissenschaftslehre	Wissenschaft	Bereich	Wissenschaftler	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Patente
Fachwissenschaft	Naturwissenschaften	Physik	Newton	248	24	39	5	566	13	327	14
		Chemie	Avogadro	39	5	348	4	128	5	315	
			Boyle			39	5				
			Mendelejew			298	1				
		Physik	Doppler	248	24	218	9	199	9		
			Faraday	104	6	218	9				
			Maxwell								
			Newton	144	18			199	9		
Universalwissenschaft				353	17	526	19	899	32	797	20
	Formalwissenschaften			169	3	526	19	251	15	763	16
		Informatik		169	3	309	6	242	9	134	0
			Dijkstra			272	0	129	9	134	0
			Knuth	169	3			113	0		253
			Turing			37	6				237
		Mathematik		217	13	9	6	629	16		
			Cauchy					190	6		
			Fibonacci				9	6			
			Gauß					236	2		
			Pythagoras			217	13				
Geisteswissenschaften				184	14			648	17	34	4
		Philosophie		184	14			648	17	34	4

Abbildung 2.1: Ausschnitt einer Datentabelle und der daraus entstandenen Pivot-Tabelle.

2.3.3 Datenfelder

Die Spalten der Datenfelder entsprechen den Spalten in der Datentabelle, auch die Überschriften werden übernommen. Hier sind die Werte bereits gruppiert und summiert.

Kapitel 3

Datenstruktur Baum für den Zeilenbereich

Für den Aufbau und die Verwaltung des Zeilenbereichs einer Pivot-Tabelle eignet sich ideal ein Baum als Datenstruktur zur Speicherung der Zeilenfelder. Im Folgenden werden die Zusammenhänge näher erläutert.

3.1 Baum

3.1.1 Definition

Die folgenden Begriffsdefinitionen basieren auf Jonathan L.Gross und Jay Yellen (JLG06).

Graph

Ein **Graph** $G = (V, E)$ ist eine mathematische Struktur bestehend aus zwei endlichen Mengen, der Knotenmenge V und der Kantenmenge E . Jede **Kante** e aus E hat ein oder zwei **Knoten** v aus V zugeordnet, diese heissen die Endpunkte von e .

gerichtete Kante e

Eine **gerichtete Kante** e ist eine Kante die von einem Startknoten zu einem Endknoten führt, die Endpunkte der Kante sind ein geordnetes Paar.

Pfad

Ein Pfad ist ein Weg von einem Startknoten zu einem Endknoten, auf dem sich keine Kanten wiederholen.

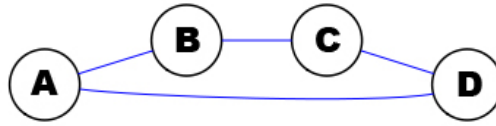


Abbildung 3.1: ungerichteter, zyklischer Graph mit der Knotenmenge $V = \{A, B, C, D\}$ und der Kantenmenge $E = \{\{A, B\}, \{B, C\}, \{C, D\}, \{A, D\}\}$.

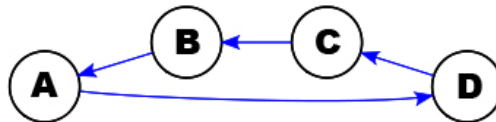


Abbildung 3.2: gerichteter, zyklischer Graph mit der Knotenmenge $V = \{A, B, C, D\}$ und der Kantenmenge $E = \{(A, B), (B, C), (C, D), (A, D)\}$.

Zyklus

Ein Zyklus ist ein Pfad von einem Startknoten zu sich selbst. Ein Graph, der Zyklen enthält, heisst zyklisch, sonst azyklisch.

Baum

Formale Definition: Ein Graph $G = (V, E)$ ist genau dann ein Baum, wenn gilt:

- G ist azyklisch
- G ist zusammenhängend

Unter dem in dieser Diplomarbeit verwendeten Begriff Baum versteht man einen gerichteten Graphen, der genau einen Knoten ohne direkten Vorgänger hat (die Wurzel) und in dem jeder Knoten von der Wurzel aus durch einen und nur einen Pfad erreicht wird (wird in der Fachliteratur als Arboreszenz bezeichnet). Die Tiefe eines Knotens ist die Länge des Pfades von der Wurzel

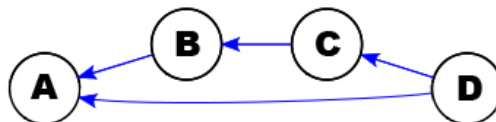


Abbildung 3.3: gerichteter, azyklischer Graph mit der Knotenmenge $V = \{A, B, C, D\}$ und der Kantenmenge $E = \{(A, B), (B, C), (C, D), (A, D)\}$, aber kein Baum (Arboreszenz).

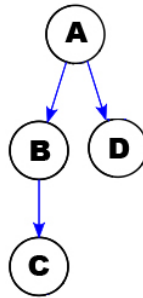


Abbildung 3.4: Baum mit der Knotenmenge $V = \{A, B, C, D\}$ und der Kantenmenge $E = \{(A, B), (B, C), (C, D), (A, D)\}$. A ist die Wurzel des Baumes, B und D sind die Kinder von A , C ist ein Kind von B . A und B sind innere Knoten, C und D sind Blätter.

zu diesem Knoten, die Höhe des Baums ist die Länge des längsten Pfades von der Wurzel. Der direkte Vorgänger eines Knotens wird als Vater bezeichnet, der direkte Nachfolger als Kind. Knoten ohne Kinder heißen Blätter, alle anderen Knoten bezeichnet man auch als innere Knoten.

3.1.2 Zeilenfelder als Baum

Die Datenstruktur Baum wird hier verwendet, um die Gruppierungsebenen des Zeilenbereiches darzustellen. Sie eignet sich sehr gut, da es mit Hilfe eines Baumes sehr einfach ist, die hierarchische Anordnung der unterschiedlichen Merkmalsausprägungen wiederzugeben. Jedes Merkmal entspricht einer Ebene im Baum (alle Knoten gleicher Tiefe befinden sich auf einer Ebene), jede Merkmalsausprägung einem Knoten im Baum, solange noch tiefere Ebenen existieren, ansonsten handelt es sich um Blätter. Diese speziellen Bezeichnungen im Baum entsprechen auch den Spezialfällen in der Implementierung: Die Wurzel, die inneren Knoten (exklusive Wurzel) und die Blätter werden jeweils gesondert behandelt.

3.1.3 Begriffsbedeutung für die Pivot-Tabelle

Wurzel

Die Wurzel entspricht nicht wirklich einem Datenobjekt aus der Datentabelle sondern ist der Überbegriff, der stellvertretend für die gesamte Pivot-Tabelle verwendet wird.

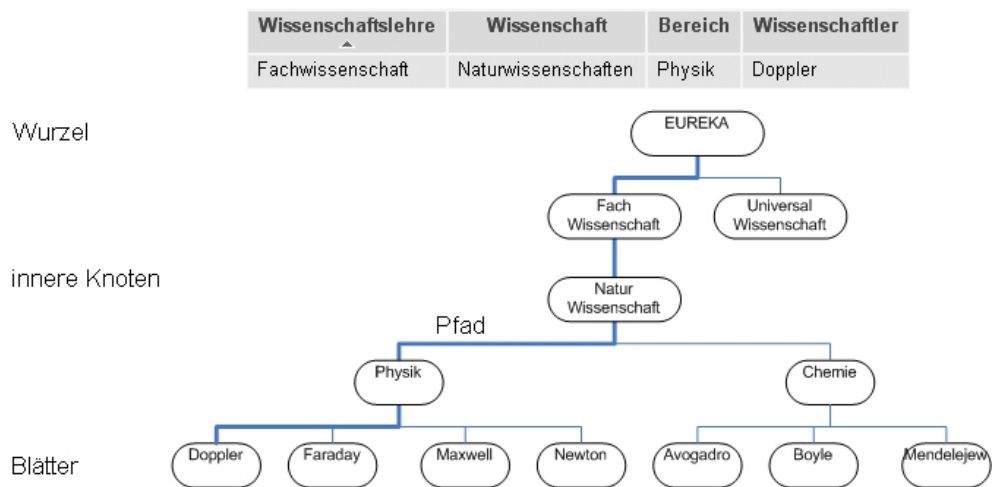


Abbildung 3.5: Eine Zeile aus einer Datentabelle und der dazugehörige Pfad im Baum.

Innere Knoten

Die Knoten entsprechen den einzelnen Merkmalsausprägungen. Knoten mit gleicher Tiefe sind Ausprägungen ein und desselben Merkmals. Ein Knoten besitzt genau einen Vater, welcher die Merkmalsausprägung der übergeordneten Ebene darstellt, und mehrere Kinder, welche die Merkmalsausprägungen der untergeordneten Ebene repräsentieren.

Blätter

Die Blätter entsprechen der letzten Gruppierungsebene. Sie besitzen keine Kinder. Ihre Tiefe entspricht der Anzahl der Spalten im jeweiligen Zeilenbereich.

Knotenpfade

Verfolgt man einen Pfad von der Wurzel zu einem Blatt, so erhält man eine Zeile aus der Datentabelle.

Abb. 3.5 zeigt ein Beispiel für einen Baum und die oben beschriebenen Begriffe.

3.2 Baumaufbau

Die Daten für die Erstellung der Pivot-Tabelle werden aus einer Datenbank gelesen und in einem Baum abgelegt. Als Abfragesprache wird SQL verwendet. Für den Aufbau des Baumes gibt es zwei Möglichkeiten.

3.2.1 Rekursiver Aufbau

Der Baum wird zuerst in die Tiefe aufgebaut. Der Aufbau erfolgt rekursiv, wie der Algorithmus `buildtree()` aufzeigt. Die einzelnen Merkmalsausprägungen werden jeweils durch folgendes SELECT ausgelesen:

```
SELECT DISTINCT 'Wissenschaft'
FROM 'datentabelle'
WHERE 'Wissenschaftslehre'='Fachwissenschaft';
```

Algorithmus 1 `buildtree` (w, MZ, L)

Eingabe: Wurzelknoten w ; Menge der Zeilenmerkmale MZ ; Anzahl der Zeilenmerkmale L

Ausgabe: Baum der Zeilenfelder

Variable(n): Index m ; Laufvariable i

```
1: //  $MZ[0]$  enthält alle Merkmalsausprägungen der ersten Gruppierungsebene
2:  $i = 0$ ;
3: für alle  $m \in MZ[0]$  {
4:    $w.\text{Kind}[i] \leftarrow m$ ;
5:   setchildren( $w.\text{Kind}[i], 1, MZ, L$ );
6:    $i \leftarrow i + 1$ ;
7: }
```

Die Prozedur `buildtree()` bekommt als Parameter die Wurzel des Baumes w übergeben und ein Array mit allen unterschiedlichen Merkmalsausprägungen $M[]$. Die Wurzel ist als Datenstruktur auch ein Knoten des Baumes, besitzt aber in der Datentabelle kein Pendant. Alle unterschiedlichen Merkmalsausprägungen aus dem Array $M[0]$ (die oberste Gruppierungsebene) werden der Reihe nach als Kinder an die Wurzel gehängt und die Unterprozedur `setchildren()` wird aufgerufen. Es wird der aktuell gesetzte Knoten und als Startebene „1“ übergeben. In der rekursiven Prozedur `setchildren()` werden wieder alle unterschiedlichen Merkmalsausprägungen $M_w[l]$ der aktuellen Ebene (Level) l , die immer inkrementiert und übergeben wird, durchgegangen. Der Vaterknoten bzw. die übergeordnete Ausprägung ist nun der

Algorithmus 2 setchildren (w, l, MZ, L)

Eingabe: aktueller Knoten w ; aktuelle Gruppierungsebene l ; Menge der Zeilenmerkmale MZ ; Anzahl der Zeilenmerkmale L

Ausgabe: Baum der Zeilenfelder

Variable(n): Index m ; Laufvariable i

```
1: //  $MZ_w[l]$  enthält alle Merkmalsausprägungen der  $l$ -ten Gruppierungsebene, die als Vaterknoten(= übergeordnete Merkmalsausprägung)  $w$  besitzen
2:  $i = 0$ ;
3: für alle  $m \in MZ_w[l]$  {
4:    $w.\text{Kind}[i] \leftarrow m$ ;
5:   falls  $l < L$  dann {
6:     setchildren( $w.\text{Kind}[i], l + 1, MZ, L$ );
7:      $i \leftarrow i + 1$ ;
8:   }
9: }
```

übergebene Knoten. Beim SELECT werden immer alle übergeordneten Merkmale abgefragt, um mehrfach vorkommende Merkmalsausprägungen in unterschiedlichen Teilbäumen richtig zu behandeln. Solange nicht die letzte Gruppierungsebene erreicht wurde, wird die Rekursion fortgesetzt.

Abb. 3.6 verdeutlicht den Aufbau eines Baumes für den Zeilenbereich der hier verwendeten Beispiel-Pivot-Tabelle. Als erstes wird der Knoten *Fachwissenschaft* angelegt. Dies passiert noch in `buildtree()`. Danach, ab diesem Punkt als rekursive Abarbeitung in `setchildren()`, wird sein erstes Kind *Naturwissenschaft*, dessen erstes Kind *Physik* und zuletzt das erste Blatt *Newton* angelegt. Weitere Blätter des Knotens *Physik* sind *Doppler*, *Faraday* und *Maxwell*. Als nächstes findet die Rekursion mit *Chemie* ein weiteres Kind von *Naturwissenschaft*, mit dazugehörigen Blättern, usw.

3.2.2 Nur ein SELECT

Die zweite Möglichkeit den Baum aufzubauen, ist mit Hilfe eines einzigen SELECT der Form:

```
SELECT DISTINCT 'Wissenschaftslehre', 'Wissenschaft',
                'Bereich', 'Wissenschaftler'
FROM 'datentabelle';
```

Dieser liefert als Ergebnis die ganze Einteilung des Zeilenbereichs. Die Zeilenanzahl entspricht der Anzahl der Blätter in dem Baum. Abb. 3.7 zeigt

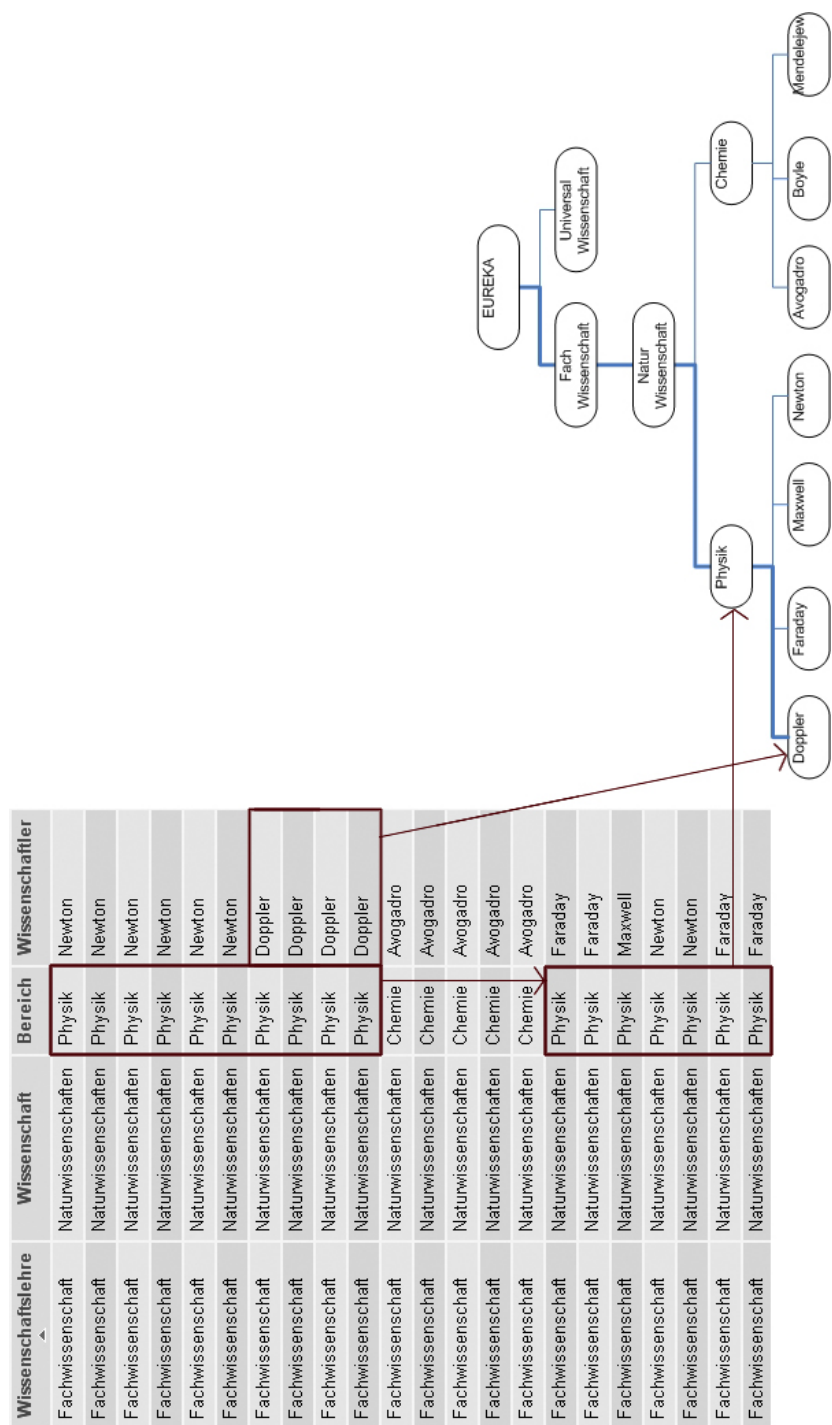


Abbildung 3.6: Teilausschnitt einer Datentabelle und des entsprechenden Baumes.

Wissenschaftslehre	Wissenschaft	Bereich	Wissenschaftler
Universalwissenschaft	Formalwissenschaften	Informatik	Turing
Universalwissenschaft	Formalwissenschaften	Informatik	Dijkstra
Universalwissenschaft	Formalwissenschaften	Informatik	Knuth
Universalwissenschaft	Formalwissenschaften	Mathematik	Pythagoras
Universalwissenschaft	Formalwissenschaften	Mathematik	Gauß
Universalwissenschaft	Formalwissenschaften	Mathematik	Fibonacci
Universalwissenschaft	Formalwissenschaften	Mathematik	Cauchy
Fachwissenschaft	Naturwissenschaften	Physik	Faraday

Abbildung 3.7: Ergebnis einer SQL Abfrage nach dem ganzen Zeilenbereich.

einen Ausschnitt daraus. Diese SQL Ergebnisse werden nun zeilenweise Abgearbeitet und alle Einträge – falls noch nicht vorhanden – in einen anfangs leeren Baum eingefügt. Für die in der Abb. 3.7 gezeigten Ergebniszeilen würde nun zuerst der Knoten *Universalwissenschaft*, dann sein Kind *Formalwissenschaften*, dessen Kind *Informatik* und schließlich das Blatt *Turing* angelegt. Bei der Abarbeitung der zweiten Ergebniszeile wird der bereits bestehende Knotenpfad *Universalwissenschaft* $\rightarrow$ *Formalwissenschaften* $\rightarrow$ *Informatik* verfolgt und das neue Blatt *Dijkstra* angelegt; genauso verfährt man beim Blatt *Knuth*. Zeile 4 des Ergebnisbereichs legt bereits beim Knoten *Formalwissenschaften* ein neues Kind *Mathematik* an, usw.

Diese Methode zum Anlegen des Baumes erfordert zwar nur ein einziges SQL-Statement, braucht aber viel Verarbeitungsoperationen beim Aufbau des Baumes. Für jede Zeile des Ergebnisbereichs werden bis zur vorletzten Ebene Abfragen durchgeführt, ob der jeweilige Knoten bereits angelegt wurde. Erst die letzte Spalte des Ergebnisbereichs, die gleichzeitig die letzte Ebene der Zeilenfelder darstellt, kann direkt als Knoten angelegt werden, da diese Einträge sich nicht wiederholen.

Um den Baumaufbau möglichst performant zu gestalten wird pro Ebene ein Array verwaltet, in dem für jeden Knoten, der bereits angelegt wurde, ein Eintrag existiert. Der Zugriff auf diesen Eintrag erfolgt über einen eindeutigen Index, der alle Merkmalsausprägungen der darüberliegenden Ebenen vereint. Somit muss zur Abfrage, ob ein Knoten existiert, nur ein Vergleich auf die Existenz im entsprechenden Array getätigt werden.

Die Anzahl der Vergleichsoperationen beim Baumaufbau dieser Methode ist gleich dem Produkt aus der Höhe ($h(t)$) des Baums (= Anzahl der verschiedenen Merkmale in den Zeilenfeldern) und der Anzahl der Blätter: $\text{Comp} = (h(t) - 1) \cdot \text{Blattanzahl}$.

3.2.3 Analyse

Die erste Methode der vielen SELECT-Statements liefert nur die Knoten, die direkt aus SQL ohne weitere Behandlung in den Baum eingefügt werden können. Sie ist vor allem dann vorzuziehen, wenn die letzte Gruppierungsebene des Zeilenbereichs viele unterschiedliche Ausprägungen hat. In diesem Fall benötigt die zweite Methode ungleich mehr Abarbeitungsoperationen. Das Beispiel einer statistischen Analyse, welche nach Bundesländern und dann direkt nach Haushalten gliedert, würde bei der ersten Methode nur 9 SELECTs für die österreichischen Bundesländer brauchen und ca. 3,5 Millionen (geschätzte Anzahl der Haushalte nach (Sta08)) Abarbeitungsschritte. Das ist natürlich ein Extremfall, soll aber das zugrundeliegende Problem verdeutlichen. Da es bei reellen Untersuchungen, die als Pivot-Tabellen dargestellt werden sollen, selten sehr viele Ausprägungen der unterschiedlichen Merkmale geben wird, werden die Laufzeitunterschiede der beiden Methoden gering sein. Im Folgenden werden beide Methoden als gleichwärtig angesehen und es wird, ausser bei den Untersuchungsergebnissen, nicht mehr zwischen ihnen unterschieden.

Kapitel 4

Rekursiver Aufbau des Spaltenbereichs

Die Schwierigkeit beim Aufbau des HTML Codes für die Ausgabe der Tabelle ist die Reihenfolge, in der die Informationen abgefragt werden können, im Gegensatz zu der Reihenfolge, in der sie ausgegeben werden. Eine HTML Seite wird immer zeilenweise nach dem top-down Prinzip verarbeitet und muss daher auch so aufgebaut werden. Das bedeutet speziell für die Pivot-Tabelle, dass immer zuerst eine ganze Zeile generiert werden muss, bevor man in die nächste springen kann. Das ist durchaus ein Problem, da die Werte für die Pivot-Tabelle nicht immer in der benötigten Reihenfolge zur Verfügung stehen. Bereits beim Aufbau des Spaltenbereichs ergibt sich ein damit verbundenes Problem.

4.1 Herkunft der Spaltenbezeichnungen

Wie in 2.3.2 bereits kurz angedeutet, werden die Spaltenfelder aus einzelnen Spalten der Datentabelle aufgebaut. Das bedeutet, dass jede Gruppierungsebene und damit jede Zeile des Spaltenbereichs aus den Einträgen in einer Spalte, d.h. den Ausprägungen eines Merkmals besteht. Der Anwender bestimmt die Spalten der Datentabelle, die für den Spaltenbereich herangezogen werden sollen. Die Anzahl der bestimmten Spalten ergibt dann die Anzahl der Gruppierungsebenen c . Die Spaltenüberschriften einer Gruppierungsebene sind dann, wenn nicht anders angegeben, die jeweils unterschiedlichen Einträge einer Spalte. Es besteht die Möglichkeit diese mit Hilfe eines Filters einzuschränken. In der Beispieldatentabelle werden die Spalten Jahr und Monat ausgewählt. Die Datentabelle umfasst die Erhebung für die Jahre 2005-2007 für jeweils alle Monate, die Pivot-Tabelle soll aber nur eine Zusam-

menfassung über jeweils das erste Trimester der Jahre 2006 und 2007 liefern. Die letzte Gruppierungsebene ist bereits die Kopfzeile des Datenbereichs, wird aber bei den unten folgenden Berechnungen ebenfalls herangezogen. Sie besteht – anders als die Ebenen des Spaltenbereichs – nicht aus den Ausprägungen eines Merkmals sondern aus unterschiedlichen Merkmalen, vgl. Kapitel 5.

4.2 colspan

Bei mehreren Gruppierungsebenen und jeweils unterschiedlich vielen Merkmalsausprägungen in den jeweiligen Ebenen besteht der Bereich der Spaltenfelder aus verschiedenen großen Zellen. Es muss sich z.B. die erste Ausprägung des ersten Merkmals über alle ihr untergeordneten Ausprägungen des zweiten Merkmals erstrecken. Ein Beispiel dazu zeigt Abb. 4.1. Der Wert für das Zellenattribut `colspan` (zum Verbinden mehrerer Spalten) der HTML Tabelle muss also zuerst für jede Ebene des Spaltenbereichs berechnet werden.

	2006								2007		
	Jänner		Februar		März		April		Jänner		Feb
Wissenschaftslehre ±	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ide
Fachwissenschaft ±	3585	264	1952	127	5013	247	2530	206	2377	167	553
Universalwissenschaft ±	4555	249	6509	363	9184	626	6489	442	3994	301	454

Abbildung 4.1: Die Merkmalsausprägungen der verschiedenen Ebenen des Spaltenbereichs erstrecken sich jeweils über unterschiedlich viele Spalten. Die unterste Ebene mit den Überschriften des Datenbereichs hat pro Merkmalsausprägung immer nur eine Spalte, d.h. die *Ideen* sowie *Patente* sind immer nur eine eigene Spalte. Jede Merkmalsausprägung der vorletzten Ebene fasst diese Variablen zusammen, d.h. je ein *Monat* erstreckt sich immer über die 2 Spalten mit den *Ideen* und den *Patenten*. Die Ebene davor, die *Jahre*, liegt wiederum jeweils über allen zugehörigen *Monaten*, d.h. je ein *Jahr* erstreckt sich über 4 *Monate* zu 2 Variablen, also 8 Spalten, usw.

Es gilt: Jede Ausprägung hat als Anzahl der verbundenen Zellen einerseits die Anzahl der ihr untergeordneten Ausprägungen, andererseits können sich diese wiederum über mehrerer Zellen erstrecken. Die Gesamtanzahl `cols` der verbundenen Zellen pro Spaltenebene `i`, wobei 1 die oberste Ebene ist, ergibt sich aus:

$$\text{cols}[i] = \text{sizeof}(\text{columnarray}[i+1]) \cdot \text{cols}[i+1] \quad (4.1)$$

`cols[i]` bezeichnet hier die Anzahl der zu verbindenden Zellen in der i -ten Spaltengruppierungsebene und `sizeof(columnarray[i])` die jeweilige Anzahl der Merkmalsausprägungen in Ebene i . Die gleichen Bezeichnungen werden im folgenden Algorithmus zur Berechnung der verbundenen Spalten in den jeweiligen Spaltengruppierungsebenen verwendet:

Algorithmus 3 `colspan ()`

Eingabe: globales Array `cols[]`; globales array `columnarray[]`

Ausgabe: Anzahl der verbundenen Zellen pro Spaltenebene i gespeichert in `cols[i]`

Variable(n): Laufvariable i ; Anzahl der Ebenen c

```

1:  $c \leftarrow \text{sizeof}(\text{columnarray}) - 1$ ;
2: cols[c] ← 1;
3: für  $i = c - 1 \dots 1$  {
4:   cols[i] ← sizeof(columnarray[i + 1]) · cols[i + 1];
5: }
```

Die allerunterste Ebene c hat keine verbundenen Spalten und erstreckt sich somit über eine Spalte, d.h. `cols[i]` mit i gleich c wird auf 1 gesetzt. Mit dieser Ebene fängt die Berechnung der jeweiligen `cols[i]` an.

Die Gesamtanzahl der Spalten C im Spaltenbereich ist dann gleich dem Produkt aus der Anzahl der unterschiedlichen Ausprägungen in jeder Gruppierungsebene ($|\text{MS}[i]|$).

$$C = \prod_{i=1}^c |\text{MS}[i]| = \text{cols}[1] \cdot \text{sizeof}(\text{columnarray}[1]) \quad (4.2)$$

Da der Spaltenbereich gleichzeitig als Kopfzeile für den Datenbereich dient, ist die Gesamtanzahl der Spalten C im Spaltenbereich gleichzeitig die Gesamtanzahl der Spalten im Datenbereich.

4.3 HTML Code

Um das Problem der HTML-Ausgabe der Spaltenfelder zu lösen, baut man die jeweiligen Codesegmente rekursiv auf. Diese werden in Arrays abgespeichert und zum gegebenen Zeitpunkt an den richtigen Stellen wieder ausgelesen.

Beim rekursiven Aufbau wird wieder in der tiefsten Ebene begonnen. Für jede Ausprägung wird der HTML-Code für eine Zelle generiert:

`<td colspan=cols[aktuelle Ebene  $l$ ]>Merkmalsausprägung  $m$ </td>`.

Algorithmus 4 `htmlgenerate( $l, c, \text{cols}[], M$ )`

Eingabe: aktuelle Gruppierungsebene l ; Anzahl der Spaltenebenen c ; Array mit den Anzahlen der verbundenen Spalten `cols[]`; Menge der Spaltenmerkmalsausprägungen MS

Ausgabe: Array mit HTML-Code `html[]`

Variable(n): Hilfsarray `hilf[]`; Laufvariablen j, i ; Kardinalität der Menge MS k

```
1: falls  $l < c$  dann {
2:   htmlgenerate( $l + 1, c, \text{cols}[], M$ )
3: }
4: für alle  $m \in MS[l]$  {
5:   html[l]  $\leftarrow$  html[l]  $\oplus$  <td colspan=cols[1]>m</td>;
6:   //  $\oplus$  wird als Symbol für Stringkonkatination verwendet
7: }
8: falls  $l > 1$  dann {
9:   für  $j = l \dots 2$  {
10:     $k \leftarrow |MS[j - 1]|;$ 
11:    für  $i = 0 \dots k$  {
12:      falls  $j == l$  dann {
13:        hilf[j - 1]  $\leftarrow$  hilf[j - 1]  $\oplus$  html[l];
14:      } sonst {
15:        hilf[j - 1]  $\leftarrow$  hilf[j - 1]  $\oplus$  hilf[l];
16:      }
17:    }
18:  }
19: }
```

Für jede Zeile des Spaltenbereichs der Pivot-Tabelle muss dieser HTML-Code so oft aneinandergesetzt werden wie es in dieser Ebene Spalten gibt.

Aufruf und Ausgabe

Nachdem der HTML-Code mittels `htmlgenerate()` generiert wurde (der Aufruf erfolgt ein einziges Mal mit der Ebene 1), kann der HTML-Code für die Pivot-Tabelle zeilenweise ausgegeben werden. Vor dem Spaltenbereich (oberhalb des Zeilenbereichs) muss nur jeweils die korrekte Anzahl an Leerzellen entsprechend der Gruppierungsebenen im Zeilenbereich ausgegeben werden.

Kapitel 5

Datenfelder

Die eigentlichen Gruppierungen, Berechnungen bzw. Auswertungen der Werte passieren im Datenbereich. Die einzelnen Werte des Datenbereichs sind aufsummierte Werte einzelner Zellen aus der Datentabelle, wobei immer nur Werte einer Spalte aber mehrerer Zeilen summiert werden. Für die Werte des Datenbereichs, die zum Berechnen einer Datenzelle herangezogen werden, gelten die Ausprägungen des jeweiligen Zeilenfeldes des Zeilenbereichs und des jeweiligen Spaltenfeldes des Spaltenbereichs. Demnach muss für die Zeilen der Datentabelle, die den in Abb. 5.2 markierten Datenwert ergeben, folgendes gelten:

Wissenschaftslehre	Wissenschaft	Bereich	Wissenschaftler	Monat	Jahr	Ideen
Fachwissenschaft	Naturwissenschaften	Physik	Newton	Jänner	2006	142
Fachwissenschaft	Naturwissenschaften	Physik	Newton	Jänner	2006	2

Abbildung 5.1: Zeilen einer Datentabelle, welche die Bedingungen für die Zelle des Datenbereichs der Pivot-Tabelle 5.2 erfüllen.

Möglichkeiten für den Datenbereich

Damit Datenauswertungen mit Pivot-Tabellen überhaupt erst interessant werden, benötigt man große Datenmengen, die zu gruppieren auch sinnvoll sind. Daraus ergibt sich aber natürlich ein Performanceproblem, da all diese Daten mit SQL Abfragen aus der Datenbank extrahiert, mit Hilfe von PHP Operationen ausgewertet, und dann schließlich noch mit HTML dargestellt werden müssen. Zusätzlich kommt noch die Übertragung der Daten zwischen den beteiligten Systemen hinzu. Möchte man die Pivot-Tabelle komplett flexibel gestalten, sodass jederzeit die Möglichkeit besteht, mehr oder

		SPALTENFELDER											
ZEILENFELDER				2006								2007	
				Jänner		Februar		März		April		Jänner	
				Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Patente
				248	24	39	5	566	13	327	14	315	1
	Wissenschaftslehre ±	Wissenschaft ±	Bereich ±	Wissenschaftler									
	Fachwissenschaft ±				248	24	39	5	566	13	327	14	315
		Naturwissenschaften ±			248	24	39	5	566	13	327	14	315
			Physik ±		248	24			218	9	199	9	
				Doppler	104	6			218	9			
				Faraday									
				Maxwell									
				Newton	144	18				199	9		
	Universalwissenschaft ±				353	17	526	19	899	32	797	20	786
		Formalwissenschaften ±			169	3	526	19	251	15	763	16	490
			Informatik ±		169	3	309	6	242	9	134	0	490

Abbildung 5.2: Datenzeile einer Pivot-Tabelle mit Aussage über die Anzahl der Ideen im Jänner 2006 (Bedingungen für den Spaltenbereich) der Forschungsgruppe „Newton“ aus dem Bereich Physik, gehörend zu den Naturwissenschaften und diese wiederum zu den Fachwissenschaften (Bedingungen des Zeilenbereichs).

weniger Ebenen anzuzeigen, dann gibt es für die Realisierung verschiedene Möglichkeiten, wie die Werte für den Datenbereich berechnet werden können. Das Endresultat ist für alle Methoden gleich. Dessen Ausgabe kann getrennt erfolgen, in dieser Arbeit wird hierfür HTML verwendet, es ist aber ebenso möglich, andere Benutzerschnittstellen (z.B. Java, XML oder Tabellenkalkulationsprogramme) anzusteuern.

Reine SQL Abfragen

Um die Werte für den Datenbereich zu erhalten, verwendet man ausschließlich SQL-Abfragen. Ein Zwischenspeicher wird nur für das kurzfristige Abspeichern des Ergebnisses einer einzigen SQL-Abfrage bis zur Ausgabe in HTML verwendet. Weiterer Zwischenspeicher wird hier nicht benötigt. Auch die Berechnungen erfolgen direkt über einzelne SQL-Abfragen.

Zwischenspeicher

Bereits bei der Erstellung des Baumes für die Zeilenfelder können alle Werte in einen Zwischenspeicher gelegt und beim Aufbau der HTML Seite direkt aus dem Speicher berechnet werden. Dabei wird sozusagen eine Kopie der benötigten Datentabelle bzw. Teile davon gemacht. Die Berechnungen erfolgen mittels PHP.

Mischung aus SQL Abfragen und Zwischenspeicher

Zuviele SQL-Statements können sich negativ auf die Laufzeit auswirken, daher ist prinzipiell eine Variante mit wenigen SQL-Abfragen zu bevorzugen.

Dies kann aber wiederum speicher- und berechnungsintensiv sein. Je nach Aggregationsverfahren und Zusammensetzung der Daten kann es sinnvoll sein, beide Methoden miteinander zu kombinieren.

Fragestellungen

In den folgenden Kapiteln werden die unterschiedlichen Methoden diskutiert und miteinander verglichen. Als Ausgangstabelle für die Berechnungen wird zuerst eine kleine, einfache Tabelle herangezogen (vgl. Abb. 5.3 und 5.4). Diese soll der Veranschaulichung dienen und die Berechnungen und Vergleiche vereinfachen, bildet aber nicht im geringsten den Umfang des Problems ab. Anschließend werden die gefundenen Lösungen anhand großer Datenmengen nochmals evaluiert.

Die Fragen, die hier vor allem beantwortet werden müssen, sind:

- Wovon hängt die Laufzeit des Aufbaus einer Pivot-Tabelle maßgeblich ab?
- Ist die Summation über PHP günstiger oder die direkte Summenbildung über SQL?
- Was ist vorzuziehen: viele gezielte SQL-Abfragen oder wenige SQL-Abfragen mit aufwendiger Berechnung im Programm?
- Hängt die Antwort auf die letzte Frage von der Anzahl und Struktur der analysierten Daten ab?
- In wie weit spielt die Anzahl der Merkmale in den Zeilen- und Spaltenfeldern eine Rolle?
- Welchen Einfluss hat die Anzahl der Merkmalsausprägungen?

			2006				2007			
			Jänner	Februar	März	April	Jänner	Februar	März	April
Wissenschaftstheorie +	Bereich +	Wissenschaftler	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Patente
			3895	264	1952	127	5013	247	2530	205
Fachwissenschaft +	Naturwissenschaften +	Chemie +	3895	264	1952	127	5013	247	2530	205
			1952	127	3432	156	1217	99	2377	167
		Arquadio			1952	127	1638	83	1217	99
		Boile			1952	127			2377	167
		Mendelejew			1794	73			1718	91
		Doppler	3895	264	1561	91	1313	107	3813	198
		Faraday	1128	100	1581	91			3165	215
		Maxwell								
Universalwissenschaft +	Formalwissenschaften +	Informatik +	2457	164			1313	107	3813	198
			1952	127	4509	853	5184	536	4060	354
		Dijkstra	1533	105	4509	853	4555	551	5424	358
		Knuth	1633	105	3886	112	2780	121	3099	204
		Turing			1670	116	1404	111	2617	212
		Cauchy			1716	96			1148	112
		Fibonacci			2923	151	1625	130	1469	100
		Gauß			1625	130			3099	204
		Pythagoras							1334	106
		Aristoteles	2922	144	4829	275	1005	84	1765	98
		Platon	2922	144	4829	275	1005	84	1441	110
		Sokrates	1233	97					1441	110
		Stoa	1689	47	2897	195			1227	96
							1377	89	1634	75
					1932	80			1811	108

Abbildung 5.3: Ergebnis-Pivot-Tabelle erstellt aus einer Datentabelle mit 1000 Einträgen ähnlich der Datentabellen aus Abb. 2.1. Hier werden wieder die Erfolge, aufgeteilt in Ideen und Patente, unterschiedlicher Forschungsgruppen – benannt nach namhaften Wissenschaftlern – analysiert. Die Analyse erstreckt sich über jeweils das erste Trimester der Jahre 2006 und 2007. Die Forschungsgruppen sind zugeteilt zu verschiedenen Bereichen, wie z.B. Physik, Mathematik usw., diese zu den entsprechenden Wissenschaften, die nach Formal- und Universalwissenschaften gegliedert sind.

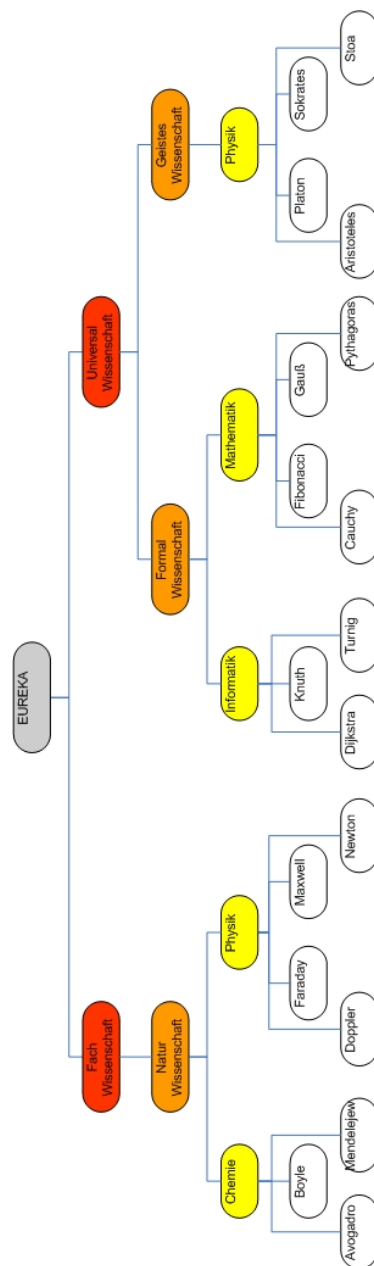


Abbildung 5.4: Der zu der in Abb. 5.3 dargestellten Pivot-Tabelle gehörende Gliederungsbaum; die jeweiligen Ebenen sind farblich entsprechend der Farben in den Zeilenebenen markiert.

Kapitel 6

Implementierung mittels einzelner SQL-Abfragen

Bei der Methode reiner SQL-Abfragen werden die Daten für den Datenbereich, wie der Name schon sagt, ausschliesslich über Datenbankabfragen gewonnen. Ein Zwischenspeicher wird nur verwendet, so weit dies für das Abspeichern der Ergebnisse einer SQL-Abfrage notwendig ist. Die Berechnungen erfolgen direkt durch SQL.

6.1 Performance

Die Verarbeitungsgeschwindigkeit hängt von verschiedenen Faktoren ab. Die **Größe der Datentabelle** ist natürlich ausschlaggebend. Je größer die Datentabelle ist, desto länger braucht eine einzelne SQL-Abfrage, unabhängig davon, wie die Daten selbst beschaffen sind und wie die Tabelle aufgebaut ist. Besteht die Datentabelle aus **vielen unterschiedlichen Merkmalsausprägungen** in den Spalten, welche für den Zeilenbereich herangezogen werden (es entstehen viele Zeilen in der Pivot-Tabelle, weil jeweils wenige Datenwerte aus der Datentabelle zusammengefasst werden), benötigt man bei der selben Datentabellengröße mehr SQL-Abfragen bzw. mehr Auswertungen als bei einer Datentabelle mit wenigen Merkmalsausprägungen in den Zeilenfeldern.

Auch die Anzahl der Spaltenfelder, d.h. der Ebenen im Spaltenbereich und deren jeweilige Merkmalsausprägungen, bestimmt die Laufzeit. All diese Merkmale, egal ob aus dem Spalten- oder Zeilenbereich, müssen auf die gewünschten Ausprägungen abgefragt werden. Dies geschieht in der WHERE-Klausel des SELECT-Statements. Die letzte Spaltengruppierungsebene, also die Variablenüberschriften des Datenbereichs, wird im Gegensatz dazu nicht in der

WHERE-Klausel sondern im SELECT-Teil aufgebaut und ist bei jeder SQL-Abfrage gleich. In diesem Beispiel lautet diese

```
SELECT SUM('Ideen'), SUM('Patente') ...
```

Sollte es noch eine zusätzliche Spaltenpalte geben, wie das z.B. bei MS Excel der Fall ist, hier aber aufgrund der Überlegungen aus Kapitel 1.5.2 nicht gemacht wird, kann man das SELECT-Statement um diese Summen erweitern und ohne weiteren Aufwand in die Pivot-Tabelle einbauen:

```
SELECT SUM('Ideen'), SUM('Patente'), SUM('Ideen'+ 'Patente') ...
```

Es gibt nun verschieden Möglichkeiten SQL einzusetzen, um die Ergebnisse für die Datenfelder der Pivot-Tabelle zu erhalten.

6.2 Durchführungsmöglichkeiten

6.2.1 Ein SQL-Statement pro Knoten und Spaltenfeld

Eine simple, aber sicher nicht performante Möglichkeit ist es, für jeden inneren Knoten und jedes Blatt (also pro einzelner Datenzeile) eine eigene SQL-Abfrage abzusetzen. Die Anzahl der SQL-Abfragen entspricht somit der Gesamtanzahl der Datenzellen, also dem Produkt aus Zeilenfeldern und Spaltenfeldern. Dieser triviale Ansatz ist MS Excel nachempfunden. Dort wird beim Einrichten einer Pivot-Tabelle Zeile für Zeile und Spalte für Spalte vorgegangen.

Ein Beispiel zur Veranschaulichung:

Angenommen man untersucht in dem in Tabelle 5.3 beschriebenen Beispiel nur die Bereiche (*Chemie*, *Physik*, etc.) und spaltet nicht noch tiefer auf die einzelnen Wissenschaftler auf, dann besitzt der zugrundeliegende Baum nur mehr 5 Blätter und insgesamt 10 Knoten statt ursprünglich 28, obwohl die Datentabellengröße unverändert bleibt. Das reduziert zwangsläufig die Anzahl der SQL-Abfragen. Wenn dann auch wirklich nur mehr die Bereiche und nicht deren Zuordnung zu Wissenschaften und zur Wissenschaftslehre interessant sind (und man deren summierte Werte nicht benötigt), bleiben nur mehr die Blätter des Baumes für die Generierung der SQL Abfragen, und das sind 5.

Je mehr Gruppierungsebenen man in den Spalten analysieren möchte, umso laufzeitintensiver wird die Auswertung. Die Anzahl der SQL Abfragen pro Knoten im Baum multipliziert sich mit der Anzahl der Spalten der vorletzten Spaltengruppierungsebene. (vgl. Gleichung 4.2, wobei hier nur bis $c - 1$ multipliziert wird). Dieser Bereich wird als Teil der WHERE-Klausel in

die SQL-Abfrage mit Hilfe der Funktion `sqlgenerate()` (siehe weiter unten) eingebaut. Dabei unterscheiden sich die einzelnen WHERE-Klauseln eben bis zur vorletzten Ebene. Für das Beispiel in 5.3 ergeben sich 8 verschiedene Spaltenbereichsbedingungen:

```
... AND 'Jahr'='2006' AND 'Monat'='Jänner'
... AND 'Jahr'='2006' AND 'Monat'='Februar'
... AND 'Jahr'='2006' AND 'Monat'='März'
... AND 'Jahr'='2006' AND 'Monat'='April'
... AND 'Jahr'='2007' AND 'Monat'='Jänner'
... AND 'Jahr'='2007' AND 'Monat'='Februar'
... AND 'Jahr'='2007' AND 'Monat'='März'
... AND 'Jahr'='2007' AND 'Monat'='April'
```

Zusammengefasst ergibt sich die Anzahl der SQL-Abfragen für eine Implementierung ohne Zuhilfenahme von Zwischenspeicher aus dem Produkt aus Anzahl aller Knoten im Gliederungsbaum, multipliziert mit der Anzahl der Spalten der vorletzten Spaltengruppierungsebene.

$$\text{sql} = |V| \cdot \prod_{i=1}^c |\text{MS}[i]| \quad (6.1)$$

$|V|$ sei die Anzahl der inneren Knoten und Blätter im Baum und $\prod_{i=1}^c |\text{MS}[i]|$ die Anzahl der Spalten (ohne Variablenüberschriften) (vgl. Gleichung 4.2). Für das hier behandelte Beispiel wären das, wenn alle Zeilengruppierungsebenen analysiert werden sollen, $28 \cdot 8 = 224$ SQL-Abfragen. Wenn sich nun die Anzahl der Wissenschaftler, Bereiche, Wissenschaften usw. nicht ändert, d.h. die Anzahl der Merkmalsausprägungen der jeweiligen Merkmale gleich bleibt, die Datentabellengröße aber zunimmt, da z.B. weitere Untersuchungsergebnisse gesammelt wurden, bleibt die Anzahl der SQL-Abfragen konstant.

Jedes SQL-Statement hat die Form:

```
SELECT SUM('Ideen'),SUM('Patente')
FROM 'datentabelle'
WHERE 'Wissenschaftler'='Newton'
AND 'Bereich'='Physik'
AND 'Wissenschaft'='Naturwissenschaften'
AND 'Wissenschaftslehre'='Fachwissenschaft'
AND 'Jahr'='2006' AND 'Monat'='Jänner'
```

Die Abfrage liefert genau eine Zeile mit den bereits aufsummierten Daten. Diese Ergebnisse werden im Zwischenspeicher abgelegt. Die Anzahl der zu untersuchenden Variablen bestimmt die Anzahl der zu speichernden Ergebnisse, in diesem Beispiel also zwei.

SUM(Ideen)	SUM(Patente)
2457	164

Abbildung 6.1: Ergebnis einer SQL-Abfrage für einen einzelnen Knoten und der vorletzten Spaltengruppierung.

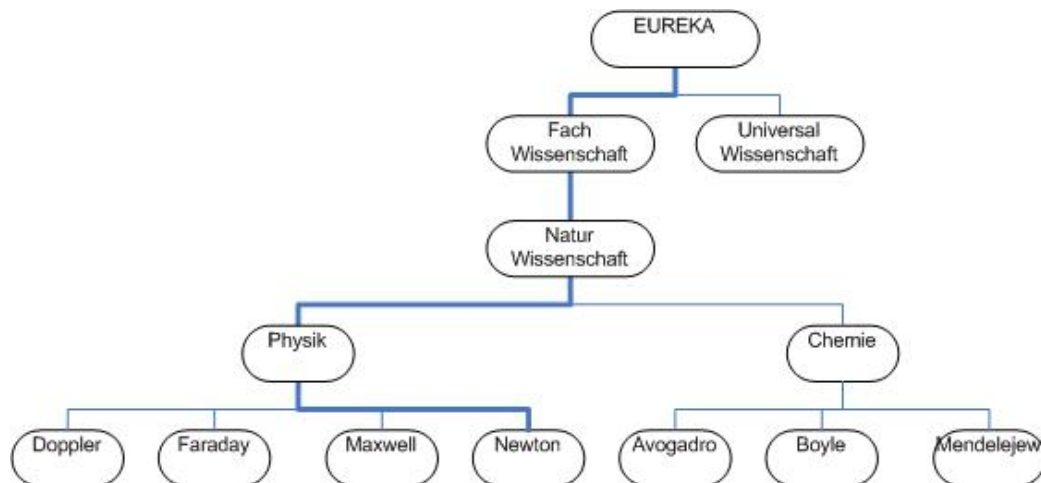


Abbildung 6.2: Pfad im Baum, der einer Zeilengruppierung entspricht.

Implementierung

Zeilenfelder als Attribut des Knotens

Bereits beim Aufbau des Baumes werden für jeden Knoten, egal ob innerer Knoten oder Blatt, die passenden Ausprägungen der Zeilenfelder als Teil einer SQL-WHERE-Klausel abgespeichert. Diese Ausprägungen entsprechen genau dem Pfad, den man von der Wurzel des Baumes bis zum jeweiligen Knoten folgen muss. Da dies auch dem Weg entspricht, der beim Aufbau des Baumes gegangen wird, d.h. in die Tiefe, werden die jeweiligen Ausprägungen der Vaterknoten gleich in der richtigen SQL-Formatierung beim jeweiligen Kind mitgespeichert. Abb. 6.2 zeigt einen solchen Pfad. Beim Aufbau des Baumes wird entlang dieses Pfades folgende WHERE-Klausel generiert: Der Wurzelknoten, Ebene 0, dient nur zur Unterscheidung verschiedener Pivot-Tabellen. Beim Aufbau des Astes, der in der Abbildung dick eingezeichnet ist, übergibt der Vaterknoten *Fachwissenschaft* an seine Kinder:

WHERE 'Wissenschaftslehre'='Fachwissenschaft'

Sein einziges Kind *Naturwissenschaft* ergänzt dieses Statement und übergibt:

```
WHERE 'Wissenschaftslehre'='Fachwissenschaft'
AND 'Wissenschaft'='Naturwissenschaft'
```

Vom Bereich *Physik* wird

```
WHERE 'Wissenschaftslehre'='Fachwissenschaft'
AND 'Wissenschaft'='Naturwissenschaft'
AND 'Bereich'='Physik'
```

an die Blätter übergeben, usw.

Aufbau des SQL Statements für den Spaltenbereich

Der Spaltenbereich muss gesondert behandelt werden. Ähnlich wie beim HTML-Code für die Tabelle des Spaltenbereichs wird der Teil der WHERE-Klausel des SQL-Statements, der den Spaltenbereich abbildet, rekursiv aufgebaut und in ein Array abgespeichert. Dabei werden alle unterschiedlichen Varianten der Zusammensetzungen der Bedingungen für die Ausprägungen des Spaltenbereichs abgelegt. Es wird in der obersten Gruppierungsebene begonnen und, ähnlich wie beim Aufbau des Baumes für den Zeilenbereich, immer zuerst in die Tiefe gegangen.

Das endgültige SELECT-Statement für einen Knoten wird dann während des Aufbaus der HTML-Seite aus den verschiedenen Teilen zusammengesetzt und ausgeführt.

Die Algorithmen 5, 6 und 7 beschreiben überblicksmäßig den Ablauf der ersten Methode. Das Zusammenspiel der wichtigsten Funktionen wird im Kapitel 9.3 beschrieben.

6.2.2 Ein SQL-Statement pro Knoten

Die Aufteilung auf die einzelnen Spaltengruppierungsfelder kann auch entfallen, denn alle Werte einer Zeile des Datenbereichs können in einem einzigen SQL-Statement abgefragt werden. Dazu müssen die Merkmale des Spaltenbereichs im SELECT-Teil eingefügt werden.

```
SELECT 'Jahr', 'Monat', SUM('Ideen'),SUM('Patente')
FROM 'datentabelle'
WHERE 'Wissenschaftler'='Newton'
AND 'Bereich'='Physik'
AND 'Wissenschaft'='Naturwissenschaften'
AND 'Wissenschaftslehre'='Fachwissenschaft'
AND 'Jahr' IN ('2006','2007')
AND 'Monat' IN ('Jänner','Februar','März','April')
GROUP BY 'Jahr', 'Monat'
```

Algorithmus 5 triviale Methode(MZ, MS)

Eingabe: Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS

Ausgabe: Pivot-Tabelle

Variable(n): Laufvariable über alle Knoten der ersten Ebene i ; aktuelle Merkmalsausprägung m

```
1:  $i = 0$ ;  
2: root=new Node();  
3: SQL-Abfrage mit Einschränkung auf  $MZ[1]$ ;  
4: für alle  $m \in MZ[1]$  {  
5:   root.children[ $i$ ]= $m$ ;  
6:   build_tree( $m, 1$ );  
7:    $i \leftarrow i + 1$ ;  
8: }  
9: build_table(root,  $MZ, MS$ );
```

Algorithmus 6 build_tree(m, l, MZ, MS)

Eingabe: übergeben Merkmalsausprägung m ; aktuell aufgebauter Level l ; Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS

Ausgabe: Gruppierungsbaum

Variable(n): Laufvariable über alle Knoten der ersten Ebene i ; aktuelle Merkmalsausprägung m

```
1:  $i = 0$ ;  
2: n=new Node(name);  
3: SQL-Abfrage mit Einschränkung auf  $MZ_m[l]$  und auf die Merkmalsausprägung  $m$ ;  
4: für alle  $m \in MZ_m[l]$  {  
5:   n.children[ $i$ ]= $m$ ;  
6:   build_tree( $m, l + 1$ );  
7:    $i \leftarrow i + 1$ ;  
8: }
```

Algorithmus 7 build_table(node, MZ , MS)

Eingabe: Wurzelknoten root; Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS

Ausgabe: Pivot-Tabelle

- 1: Ausgabe node.name;
 - 2: **für alle** $m \in MS$ {
 - 3: SQL-Abfrage mit Einschränkung auf den aktuellen Knoten node und Pfad node.description und auf die Merkmalsausprägung m ;
 - 4: Ausgabe der Ergebnisse der SQL-Abfrage;
 - 5: }
 - 6: **für alle** node.children n {
 - 7: build_table(n , MZ , MS);
 - 8: }
-

Als Ergebnis bekommt man alle Einträge zu einer *Forscherguppe* (siehe Abb. 6.3). Problematisch wird allerdings die Zuordnung der Werte zu den richti-

Jahr	Monat	SUM('Ideen')	SUM('Patente')
2006	April	1217	99
2006	Februar	1952	127
2006	März	1638	83
2007	April	1411	136
2007	Jänner	2377	167

Abbildung 6.3: Ergebnis der SQL-Abfrage für einen Knoten gruppiert nach Jahr und Monat.

gen Spalten. Man kann zwar bestimmen, dass die Abfrage-Ergebnisse sortiert nach den Spaltenfeldern ausgegeben werden sollen, dies muss aber nicht immer dem Aufbau der Pivot-Tabelle entsprechen. Will man z.B. Monate anzeigen und sind diese namentlich und nicht numerisch abgelegt, entspricht deren richtige Reihenfolge nicht deren alphabetischer Reihenfolge. Um nun die Ergebniswerte den richtigen Spalten zuzuordnen, müssten diese zuerst abgespeichert werden, um sie dann beim HTML-Aufbau an der richtigen Stelle ausgeben zu können.

Diese Methode liefert also maximal so viele Ergebniszeilen wie die Gesamtanzahl C der Spalten des Spaltenbereichs (wenn hier jede Forschergruppe in jedem Monat jedes Jahr Ergebnisse liefert, ansonsten fehlen diese Zeilen im Abfrageergebnis und in der Folge bleiben die Datenzellen leer) (vgl. Gleichung 4.2). Jede Ergebniszeile hat wieder so viele Spalten wie die Anzahl

der zu untersuchenden Variablen, im Beispiel also zwei. Das Abfrageergebnis wird zwischengespeichert, um dann gleich wieder ausgegeben zu werden. Der Zwischenspeicher kann danach sofort wieder neu verwendet werden. Die belegte Größe ist also immer maximal C mal der Anzahl der zu untersuchenden Variablen.

Die Algorithmen 5 und 6 aus Methode 1 können genauso für die verbesserte Methode mit nur einer SQL-Abfrage pro Knoten verwendet werden. Es ändert sich nur der Algorithmus `build_table()`.

Algorithmus 8 `build_table(node,  $MZ$ ,  $MS$ )`

Eingabe: Wurzelknoten `root`; Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS

Ausgabe: Pivot-Tabelle

```

1: Ausgabe node.name;
2: SQL-Abfrage mit Einschränkung auf den aktuellen Knoten und Pfad
3: data[]=Array der Ergebnisse der SQL-Abfrage;
4: für alle  $m \in MS$  {
5:   Ausgabe data[m];
6: }
7: für alle node.children  $n$  {
8:   build_table(n,  $MZ$ ,  $MS$ );
9: }
```

6.3 Verbesserung der SQL Performance

Die Laufzeit kann bereits optimiert werden ohne dass man versucht, mit Hilfe des Zwischenspeichers die Performance zu verbessern. Zawodny (ZB04) und Kofler (MK05) haben verschieden Möglichkeiten aufgezeigt, einzig durch Optimierung der Datenbank und der SQL-Abfragen die MySQL-Performance zu steigern.

6.3.1 MySQL-Tabellentypen

Noch vor dem Anlegen einer Datentabelle in MySQL ist es wichtig, den richtigen Tabellentyp zu wählen. MySQL bietet drei unterschiedliche Haupttabellentypen, die je nach Anwendungsfall zu wählen sind: MyISAM, BDB und InnoDB. Der Hauptunterschied zwischen diesen Typen, ohne auf diese genauer einzugehen, ist die Behandlung von konkurrierenden, d.h. gleichzeitigen Lese- und Schreibzugriffen auf Datensätze und die damit zusam-

menhängenden Locks (Sperren der Tabellen bzw. Tabellenteile), damit kein gleichzeitiger Lese- und Schreibzugriff passieren kann. So wichtig Locks auch sind kosten sie natürlich Performance, da ein Mehraufwand entsteht um einen Lock zu setzen, nachzuprüfen, ob ein Lock auf einem Datenbereich existiert und einen Lock wieder frei zu geben. Je besser und komplexer die eingesetzte Lösungsmethode bei Konkurrenz (gleichzeitiger Zugriff auf einen Datensatz) ist, desto laufzeitintensiver sind die SQL-Abfragen auf die Tabelle. MyISAM Tabellen verwendet Locks immer auf ganze Tabellen, d.h. wenn ein Schreibzugriff auf einen Datensatz einer Tabelle stattfindet, kann auf der ganzen Tabelle nicht mehr gelesen bzw. geschrieben werden. Diese Art der Tabelle ist geeignet für Applikationen mit vielen Lese- und wenigen Schreibzugriffen. BDB Tabellen verwenden sogenannte Page Locks, d.h. es wird immer nur eine Seite der Tabelle gesperrt, nicht die ganze Tabelle. Das ist allerdings eine problematische Lösung, wenn oft neue Datensätze eingefügt werden, da dabei immer die letzte Seite gesperrt ist. InnoDB Tabellen haben den kompliziertesten Lösungsmechanismus für Konkurrenzen und sperren immer nur die betroffene Zeile einer Tabelle. Sie sind geeignet für Applikationen mit sehr vielen Schreibzugriffen, aber dadurch auch am aufwendigsten und laufzeitintensivsten. Da bei der Erstellung der Pivot-Tabelle reine Lesezugriffe auf die Datentabelle erfolgen, ist die Anwendung von komplizierten Lösungsmethoden nicht notwendig. Der beste Tabellentyp für die Datentabellen ist somit MyISAM, sozusagen der einfachste Tabellentyp, aber in diesem konkreten Anwendungsfall auch der effizienteste. Für detaillierte Analyse der MySQL-Tabellentypen vgl. (ZB04).

6.3.2 Index

Die Indizierung der Datentabelle entsprechend der gewünschten Ergebnis-Pivot-Tabelle ist eine der einfachsten Methoden, die SQL Performance zu steigern. Bei dieser Methode wird die Suche nach Einträgen in der MySQL Datenbank selbst beschleunigt. Die Suche nach Daten in einer nicht indizierten Datenbank ist vergleichbar mit der Suche nach Telefonnummern in einem Telefonbuch. Um die Telefonnummer 12-34-56-78 zu finden, muss schlimmstenfalls das ganze Telefonbuch durchforstet werden. Das passiert mit linearer Laufzeit $O(n)$, d.h. man muss jede einzelne Telefonnummer im Telefonbuch vergleichen. Da die Telefonnummern aber einzigartig sind, kann man eventuell schon vorher abbrechen, sobald der Eintrag gefunden wurde. Legt man das Beispiel nun auf die Datenbank um, dann dauert die Suche auf jeden Fall $O(n)$, da gerade bei Datentabellen, welche durch eine Pivot-Tabelle gruppiert werden sollen, die Einträge passend zum Suchkriterium mehrmals vorkommen können und somit immer die gesamte Tabelle durchsucht werden muss.

Viel leichter und effizienter ist die Suche in einem Telefonbuch nach dem Nachnamen. Da diese Einträge bereits sortiert sind, kann man ungefähr abschätzen, in welchem Bereich des Telefonbuchs der Eintrag erwartet werden kann und diesen Bereich dann mit Hilfe einer „händischen“ binären Suche¹ bis zum Erfolg weiter durchsuchen. Einträge in einer Datenbank sind jedoch standardmäßig nicht sortiert (neue Einträge werden am Ende eingetragen). Abhilfe schaffen Indizes, die auf die Spalten gelegt werden, nach deren Einträgen oft gesucht wird. Die Datenbank legt sich dann eine sortierte Liste mit diesen Einträgen und dem Ort, wo diese in der Tabelle zu finden sind, ab. Um nicht allzu viel Speicherkapazität zu verbrauchen, können bei alphanumerischen Datenbankfeldern auch nur die ersten x-Buchstaben der Einträge abgelegt werden. Mit Hilfe eines Index schließt MySQL Zeilen aus, in denen die Einträge sicher nicht vorkommen können und durchsucht den Rest. Für die Pivot-Tabellen-Analyse ist es sinnvoll, den Index auf die Spalte der Datentabelle zu legen, die der letzten Gruppierungsebene des Zeilenbereichs entspricht. Im Beispiel, das bei dieser Arbeit verwendet wird, wären das die Spalte der Forschergruppen bezeichnet mit *Wissenschaftler*. Hier reichen die ersten 4 Buchstaben, um die Bezeichnungen der *Wissenschaftler* zu unterscheiden.

6.3.3 Query Cache

Eine weitere Optimierungsmethode ist das Cachen der SQL-Abfrage-Ergebnisse durch MySQL selbst, dazu muss die `query_cache_size` Variable in der Konfigurationsdatei gesetzt werden. Dadurch cacht MySQL selbst die Abfragen und dazugehörige Abfrageergebnisse. Sollte eine SQL-Abfrage eins zu eins wiederholt werden (MySQL macht dabei einen Textvergleich), wird nicht die Datenbanktabelle neu durchsucht sondern die Ergebnisse aus dem Cache gelesen. Das erspart vor allem bei Abfragen mit komplizierten WHERE-Klauseln viel Suchzeit. Allerdings gilt das natürlich nur für Abfragen, welche zu bereits vollständig identisch abgesetzten wurden. Beim einmaligen Aufbau der Pivot-Tabelle spielt diese Optimierungsmethode also keine Rolle. Wird jedoch oft auf- und zugeklappt, also eine ganze Gruppierungsebene ausgeblendet und dann wieder eingeblendet, wiederholen sich die SQL-Abfragen. In diesem Fall hat diese Konfigurationseinstellung auf jedenfall Einfluss auf die Abarbeitungszeit der SQL-Abfragen, sofern die Daten neu aus der Datenbank generiert werden und nicht, wie später betrachtet, in PHP gecached werden.

¹ Der jeweilige Suchbereich wird immer halbiert - durch das mittlere Element wird bestimmt, in welcher Hälfte der gewünschte Eintrag zu Suchen ist - diese wird wieder halbiert usw. Weitere Informationen zu binärer Suche vgl. (THC07).

Kapitel 7

Cachen

Auf das Problem der SQL-Abfragen wird auch in (Sch05) verwiesen. Datenbankabfragen sind oft der langsamste Teil einer Online-Applikation. Die Beseitigung bzw. Reduzierung dieser aufwändigen Datenbank-Aufrufe erhöht die Performance signifikant. Cachen kann hier eine Lösung sein.

7.1 Cachen

Cachen bedeutet die Speicherung von Daten, die man später verwenden möchte. Man speichert vor allem häufig verwendete Daten, so dass man später schneller auf diese zugreifen kann als ohne Cache (Speicherzugriffe sind im Allgemeinen die schnellste Zugriffsform).

Ein Beispiel, das Schlossnagle (Sch05) für das Cachen auch außerhalb der Programmier- und Computerwelt verwendet, wäre das System von Telefonnummern. Die Telefongesellschaft verteilt, auch heutzutage, regelmäßig Telefonbücher. Diese Bücher sind dicke, alphabetisch sortierte Wälzer, in denen man jede Telefonnummer, die registriert ist und deren Besitzer zu der Veröffentlichung eingewilligt hat, finden kann, aber es braucht lange, sie durchzublätern und darin zu suchen. Das ist vergleichbar mit einer sehr großen Datentabelle, die natürlich auch sehr lange Zugriffszeiten hat. Um schnelleren Zugriff auf häufig verwendete Telefonnummern (z.B. von Familie und Freunden) zu haben, legt man sich diese im Kurzwahlspeicher des Telefons ab oder schreibt sie in einer eigenen Liste zusammen. Der Kurzwahlspeicher bzw. die Telefonliste sind vergleichbar mit dem schnellen Zwischenspeicher, dem Cache.

Die Schwierigkeit liegt nun darin zu erkennen, welche Daten cachbare Daten sind, und wo es sinnvoll ist, die Daten zwischenzuspeichern und wo nicht. Das hängt zum Großteil davon ab, wie und wofür die Pivot-Tabelle verwendet

wird. Angenommen die Analyse mittels der Pivot-Tabelle wird nicht nur ein einziges Mal ausgegeben, mit allen zu untersuchenden Merkmalen und deren Merkmalsausprägungen, sondern wird zu Vergleichszwecken verwendet und man analysiert auch die einzelnen Ebenen für sich alleine und dann wieder die Gesamtheit, dann ist das Cachen nicht nur sinnvoll sondern ist unbedingt zu empfehlen. Abb. 7.1 zeigt zusätzliche Varianten des Zeilenbereichs zu der vollständig aufgeklappten Pivot-Tabelle aus Abb. 5.3. Ist das aber nicht der Fall und es wird die Pivot-Tabelle ein einziges Mal aufgerufen, kann man diese trotzdem aus dem Zwischenspeicher aufbauen und nicht wie im vorigen Kapitel beschrieben mit vielen SQL-Abfragen.

Bereits beim Aufbau des Baumes werden SQL-Abfragen generiert. Sie fragen jedoch aus der Daten-Tabelle nur die Merkmalsausprägungen für jedes Merkmal ab und dienen dem Aufbau des Zeilenbereichs. Man kann aber die SQL-Abfragen der letzten Ebene so weit erweitern, dass nicht nur die Merkmalsausprägungen sondern auch gleichzeitig die Werte des Datenbereichs als Ergebnis zurückgeliefert werden. Diese müssen dann zwischengespeichert werden, da der Aufbau der HTML-Seite erst später erfolgt.

Zwischenspeicher

Das Ergebnis der Abfrage wird in einem mehrdimensionalen Array abgespeichert. Eine Ergebniszeile sieht dann wie in Abb. 7.2 aus.

Das mehrdimensionale Array hat als Indizes einerseits den Knoten selbst und dann jeweils die Merkmale der Spaltengruppierungen. Die Daten aus der Ergebniszeile in Abb. 7.2 werden abgespeichert unter

```
DATEN[id_{Doppler}][2006][Jänner][Ideen]=1128
DATEN[id_{Doppler}][2006][Jänner][Patente]=100
```

Die Tabelle 7.3 verdeutlicht den Aufbau des mehrdimensionalen Arrays.

7.2 Durchführungsmöglichkeiten

7.2.1 Einzelne Abfragen abspeichern

Um die Werte im Nachhinein den richtigen Spaltenmerkmalen zuordnen zu können, müssen diese nicht nur in der WHERE-Klausel vorkommen, sondern müssen als Spalten der Ergebnis-Tabelle der SQL-Abfrage zurückgeliefert werden. Das bedeutet, dass der SELECT-Teil zusätzlich zu der Abfrage nach den Merkmalsausprägungen der letzten Zeilengliederungsebene nicht nur aus den Merkmalsausprägungen der letzten Spaltengliederungsebene, sondern

	2006								2007		
	Jänner		Februar		März		April		Jänner		Feb
Wissenschaftslehre ±	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ide
Fachwissenschaft ±	3585	264	1952	127	5013	247	2530	206	2377	167	553
Universalwissenschaft ±	4555	249	6509	363	9184	626	6489	442	3994	301	454

		2006							
		Jänner		Februar		März		April	
Wissenschaftslehre ±	Wissenschaft ±	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	P
Fachwissenschaft ±		3585	264	1952	127	5013	247	2530	20
	Naturwissenschaften ±	3585	264	1952	127	5013	247	2530	20
Universalwissenschaft ±		4555	249	6509	363	9184	626	6489	4
	Formalwissenschaften ±	1633	105	6509	363	4355	351	5424	35
	Geisteswissenschaften ±	2922	144			4829	275	1065	84

			2006							
			Jänner		Februar		März			
Wissenschaftslehre ±	Wissenschaft ±	Bereich	Ideen	Patente	Ideen	Patente	Ideen	Patente	Ideen	Pat
Fachwissenschaft ±			3585	264	1952	127	5013	247		
	Naturwissenschaften ±		3585	264	1952	127	5013	247		
		Chemie			1952	127	3432	156		
		Physik	3585	264			1581	91		
Universalwissenschaft ±			4555	249	6509	363	9184	626		
	Formalwissenschaften ±		1633	105	6509	363	4355	351		
		Informatik	1633	105	3586	212	2730	221		
		Mathematik			2923	151	1625	130		
	Geisteswissenschaften ±		2922	144			4829	275		
		Philosophie	2922	144			4829	275		

Abbildung 7.1: Pivot-Tabelle mit verschiedenen Ansichten des Zeilenbereichs: Die oberste Abbildung zeigt nur die Erfolge gruppiert nach Wissenschaftslehre mit den zwei Merkmalsausprägungen *Fachwissenschaft* und *Universalwissenschaft*. Die mittlere Abbildung präsentiert eine Ebene mehr und zeigt das Merkmal *Wissenschaft* gruppiert nach der *Wissenschaftslehre*, hier sind es insgesamt drei Ausprägungen. Schließlich zeigt die unterste Abbildung auch den konkreten *Wissenschaftsbereich* an.

Wissenschaftler	Jahr	Monat	SUM(`Ideen`)	SUM(`Patente`)
Doppler	2006	Jänner	1128	100

Abbildung 7.2: Eine Ergebniszeile der erweiterten SQL-Abfrage, die nicht nur die Kinder eines Knotens im Baum liefert, sondern auch gleichzeitig die Daten, die im Zwischenspeicher abgelegt werden, um sich später weitere SQL-Abfragen zu ersparen.

	Jänner	Februar	März	April
2006	Ideen Patente	Ideen Patente	Ideen Patente	Ideen Patente
2007	Ideen Patent	Ideen Patente	Ideen Patente	Ideen Patente

Abbildung 7.3: Ein mehrdimensionales Array zum Cachen der Daten für den Datenbereich wird jedem Knoten im Baum als zusätzliches Attribut zugeordnet.

auch aus den Merkmalen aller anderen Spaltengliederungsebenen aufgebaut wird. Betrachtet man wieder die Datenzelle aus Abb. 5.2, so lautet die ursprüngliche SQL-Abfrage nach den Kindern zum Aufbau des Baumes:

```
SELECT 'Wissenschaftler'
FROM 'datentabelle'
WHERE 'Bereich'='Physik'
AND 'Wissenschaft'='Naturwissenschaften'
AND 'Wissenschaftslehre'='Fachwissenschaft'
```

Diese Abfrage liefert als Ergebnis alle Merkmalsausprägungen zum Merkmal *Wissenschaftler*, die als Vater *Physik* haben. Angemerkt sei hier auch, dass in dem hier behandelten Beispiel die wiederholte Abfrage nach den weiter darüber liegenden Ebenen nicht notwendig wäre, da die Zuordnungen eindeutig sind. Es existiert z.B. nur ein Bereich *Physik* und hat den eindeutigen Vaterknoten *Naturwissenschaften*. Würde man hier aber noch als Zwischenebene das Merkmal *Nationalität* zwischen die Merkmalen *Bereich* und *Wissenschaftler* einfügen, so wäre diese Zuordnung nicht mehr eindeutig. Es würde dann nicht reichen

```
WHERE 'Nationalität'='Deutschland'
```

abzufragen, da es in allen *Bereichen* deutsche Gruppen geben kann. Um die Abfragen tauglich für alle Tabellen zu machen, wird immer bis zur obersten Zeilengruppierungsebene abgefragt.

Erweitert man nun das SQL-Statement so, dass alle notwendigen Daten abgefragt werden, lautet dieses:

```
SELECT 'Wissenschaftler','Jahr','Monat',
SUM('Ideen'), SUM('Patente')
```

```

FROM 'datentabelle'
  WHERE 'Bereich'='Physik'
  AND 'Wissenschaft'='Naturwissenschaften'
  AND 'Wissenschaftslehre'='Fachwissenschaft'
  AND 'Jahr' IN ('2006','2007')
  AND 'Monat' IN ('Jänner','Februar','März','April')
  GROUP BY 'Wissenschaftler','Jahr','Monat'

```

Als Ergebnis erhält man alle Einträge aus der Datentabelle, die Kinder vom Knoten *Bereich* enthalten und die Jahres- und Monats-Bedingungen erfüllen. Hier werden Einträge, die keine Analyse-Ergebnisse liefern, bereits gefiltert, d.h. es entstehen keine leeren Zeilen in der Pivot-Tabelle. Sollte das jedoch gewünscht sein, um immer eine vollständige Pivot-Tabelle mit allen Merkmalsausprägungen aller Merkmale zu erhalten, muss der Umweg über zwei Abfragen gegangen werden: einerseits die Abfrage nach den Kindern des Knotens, wie bei der Methode reiner SQL-Abfragen, und andererseits nach den Daten für den Datenbereich mit den jeweiligen Bedingungen. Diese zwei Abfragen verbindet man mittels eines **OUTER JOIN** zu einer SQL-Abfrage. Die so erzielten Ergebnisse werden nun als weiteres Attribut des Blattes gespeichert. Die Anzahl der gespeicherten Daten pro Blatt entspricht max. der Anzahl der Spalten des Spaltenbereichs. Es kann durchaus sein, dass keine Einträge in der Datentabelle zu bestimmten Bedingungsgruppen gefunden werden, wenn z.B. eine Forschergruppe in einem Jahr gar nicht tätig war. Wenn das nicht der Fall ist oder wenn 0 als Dummy (Leer-)Einträge für diese gespeichert werden ergibt sich, dass der Zwischenspeicherbereich pro Blatt gleich ist der Anzahl der Spalten aus Gleichung 4.2. Die Berechnung aus der Gleichung 4.1 für die Beispiel-Pivot-Tabelle 5.3 ergibt für die jeweiligen Spaltengruppierungsebenen: $\text{cols}[2]=2$ und $\text{cols}[1]=2 \cdot 4=8$ und insgesamt $C=1 \cdot 2 \cdot 8=16$ Spalten. Für insgesamt 18 Blätter, also 18 unterschiedliche Merkmalsausprägungen des Merkmals *Wissenschaftler*, ergibt sich dann ein Zwischenspeicher der Größe $18 \cdot 16 = 128$. Wobei hier nicht die tatsächliche Speicherplatz-Größe (in Byte) sondern die Gesamtanzahl der zwischengespeicherten Werte gezählt wird.

Innere Knoten

Eine Möglichkeit, die Datenbereichswerte für die inneren Knoten zu berechnen, ist die Summation der bereits erhaltenen Werte für die Blätter. Das kann leider nicht erst bei der Ausgabe des HTML-Codes passieren, da hier die Daten, die aufsummiert werden müssen, immer erst eine Ebene später zur Verfügung stehen, die Pivot-Tabelle aber von der ersten (mit nur einem Merkmal) bis zur untersten (mit allen Merkmalen) Ebene ausgegeben wird.

Das Problem wird anhand des Beispiels klarer: Es wird zuerst die Gruppierung nach *Wissenschaftslehre* und hier zuerst die *Fachwissenschaft* ausgegeben. In der ersten Zeile des Datenbereichs stehen die Summen der Werte der darunterliegenden Gruppierungsebenen, die man an dieser Stelle aber gar nicht abfragen kann, ohne wieder unnötig in die Tiefe des Baumes zu gehen. Das Berechnen dieser Summen muss also wieder während des Aufbaus des Baumes stattfinden und in ein mehrdimensionales Array abgelegt werden. Es wird für jeden Knoten gleich zu Beginn angelegt und erst nachdem die Blätter des jeweiligen Pfades gesetzt wurden befüllt (diese Art der Traversierung eines Baumes nennt man auch „Post order“).

Die zweite Möglichkeiten, die Werte für die inneren Knoten zu bekommen, ist eine Mischung aus Verwendung eines Zwischenspeichers und SQL-Abfragen. Die in Kapitel 7.2.1 beschriebene Variante für die Blätter lässt sich nämlich eins zu eins auch auf alle inneren Knoten anwenden.

Die Algorithmen 9, 10 und 11 fassen den Ablauf der dritten Methode zusammen.

Algorithmus 9 dritte Methode(MZ, MS)

Eingabe: Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS

Ausgabe: Pivot-Tabelle

Variable(n): Laufvariable über alle Knoten der ersten Ebene i ; aktuelle Merkmalsausprägung m

```

1:  $i = 0$ ;
2:  $root = \text{new Node}()$ ;
3: SQL-Abfrage mit Einschränkung auf  $MZ[1]$ ;
4: für alle  $m \in MZ[1]$  {
5:   für alle  $n \in MS$  {
6:      $data[m][] = \text{Array der Ergebnisse der SQL-Abfrage für die Aus-}$ 
        $\text{prägung } m$ ;
7:   }
8:    $root.children[i] = m$ ;
9:    $\text{build\_tree}(m, 1)$ ;
10:   $i \leftarrow i + 1$ ;
11: }
12:  $\text{build\_table}(root, MZ, MS)$ ;

```

Algorithmus 10 $\text{build_tree}(m, l, MZ, MS)$

Eingabe: übergeben Merkmalsausprägung m ; aktuell aufgebauter Level l ;
Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS

Ausgabe: Gruppierungsbaum

Variable(n): Laufvariable über alle Knoten der ersten Ebene i ; aktuelle Merkmalsausprägung m

```
1:  $i = 0$ ;  
2:  $n = \text{new Node}(\text{name})$ ;  
3: SQL-Abfrage mit Einschränkung auf  $MZ_m[l]$  und auf die Merkmalsausprägung  $m$ ;  
4: für alle  $m \in MZ_m[l]$  {  
5:   für alle  $n \in MS$  {  
6:      $\text{data}[m][n] = \text{Array der Ergebnisse der SQL-Abfrage für die Ausprägung } m$ ;  
7:   }  
8:    $n.\text{children}[i] = m$ ;  
9:    $\text{build\_tree}(m, l + 1)$ ;  
10:   $i \leftarrow i + 1$ ;  
11: }
```

Algorithmus 11 $\text{build_table}(\text{node}, MZ, MS)$

Eingabe: Wurzelknoten root ; Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS

Ausgabe: Pivot-Tabelle

```
1: Ausgabe  $\text{node.name}$ ;  
2: für alle  $m \in MS$  {  
3:   Ausgabe  $\text{data}[\text{node.name}][m]$ ;  
4: }  
5: für alle  $\text{node.children } n$  {  
6:    $\text{build\_table}(n, MZ, MS)$ ;  
7: }
```

7.2.2 Zwischenspeichern der Werte aus einem einzigen SQL-Statement

Eine weitere Möglichkeit besteht darin, sich sozusagen eine vollständige Kopie der benötigten Teile der Datentabelle in den Zwischenspeicher zu legen und aus diesem einerseits den Baum aufzubauen und andererseits auch die Daten für den Datenbereich zu berechnen. Diese Methode ist nicht nur sehr speicherintensiv sondern benötigt auch PHP Summationen und PHP Vergleiche. Der Vorteil ist aber, dass nur eine einzige SQL-Abfrage benötigt wird. Diese muss zusätzlich zu den Variablenüberschriften im SELECT-Teil alle Ausprägungen der Zeilenfelder und die Merkmale des Spaltenbereichs enthalten. Nach diesen wird auch gruppiert.

```
SELECT 'Wissenschaftslehre', 'Wissenschaft', 'Bereich',
      'Wissenschaftler',
      'Jahr', 'Monat',
      SUM('Ideen'), SUM('Patente')
```

Die WHERE-Klausel schränkt dann nur mehr auf die Merkmalsausprägungen des Spaltenbereichs ein.

```
WHERE 'Jahr' IN ('2006', '2007')
      AND 'Monat' IN ('Jänner', 'Februar', 'März', 'April')
```

Durch die SQL-Abfrage werden alle Ausprägungen der Zeilenmerkmale als Spalten des SQL-Ergebnisses ausgegeben. Diese Ergebnisse kann man verwenden, um den Baum aufzubauen. Der Unterschied zu den anderen Methoden ist jedoch, dass sich die Ausprägungen bei den Ergebniszeilen wiederholen und nicht fertig gruppiert für den Baumaufbau zurückgegeben werden. Abb. 7.4 zeigt einen Ausschnitt aus dem Ergebnis der SQL-Abfrage nach der vollständigen, benötigten Datentabelle. Man sieht hier, dass sich die Einträge der ersten vier Spalten immer wieder wiederholen. Um daraus nun den Gliederungsbaum aufzubauen, muss beim Abarbeiten jeder Zeile, die SQL zurück liefert, abgefragt werden, ob die jeweiligen Ausprägungen bereits als Knoten im Baum angelegt wurden. Das sind auf jeden Fall so viele Vergleiche wie Anzahl der Ergebniszeilen (EZ) mal Anzahl der Merkmale des Zeilenbereichs (MZ), wobei max. für jedes Blatt ($|B|$) und für jede Ausprägung der Spaltengruppierungsebenen (die Variablenüberschriften nicht gerechnet) eine Ergebniszeile ausgegeben wird. Das ergibt insgesamt:

$$\text{Comp} = |EZ| \cdot |MZ| = |B| \cdot \prod_{i=1}^c |\text{MS}[i]| \cdot |MZ| \quad (7.1)$$

Wissenschaftslehre	Wissenschaft	Bereich	Wissenschaftler	Jahr	Monat	SUM(`Ideen`)	SUM(`Patente`)
Fachwissenschaft	Naturwissenschaften	Physik	Doppler	2006	Jänner	1128	100
Fachwissenschaft	Naturwissenschaften	Physik	Doppler	2006	März	1581	91
Fachwissenschaft	Naturwissenschaften	Physik	Doppler	2007	März	3165	215
Fachwissenschaft	Naturwissenschaften	Physik	Newton	2006	April	1313	107
Fachwissenschaft	Naturwissenschaften	Physik	Newton	2006	Jänner	2457	164
Fachwissenschaft	Naturwissenschaften	Physik	Newton	2007	Februar	3813	198
Universalwissenschaft	Formalwissenschaften	Informatik	Dijkstra	2006	April	1151	114
Universalwissenschaft	Formalwissenschaften	Informatik	Dijkstra	2006	Februar	1870	116
Universalwissenschaft	Formalwissenschaften	Informatik	Dijkstra	2006	März	1404	111

Abbildung 7.4: Ein einziges SQL liefert alle notwendigen Daten für den Baumaufbau und den Datenbereich.

Auch die Summationen dürfen nicht unterschätzt werden. Es wird für jeden Knoten so oft summiert wie es Spalten C im Spaltenbereich gibt.

$$\text{Sum} = |V| \cdot C \quad (7.2)$$

Der Algorithmus zum Aufbau der Pivot-Tabelle `build_table()` entspricht dem Algorithmus aus der Methode 3. Der Aufbau des Baumes unterscheidet sich nun aber grundlegend von den bereits beschriebenen Methoden. Der Aufbau der Pivot-Tabelle nach dem vierten Ansatz wird in Pseudocode 12, 13 und 14 beschrieben.

Algorithmus 12 vierte Methode(MZ, MS)

Eingabe: Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS , Ergebnisse der SQL-Abfrage SQL

Ausgabe: Pivot-Tabelle

Variable(n): Laufvariable über alle Knoten der ersten Ebene i ; aktuelle Merkmalsausprägung m

- 1: $i = 0$;
 - 2: `root=new Node()`;
 - 3: SQL-Abfrage mit Einschränkung auf alle MZ und alle MS ;
 - 4: `build_tree(m, 1, MZ, MS, SQL)`;
 - 5: `build_table(root, MZ, MS)`;
-

7.2.3 Weitere Möglichkeiten

Es gibt noch weitere Mischungen dieser Methoden, z.B. eine SQL-Abfrage für den Baumaufbau, welche die Ausprägungen richtig gruppiert zurück liefert und die vielen PHP Vergleiche spart, und eine weitere SQL-Abfrage für

Algorithmus 13 $\text{build_tree}(m, l, MZ, MS, SQL)$

Eingabe: übergeben Merkmalsausprägung m ; aktuell aufgebauter Level l ;
Menge der Zeilenmerkmalsausprägungen MZ ; Menge der Spaltenmerkmalsausprägungen MS , Ergebnisse der SQL-Abfrage SQL

Ausgabe: Gruppierungsbaum

Variable(n): Laufvariable über alle Knoten der ersten Ebene i ; aktuelle Merkmalsausprägung m

```
1: für alle Ergebniszeilen aus SQL {
2:    $n = \text{root}$ ;
3:   für  $i = 1 \dots |MZ|$  {
4:     falls  $\exists$  Knoten aus  $SQL[i]$  dann {
5:       // der Knoten passend zu der abgearbeiteten SQL Ergebniszeile
       wird zurückgegeben
6:        $n.\text{child} \leftarrow \text{Node}(SQL[i])$ ;
7:       für alle  $m \in MS$  {
8:          $\text{data}[n][m] \leftarrow \text{data}[n][m] + \text{Werte aus der SQL-Abfrage}$ 
9:       }
10:    } sonst {
11:       $n.\text{child} \leftarrow \text{new Node}(SQL[i])$ ;
12:      für alle  $m \in MS$  {
13:         $\text{data}[n][m] \leftarrow \text{Werte aus der SQL-Abfrage}$ 
14:      }
15:    }
16:     $n \leftarrow n.\text{child}$ 
17:  }
18: }
```

die Werte und Summationen für die Datenzellen. Im Endeffekt hängt die Wahl der Methode auch sehr von den Anwendungsumgebungen und dem Benutzerverhalten ab, wie das folgende Kapitel zeigen wird.

Kapitel 8

Speicheroptimierung vs. Laufzeitoptimierung - Vergleichsergebnisse

8.1 Einleitung

Im Folgenden werden die in den vorigen Kapiteln vorgestellten Methoden miteinander verglichen. Es werden jeweils folgende Kriterien untersucht:

- **SQL-Abfragen (SQL)** Anzahl der benötigten SQL-Abfragen.
- **Ergebniszeilen (EZ)** Anzahl der von MySQL zurückgelieferten Ergebniszeilen.
- **Ergebnisspalten (ES)** Anzahl der von MySQL zurückgelieferten Ergebnisspalten.
- **Datentransfer (DT)** Die Ergebnisse der SQL Abfragen müssen vom Datenbankserver zum Webserver übertragen werden, die Anzahl dieser übertragenen Daten ist oft relevant, falls die Übertragungsgeschwindigkeit sehr niedrig ist. DT gibt nur die Anzahl der Werte an, je nach Datentyp und Betriebssystem können das unterschiedlich viele Bytes sein.
- **Zwischenspeicher (ZS)** Anzahl der benötigten Arrayfelder, die in PHP angelegt und befüllt werden.
- **PHP Summation (PHP-Sum)** Anzahl der Summationen, die über PHP durchgeführt werden.

- **PHP Vergleiche (PHP-Comp)** Anzahl der Vergleichsoperationen, die in PHP durchgeführt werden müssen, um z.B. die richtige Stelle im Baum zu finden.

Die benötigten SQL-Abfragen und auch die jeweils unterschiedlichen Ergebniszeilen werden nochmals angeführt, um die Unterschiede in deren Komplexität aufzuzeigen. Die untersuchten Datentabellen haben jeweils 10^3 , 10^4 und 10^5 Einträge, wobei die Anzahl der unterschiedlichen Merkmalsausprägungen variiert. Schließlich werden die Methoden auch auf einer Tabelle mit realen Daten getestet, welche den Anforderungen von CARE (vgl. Kapitel 1.7.1) entspricht.

8.1.1 Testen

Beim Testen wurde speziell auf die von (ZB04) beschriebenen relevanten Punkte für Datenbanktests geachtet.

- Nur einen Parameter pro Test ändern: es werden zuerst Datentabellen ohne Index, dann mit Index getestet, erst beim nächsten Durchlauf der Tabellentyp geändert.
- Iterativ Testen: keine riesigen Sprünge bei der Vergrößerung der Datentabelle machen.
- Test immer wiederholen: es wird mind. 4x getestet, da die Rechner unterschiedlich belastet werden, vor allem der entfernte Server, der gleichzeitig im Echtbetrieb als Webserver arbeitet, hat natürlich gleichzeitige Belastungen, die von den Tests unabhängig sind und sich ändern können. Beim wiederholten Testen ist vor allem auch zu beachten, dass hier der in Kapitel 6.3.3 beschriebene Query Cache abgeschaltet wird, da sonst die Ergebnisse der Wiederholungen verfälscht werden.
- Realdaten verwenden: zusätzlich zu den generierten Datentabellen werden auch reale Tabellen von CARE getestet.
- Nicht zu viele Clients verwenden: getestet wird auf genau einem Client.

8.1.2 Technische Daten

Die Ergebnisse wurden auf drei Umgebungen getestet, um die drei üblichen Anwendungsfälle für den Aufbau einer Pivot-Tabelle aus einer in einer Datenbank gespeicherten Datentabelle zu simulieren:

1. Umgebung 1 (UG1): lokaler Datenbank- und Webserver

Rechner Intel Pentium 2x 1,66GHz, 2GB RAM, Windows XP

Webserver Apache 2.2.4

PHP Version 5.2.4

MySQL Version 5.0.45

2. Umgebung 2 (UG2): entfernter Datenbankserver und lokaler Webserver: Der Rechner aus Punkt 1 wird als lokaler Webserver eingesetzt, der Rechner aus Punkt 3 als entfernter Datenbankserver

3. Umgebung 3 (UG3): entfernter Datenbank- und Webserver

Rechner Dual XEON, 2x 2,8GHz, L2 Cache 2GB RAM, Festplatten SCSI Rate5, Linux 2.4.20

Webserver Apache 2.0.44

PHP Version 5.2.1

MySQL Version 5.0.27

Während der lokale Datenbank- und Webserver keine weiteren Belastung während der Testphase trägt, trifft das auf den externen Server nicht zu, da dieser im Produktivbetrieb steht und auch andere Anwendungen gleichzeitig bearbeitet.

Die Laufzeitmessung erfolgt mittels Timestamps, die jeweils den Baumaufbau, den Aufbau der HTML Tabelle und die Gesamtlaufzeit messen. Die Werte werden auf Zehntel Millisekunden gerundet.

8.2 Formale Analyse

8.2.1 Methode 1: je eine SQL-Abfrage pro Knoten und pro Spalte

Bei der Generierung des HTML-Codes werden an den entsprechenden Stellen die SQL-Statements generiert und abgesetzt. Dabei wird für jeden Knoten des Gliederungsbaumes und jede Merkmalsausprägung des Spaltenbereichs bis zur vorletzten Ebene eine Abfrage an den SQL-Server geschickt.

SQL

Jedes SQL-Statement hat, bezogen auf das in dieser Arbeit verwendete Beispiel, die Form:

```
SELECT SUM('Ideen'),SUM('Patente')
FROM 'datentabelle'
WHERE 'Wissenschaftler'='Newton'
AND 'Bereich'='Physik'
AND 'Wissenschaft'='Naturwissenschaften'
AND 'Wissenschaftslehre'='Fachwissenschaft'
AND 'Jahr'='2006' AND 'Monat'='Jänner'
```

Der SQL-Server liefert folgendes zurück:

SUM(Ideen)	SUM(Patente)
2457	164

Abbildung 8.1: Ergebnis einer SQL-Abfrage für einen einzelnen Knoten und der vorletzten Spaltengruppierung.

Theoretische Analyse

Tabelle 8.1 zeigt die performancebestimmenden Größen für diese Methode formal und anhand des Beispiel aus 5.3, dabei ist $|V|$ die Anzahl der Knoten im Gliederungsbaum, $|MS[i]|$ die Anzahl der Merkmalsausprägungen in der i -ten Spaltengruppierungsebene und c die Anzahl der Spaltengruppierungsebenen inklusive der Ebene mit den Variablenüberschriften.

Tabelle 8.1: Theoretische Analyse der Methode reiner SQL-Abfragen: pro Knoten pro Spalte.

	Formal	Beispiel
SQL	$1 + V \cdot (\prod_{i=1}^{c-1} MS[i] + 1) - B $	$1 + 28 \cdot 9 - 18 = 235$
EZ	1	1
ES	$ MS[c] $	2
DT	$EZ \cdot ES \cdot SQL$	$1 \cdot 2 \cdot 235 = 470$
ZS	0	0
PHP-Sum	0	0
PHP-Comp	0	0

Diskussion

Diese Implementierungsmethode ist die trivialste Methode, welche die Berechnungen vom untersten Blatt bis zur obersten Zeilengruppierungsebene auf Einzelprobleme zerlegt. Dadurch entstehen sehr viele laufzeitintensive SQL-Abfragen. Es handelt sich sicherlich um keine optimale Variante. Ihr Vorteil sind die relativ wenigen Datensätze, die im Falle eines getrennten Datenbank- und Webservers übertragen werden müssen. Die Anzahl dieser hängt einzig von der Anzahl der zu untersuchenden Variablen ab und ist meist gering. Allerdings erzeugen auch die SQL-Statements selbst einen Datenverkehr.

Günstig für diese Methode sind möglichst wenige Merkmalsausprägungen pro Merkmal sowohl im Zeilen- wie auch Spaltenbereich.

8.2.2 Methode 2: je eine SQL-Abfrage pro Knoten

Bei der Generierung des HTML-Codes werden an den entsprechenden Stellen die SQL-Statements generiert und abgesetzt. Dabei wird für jeden Knoten des Baumes eine Abfrage an den SQL-Server geschickt, d.h. alle Daten einer Zeile des Datenbereichs der Pivottabelle werden jeweils mittels eines SQL-Statements abgefragt.

SQL

Jedes SQL-Statement hat die Form:

```
SELECT 'Jahr', 'Monat', SUM('Ideen'),SUM('Patente')
FROM 'datentabelle'
WHERE 'Wissenschaftler'='Newton'
AND 'Bereich'='Physik'
AND 'Wissenschaft'='Naturwissenschaften'
AND 'Wissenschaftslehre'='Fachwissenschaft'
AND 'Jahr' IN ('2006','2007')
AND 'Monat' IN ('Jänner','Februar','März','April')
GROUP BY 'Jahr', 'Monat'
```

Das SQL-Statement liefert folgendes zurück:

Theoretische Analyse

Tabelle 8.2 zeigt die performancebeeinträchtigenden Größen für diese Methode formal und anhand des Beispiels aus 5.3.

Jahr	Monat	SUM('Ideen')	SUM('Patente')
2006	April	1217	99
2006	Februar	1952	127
2006	März	1638	83
2007	April	1411	136
2007	Jänner	2377	167

Abbildung 8.2: Ergebnis der SQL-Abfrage für einen Knoten gruppiert nach Jahr und Monat.

Tabelle 8.2: Theoretische Analyse der Methode mit jeweils einer SQL-Abfrage pro Knoten.

	Formal	Beispiel
SQL	$1 + 2 \cdot V - B $	$1 + 56 - 18 = 39$
EZ	$\prod_{i=1}^{c-1} MS[i] $	8
ES	$ MS[c] $	2
DT	$EZ \cdot ES \cdot SQL$	$8 \cdot 2 \cdot 39 = 624$
ZS	$\prod_{i=1}^c MS[i] $	16
PHP-Sum	0	0
PHP-Comp	0	0

Diskussion

Diese Implementierungsmethode benötigt im Vergleich zur trivialen Methode aus 8.2.1 pro Knoten um so viele SQL-Abfragen weniger, wie es Ausprägungen in den Spaltenmerkmalen ohne der Variablenüberschriften gibt. Der geringe zusätzliche Speicherplatzverbrauch ist dabei fast zu vernachlässigen, das ist einer der Vorteile dieser Methode gegenüber den Nachfolgenden. Bei einer sehr komplexen Datentabelle mit vielen inneren Knoten müssen hier trotzdem immer nur die Werte einer einzigen SQL-Abfrage abgespeichert werden, im Gegensatz zu den nachfolgenden Methoden, bei denen der Speicher von der Anzahl der Knoten und Spalten abhängt. Da aber meist der Speicher nicht das limitierende Problem ist, ist diese Methode nur in Einzelfällen zu bevorzugen.

Wie auch bei der vorigen Methode sind möglichst wenige Merkmale und Merkmalsausprägungen im Zeilenbereich günstig.

8.2.3 Methode 3: Zwischenspeicher mit mehreren SQL-Abfragen

Beim Aufbau des Gliederungsbaumes werden zusätzlich zu den einzelnen Knoten, d.h. zu den Ausprägungen eines Merkmals, auch gleichzeitig die Werte für den Datenbereich abgefragt und gespeichert.

SQL

Jedes SQL-Statement hat die Form:

```
SELECT 'Wissenschaftler', 'Jahr', 'Monat',  
      SUM('Ideen'), SUM('Patente')  
FROM 'datentabelle'  
  WHERE 'Bereich'='Physik'  
    AND 'Wissenschaft'='Naturwissenschaften'  
    AND 'Wissenschaftslehre'='Fachwissenschaft'  
    AND 'Jahr' IN ('2006','2007')  
    AND 'Monat' IN ('Jänner','Februar','März','April')  
GROUP BY 'Jahr', 'Monat'
```

Die Datenbank liefert folgendes zurück:

Wissenschaftler	Jahr	Monat	SUM(`Ideen`)	SUM(`Patente`)
Newton	2006	April	1313	107
Newton	2006	Jänner	3585	264
Doppler	2006	März	1581	91
Newton	2007	Februar	3813	198
Doppler	2007	März	3165	215

Abbildung 8.3: Ergebnis der SQL-Abfrage die direkt beim Baumaufbau die Werte für den Datenbereich liefert (Ausschnitt).

Theoretische Analyse

Tabelle 8.3 zeigt die performancebestimmenden Größen für diese Methode formal und anhand des Beispiels aus 5.3. Hier werden zwei weitere Variablen benötigt: $|MZ[i]|$ bezeichnet die Anzahl der Merkmalsausprägungen des Zeilenbereichs der i -ten Ebene und z die Anzahl der Zeilengruppierungsebenen.

Tabelle 8.3: Theoretische Analyse der Methode des Zwischenspeicherns mit mehreren SQL-Abfragen.

	Formal	Beispiel
SQL	$1 + \sum_{i=1}^{z-1} MZ[i] $	11
EZ	$ V \cdot \prod_{i=1}^{c-1} MS[i] $	$28 \cdot 8 = 224$
ES	$1 + MS[c] $	3
DT	EZ · ES	$224 \cdot 3 = 772$
ZS	$ V \cdot \prod_{i=1}^c MS[i] $	$28 \cdot 16 = 448$
PHP-Sum	0	0
PHP-Comp	EZ · 1	224

Diskussion

Da SQL-Abfragen im allgemeinen sehr laufzeitintensiv im Vergleich zu Berechnungs- und Vergleichsoperationen in PHP sind, sind die Methoden des Zwischenspeicherns auf jeden Fall vorzuziehen, da hier wesentlich weniger SQL-Abfragen abgesetzt werden müssen. Die Anzahl der auf einmal zu übertragenden Daten und die Größe des Zwischenspeichers nimmt hier allerdings zu. Die dritte Methode ist die Methode des Zwischenweges. Es gibt immer noch pro inneren Knoten jeweils eine SQL-Abfrage, diese baut aber den Baum auf und speichert gleichzeitig die Zwischenwerte. Die Daten, die pro SQL-Statement abzuarbeiten und zu übertragen sind, sind zwar schon mehr als bei den ersten beiden Methoden, aber im Vergleich zum vierten Ansatz immer noch relativ wenig.

Ein weiterer Vorteil aller drei bis jetzt analysierten Methoden sind die Summationen bzw. allgemein alle Berechnungsoperationen, die man hier direkt an MySQL weitergibt und bereits die Ergebnisse für jeden Knoten fertig im Zwischenspeicher ablegt.

Die zwei Größen, die bei dieser Methode die Performance beeinträchtigen sind einerseits die Daten zur Übertragung, die bei einer schlechten Übertragungsrate eine große Rolle spielen, und die Vergleiche. Diese sind zwar nicht sehr laufzeitintensiv, können aber bei einer Datentabelle mit sehr vielen Ausprägungen zunehmend ins Gewicht fallen.

8.2.4 Methode 4: Zwischenspeicher mit einer einzigen SQL-Abfrage

Eine einzige SQL-Abfrage, die in manchen Fällen eine vollständige Kopie der benötigten Teile der Datentabelle zurückliefert, wird verwendet, um die Daten der Blätter und der gesamten Baumstruktur zu erhalten. Die Summen

für die Blätter werden durch die MySQL-Abfrage gebildet, alle Berechnungen für deren Vaterknoten und alle anderen inneren Knoten erfolgen in PHP.

SQL

Jedes SQL-Statement hat die Form:

```
SELECT 'Wissenschaftslehre','Wissenschaft','Bereich',
      'Wissenschaftler','Jahr','Monat',
      SUM('Ideen'), SUM('Patente')
FROM 'datentabelle'
WHERE 'Jahr' IN ('2006','2007')
AND 'Monat' IN ('Jänner','Februar','März','April')
GROUP BY 'Wissenschaftslehre','Wissenschaft','Bereich',
        'Wissenschaftler','Jahr','Monat'
```

Ein solcher Aufruf liefert aus der Datenbank folgendes zurück:

Wissenschaftslehre	Wissenschaft	Bereich	Wissenschaftler	Jahr	Monat	SUM('Ideen')	SUM('Patente')
Fachwissenschaft	Naturwissenschaften	Physik	Doppler	2006	Jänner	1128	100
Fachwissenschaft	Naturwissenschaften	Physik	Doppler	2006	März	1581	91
Fachwissenschaft	Naturwissenschaften	Physik	Doppler	2007	März	3165	215
Fachwissenschaft	Naturwissenschaften	Physik	Newton	2006	April	1313	107
Fachwissenschaft	Naturwissenschaften	Physik	Newton	2006	Jänner	2457	164
Fachwissenschaft	Naturwissenschaften	Physik	Newton	2007	Februar	3813	198
Universalwissenschaft	Formalwissenschaften	Informatik	Dijkstra	2006	April	1151	114
Universalwissenschaft	Formalwissenschaften	Informatik	Dijkstra	2006	Februar	1870	116
Universalwissenschaft	Formalwissenschaften	Informatik	Dijkstra	2006	März	1404	111

Abbildung 8.4: Ein einziges SQL-Statement liefert alle notwendigen Daten für den Baumaufbau und den Datenbereich (Ausschnitt).

Theoretische Analyse

In Tabelle 8.4 werden wieder die performancebeeinträchtigenden Größen für diese Methode formal und anhand des Beispiel aus 5.3 zusammengefasst. $|B|$ bezeichnet hier die Gesamtanzahl der Blätter des Gruppierungsbaumes für den Aufbau der Pivot-Tabelle.

Diskussion

Diese Methode mit nur einer einzigen SQL-Abfrage ist sicherlich in den meisten Fällen die schnellste, auch wenn hier die Speicherbelastung und der Datentransfer nicht unterschätzt werden dürfen. Sollten die Daten von einem

Tabelle 8.4: Theoretische Analyse der Methode des Zwischenspeicherns mit einer SQL-Abfrage.

	Formal	Beispiel
SQL	1	1
EZ	$ B \cdot \prod_{i=1}^{c-1} MS[i] $	$18 \cdot 8 = 144$
ES	$ MZ + MS + MS[c] $	$4 + 2 + 2 = 8$
DT	$EZ \cdot ES \cdot SQL$	$144 \cdot 8 \cdot 1 = 1152$
ZS	$ V \cdot \prod_{i=1}^c MS[i] $	$28 \cdot 16 = 448$
PHP-Sum	$(B \cdot MZ - V) \cdot \prod_{i=1}^c MS[i] $	$(18 \cdot 4 - 28) \cdot 16 = 704$
PHP-Comp	$ MZ \cdot EZ$	$4 \cdot 144 = 576$

Datenbankserver zu einem Webserver transferiert werden müssen bevor sie verarbeitet werden können, kann es hier bei großen Datentabellen und langsamer Übertragungsgeschwindigkeit zu Problemen kommen.

Im Vergleich zur dritten Methode punktet sie mit der einzigen SQL-Abfrage, die aber wieder bei einer sehr großen Datentabelle mit vielen Ausprägungen ein Nachteil sein kann bzw. sogar ein Limit darstellen kann.

Ein weiterer Nachteil sind die Summationen bzw. Berechnungsoperationen, die, ausser bei den Blättern, nicht mehr als Ergebnis der SQL-Abfrage zurückgeliefert werden sondern mit PHP berechnet werden müssen. Um den Baum nicht nochmals durchgehen zu müssen, werden die Summen direkt beim Aufbau des Baumes berechnet, im zu jedem inneren Knoten entlang des Pfades, welcher der ausgegebenen Ergebniszeile entspricht, der aktuelle Wert des Blattes hinzugerechnet wird. Das ergibt für jedes Blatt, jedes Merkmal des Zeilenbereichs und jede Spalte des Spaltenbereichs eine Summation:

$$\text{PHP-Sum} = |B| \cdot |MZ| \cdot \prod_{i=1}^c |MS[i]| \quad (8.1)$$

Davon müssen allerdings die Blätter je einmal pro Spalte abgezogen werden, da diese beim ersten Zugriff nur gespeichert und nicht summiert werden:

$$\text{PHP-Sum} = (|B| \cdot |MZ| - |B|) \cdot \prod_{i=1}^c |MS[i]| \quad (8.2)$$

Analog ist jeder erste Zugriff bei einem inneren Knoten ebenfalls eine Speicheroperation und keine Summation:

$$\text{PHP-Sum} = (|B| \cdot |MZ| - |B| - (|V| - |B|)) \cdot \prod_{i=1}^c |MS[i]| \quad (8.3)$$

Wichtig ist auch die Art der Auswertung, die mit der Pivot-Tabelle dargestellt werden soll. Methoden eins bis drei unterstützen ohne weiteren Programmieraufwand alle Aggregationsoperationen, die MySQL unterstützt, dabei sind vor allem die Minimum- und Maximumbestimmung, die Durchschnittsbildung und statistische Auswertungen wie Standardabweichung und Varianz interessant. Bei der vierten Methode muss die gewünschte Operation eigens implementiert werden. Bei komplizierten Berechnungen steigt dadurch natürlich auch die Laufzeit und der benötigte Speicher. Berechnungen für innere Knoten, die auf den Ergebnissen der jeweiligen Kinder aufbauen, wie z.B. Maximum- und Minimumberechnung, sind dabei ähnlich performant wie die Summenbildung. Bei diesen kann das Ergebnis für die Blätter bereits durch Berechnung mittels SQL-Abfrage ermittelt werden. Die jeweiligen Gruppierungsergebnisse in den Vaterknoten werden aus den Ergebnissen der Kinder ermittelt. Es gilt: Die Summe der Werte einer Menge A addiert zur Summe der Werte einer Menge B ist gleich der Summe der Werte der Vereinigungsmenge aus A und B . Dies gilt analog für Minimum und Maximum.

$$sum(A) + sum(B) = sum(A \cup B) \quad (8.4)$$

$$min(min(A), min(B)) = min(A \cup B) \quad (8.5)$$

$$max(max(A), max(B)) = max(A \cup B) \quad (8.6)$$

Die Gleichung 8.6 kann aber nicht ohne weiteres auf alle Funktionen verallgemeinert werden. Sie gilt vor allem dann nicht, wenn zur Berechnung alle Einzelwerte benötigt werden, wie z.B. bei der Berechnung des Mittelwerts, der Varianz oder der Standardabweichung.

$$\mu(\mu(A), \mu(B)) \neq \mu(A \cup B) \quad (8.7)$$

$$\sigma(\sigma(A), \sigma(B)) \neq \sigma(A \cup B) \quad (8.8)$$

$$\sigma^2(\sigma^2(A), \sigma^2(B)) \neq \sigma^2(A \cup B) \quad (8.9)$$

Um den Mittelwert eines inneren Knotens zu berechnen, benötigt man zusätzlich zu den Mittelwerten der Kinderknoten noch die Anzahl der bereits inkludierten Werte. Dann lässt sich der Mittelwert durch die folgende Formel berechnen:

$$\mu(A, B) = \frac{\mu(A) \cdot |A| + \mu(B) \cdot |B|}{|A| + |B|} \quad (8.10)$$

In diesem Fall ist der Berechnungs- und Speicheraufwand bei der vierten Methode geringfügig größer als bei der Summation. Wenn die gewünschte Auswertung aber auf allen Werten basiert, steigt der Aufwand für die Berechnung wesentlich, da dann auch alle Werte der Kinder an den Vaterknoten

übergeben werden müssen. Als Beispiel für eine komplexe Berechnung dient die Varianz:

$$\sigma^2(A) = \frac{\sum_{i=1}^{|A|} (A[i] - \mu(A))^2}{|A|} \quad (8.11)$$

Das bedeutet, dass bei mehreren Einträgen in der Datentabelle für ein Blatt nicht nur z.B. die Summe gespeichert werden muss, sondern alle Einzelwerte. Das vergrößert den Speicheraufwand immens. Auch für die Berechnungen bei den Vaterknoten können nicht die Ergebnisse der Kinder verwendet werden, sondern es sind immer wieder alle Einzelwerte notwendig. Dadurch steigt die Laufzeit erheblich.

8.3 Testergebnisse

In Tabelle 8.5 werden die vier Methoden anhand ihrer Parameter für den Datenbanktransfer, Speicherverbrauch und Berechnungen verglichen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen dargestellt. Jedes Ergebnis zeigt den Mittelwert aus mindestens vier Testdurchläufen in Sekunden. Der obere Wert gibt die Dauer des Aufbaus des Gliederungsbaumes an, während der untere Wert die Zeit des HTML-Aufbaus zeigt. Vor allem ist aber natürlich die Gesamtlaufzeit wichtig.

Als Grundlage für die erstellte Pivottabelle dient eine Datentabelle mit 10^4 Einträgen. Diese Tabelle wurde zuerst als InnoDB und danach zum Vergleich als MyISAM-Tabelle (jeweils ohne Indizes) getestet. Im dritten Durchlauf wurden auf die MyISAM-Tabelle, entsprechend den verschiedenen WHERE-Klauseln bei den unterschiedlichen Methoden, folgende Spaltenkombinationen mit je einem Index belegt:

Wissenschaftslehre

Wissenschaftslehre,Wissenschaftslehre

Wissenschaftslehre,Wissenschaftslehre,Bereich

Wissenschaftslehre,Wissenschaftslehre,Bereich,Wissenschaftler

Wissenschaftslehre,Wissenschaftslehre,Bereich,Wissenschaftler,Jahr,Monat

Jahr,Monat

An den Ergebnissen kann man deutlich erkennen, welche Performancesteigerungen allein die durch MySQL angebotenen Verbesserungen mit sich bringen. Diese sind vor allem bei Methode 1 und 2, d.h. bei den Methoden, welche viele kleine SQL-Abfragen absetzen, sichtbar. Da die vierte Methode nur ein einziges SQL-Statement absetzt und dieses einen sehr großen Teil der Datentabelle zurückliefert, bringt die Indizierung nicht viel Performanceverbesserung (die Indizes werden nicht verwendet). Sollte man auf nicht indizier-

ten Tabellen arbeiten bzw. nicht über die Indizierungsarten Bescheid wissen, empfiehlt sich auf jeden Fall die vierte Methode. Diese ist im Wesentlichen unabhängig vom gewählten Tabellentyp und in diesem Beispiel um vieles schneller als die anderen Methoden. Auch die Tatsache, dass der MySQL-Tabellentyp InnoDB fast um das dreifache langsamer ist bei den Methoden 1 bis 3 als die optimierte MyISAM-Tabelle und man oft nicht die Möglichkeit hat, den Tabellentyp zu ändern, spricht für die Methode mit einem SQL-Statement und einem sehr großen Zwischenspeicher. Der Baumaufbau ist bei den Methoden 1, 2 und 3 fast gleich, mit dem Unterschied, dass die dritte Methode gleich die Daten für den Datenbereich mit einliest. Diese werden bei letzterer Methode, wie auch bei der vierten, im Zwischenspeicher abgelegt. Dadurch entfallen die SQL-Abfragen beim Aufbau der HTML-Tabelle, entsprechend ist der HTML-Aufbau bei diesen Methoden wesentlich schneller als bei den Methoden 1 und 2.

Für alle weiteren Tests werden indizierte MyISAM Datentabellen verwendet.

Tabelle 8.6 zeigt den linearen Performanceverlust jeweils bei Verzehnfachung der Anzahl der Einträge in der Datentabelle. Die jeweiligen Merkmalsausprägungen bleiben bei den Datentabellen gleich. Es kommen immer nur neue Einträge zu den bereits bestehenden hinzu. Auf das Beispiel der Forschergruppen abgebildet heißt das, dass pro Beobachtungszeitraum viel mehr Einträge zu den Forschungsergebnissen der einzelnen Forschergruppen existieren, aber weder neue Wissenschaftsbereiche noch neue Gruppen entstehen. Das ist in der Realität sehr oft der Fall, z.B. wenn es um die Auswertung von Umsätzen der Abteilungen in einer Firma geht. Abb. 8.5 verdeutlicht diese Linearität.

Tabelle 8.7 zeigt die Ergebnisse des Pivot-Tabellen-Aufbaus aus einer Datentabelle mit 10^5 Einträgen, die im Vergleich zu der gleichgroßen Datentabelle aus Tabelle 8.6 sehr viele Merkmalsausprägungen besitzt. Die Anzahl der Blätter pro Knoten der vorletzten Ebene ist aber wieder vergleichbar mit den bereits untersuchten Datentabellen, d.h. es gibt mehr Gruppierungen bei den Datentabellen als in den vorigen Tests, aber die Anzahl der Kinder pro Knoten ist vergleichbar. Dieser Test simuliert den Fall, dass nicht mehr Ergebnisse der bestehenden Forschergruppen eingereicht werden, sondern es kommen weitere Forschergruppen, Bereiche, Wissenschaften und Wissenschaftslehren hinzu bzw. neue Abteilungen, die sich wieder in einer ähnlichen Struktur gliedern, wie die bereits bestehenden. Vergleicht man nun die 10^5 Datentabelle mit wenigen Ausprägungen und die gleich große Datentabelle mit vielen Ausprägungen fällt auf, dass die Performance bei der vierten Methode in Relation am größten abnimmt. Daraus kann man schließen, dass sich die Komplexität des Tabellenaufbaus besonders stark auf Methode 4 aus-

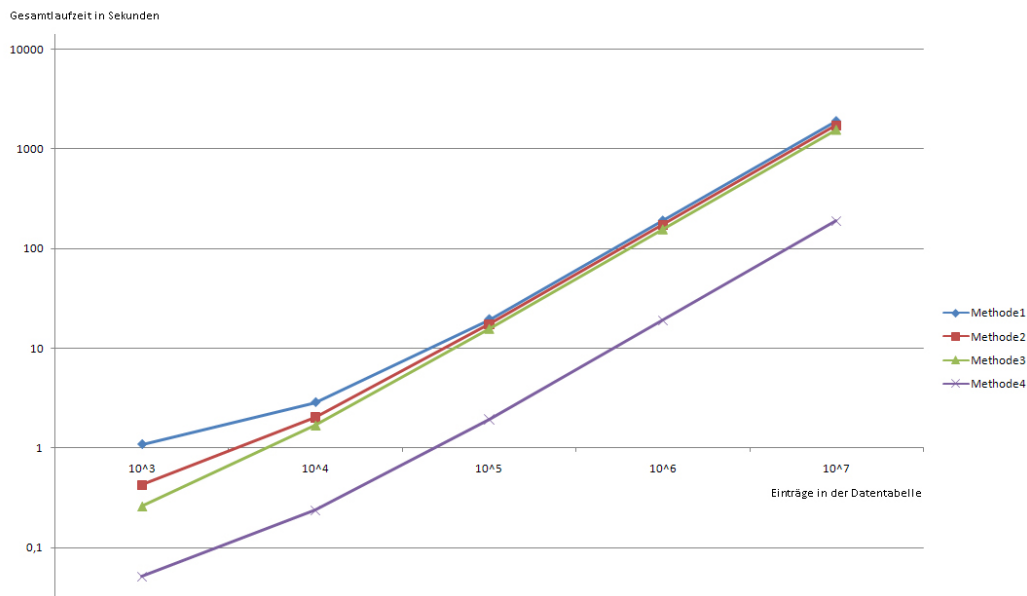


Abbildung 8.5: Die Ergebnisse aus Tabelle 8.6 auf einer doppelt logarithmischen Skala grafisch gegenübergestellt. Bis auf den Overhead, der als konstanter Wert weitgehend unabhängig von der Anzahl der Datensätze bleibt, steigen die Laufzeiten im Vergleich zu den Einträgen der Datentabelle linear an.

wirkt. Bei den Tests mit noch größeren Tabellen (10^6 Einträge) mit ähnlich komplexen Gruppierungsebenen fällt die vierte Methode ganz aus, da hier offensichtlich Laufzeit- bzw. Speicher-Limits von MySQL in Verbindung mit PHP erreicht werden.

Im unteren Bereich der Tabelle 8.7 wird der MySQL Query-Cache (der interne Zwischenspeicher für Ergebnisse von häufig wiederholten Abfragen – siehe 6.3.3) verwendet, so sind Methoden 2, 3 und 4 ab dem zweiten Aufruf nahezu gleich performant. Bei kleineren Datentabellen, wie zum Beispiel bei der Tabelle aus 8.3, sind die Methoden 2 und 3 sogar schneller als die vierte Methode (Gesamtlaufzeit in Sekunden Methode 2: 0.1328, Methode 3: 1.1729; Methode 4: 2.864). Die Laufzeit hängt hier fast nur mehr von der Dauer des HTML-Aufbaus ab ¹. Diese Erkenntnis ist durchaus interessant für häufig wiederholte Auswertungen mittels Pivot-Tabellen, vor allem dann, wenn viele unterschiedliche Benutzer die gleichen Daten abrufen und dadurch eine Zwischenspeicherung in der PHP-Session (Laufzeitumgebung pro Benutzer)

¹ Bei der trivialen Methode 1 bringt diese interne MySQL Verbesserung bei sehr vielen Abfragen keine Laufzeitverkürzung, da der Abfrage-Zwischenspeicher bei großen Tabellen so viele Ergebnisse nicht speichern kann.

nicht möglich ist.

Um aus der Datentabelle, die Ausgangstabelle für die Ergebnisse aus Tabelle 8.7 war, eine gleichwärtige, d.h. mit gleich vielen Merkmalen im Zeilen- wie auch im Spaltenbereich, Pivot-Tabelle mittels Microsoft Excel zu erstellen, benötigt diese Software im Schnitt 17,975ms. Sie liegt also deutlich über der vierten Methode, ist aber in dem Fall performanter als die dritte Methode. Bei kleineren Datentabellen ist der Vergleich schwierig, da die Ergebnislaufzeiten im Mikrosekundenbereich liegen und ohne Timestamps, die in Excel nicht gesetzt werden können, nicht messbar sind.

Die Auswirkungen von unterschiedlichen Aggregationsmethoden werden in Abb. 8.8 deutlich. Es werden Summenbildung und Varianz verglichen. Wie schon besprochen müssen für die Berechnung der Varianz alle einzelnen Datenwerte zur Verfügung stehen. Das bedeutet, dass bei Methode 4 pro Datenzeile nicht nur ein Wert von der Datenbank geliefert wird sondern alle Einzelwerte. Das erhöht den Datentransfer bei großen Tabellen enorm. Desweiteren müssen diese Einzelwerte für die Berechnung zwischengespeichert werden, wodurch auch der Speicherverbrauch steigt. Und schließlich wird pro Knoten eine aufwendige Berechnung über alle zugrundeliegenden Einzelwerte durchgeführt. Die Laufzeiten zeigen ganz deutlich, dass hier die Methode 4 nicht sehr effizient ist. In Methode 3 (wie auch bei den Methoden 1 und 2) werden die Berechnungen direkt in der Datenbank durchgeführt. Durch die performante Implementierung und den schnellen Zugriff auf die Daten in MySQL ist die Varianzberechnung nahezu gleich schnell wie die Summenbildung.

Tabelle 8.5: Datentabelle mit 10^4 Einträgen: 4 Zeilenmerkmale, davon 100 innere Ausprägungen und 180 Endausprägungen; 3 Spaltengruppierungen mit insgesamt 16 Spalten. Der obere Bereich zeigt den Umfang der Datenbanktransfers, des Speicherverbrauchs und die Anzahl der Berechnungen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen in Sekunden dargestellt. UG1, UG2 und UG3 sind drei verschiedene Test-Umgebungen.

	Methode 1			Methode 2			Methode 3			Methode 4		
Baumaufbau												
SQL-Abfragen		101			101			101			1	
Ø EZ pro SQL		2,8			2,8			11,4			450	
Ø ES pro SQL		1			1			5			8	
Ø Daten/SQL		2,8			2,8			57			3600	
Datenaufbau												
SQL-Abfragen		2240			280			0			0	
Ø EZ pro SQL		0,5			4,1			0			0	
Ø ES pro SQL		3			4			0			0	
Ø Daten/SQL		1,5			16,4			0			0	
Cache												
ZS		0			4			2300			2300	
PHP-Sum		0			0			0			2700	
PHP-Comp		0			0			700			1800	
Laufzeiten	UG1	UG2	UG3	UG1	UG2	UG3	UG1	UG2	UG3	UG1	UG2	UG3
InnoDB												
T(Baumaufbau)	3,5289		6,0174	3,5131		5,5564	6,3041		9,502	0,3064		0,2913
T(HTML – Aufbau)	76,1313		131,9854	10,3289		18,259	0,0204		3,8656	0,0233		3,3633
T(Gesamt)	79,6784		138,0202	13,8602		23,8357	6,3445		13,4042	0,3489		3,6748
MyISAM												
T(Baumaufbau)	1,7870		2,0723	1,7968		2,0889	3,2641		3,6436	0,2672		0,3525
T(HTML – Aufbau)	32,3646		40,5170	4,8708		6,0143	0,0221		2,2869	0,024		2,0085
T(Gesamt)	34,1703		42,5994	6,6857		8,1218	3,3057		5,9449	0,3104		2,3736
MyISAM + Index												
T(Baumaufbau)	1,0514	2,8034	0,9912	1,0493	3,4215	1,0430	2,4751	7,3561	2,4018	0,2695	0,5884	0,2413
T(HTML – Aufbau)	12,1768	59,5568	13,3724	3,1778	11,8116	4,5816	0,0212	0,0204	2,5504	0,0236	0,0227	2,8717
T(Gesamt)	13,2465	62,4308	14,3820	4,2456	15,2969	5,6323	2,5158	7,4382	4,9639	0,3125	0,6720	3,1296

Tabelle 8.6: Vergleich von drei Datentabellen mit 10^3 , 10^4 , 10^5 Einträgen: 4 Zeilenmerkmale, davon 10 innere Ausprägungen und 18 Endausprägungen; 3 Spaltengruppierungen mit insgesamt 16 Spalten. Der obere Bereich zeigt den Umfang der Datenbanktransfers, des Speicherverbrauchs und die Anzahl der Berechnungen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen in Sekunden dargestellt.

	Methode 1				Methode 2				Methode 3				Methode 4			
Baumaufbau																
SQL-Abfragen	11				11				11				1			
∅ EZ pro SQL	2,5				2,5				10,5				45			
∅ ES pro SQL	1				1				5				8			
∅ Daten pro SQL	2,5				2,5				52,3				360			
Datenaufbau																
SQL-Abfragen	224				28				0				0			
∅ EZ pro SQL	0,5				4,1				0				0			
∅ ES pro SQL	3				4				0				0			
∅ Daten pro SQL	1,5				16,4				0				0			
Cache																
ZS	0				4				230				230			
PHP-Sum	0				0				0				270			
PHP-Comp	0				0				70				180			
Laufzeiten	UG1	UG2	UG3		UG1	UG2	UG3		UG1	UG2	UG3		UG1	UG2	UG3	
10^3 Einträge																
T (Baumaufbau)	0,1067	0,3485	0,1576		0,1058	0,3665	0,2131		0,2396	0,6055	0,4341		0,0324	0,0772	0,0612	
T (HTML – Aufbau)	0,9576	5,1442	1,7072		0,3028	1,0268	0,582		0,0022	0,0021	0,0276		0,0023	0,0024	0,0363	
T (Gesamt)	1,0800	5,5950	1,8775		0,4246	1,4633	0,8073		0,2586	0,6694	0,47		0,051	0,1348	0,1047	
10^4 Einträge																
T (Baumaufbau)	0,7202	0,9036	1,1034		0,7207	0,8044	1,1164		1,6644	2,2736	2,1387		0,2047	0,2236	0,2576	
T (HTML – Aufbau)	2,1188	6,1768	3,5211		1,2901	1,7922	2,2853		0,0022	0,0021	0,0159		0,0023	0,0023	0,0165	
T (Gesamt)	2,8552	7,1451	4,638		2,0269	2,701	3,419		1,6833	2,3454	2,1709		0,2346	0,2941	0,2818	
10^5 Einträge																
T (Baumaufbau)	6,7344	6,5867	10,3528		6,729	6,9117	6,7338		15,6634	13,6264	15,0624		1,9178	1,7511	1,9811	
T (HTML – Aufbau)	12,7141	15,9073	14,3728		10,6161	10,1147	9,6918		0,0022	0,0022	0,0514		0,0024	0,0024	0,0657	
T (Gesamt)	19,4641	22,5503	24,7562		17,3607	17,0844	16,4493		15,682	13,6852	15,147		1,9346	1,808	2,0699	

Tabelle 8.7: Datentabelle mit 10^5 Einträgen: 4 Zeilenmerkmale, davon 1000 innere Ausprägungen und 1800 Endausprägungen; 3 Spaltengruppierungen mit insgesamt 16 Spalten. Der obere Bereich zeigt den Umfang der Datenbanktransfers, des Speicherverbrauchs und die Anzahl der Berechnungen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen in Sekunden dargestellt.

	Methode 1				Methode 2				Methode 3				Methode 4			
Baumaufbau	911				911				911				1			
SQL-Abfragen	2,8				2,8				11,4				4095			
Ø EZ pro SQL	1				1				5				8			
Ø ES pro SQL	2,8				2,8				57				32760			
Ø Daten/SQL																
Datenaufbau	2548				280				0				0			
SQL-Abfragen	0,5				4,1				0				0			
Ø EZ pro SQL	3				4				0				0			
Ø ES pro SQL	1,5				16,4				0				0			
Ø Daten/SQL																
Cache	0				4				20930				20930			
ZS	0				0				0				24570			
PHP-Sum	0				0				6370				16380			
PHP-Comp																
Laufzeiten	UG1	UG2	UG3		UG1	UG2	UG3		UG1	UG2	UG3		UG1	UG2	UG3	
T (Baumaufbau)	10,5850		16,5968		10,6048	28,5316	13,7769		24,6291	60,1342	31,4778		5,5025	4,5337	3,0889	
T (HTML – Aufbau)	118,7874		172,7514		31,828	100,1535	53,295		3,8333	5,0321	6,1625		3,6892	4,4865	6,2691	
T (Gesamt)	129,4134		189,4773		42,4746	128,7457	67,1312		29,0505	65,2274	37,7248		9,2537	9,1238	9,4281	
weiterer Aufruf mit internem MySQL Query-Cache																
Laufzeiten	UG1				UG1				UG1				UG1			
T (Baumaufbau)	–				0,2082				0,4805				0,4422			
T (HTML – Aufbau)	–				4,7662				4,4222				4,1757			
T (Gesamt)	–				5,0211				4,9623				4,6773			

Tabelle 8.8: Vergleich der Berechnung von Summen und Varianzen zwischen verschiedenen aufgebauten Datentabellen mit 10^4 Einträgen. Der obere Bereich zeigt die Parameter für den Datenbanktransfer, Speicherverbrauch und die Berechnungen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen in Sekunden dargestellt.

	10 ⁴ Einträge			10 ⁴ Einträge; komplexe Tabelle			10 ⁴ Einträge; vollständige Tabelle				
	Methode 3		Methode 4	Methode 3		Methode 4	Methode 3		Methode 4		
Baumaufbau SQL-Abfragen ∅ EZ pro SQL ∅ ES pro SQL ∅ Daten pro SQL	11		1	101	1	11		1			
	10,5		45 4720	11,4	450 4720	20		450 6648			
	5		8	5	8	5		8			
	52,3		360 37760	57	3600 37760	101,8		3600 53184			
Datenaufbau SQL-Abfragen ∅ EZ pro SQL ∅ ES pro SQL ∅ Daten pro SQL	0		0	0	0	0		0			
	0		0	0	0	0		0			
	0		0	0	0	0		0			
	0		0	0	0	0		0			
Cache ZS PHP-Sum PHP-Comp	230		2300	37760	448			53184			
	0		130 75520	0	1300 75520	0		704 106368			
	115		180 18880	1150	1800 18880	224		576 26592			
Laufzeiten	SUM	VAR	SUM	VAR	SUM	VAR	SUM	VAR			
T(Baumaufbau)	1,6644	1,6679	0,2047	2,4751	2,5189	0,2695	6,1101	1,9414	1,9374	0,2786	8,7223
T(HTML – Aufbau)	0,0022	0,0021	0,0023	0,0212	0,0021	0,0023	0,1018	0,0026	0,0029	0,0027	0,0793
T(Gesamt)	1,6833	1,6875	0,2346	2,5158	2,5677	0,3125	6,2618	1,9589	1,9776	0,2971	8,8406

Kapitel 9

Realisierung

Das webbasierte Tool zum Erstellen von Pivot-Tabellen ist eine PHP-Anwendung, welche HTML-Code generiert. Die Datentabellen werden bis auf weiteres aus einer MySQL Datenbank gelesen.

Im Folgenden werden kurz die wichtigsten Elemente des Programms erklärt. Anschließend folgt eine Beschreibung des Zusammenspiels der wichtigsten Funktionen der Methode 3 und 4.

9.1 Variablen

Datenbank-Konfiguration

Die Datenbankverbindungseigenschaften stehen in `config.php`: HOST, USER, PASSWORD, DATENBANK. Alle weiteren Variablen sind in `variables.php` zu deklarieren.

Eingabevariablen

Die Spalten- und Zeilenfelder aus denen die Pivot-Tabelle bestehen soll, müssen vorab definiert werden. Im Moment geschieht dies noch direkt in `variables.php`.

`$datatable` Datentabelle in der Datenbank, die als Ausgangstabelle für die Pivot-Tabelle dienen soll. Die Datentabelle wird im Moment noch direkt in php-MyAdmin eingegeben bzw. importiert. Ein Upload Tool für eine externe Datei in gängigen Formaten fehlt in der Applikation.

`$row_array[]` Liste der Merkmale der Zeilenfelder

`$column_array[]` Liste der Merkmale der Spaltenfelder

\$agregat Gewünschte mathematische Operation, die für die Berechnungen gewählt werden soll; default ist SUM

9.2 Objekt node

Die Implementierung verwendet die Klasse **node** (vgl. Abb. 9.1), die sich auf die Wurzel des Baumes, seine inneren Knoten und die Blätter vererbt. Die Wurzel sowie die inneren Knoten haben als zusätzliche Attribute ihre Kinder, die inneren Knoten und die Blätter jeweils einen Vaterknoten.

Für das Auf- und Zuklappen von Ebenen bzw. auch nur einzelner Knoten werden die Funktionen **open()** und **close()** verwendet. Dabei wird das Attribut **node.open** auf 1 gesetzt, wenn dieser aufgeklappt ist, andernfalls auf 0. Wird eine ganze Ebene zugeklappt, so werden die Knoten aller darunter liegenden Ebenen nicht angezeigt. Etwaige Berechnungen oder SQL-Abfragen kommen in diesem Fall nicht vor.

Mit **set_description()** werden je nach Methode Eigenschaften für die WHERE-Klauseln oder für Indizes des Zwischenspeichers abgelegt.

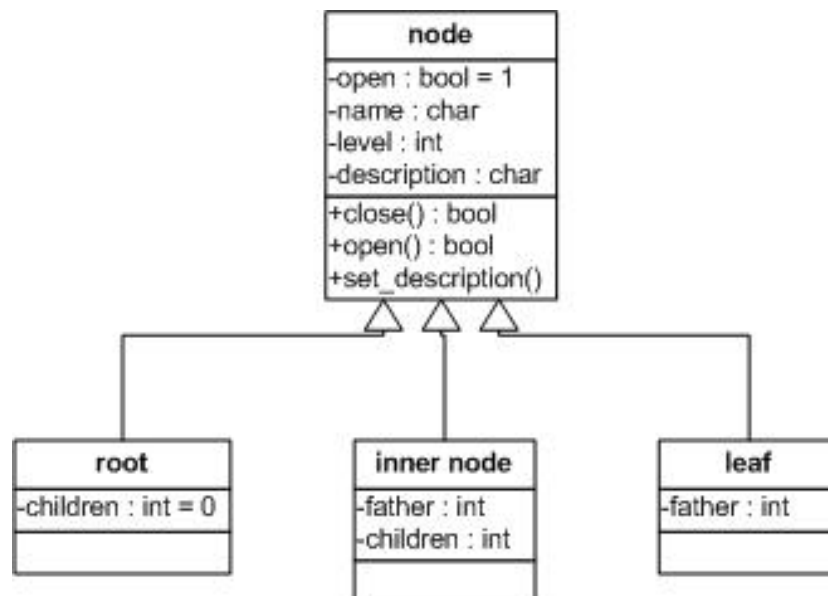


Abbildung 9.1: Klassendiagramm für die Klasse node.

9.3 Funktionen

Im Folgenden werden die wichtigsten Funktionen der Methoden 3 und 4 beschrieben (vgl. Abb. 9.2).

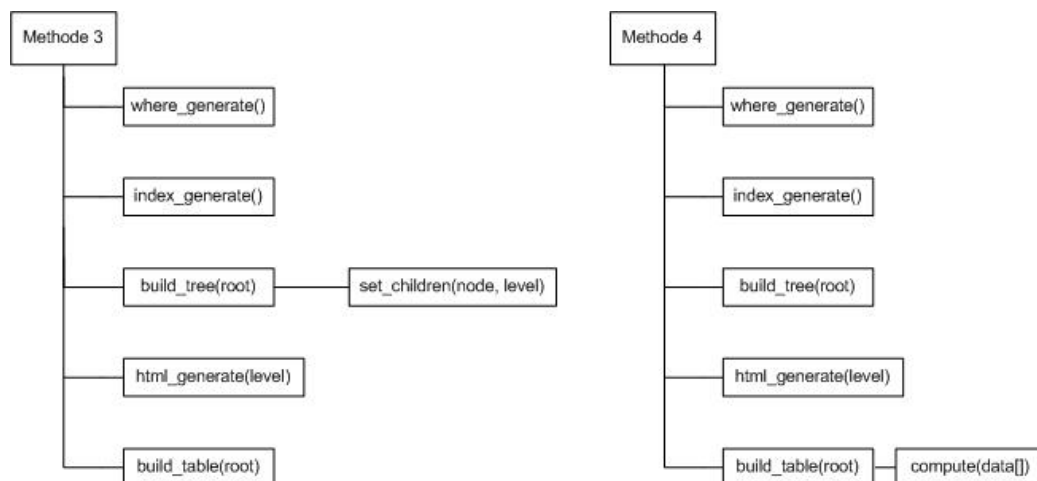


Abbildung 9.2: Zusammenspiel der wichtigsten implementierten Funktionen für die Methode 3 und 4.

`where_generate()`, `index_generate()`, `html_generate()` sind rekursive Funktionen und in den Methoden 3 und 4 gleich aufgebaut:

`where_generate()` generiert die WHERE-Klausel aus den Ausprägungen des Spaltenbereichs. Übergabeparameter ist die jeweils aktuelle Spaltengruppierungsebene. Der Aufruf erfolgt zu Beginn mit `where_generate(1)`.

`index_generate()` generiert ein Array mit Indizes aus den Ausprägungen des Spaltenbereichs. Dieses wird verwendet, um die Datenwerte für die jeweiligen Knoten abzuspeichern und dann wieder abzurufen. Übergabeparameter ist die jeweils aktuelle Spaltengruppierungsebene. Der Aufruf erfolgt zu Beginn mit `index_generate(1)`.

`html_generate()` generiert den HTML-Code für den Spaltenbereich aus den dementsprechenden Ausprägungen. Dabei werden die zu verbindenden Zellen berücksichtigt. Übergabeparameter ist die jeweils aktuelle Spaltengruppierungsebene. Der Aufruf erfolgt zu Beginn mit `html_generate(1)`.

Die Funktionen `build_tree()` und `set_children()` der dritten Methode bauen den Baum auf. Sie sind sehr ähnlich implementiert. `build_tree()` behandelt den Wurzelknoten und wird vom Hauptprogramm mit der Wurzel (Objekt der Klasse `node`) als Übergabeparameter aufgerufen. Diese ruft wiederum `set_children()` auf und übergibt als Parameter der Reihe nach die Kinder der Wurzel und immer den Wert `1` als die aktuelle Ebene.

`set_children()` baut den Baum von der ersten Ebene der Zeilengruppierungsmerkmale bis zu den Blättern rekursiv auf und übergibt jeweils die Kinder des aktuellen Knotens und die Zählvariable der aktuellen Ebene.

Die vierte Methode arbeitet mit der Funktion `build_tree()` das Ergebnis eines einzigen SQL-Statements ab und baut daraus den Gliederungsbaum und den Zwischenspeicher mit den Datenwerten auf. Sie wird ähnlich wie bei Methode 3 im Hauptprogramm mit dem Wurzelknoten als Parameter aufgerufen, ist aber nicht rekursiv, da sie sequentiell die Ergebniszeilen abarbeitet.

Die endgültige Ausgabe der Pivot-Tabelle erfolgt bei beiden Ansätzen gleich in der Funktion `build_table()`, die wieder mit dem Wurzelknoten als Parameter vom Hauptprogramm aufgerufen wird. Falls komplexe Berechnungsoperationen für die Analyse notwendig sind, wird vor der Ausgabe in der Methode 4 noch eine Berechnungsroutine `compute()` mit dem Array der für den aktuellen Knoten abgespeicherten Werten aufgerufen.

Kapitel 10

Schlussfolgerungen

Das Projektziel war es, eine effiziente, plattformunabhängige Softwarelösung zu schaffen, welche aus Datentabellen übersichtliche Pivot-Tabellen generiert. Als Ergebnis dieser Arbeit wurde ein PHP-Programm entwickelt, welches Daten aus einer MySQL-Datenbank ausliest und aufbereitet. Die Software wurde beim Auftraggeber CARE Österreich erfolgreich zum Einsatz gebracht.

10.1 Wahl der Methode

Nachdem bei der Verarbeitung großer Datenmengen die Performance eine wichtige Rolle spielt, wurden verschiedene Lösungswege untersucht. Die Ergebnisse haben gezeigt, dass die Wahl der Methode in erster Linie von der Art der Datentabelle und der geforderten Aggregationsfunktion abhängt. Vor allem die vorgestellten Methoden 3 (Mischung aus SQL-Abfragen und Zwischenspeicher) und 4 (eine einzige SQL-Abfrage, deren Ergebnisse für die HTML Ausgabe sowie weitere Berechnungen verwendet werden) konkurrieren bezüglich Laufzeiten und Speicherbedarf. In sehr vielen Fällen handelt es sich um Datentabellen mit sehr vielen Merkmalsausprägungen in der untersten Zeilengruppierungsebene. Gerade bei einer solchen Datenverteilung sind die beiden Methoden vergleichbar. Ein Beispiel dafür sind Untersuchungen im Bereich der Demografie, hier haben Datentabellen meist als unterste Gliederungsebene die Bevölkerung selbst, ob namentlich genannt oder nur in Pseudonymen. Das statistische Zentralamt sammelt Daten von jedem einzelnen Bewohner ein und gliedert diese dann z.B. geografisch nach Bezirken, Bundesländern, demografisch nach Alter, Größe, ökologisch nach Einkommen usw. In all diesen Fällen gibt es vergleichsweise wenige Ausprägungen bei den Merkmalen der ersten bis zur vorletzten Gliederungsebene des Zeilenbereichs, 9 Bundesländer, 10 Altersklassen, Einkommensklassen, aber im

Verhältnis dazu viele Ausprägungen des untersten Merkmals. Meist interessieren dann auch die konkreten Daten der untersten Eben gar nicht, da man ja keine Aussagen über einen Einzelnen treffen will, sondern auf Entwicklungen bei der Gesamtheit schließen möchte. Gerade in diesem Fall wirken sich dann Aggregationsfunktionen (wie z.B. die Varianz), welche alle zugrundeliegenden Daten für die Berechnung benötigen, nachteilig auf Methode 4 aus. Für einfache Funktionen (wie z.B. die Summenbildung) ist jedoch Methode 4 bezogen auf die Gesamt-Laufzeit konkurrenzlos und bei großen Datentabellen auch performanter als Microsoft Excel, die auf dem Gebiet der Pivot-Tabellen-Erstellung bisher bekannteste Software.

10.2 Future Work

Das im Rahmen dieser Diplomarbeit entwickelte Tool zur Online-Datenanalyse mit Hilfe von Pivot-Tabellen basiert auf den alten und etablierten Webtechnologien HTML, PHP, MySQL. Um dem Betrachter dieser Pivot-Tabellen noch mehr Komfort und Flexibilität zu geben bietet es sich an, dieses Tool auf WEB2.0 Standard zu bringen und mit AJAX-Technologien zu arbeiten.

AJAX steht für *Asynchronous Javascript And Extensible Markup Language* und ist ein seit 2005 bekannter Begriff. Auf wen dieser zurückzuführen ist, ist nicht ganz klar, wurde aber zum ersten Mal von Jesse James Garrett in einem Beitrag über einen neuen Ansatz für Internetanwendungen, der sich an der Funktionsweise von Desktop-Programmen orientiert, verwendet (vgl.(Car06)). Eine einfache Internetseite, so wie die Pivot-Tabelle eine ist – ohne Frames, ohne Flash, ohne JavaApplets – ist prinzipiell statisch. Gibt man im Browser die Webadresse der Pivot-Tabelle ein, wird eine Verbindung zum Webserver, auf dem diese liegt, aufgebaut und die entsprechenden Daten werden geladen. Zusätzlich laufen natürlich im Hintergrund noch die SQL-Abfragen. Trotz Optimierung benötigt dieser Ladevorgang einige Zeit. Zuerst wird der Spaltenbereich aufgebaut, dann zeilenweise der Zeilenbereich und die zusätzlichen Datenfelder. Entscheidet sich der Betrachter der Analyse eine Ebene weniger im Bereich der Zeilengruppierungen anzuzeigen, z.B. nicht mehr nach *Wissenschaftsgruppen* sondern nur nach *Bereichen* zu gliedern, und entfernt die *Wissenschaftler*-Ebene, wird wieder die ganze Seite neu geladen. Auch wenn nun die Werte aus dem Zwischenspeicher kommen, müssen trotzdem bereits angezeigte Teile wieder neu geladen werden und verbrauchen daher Zeit. Auch der Spaltenbereich, der immer gleich ist, muss trotzdem neu geladen werden. Via AJAX ist es möglich, eine Verbindung zum Webserver aufzubauen und nur die Teile zu holen, die sich geändert

				2006					
				Jänner		Februar		März	
Wissenschaftslehre ±	Wissenschaft ±	Bereich ±	Wissenschaftler	Ideen	Patente	Ideen	Patente	Ideen	Pat
Fachwissenschaft ±				3585	264	1952	127	5013	247
	Naturwissenschaften ±			3585	264	1952	127	5013	247
		Chemie ±				1952	127	3432	156
		Physik ±		3585	264			1581	91
			Doppler	1128	100			1581	91
			Faraday						
			Maxwell						
			Newton	2457	164				
Universalwissenschaft ±				4555	249	6509	363	9184	626
	Formalwissenschaften ±			1633	105	6509	363	4355	351
		Informatik ±		1633	105	3586	212	2730	221
		Mathematik ±				2923	151	1625	130
	Geisteswissenschaften ±			2922	144			4829	275
		Philosophie ±		2922	144			4829	275

Abbildung 10.1: Pivot-Tabelle, die bis zur Zeilenebene des Bereiches analysiert wird und in der nur zusätzlich die Daten der Wissenschaftsgruppen des Bereiches Physik angezeigt werden.

haben, sei es der ganze Datenbereich oder nur Teile davon, und diese in die bereits bestehende Seiten an klar definierten Stellen im Browser einzufügen. Ein Beispiel: Der Betrachter analysiert gerade die *Bereiche* ohne Aufspaltung auf die *Wissenschaftsgruppen* und stellt fest, dass der Bereich *Physik* eine sehr hohe Ideenquote hat und möchte nur für diesen Bereich die Aufspaltung auf die *Forschungsgruppen* haben, um zu wissen, welche davon so erfolgreich ist. Abb. 10.1 zeigt die dazugehörige Pivot-Tabellen-Ansicht.

Durch AJAX müssen nicht mehr alle Daten des Datenbereiches geladen werden, sondern es werden nur die neu dazugekommenen Daten aus der Datenbank geholt und vom Webserver übertragen. In dem oben beschriebenen Beispiel wären das die einzelnen Daten der *Wissenschaftsgruppen* des Bereiches *Physik*, die nur aus dem Zwischenspeicher geholt werden müssen (und die aktuellen Daten für das *Jahr 2008*, die neu aus der Datenbank abgefragt werden müssen). Ein Neuladen der ganzen Seite ist nicht mehr notwendig. Das bringt natürlich enorme Performance-Steigerung und würde die „Web-basierte Darstellung großer Datenmengen als Pivot-Tabelle mithilfe ressourcenoptimierter Aggregationsverfahren“ wesentlich attraktiver machen!

Tabellenverzeichnis

8.1	Theoretische Analyse der Methode reiner SQL-Abfragen: pro Knoten pro Spalte.	67
8.2	Theoretische Analyse der Methode mit jeweils einer SQL-Abfrage pro Knoten.	69
8.3	Theoretische Analyse der Methode des Zwischenspeicherns mit mehreren SQL-Abfragen.	71
8.4	Theoretische Analyse der Methode des Zwischenspeicherns mit einer SQL-Abfrage.	73
8.5	Datentabelle mit 10^4 Einträgen: 4 Zeilenmerkmale, davon 100 innere Ausprägungen und 180 Endausprägungen; 3 Spaltengruppierungen mit insgesamt 16 Spalten. Der obere Bereich zeigt den Umfang der Datenbanktransfers, des Speicherverbrauchs und die Anzahl der Berechnungen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen in Sekunden dargestellt. UG1, UG2 und UG3 sind drei verschiedene Test-Umgebungen.	79
8.6	Vergleich von drei Datentabellen mit $10^3, 10^4, 10^5$ Einträgen: 4 Zeilenmerkmale, davon 10 innere Ausprägungen und 18 Endausprägungen; 3 Spaltengruppierungen mit insgesamt 16 Spalten. Der obere Bereich zeigt den Umfang der Datenbanktransfers, des Speicherverbrauchs und die Anzahl der Berechnungen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen in Sekunden dargestellt.	80
8.7	Datentabelle mit 10^5 Einträgen: 4 Zeilenmerkmale, davon 1000 innere Ausprägungen und 1800 Endausprägungen; 3 Spaltengruppierungen mit insgesamt 16 Spalten. Der obere Bereich zeigt den Umfang der Datenbanktransfers, des Speicherverbrauchs und die Anzahl der Berechnungen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen in Sekunden dargestellt.	81

8.8	Vergleich der Berechnung von Summen und Varianzen zwischen verschieden aufgebauten Datentabellen mit 10^4 Einträgen. Der obere Bereich zeigt die Parameter für den Datenbanktransfer, Speicherverbrauch und die Berechnungen. Im unteren Bereich werden die Ergebnisse der Laufzeitmessungen in Sekunden dargestellt.	82
-----	---	----

Abbildungsverzeichnis

1.1	Horizontale Tabelle.	7
1.2	Beispiel einer Kreuz-Tabelle.	8
1.3	Aufbau einer Pivot-Tabelle.	10
1.4	Pivot-Tabelle mit Spaltensummen.	11
1.5	MS Excel Pivot-Tabelle.	14
1.6	MS Access Pivot-Tabelle.	15
1.7	Ein Beispiel einer CARE Pivot-Tabelle.	18
2.1	Daten- und Pivot-Tabelle.	23
3.1	ungerichteter, zyklischer Graph.	25
3.2	gerichteter, zyklischer Graph.	25
3.3	gerichteter, azyklischer Graph.	25
3.4	Baum.	26
3.5	Begriff Baum für die Pivot-Tabelle.	27
3.6	Ausschnitt einer Datentabelle und des entsprechenden Baumes.	30
3.7	Ergebnis einer SQL Abfrage nach dem ganzen Zeilenbereich.	31
4.1	Ausprägungen der verschiedenen Ebenen des Spaltenbereichs.	34
5.1	Zeilen einer Datentabelle, welche die Bedingungen für die Zelle des Datenbereichs der Pivot-Tabelle 5.2 erfüllen.	38
5.2	Datenzeile einer Pivot-Tabelle.	39
5.3	Ergebnis-Pivot-Tabelle erstellt aus einer Datentabelle mit 1000 Einträgen.	41
5.4	Der zu der in Abb. 5.3 dargestellten Pivot-Tabelle gehörende Gliederungsbaum.	42
6.1	Ergebnis einer SQL-Abfrage für einen einzelnen Knoten und der vorletzten Spaltengruppierung.	46
6.2	Pfad im Baum, der einer Zeilengruppierung entspricht.	46
6.3	Ergebnis der SQL-Abfrage für einen Knoten gruppiert nach Jahr und Monat.	49

7.1	Pivot-Tabelle mit verschiedenen Ansichten des Zeilenbereichs .	55
7.2	Ergebniszeile der erweiterten SQL-Abfrage.	55
7.3	Mehrdimensionales Array zum Cachen.	56
7.4	Ein einziges SQL liefert alle notwendigen Daten für den Baumaufbau und den Datenbereich.	61
8.1	Ergebnis einer SQL-Abfrage für einen einzelnen Knoten und der vorletzten Spaltengruppierung.	67
8.2	Ergebnis der SQL-Abfrage für einen Knoten gruppiert nach Jahr und Monat.	69
8.3	Ergebnis der SQL-Abfrage die direkt beim Baumaufbau die Werte für den Datenbereich liefert (Ausschnitt).	70
8.4	Ein einziges SQL-Statement liefert alle notwendigen Daten für den Baumaufbau und den Datenbereich (Ausschnitt).	72
8.5	Die Ergebnisse aus Tabelle 8.6 auf einer doppelt logarithmischen Skala grafisch gegenübergestellt. Bis auf den Overhead, der als konstanter Wert weitgehend unabhängig von der Anzahl der Datensätze bleibt, steigen die Laufzeiten im Vergleich zu den Einträgen der Datentabelle linear an.	77
9.1	Klassendiagramm für die Klasse node.	84
9.2	Zusammenspiel der wichtigsten implementierten Funktionen für die Methode 3 und 4.	85
10.1	Pivot-Tabelle, die bis zur Zeilenebene des Bereiches analysiert wird und in der nur zusätzlich die Daten der Wissenschaftsgruppen des Bereiches Physik angezeigt werden.	89

Literaturverzeichnis

- [baa08] *baaGrid*. <http://phpclasses.byting.at/browse/package/1298.html>. Version: 2008
- [Car06] CARL, Denny: *Praxiswissen AJAX*. O'Reilly, 2006. – ISBN 3-8972-1451-2
- [Ehr86] EHRENBURG, A.S.C.: Reading a Table: An Example. In: *Applied Statistics* 35 (1986), 1, Nr. 35, S. 237
- [HZ70] HANS ZEISEL, Carmen L.: *Die Sprache der Zahlen*. Kiepenheuer & Witsch, 1970
- [JLG06] JONATHAN L. GROSS, Jay Y.: *Graph Theory and Its Applications*. Chapman & Hall /CRC, 2006. – ISBN 1-5848-8505-X
- [KB05] KLAUS BACKHAUS, Bernd Erichson und Rolf W. Wulff Plinke P. Wulff Plinke: *Multivariate Analysemethoden*. Springer, 2005. – ISBN 3-540-27870-2
- [LM94] LUDWIG-MAYERHOFER, Wolfgang: Kleine Anmerkung, die Verbesserung der Darstellung von Kreuztabellen betreffend. In: *Kölner Zeitschrift für Soziologie und Sozialpsychologie* (1994), 1, Nr. 1, S. 122
- [Mic08] *Microsoft Hilfe und Support*. <http://support.microsoft.com/kb/820742/>. Version: 2008
- [MK05] MICHAEL KOFLER, Bernd Ö.: *PHP5 & MySQL5*. Addison - Wesley, 2005. – ISBN 3-8273-2190-5
- [RE05] REINHOLD ESTERBAUER, Bernd O.: Theorie der Natur- und Kulturwissenschaften / Fakultätsvertretung Theologie der Karl Franzens Universität Graz. Version: 2005. http://oeh-fv-theo.uni-graz.at/_pdf/4652fdec713fc.pdf. 2005. – Forschungsbericht

- [Sch04] SCHWABE, Rainer W.: *Excel 2003, spielend einfach kalkulieren*. Markt+Technik Verlag, 2004. – ISBN 3–8272–6584–3
- [Sch05] SCHLOSSNAGLE, Georg: *Professionelle PHP5 Programmierung*. Addison - Wesley, 2005. – ISBN 3–8273–2198–0
- [Sch07] *Kapitel Pivot-Tabellenberichte: Nutzen und Möglichkeiten*. In: SCHELS, Ignatz: *ControllerPLUS Excel: „Excel-Know-how“*. Marlene Str. 2, 77656 Offenburg : Verlag für ControllingWissen AG, 2007
- [Sta08] *Statistik Austria*. http://www.statistik.at/web_de/statistiken/bevoelkerung/haushalte_familien_lebensformen. Version: 2008
- [SZ03] SZ: Der Datenknecht. In: *PCTip* (2003), 9, Nr. 9, 34. <http://www.pctipp.ch/>
- [THC07] THOMAS H. CORMEN, Ronald L. Rivest Clifford S. Charles E. Leiserson L. Charles E. Leiserson: *Algorithmen- eine Einführung*. Oldenbourg Wissenschaftsverlag, 2007. – ISBN 3–4865–8262–3
- [ZB04] ZAWODNY, Jeremy D. ; BALLING, Derek J.: *High Performance MySQL*. O'Reilly & Associates, 2004. – ISBN 0–5960–0306–4