TECHNISCHE UNIVERSITÄT
ILMENAU

Institut für Praktische Informatik und Medieninformatik
Fakultät für Informatik und Automatisierung
Fachgebiet Datenbanken und Informationssysteme

Bachelorarbeit

Automatisierter Entwurf eines Schemas für Bitwise Dimensional Clustering

(Automated Design of Bitwise Dimensional Clustering)

vorgelegt von:	Steve Göring
Matrikel:	43952
Betreuer:	Prof. Dr.–Ing. Kai-Uwe Sattler Dipl.–Inf. Stephan Baumann

Ilmenau, den 30. Januar 2012

Danksagung

„Leider läßt sich eine wahrhafte Dankbarkeit mit Worten nicht ausdrücken."
Johann Wolfgang von Goethe

Besonderen Dank möchte ich meinen Betreuern Dipl.–Inf. Stephan Baumann und Prof. Dr.–Ing. Kai-Uwe Sattler aussprechen. Desweiteren bedanke ich mich bei meinen Freunden in Ilmenau, die mir auch in den stressigsten Situationen immer einen anderen Blickwinkel verschafft haben. Bei Martin Meyer möchte ich mich besonders bedanken, ohne ihn wäre ich an vielen Wochentagen verhungert ;-).

Zu guter Letzt gehört meiner Mutter Petra Göring und meinem Onkel Rolf Göring Dank, da sie mir das Studium ermöglicht haben und mir immer zur Seite standen.

Besonders wichtig für diese Bachelorarbeit war außerdem die Entdeckung des Kaffees und des 26 Stunden Tages, denn ohne beide wären große Teile dieser Arbeit nie vollständig geworden.

„Ich habe solange ein Motivationsproblem, bis ich ein Zeitproblem habe!"

Unbekannt

Inhaltsverzeichnis

Abbildungsverzeichnis

Phantasie ist wichtiger als Wissen, denn Wissen ist begrenzt.

Albert Einstein

1 Einleitung

1.1 Motivation

Jeden Tag werden mehr und mehr Daten von Internetnutzern, Geräten oder anderen Quellen erzeugt. Firmen und Organisationen wissen, dass es notwendig ist, diese Daten zu analysieren, egal ob sie Geschäftsprozessen oder öffentlichen Quellen entspringen. Sie legen daher Data-Warehouses an. Die Daten aus den unterschiedlichen Quellen werden für das Data-Warehouse zusammengefasst, transformiert und anschließend in einer Datenbank abgespeichert. Die Datenbank muss dabei Terabytes an Daten schnell verarbeiten können.

Bei einem Online Versandhaus werden zum Beispiel täglich:

- ▷ tausende Bestellungen versendet
- ▷ neue Kunden registriert
- ▷ neue Verkäufer eingetragen
- ▷ weitere Produkte angeboten

und vieles mehr.

Um all diese Daten abzuspeichern bedarf es einem cleveren Datenbankschema und einem performanten Datenbanksystem.

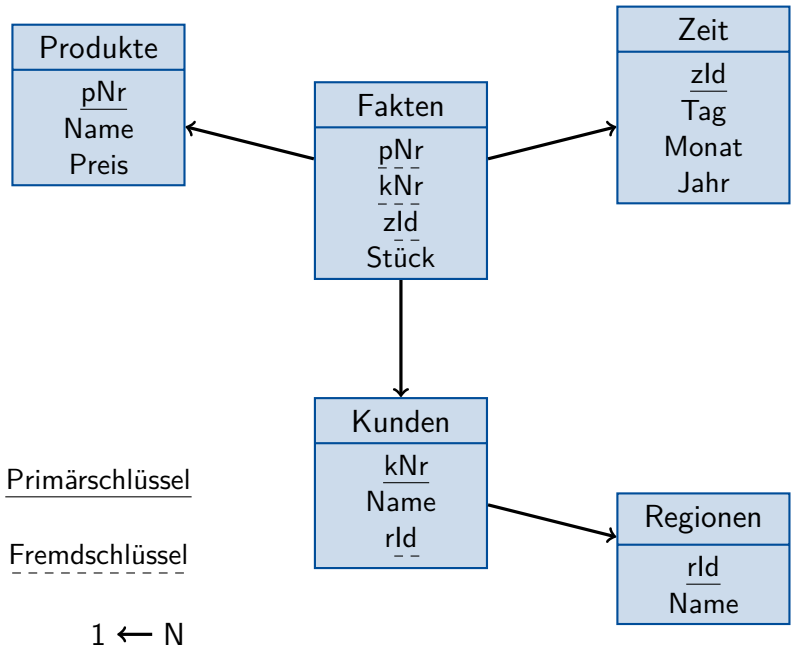


Abbildung 1.1: Data-Warehouse am Beispiel Online Versandhaus

Für ein solches Online Versandhaus ergibt sich beispielsweise das in Abbildung 1.1 dargestellte und vereinfachte Datenbankschema.

Eine Tabelle `Kunden` speichert alle Kundeninformationen (z.B. Name, Kundennummer, Wohnregion, ...). Außerdem gibt es noch eine Tabelle für alle `Produkte` und für die `Zeit`. Die Tabelle `Fakten` sammelt Informationen über den Verkauf, gespeichert wird dabei, welcher Kunde wie viele Produkte zu welchem Zeitpunkt gekauft hat.

Nehmen wir beispielsweise an, dass die Kunden-, die Produkt- und die Zeittabelle jeweils 1000 Einträge haben. So ergäbe sich bei einer Anfrage, um den gesamten Erlös in einem bestimmten Zeitraum zu ermitteln, ein Verbund über die 3 Tabellen. Im schlechtesten Fall erzeugt man –bei dem vereinfachten Beispiel– damit eine Anfrage über $1000 \cdot 1000 \cdot 1000 = 1$ Mio. Datensätzen. In realen Szenarien sind mehrere tausend Datensätze pro Tabelle gespeichert, was bedeutet, dass noch größere Datenmengen verarbeitet werden müssen.

In der Regel wird nicht nur ein Typ von Anfragen stattfinden. Es ergeben sich dadurch viele unterschiedliche Anfragen, die über fast alle Tabellen jeweils einen Join durchführen müssen. Wichtig für die Firmen und Organisation ist eine schnelle Analyse der Daten. Aus diesem Grund muss das Datenbanksystem die Anfragen performant ausführen. Reine relationale Datenbanksysteme können die Datenmengen, die in einem Data-Warehouse anfallen, nicht ohne Erweiterungen performant bewältigen. Ein Erweiterungskonzept ist das multidimensionale Clustern der Daten. Beim multidimensionalen Clustern werden Daten zunächst in Fakten und Dimensionen eingeteilt. Die Fakten werden dabei durch die Dimensionsfestlegungen geclustert, d.h. Daten, die durch die gleiche Dimension beschrieben werden, liegen auch physikalisch dicht beieinander. Die Informationen der Cluster werden in den Anfragen benutzt, um die Datenmenge zu reduzieren, wodurch sie schneller ablaufen. In einem typischen Data-Warehouse Szenario sind die Daten in typischen Schemata eingeteilt, wie z.B. Stern- oder Schneeflocken-Schema.

1.2 Data-Warehouse Datenbank-Schema

1.2.1 Sternschema

Das Sternschema ist ein einfaches Data-Warehouse-Schema. Es besteht aus einer zentralen Faktentabelle und vielen einzelnen, nicht normalisierten, Dimensionstabellen (siehe Abbildung 1.2). Die Faktentabelle hat zu jeder Dimensionstabelle (`Dim1` bis `DimX`) über Fremdschlüssel eine N:1 Beziehung (z.B. `fk1` zu `pk1`). Außer den Fremdschlüsseln zu den Dimensionstabellen besitzt sie noch weitere Attribute. Sie tragen die Informationen und werden Fakten genannt.

Ein großer Vorteil des Sternschemas ist, dass man für Anfragen immer nur Joins über alle Dimensionstabellen ausführen muss, die sich genau eine Ebene tiefer als die Faktentabelle befinden. Nachteilig dagegen ist eindeutig die redundante Speicherung der Daten in den Dimensionstabellen, weil sie nicht normalisiert sind. Das Einfügen neuer bzw. Ändern alter Daten wird jedoch durch die Redundanzen erschwert.

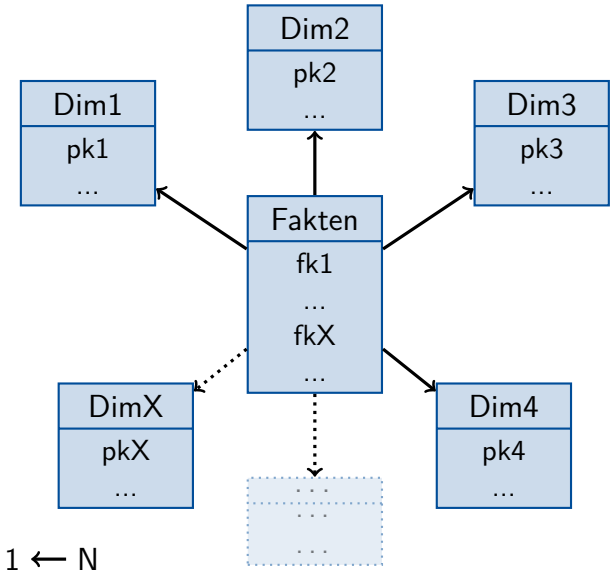


Abbildung 1.2: Sternschema

1.2.2 Schneeflockenschema

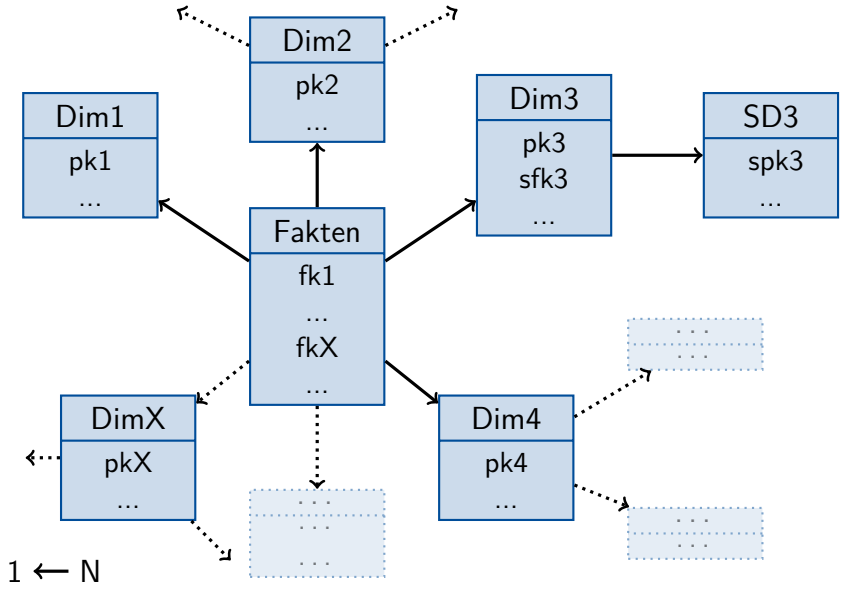


Abbildung 1.3: Schneeflockenschema

Im Vergleich zum Sternschema normalisiert man beim Schneeflockenschema die Dimensionstabellen (siehe Abbildung 1.3). Es wird dadurch der Nachteil der redundanten Speicherung ausgeglichen. Die Anfragen werden jedoch komplizierter, weil die Joins über mehrere Zwischentabellen stattfinden. In Abbildung 1.3 wird exemplarisch die Tabelle `Dim3` weiter in eine Subdimension `SD3` zerlegt (erreichbar über $sfk3 \rightarrow spk3$).

Das Anfangsbeispiel (Abbildung 1.1) ist ein typischer Vertreter des Schneeflockenschemas. Ein reines Sternschema würde man erhalten, wenn man die Regionstabelle mit der Kundentabelle zu einer Tabelle zusammenfasst.

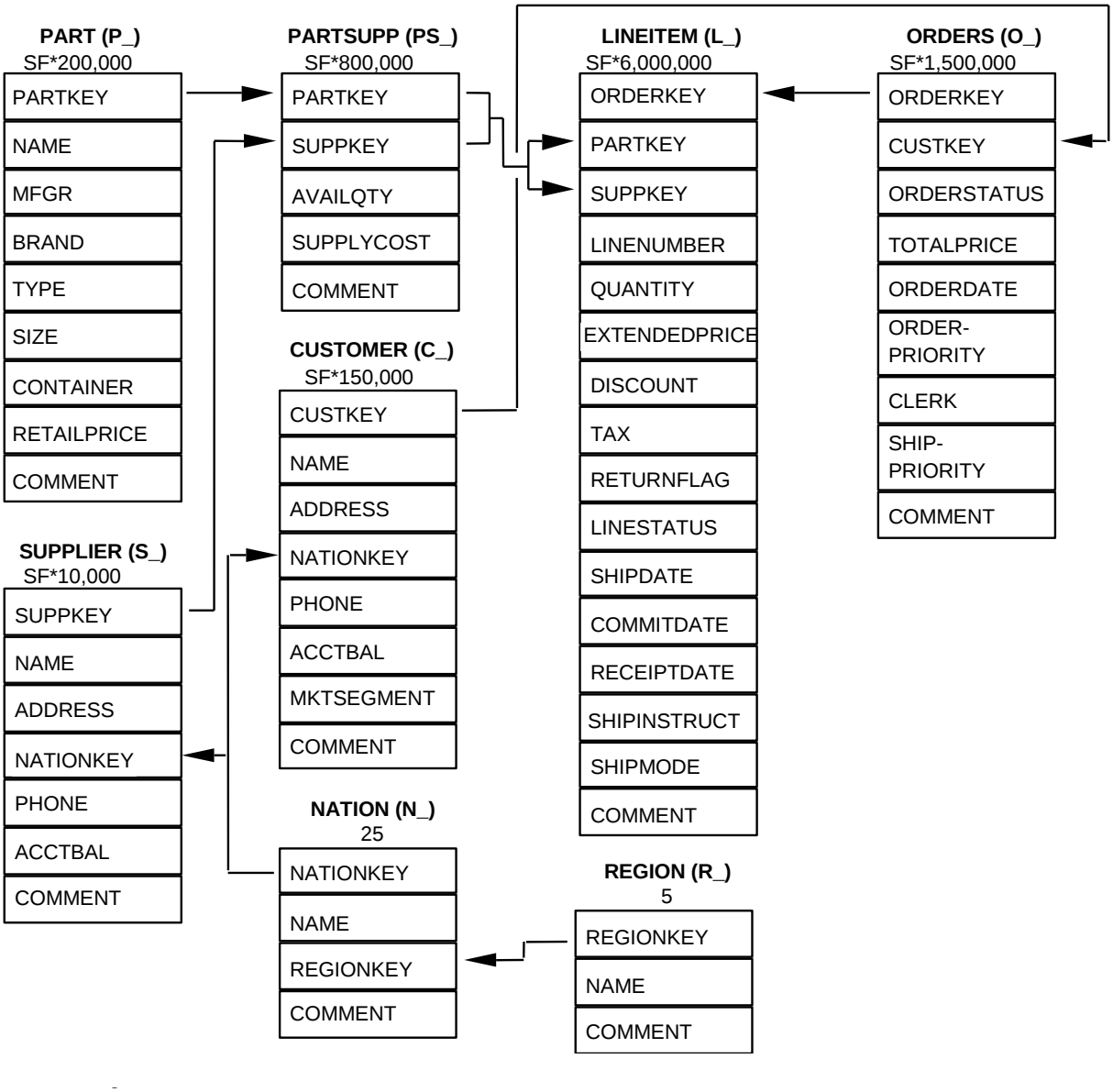


Abbildung 1.4: TPC-H Schema, [TPC, Seite 12]

1.3 TPC-H

Zum Vergleichen von relationalen Datenbanksystemen gibt es z.B. den TPC-H Benchmark. TPC-H ist ein sogenannter Decision-Support[1] Benchmark. Er beinhaltet geschäfts-orientierte Ad-hoc Anfragen, d.h. Anfragen ohne Vorwissen. Die Anfragen und Datenmenge der Datenbank wurden so ausgewählt, dass sie industriellen Anforderungen genügen. Gleichzeitig wurde versucht, die Implementierung so einfach wie möglich zu gestalten . Der Benchmark veranschaulicht Decision-Support-Systeme, die:

- ▷ große Datenmengen untersuchen
- ▷ sehr komplexe Anfragen stellen.

Die Abbildung 1.4 zeigt das Datenbankschema des TPC-H Tests. Es besteht aus acht Tabellen. Die in der Abbildung dargestellten Beziehungen zwischen den Tabellen sind $1 \rightarrow N$ Beziehungen. In der Grafik bezeichnet SF den Skalierungsfaktor, z.B. ergibt Skalierungsfaktor $SF = 1$ einen Datenbankinhalt von ca. 1 Gigabyte [TPC].

1.4 Vectorwise

Die in dieser Arbeit erreichten Erkenntnisse werden in dem Datenbanksystem Vectorwise implementiert. Vectorwise ist eine kommerzielles relationales Datenbanksystem der Firma Actian[2] und ist eine Weiterentwicklung von X100 (siehe [ZB, ZBNH05]). Vectorwise ist ein so genannter Columnstore, d.h. die Datensätze werden spaltenweise auf der Festplatte gespeichert [Act11]. Im Vergleich zu anderen klassischen relationalen Datenbanksystemen ist Vectorwise schneller, da es Funktionen moderner CPUs zur Performanzsteigerung benutzt.

Vectorwise ist (siehe [Act11]):

- ▷ vektor-orientiert: eine Operation wird auf eine Menge von Daten angewendet
- ▷ CPU-Cache optimiert: die Daten werden nach Möglichkeit im CPU-Cache verarbeitet, dadurch erhöht sich die Geschwindigkeit im Vergleich zur Verarbeitung im RAM
- ▷ spalten-orientiert: die Daten werden spaltenweise auf der Festplatte gespeichert, daher werden die Festplattenzugriffe minimiert
- ▷ parallelisiert: alle vorhandenen physikalischen CPUs werden bei Anfragen ausgenutzt, Anfragen können aus dem Grund parallel ausgeführt werden, wodurch die Performance steigt
- ▷ komprimiert.

Vectorwise platziert sich bei dem TPC-H Datenbanktest (100 GB Daten) unter den Top-5[3] der Datenbanksysteme.

[1] System, welches zur Entscheidungsunterstützung benutzt wird, ein Data-Warehouse Szenario
[2] http://www.actian.com/products/vectorwise Stand 30.01.2012
[3] http://www.tpc.org/tpch/results/tpch_perf_results.asp Stand 30.01.2012

1.5 Bitwise Dimensional Clustering

Allgemein. Bitwise Dimensional Clustering (kurz BDC) ist ein multidimensionales Speichersystem für Columnstores. Es findet momentan Anwendung im Kern von Vectorwise, kann aber theoretisch auch für andere Columnstores benutzt werden. BDC stellt eine Erweiterungstechnik für das multidimensionale Clustern von Daten in relationalen Datenbanken dar. Es transformiert dabei das multidimensionale Clusterungsproblem in ein Tupel Ordnungsproblem unabhängig von der physikalischen Speicherung [BBS11].

Dimensionen. BDC bildet die Dimensionen auf Integerzahlen ab. Das bedeutet, eine BDC geclusterte Tabelle erhält einen Sortierschlüssel `__bdc__`, der aus den Dimensionsinformationen durch bitweise Verschachtelung der Dimensionszahlen abgeleitet ist. Die Verschachtelung der einzelnen Dimensionen wird durch Round-Robin realisiert.

Dimensionsbins. Ein Dimensionsbin ist eine Menge von Datenstäzen die durch die Clusterung den gleichen `__bdc__`Wert bekommen. Die Datensätze ähneln sich daher in ihren Dimensionen und werden in einem Cluster (Bin) zusammengefasst.

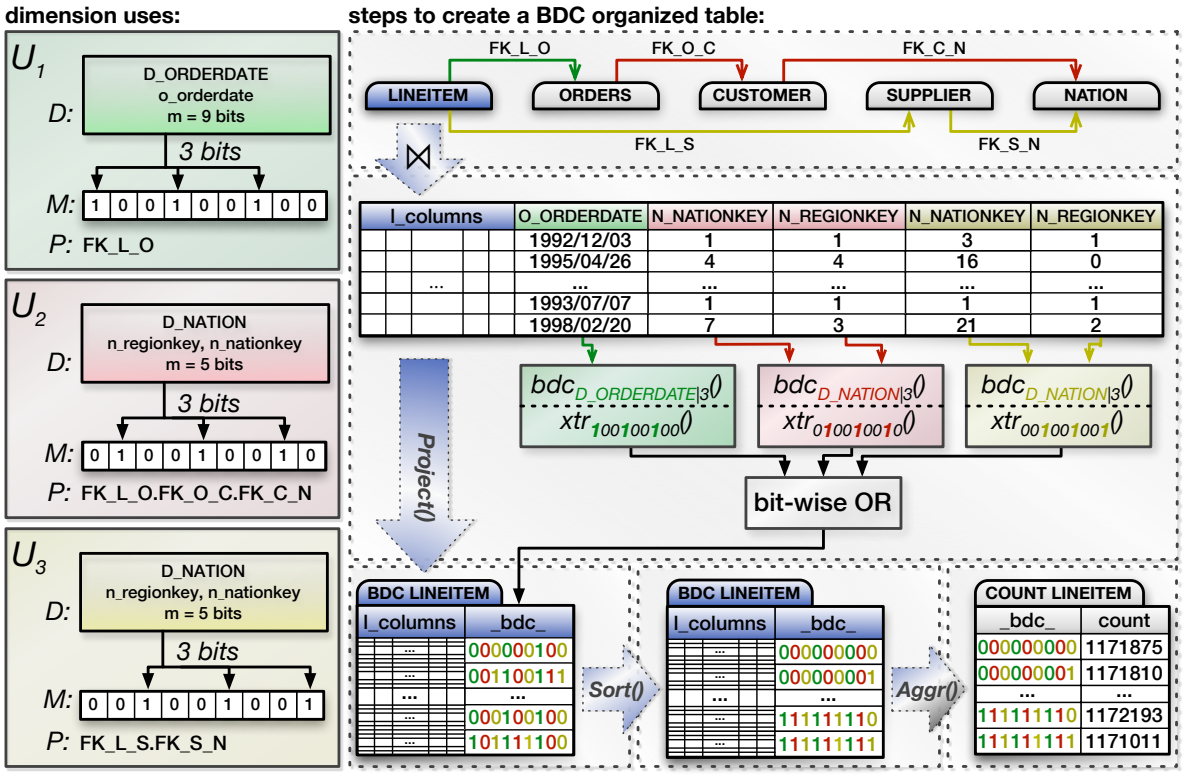


Abbildung 1.5: Erzeugen einer BDC-Tabelle, [BBS11]

Erzeugen des Schemas. In Abbildung 1.5 wird der Prozess beim Erzeugen einer BDC-Tabelle im TPC-H Schema dargestellt. Die Tabelle `lineitem` (`L_`) soll geclustert werden. Zuvor wurden 3 Dimensionen festgelegt. U_1, U_2 und U_3. Die Dimension U_1 bezieht sich auf

die Tabelle `Orders`. Sie benutzt die 9 Bits der Spalte `orderdate` zum Erzeugen der `__bdc__`-Werte. Drei der 9 Bits gehen in die Clusterung von `lineitem` über die Maske `100100100` ein. Es ist außerdem vermerkt, über welchen Pfad `FK_L_O` (Fremdschlüssel von `lineitem` zu `oders`) man die Dimensionstabelle erreicht. Gleiche Festlegungen werden für U_2 und U_3 angestellt. Die in der BDC-Tabelle zur Sortierung benutzte `__bdc__`- Spalte wird im nächsten Schritt festgelegt. Die Tabelle `lineitem` und die Dimensionstabellen U_1,U_2 und U_3 werden durch einen natürlichen Verbund zusammengeführt. Anschließend erfolgt das Extrahieren der festgelegten Dimensionsbits (mit `XOR`) und Erzeugen der `__bdc__`- Spalte durch `OR` der einzelnen Dimensionsbits. Am Ende wird die BDC-Tabelle `lineitem` nach der `__bdc__`- Spalte sortiert. Es wird außerdem noch eine `count` Tabelle angelegt, welche die Anzahl an Datensätze in einem Bin speichert.

Verarbeitung. Zum Verarbeiten der Daten benutzt BDC so genannte Sandwich-Operatoren, die auf vorhandenen und unveränderten Sortier-, Aggregations- und HashJoin-Operatoren aufbauen.

Im Folgenden werden die wichtigsten Grundoperatoren zum Anlegen eines BDC geclusterten Datenbankschemas erklärt. Alle anderen Operatoren können im BDC-Paper [BBS11] nachgelesen werden.

1.5.1 Dimensionen registrieren

In der BDC Implementierung werden zwei Funktionen zum Festlegen von Dimensionen zur Verfügung gestellt:

- ▷ `bdc_dim_create(dimname:str,srctab:str,[dimkey:clist],cbits:int)`: Dimensionsbins werden durch vorhandene Daten mit gleichverteilten Histogrammen erzeugt.
- ▷ `bdc_dim_register(dimname:str,srctab:str,[dimkey:clist],...)`: `srctab` enthält bereits alle möglichen Werte für die Dimension.

Der Parameter `dimname` ist dabei immer ein eindeutiger Name für die Dimension, `srctab` ist die Dimensionstabelle und `dimkey` stellt eine Liste von Dimensionsspalten dar. Bei bdc_dim_create(...) benötigt man zusätzlich noch den Parameter `cbits`, der angibt wie viele Dimensionswerte angelegt werden sollen (2^{cbits} viele).

1.5.2 BDC Tabelle erzeugen

Wenn alle Dimensionen angelegt sind, kann man die BDC-geclusterten Tabellen anlegen mit:

- ▷ bdc_create(bdctab:str , srctab : str , dimspec_1..dimspec_N :str, iobits)

Die Tabelle `srctab` wird dabei mit den `dimspec_1...dimspec_N` Informationen zur BDC-Tabelle `bdctab`. Die `iobits` geben an, wie viele Bits zur Clusterung benutzt werden. Der Parameter `dimspec_1...dimspec_N` ist eine Zeichenkette. Der i-te Teil `dimspec_i` setzt sich dabei nach dem Muster `dimspec_i = ":path:dimname:mask:"` zusammen. Die einzelnen Bestandteile sind:

- ▷ Dimensionsname `dimname`:
  - ◇ eindeutiger Name der Dimension
- ▷ Dimensionspfad `path`:
  - ◇ gibt an, über welche Zwischentabllen man die Dimensionenstabelle erreicht
  - ◇ `path = fk_1.fk_2....fk_M.clist_dimkey` (`fk_i`= Fremdschlüssel zur nächsten Tabelle)
- ▷ Dimensionsmasken `mask`:
  - ◇ legen fest welche Bits der Dimensionen für die Clusterung benutzt werden
  - ◇ werden durch das Round-Robin-Interleaving festgelegt
  - ◇ dürfen sich nicht überlappen, d.h. für alle Masken M gilt:

$$\bigoplus_{i \in M} int(i) = r$$

$r = 0$ oder bei $|M| = 1 \Rightarrow i \in M, i =$ "11...11"

  - ◇ sind gleich lang: $M = \{i_1, ..., i_m\}$

$$l = len(i_1) = len(i_2) = ... = len(i_m)$$

  - ◇ für alle Masken M gilt außerdem:

$$\sum_{i \in M} int(i) = 2^l - 1 = int(\underbrace{\text{"11..11"}}_{\text{l mal}})$$

1.5.3 Beispiel

Für das Anfangsbeispiel (siehe Abbildung 1.1) könnte man z.B. folgendes BDC-Schema aufstellen:

Dimensionen

Dimname	Tabelle	Dimkey	Bits	Datensätze
DIM_Regionen_rId	Regionen	rId	3	8
DIM_Produkte_pNr	Produkte	pNr	10	1024

`Regionen` und `Produkte` werden als Dimensionen festgelegt. Dabei wird die Regionsdimension mit 3 Bits angelegt, da sie 8 Datensätze umfasst. Die Produktdimension bekommt 10 Bits um 1024 Datensätze zu beschreiben.

BDC-Tabellen

BDC-Tab	Tabelle	Dimspec
_Regionen	Regionen	:'rId':'DIM_Regionen_rId':'111':
_Produkte	Produkte	:'pNr':'DIM_Produkte_pNr':'1111111111':
_Fakten	Fakten	:'pNr':'DIM_Produkte_pNr':'1010101111':
		:'kNr.rId':'DIM_Region_rId':'0101010000':
...	...	...

Exemplarisch wurden nur drei Tabellen als BDC-Tabellen angelegt. Die beiden Dimensionstabellen `Regionen` und `Produkte` sind dabei unkritisch, da sie nur ihre eigene Dimension zur Clusterung benutzen. Dagegen benutzt die Tabelle `Fakten` die zwei zuvor registrierten Dimensionen. Die Regionsdimension wird dabei über den Pfad `kNr.rId` erreicht und geht mit der Maske `0101010000` in die Clusterung der Tabelle ein. Die Maske ist nicht komplett, da die Regionsdimension nur 3 Bits verwendet. Das bedeutet, dass die Regionsdimension komplett in der Clusterung benutzt wird, dagegen werden aus der Produktdimension nur 7 der registrierten 10 Bits verwendet.

Inhalt der BDC-Tabellen

Regionen

rId	Name	__bdc__
1	Europa	000
2	Afrika	001
3	Asien	010
...	...	...
7	Amerika	110
8	Australien	111

Produkte

pNr	Name	Preis	__bdc__
1	Per Anhalter durch die Galaxis	42	0000000000
2	StarTrek 10	...	0000000001
3	SSD	...	0000000010
...	...	...	...
...	...	...	...
...	...	...	1111111111

Fakten

pNr	rId	....	....	__bdc__										Bin
1	1	...	...	0	0	0	0	0	0	0	0	0	0	
2	1	...	...	0	0	0	0	0	0	0	0	0	0	
3	1	...	...	0	0	0	0	0	0	0	0	0	0	0
...	1	...	...	0	0	0	0	0	0	0	0	0	0	
8	1	...	...	0	0	0	0	0	0	0	0	0	0	
...	...	...	...	.	.	.	.	.	.	.	.	.	.	
33	2	...	...	0	0	0	0	0	1	0	0	1	0	
...	2	...	...	0	0	0	0	0	1	0	0	1	0	34
40	2	...	...	0	0	0	0	0	1	0	0	1	0	
...	...	...	...	.	.	.	.	.	.	.	.	.	.	

Die `__bdc__`- Spalte in der Faktentabelle setzt sich aus den Dimensionen `DIM_Regionen` und

DIM_Produkte durch Round-Robin Interleaving zusammen. Die blau markierten Bits in der __bdc__- Spalte ergeben immer den __bdc__- Wert der verwendeten rId aus der Regionen BDC-Tabelle. Man erkennt, dass die letzten drei Bits der Produktnummer nicht zur Clusterung der Fakten-Tabelle benutzt werden. Die Produkte, die in den ersten sieben Bits übereinstimmen, bilden immer – mit einer Regionsid– einen Dimensionsbin.

1.6 Problemstellung

Das einfache Anfangsbeispiel ist laut 1.2.2 ein Schneeflockenschema, daher könnte man die Tabellen Produkte, Zeit und Kunden als Dimensionen festlegen. Im vorhergehenden Abschnitt 1.5.3 wurden aber nur die Tabellen Regionen und Produkte als Dimensionen festgelegt. Je nach Datenbankschema, Anzahl der Datensätze, Anfragetypen und Festplatten kann es demnach vorteilhaft sein andere Dimensionen festzulegen. In der momentanen BDC-Implementierung muss man außerdem die Dimensionen manuell registrieren und die BDC-Tabellen selbst erzeugen. Der Vorgang kann aber je nach Datenbankschema aufwändig sein, d.h. beispielsweise dass der Benutzer gewisse Tabellen nicht unbedingt als Dimensionskandidant erkennen kann, es aber durchaus vorteilhaft wäre, die Tabelle als Dimension zu benutzen. Demzufolge muss man durch einen Test herausfinden, welche Entscheidung zu treffen ist.

Die folgende Arbeit beschäftigt sich mit dem automatisierten Erstellen eines solchen multidimensionalem Schemas. Dabei sind die gewonnenen Erkenntnisse nicht unbedingt auf BDC beschränkt.

Wichtige Fragen sind zum Beispiel:

- ▷ Woran erkennt man eine Dimension bzw. Dimensionstabelle?
- ▷ Welche Dimensionen benutzt man zum Clustern?
- ▷ Welche Tabellen clustert man?
- ▷ Wie viele Dimensionen fließen in die Clusterung einer Tabelle ein?
- ▷ In welcher Reihenfolge verwendet man die Dimensionen bei der Clusterung einer Tabelle?

Alles Gescheite ist schon gedacht worden. Man muß nur versuchen, es noch einmal zu denken.

JOHANN WOLFGANG VON GOETHE

2 Verwandte Themen

2.1 Multidimensionale Clusterung in DB2

DB2 von IBM benutzt auch einen multidimensionalen Clusterungsansatz zur Performance-Steigerung bei Data-Warehouse Anwendungen (das Verfahren wird MDC genannt). Im Gegensatz zu BDC wird bei DB2 aber nicht auf der logischen Ebene geclustert, sondern auf Recordebene. Datensätze mit den gleichen Dimensionseinträgen werden dabei in den gleichen physikalischen Blöcken abgelegt. Block Indizes dienen zum schnelleren Zugriff auf die Blöcke (siehe [CFL+07, PBM+03]).

Die Grundvariante von MDC bei DB2 ist nicht automatisiert, aber DB2 bietet dem Benutzer einen Advisor (Ratgeber), der automatisch Dimensionskandidaten ermittelt und vorschlägt. In [LB04] wird die Technik des Advisors beschrieben und anschließend mit den Dimensionsfestlegungen von Experten verglichen.

Dimensionskandidaten werden dort durch die Analyse der Anfragen ermittelt, z.B. durch `GROUP BY`, `ORDER BY`, `WHERE`,... Angaben.

Interessant zu erwähnen sind die Tests zwischen den durch Experten festgelegten und automatisiert erzeugten Dimensionen. Es stellt sich dabei heraus, dass die automatisierten Dimensionen ungefähr genauso gut sind, wie die manuellen.

Es bleibt zu erwähnen, dass nicht jeder Anwender ein Experte auf dem Gebiet der multidimensionalen Clusterung ist. Daher kann nicht jeder Anwender bessere oder gleich gute Dimensionen festlegen. Eine Automatisierung ist demzufolge hilfreich.

2.2 Automatische Partitionierung - AutoPart

In [PA04] wird ein Verfahren zum automatischen Partitionieren (AutoPart) großer wissenschaftlicher Datenbanken beschrieben. Partitionierung ist ein weiterer Ansatz um große Datenmengen performant in Datenbanken zu behandeln. Das beschriebene Verfahren beschleunigt die Anfragen im Durchschnitt um den Faktor zwei. AutoPart benötigt zur Partitionierung eine Menge von Anfragen, Tabellen und Parameter die angeben, wie viel Speicher zur Spaltenreplikation benutzt werden darf und erzeugt anschließend ein neues partitioniertes Datenbankschema.

2.3 Andere Ansätze

Im Microsoft SQL Server gibt es Assistenten zum Anlegen von Indizes bzw. Dimensionen. Die Funktionalität wird in [CCG+99] beschrieben. Für die meisten Datenbanksysteme gibt es Index-Assistenten, die auch in Data-Warehouse Szenarien die Performanz steigern können. Die Vorschläge der Assistenten beruhen auf Daten- und Anfrageanalysen.

In [CN07] sind weitere Informationen zum Thema "self tuning in databases" zu finden.

2.4 Auswertung

Abschließend ist zu sagen, dass die aufgeführten Verfahren nicht komplett für BDC übernommen werden können. Problematisch an den Lösungen ist grundsätzlich, dass Anfrageanalysen stattfinden. Das in dieser Arbeit beschriebene Verfahren soll aber schemabasierend sein und keinerlei Vorwissen über die Anfragen (außer eventuell durch manuell festgelegte Dimensionen) verwenden.

Die Lösung ist immer einfach, man muss sie nur finden.

Alexander Solschenizyn

3 Lösungsansätze

3.1 Voraussetzungen

Allgemein. Für das automatische Erzeugen eines BDC-Schemas benötigt man das Datenbankschema in Form von Tabellen und Fremdschlüsselbeziehungen und zusätzlich statistische Informationen. Statistiken können entweder aus bereits vorhanden Daten in den Tabellen ermittelt werden oder in Form von Histogrammen über die Verteilung der Daten angegeben werden. Zusätzlich können noch Hinweise für Dimensionen als Parameter übergeben werden.

Das automatische Erzeugen eines BDC-Schemas wird im Folgenden immer als BDC_Auto()-Vorgang bezeichnet.

Varianten. Im weiteren Verlauf werden zwei Varianten für das Erzeugen des BDC Schemas beschrieben.

Die Variante 1 (Grundvariante 3.2) benutzt nur Primärschlüssel und Fremdschlüssel für Dimensionsentscheidungen, da keinerlei zusätzliche Informationen in das System gebracht wurden.

Im Gegensatz dazu werden bei Variante 2 (Erweiterung 3.3) zusätzliche Informationen zu möglichen Wünschdimensionen übergeben. Die Übergabe der Dimensionen erfolgt dabei durch die Angabe der Tabellen und der Spalten.

Ablauf. Es ergibt sich der typischer Ablauf beim Erzeugen des BDC-Schemas:

1. Analyse und Transformation des Schemas
2. Dimensionen finden und registrieren, eventuell Dimensionswünsche berücksichtigen
3. für jede Tabelle im Schema:
   a) Sammeln von Dimensionen für die Tabelle
   b) Erzeugen der BDC-Tabelle

Festlegungen. Ein beliebiges Datenbankschema wird als ein gerichteter Multigraph interpretieren. Ein Multigraph ist definiert durch:

$$G = (V, E)$$

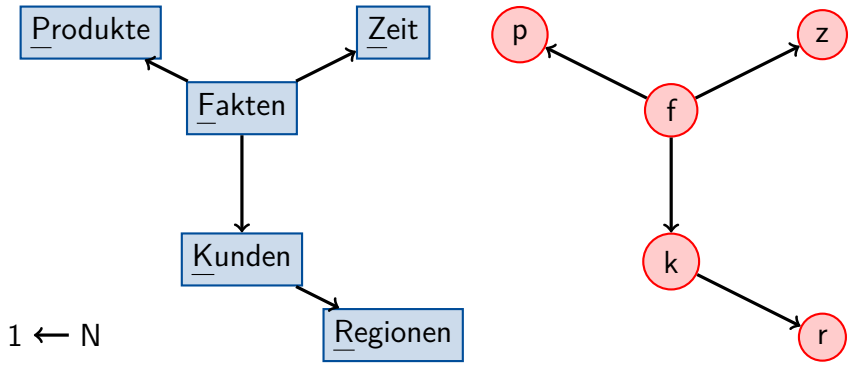
$V =$ Menge der Knoten

$E =$ Multimenge der Kanten $= \{(x, y) \mid x, y \in V\}$

Die Besonderheit einer Multimenge ist, dass die Elemente der Menge mehrfach vorkommen können. Eine Kante $(x, y) \in E$ ist ein geordnetes Tupel, wobei x der Start- und y der Endknoten ist.

Zum Interpretieren des Schemas als Multigraph (im weiteren Verlauf ist der Begriff Graph gleichbedeutend dem Begriff Multigraph) müssen die Attribute der Tabellen ignoriert werden.

Die jeweiligen Fremdschlüsselbeziehungen werden dann zu gerichteten Kanten im Graphen. Die Kante $(x, y) = x \to y$ bedeutet dabei: x ist Fremdschlüssel zum Primärschlüssel y.



(a) Schema **(b)** Graph

Abbildung 3.1: Data- Warehouse am Beispiel Online Versandhaus als Graph

Das Anfangsbeispiel (Abbildung 1.1, Seite 1) kann also zu folgendem Graphen (siehe Abbildung 3.1) transformiert werden:

$$
\begin{aligned}
&G = (V, E)\\
&V = \{Produkte, Fakten, Zeit, Kunden, Regionen\}\\
&E = \{(Fakten.pNr, Produkte.pNr),\\
&\quad (Fakten.zId, Zeit.zId),\\
&\quad (Fakten.kNr, Kunden.kNr),\\
&\quad (Kunden.rId, Regionen.rId)\}\\
&E_{simple} = \{(Fakten, Produkte), (Fakten, Zeit), (Fakten, Kunden), (Kunden, Regionen)\}
\end{aligned}
$$

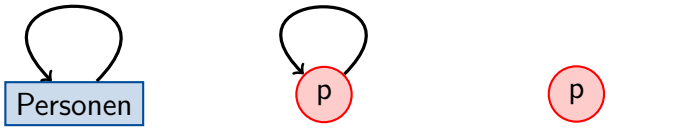
V ist die Knoten- bzw. Tabellenmenge, E ist die Multimenge der Kanten zwischen den einzelnen Knoten. Die Menge E_{simple} stellt dabei eine verkürzte Schreibweise dar, die konkreten Fremd- bzw. Primärschlüssel werden dort nicht mit angegeben.
Per Definition gilt

$$(x, y) \in E_{simple} \Leftrightarrow (x.foreignkey(y), y.primarykey) \in E$$

Allgemein gilt folglich für jedes Datenbankschema:

$$
\begin{aligned}
&G = (V, E)\\
&V = \{x \mid x \text{ ist Tabelle im Schema }\}\\
&E = \{(x, y) \mid \text{Tabelle } x \text{ hat einen Fremdschlüssel zum Primärschlüssel der Tabelle } y\}
\end{aligned}
$$

Annahmen. Man kann nun für den Multigraphen $G = (V, E)$ annehmen, dass er zusammenhängend ist, denn falls er es nicht wäre, könnte man die jeweiligen Subgraphen als getrennte Datenbankschemata betrachten und den BDC_Auto() Vorgang dort mehrmals hintereinander ausführen.



(a) Schema **(b)** Graph **(c)** Transformation

Abbildung 3.2: Einfacher Zyklus: „Person kennt Person“

Darüber hinaus wird die Annahme getroffen, dass der Graph G zyklusfrei ist. Es gibt aber Datenbankschemas, die nicht zyklusfrei sind, wie bereits ein einfaches Beispiel –dargestellt in Abbildung 3.2– demonstriert. Die Relation Personen verfügt über eine N zu 1 Beziehung zu sich selbst. Inhaltlich spiegelt sie das Verhältnis zwischen einer Gruppe von mehreren Personen, die alle einen gemeinsamen Freund haben, wider. Eine Transformation des Graphen in eine zyklusfreie Version ist nötig.

Transformation. Man kann zwei Fälle von Zyklen unterscheiden: Eigenschleifen (Fall 1) und gerichtete Kreise über mehrere Knoten (Fall 2).

Fall 1: rekursive Tabellen: (Beispiel Abbildung 3.2)
Die Eigenschleifen lassen sich einfach an den Kanten erkennen. Eine mögliche Clusterung kommt nur für die Grundtabelle in Frage, da alle anderen rekursiven Joins sich dann wiederum auf die gleiche Tabelle beziehen. Man kann also die Eigenkanten, die im Schema-Graphen vorhanden sind, weglassen:

$$G_{trans} = (V, E_{trans})$$

$$E_{trans} = E \setminus \{ \underbrace{(x, x)}_{\text{Eigenschleife}} \mid \forall x \in V \}$$

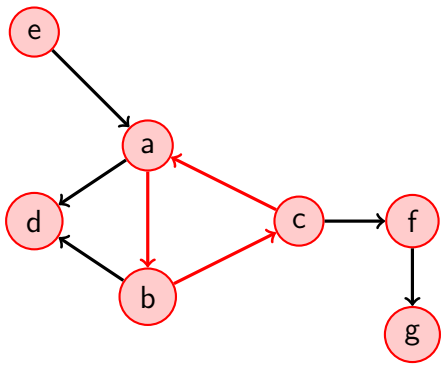
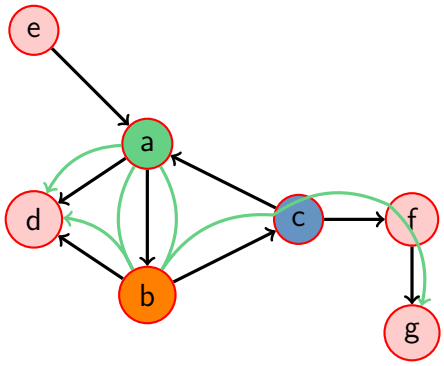


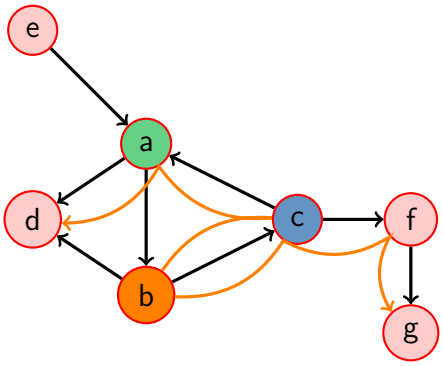
Abbildung 3.3: Zyklus: Graph

Fall 2: gerichtete Zyklen über mehrere Zwischentabellen: (Beispiel Abbildung 3.3)
Die Kreise können einfach durch eine modifizierte Tiefensuche erkannt werden. Gleichzeitig ermittelt sie die Knoten, die den Kreis bilden.

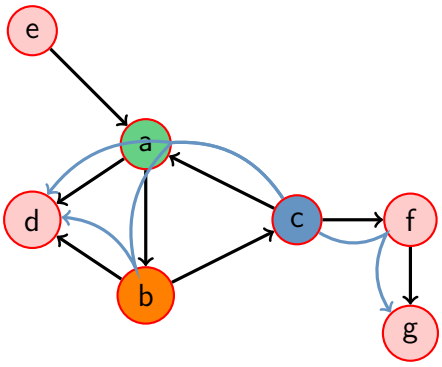
Sei $G = (V, E)$ der ursprüngliche Graph ohne Eigenkanten, P Menge der Pfade in G, $O \subset V$ die Menge der Knoten eines Kreises und $E_O \subset E$ die Kanten des Kreises.



(a) Tiefensuche Knoten a



(b) Tiefensuche Knoten b



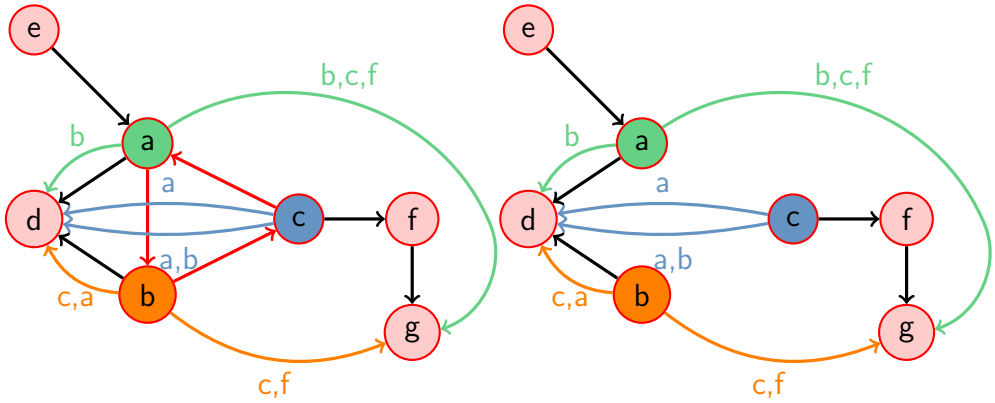
(c) Tiefensuche Knoten c

Abbildung 3.4: Transformation (1) „Zyklus über mehrere Tabellen“

Für jeden Knoten $o \in O$ wird nun eine Tiefensuche durchgeführt (vergleiche dazu Abbildung 3.4 (a)-(c)), sie liefert als Ergebnis die kürzesten Pfade $p_i = o \rightarrow ... \rightarrow b_i$ zu den erreichbaren Blättern b_i des Graphens. Der Knoten o kann somit über einen Pfad im Graphen b_i erreichen. Für alle Pfade $p_i = (o, ..., b_i)$ werden nun neue Kanten (o, b_i) (Abkürzungen) in die Menge E' eingefügt. Die Kanten des Kreises E_O werden später im transformierten Graph gelöscht.

Beispiel. In Abbildung 3.3 ist ein Beispiel dargestellt. Es ist zu erkennen, dass die Knoten a, b und c einen Kreis bilden. In Abbildung 3.4 (a) ist der Tiefensuchdurchlauf für Knoten a im Graphen dargestellt. Für den Knoten a wird zuerst der Pfad nach d gefunden, anschließend der Pfad (a, b, d) und am Ende der Pfad (a, b, c, f, g). In den Grafiken 3.4 (b)-(c) sind die Tiefensuchdurchläufe für Knoten b und c dargestellt.

Im nächsten Schritt werden neue Kanten für die Abkürzungen hinzugefügt. In Grafik 3.5 (a) sind die verkürzten Pfade ohne Start und Endknoten an den Kanten als Beschriftung markiert. Beispielsweise bedeutet die Beschriftung an der Kante von a zu g, dass der ursprüngliche Pfad



(a) verkürzte Pfade über den Kreis **(b)** Transformierter zyklusfreier Graph

Abbildung 3.5: Transformation (2) „Zyklus über mehrere Tabellen“

$(a, \underbrace{b, c, f}_{\text{Beschriftung}}, g)$ war. Die Abbildung 3.5 (b) zeigt den komplett transformierten Graphen aus Abbildung 3.3.

Allgemein. Für den transformierten Graphen G ergibt sich dann:

$$G_{trans} = (V, E_{trans})$$

$$E_{trans} = E' \cup E \setminus E_O$$

Außerdem wird eine Abbildung definiert:

$$\Delta : E_{trans} \mapsto Pot(P)$$

$Pot(P)$ bezeichnet die Potenzmenge von P

mit:

$$\Delta(a,b) = \begin{cases} \{(a,b)\} \cup \{(a,...,b),...,(a,...,b)\} & \text{,wenn } (a,b) \in E \wedge (a,b) \in E' \text{ (Fall 1)} \\ \{(a,b)\} & \text{,wenn } (a,b) \in E \wedge (a,b) \notin E' \text{ (Fall 2)} \\ \{(a,..,b),...,(a,...,b)\} & \text{,wenn } (a,b) \notin E \wedge (a,b) \in E' \text{ (Fall 3)} \end{cases}$$

$\{(a,...,b),...,(a,...,b)\} =$ ist dabei die Menge aller gefundenen Pfade von a nach b über verschiedene Zwischenknoten

Δ ist die Pfadspeicherungsfunktion. Erhält sie als Eingabe eine Kante aus dem ursprünglichen Graphen, gibt sie diese Kante zurück. Bei einer durch die Transformation entstandenen Kante, wird der durch die Tiefensuche ermittelten Pfad $(a,...,b)$ und eventuell eine originale Kante zurückgegeben. Δ ist für das spätere Rekonstruieren der ursprünglichen kreis-behafteten Wege der Kanten in E' notwendig.

Beispiel. Die Δ Funktion für das Beispiel in Grafik 3.5 (b) ist:

$\Delta(e, a) = \{(e, a)\}$ (Fall 2)

$\Delta(a, d) = \{(a, d), (a, b, d)\}$ (Fall 1)

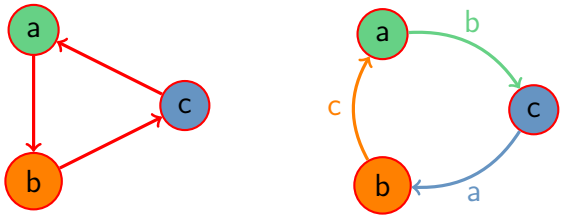
$\Delta(a, b) = \{(a, b)\}$

$\Delta(a, g) = \{(a, b, c, f, g)\}$ (Fall 3)

...

$\Delta(a, d) = \{(a, d), (a, b, d)\}$

$\Delta(c, d) = \{(c, a, d), (c, a, b, d)\}$



(a) Kreis **(b)** Transformierter Kreis

Abbildung 3.6: Transformation Kreis

Sonderfall. Falls $O = V$, d.h. der Graph besteht nur aus dem Kreis, erzeugt das beschriebene Verfahren einen neuen Kreis. In Abbildung 3.6 ist ein einfacher Graph, der nur aus einem Kreis besteht, dargestellt. Die Knoten a bis c bilden den einfachen Kreis. In der transformierten Version des Graphen erkennt man, dass erneut ein Kreis zwischen Knoten a bis c entstanden ist. Problematisch an der Transformation ist daher, dass es im ursprünglichen Graphen bereits keinerlei Blätter gibt. Abkürzungen, die den Kreis auflösen würden, können aus dem Grund nicht erzeugt werden. Dieser Sonderfall wird nicht näher betrachtet.

Zusammenfassung. Mit dem beschriebenen Verfahren kann man nun alle Zyklen nacheinander entfernen. Nach der erfolgreichen Transformation erhält man also für ein beliebiges Datenbankschema einen zusammenhängenden, gerichteten und zyklusfreien Graphen $G = (V, E, \Delta)$, mit der Erweiterung um Δ als Pfadspeicherungsfunktion. Im weiteren Verlauf wird grundsätzlich nur der transformierte Multigraph betrachtet.

3.2 Grundvariante

3.2.1 Dimensionen

Finden. Im ersten Schritt muss man mögliche Dimensionen finden. Ein potenzieller Dimensionsknoten (im Folgenden nur noch Dimensionsknoten oder Blatt) ist dabei ein Knoten der das Ende eines Pfades darstellt.

In jedem Fall existieren solche Knoten, denn der Multigraph ist nach der Transfomration zyklusfrei und zusammenhängend.

Das bedeutet, eine Dimension ist ein Knoten, der keine Kinder hat. Was in der retransformierten Version heißt, dass eine Dimensionstabelle eine Tabelle ist, bei der der Primärschlüssel in keiner Fremdschlüsselbeziehungen vorkommt. Die jeweiligen potenziellen Dimensionen werden im Prozess gesammelt und aufsteigend sortiert. Das Sortierkriterium ist die Anzahl verschiedener Werte der Dimensionsspalte in der Dimensionstabelle.

Das Finden von Knoten ohne Kinder kann ohne Tiefensuche realisiert werden:

$$D = V \setminus \{x \mid \underbrace{\exists y \in V : (x, y) \in E}_{\text{x hat Fremdschlüssel}} \} \tag{3.1}$$

D ist dabei die gesuchte Knotenmenge (Dimensionskandidaten) und wird damit charakterisiert, dass die Knoten keine wegführende Kanten haben.

Im Anfangsbeispiel ergeben sich nun folgende Dimensionskandidaten:

$$D = \{Produkte, Regionen, Zeit\}$$

Ein Dimensionskandidat wird durch d_i dargestellt.

Initiale Registrierung. Nachdem nun die Dimensionskandidaten feststehen, kann man sie registrieren. Bei BDC braucht man dazu aber auch noch die Anzahl der Bits ($cbits$), die beim Erzeugen der Dimension verwendet werden sollen. Sie werden duch eine einfache Abfrage ermittelt oder aus dem Histogramm ausgelesen und ergeben sich aus der Anzahl verschiedener Werte $|d_i|$ der Dimension.

$$cbits = \min\{\lfloor \log_2 |d_i| \rfloor, \lfloor \log_2(\max(d_i) - \min(d_i) + 1) \rfloor, \underbrace{max_{cbits}}_{\text{Obergrenze}}\} \tag{3.2}$$

In der einfachen Variante sind die Dimensionsspalten gleichzeitig auch die Primärschlüsselspalten der Tabellen. Für jede Dimensionstabelle wird nun der Primärschlüssel als Dimension mit $cbits$ registriert, sie bekommt nach folgendem Muster einen eindeutigen Namen:

$$dimname(d_i) = "DIM\_" + tablename(d_i) + columnname(d_i) \tag{3.3}$$

Für unser Anfangsbeispiel ergibt sich nun folgender Graph (siehe Abbildung 3.7, Dimensionen sind nicht farbig gefüllt).

Die Dimensionstabellen werden anschließend als BDC-Tabellen registriert, sie benutzen dabei

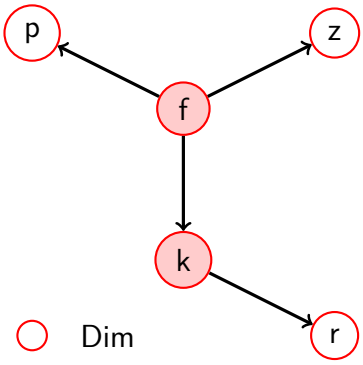


Abbildung 3.7: Data- Warehouse am Beispiel Online Versandhaus-"Initiale Dimensionen"

ihre eigene Dimension zur Clusterung. Für die Registrierung einer BDC-Tabelle benötigt man noch eine Dimensionsmaske und einen numerischen Parameter, der angibt, wie viele Bits zur Clusterung benutzt werden ($iobits$).

Dieser Parameter ist abhängig von der Blockgröße und der Tabelle. Eine obere Grenze dafür wird aus dem System ermittelt und im Folgenden mit $max_{iobits}(table)$ bezeichnet. Die Maske ergibt sich dann automatisch, da nur eine Dimension verwendet wird.

$$mask(d_i) = str(bin(2^{\text{iobits}} - 1)) = \underbrace{\text{“11..11“}}_{\text{iobits viele}}$$

3.2.2 BDC- Tabellen

Erzeugen der BDC- Tabellen. Nun muss man also nur noch $N = V \setminus D$ Tabellen betrachten. Für jede noch offene Tabelle muss nun eine BDC-Tabelle erzeugt werden, dabei muss man beachten, dass die Dimensions- bzw. Nachbartabellen bereits abgearbeitet sind. Man sucht also eine Tabelle $n \in N$, für die gilt:

$$\forall (n, x) \in E : x \notin N$$

Eine Tabelle mit der genannten Eigenschaft muss existieren, weil es Tabellen geben muss, die die Blätter des Graphen als Kinder haben.

Sei nun n eine Tabelle mit der gewünschten Eigenschaft. Die bereits abgearbeiteten Nachbartabellen von n sind in der Menge K gespeichert. So ergibt sich für die Erzeugung der BDC-Tabelle der Ablauf:

- ▷ für jede Tabelle $k \in K$
  - ◇ ermittle benutzte Dimensionen für k
  - ◇ sammele Dimensionen und ermittle Pfade durch Kante $e = (n, k)$ mit Hilfe der Δ Funktion und den im Knoten k gespeicherten Pfade zu Dimensionstabellen
- ▷ sortiere Dimensionen aus
- ▷ für jede Dimension: erzeuge Maske und ermittle benötigte Bits
- ▷ führe Round-Robin Interleaving auf alle Dimensionen aus, um die Masken zu ermitteln

- ▷ erzeuge BDC- Tabelle
- ▷ lösche n aus N und wiederhole Verfahren bis $N = \emptyset$

Aussortieren. Bei dem nun beschriebenen Verfahren kann es nun passieren, dass wichtige Dimensionen nur mit wenigen Bits in die Clusterung eingehen oder unter Umständen alle Dimensionen nur mit einem Bit. Zur Vermeidung dieser Situation sollten bestimmte Dimensionen aussortiert werden. Dazu sortiert man die Dimensionen in aufsteigender Reihe nach der Größe der benötigten Bits. Für die maximale Bitzahl wurde eine obere Grenze mit $max_{iobits}(table)$ festgelegt. Man wählt nun die ersten Dimensionen nach dem Greedy Ansatz bis man die maximale Bitzahl erreicht hat. Prinzipiell ist es besser weniger Dimensionen mit mehr Bits einzusetzen, als alle Dimensionen mit ganz wenigen.

Beispiel. In unserem Anfangsbeispiel wählen wir nun den Knoten k `Kunden` und erzeugen die BDC-Tabelle mit der benutzten Dimension `DIM_Region_rId` über den Pfad:

$$foreignkey(k).dimkey(r)$$

Anschließend können wir nun den Knoten f `Fakten` bearbeiten. Für f ergeben sich die aufgelisteten, bereits abgearbeiteten Nachbarn und Dimensionen:

- ▷ p ,z sind Dimensionsknoten, die Pfade ergeben sich durch:
  - ◇ $foreignkey(f, p).dimkey(p)$ für p und
  - ◇ $foreignkey(f, z).dimkey(z)$ für z
- ▷ k ist Nachbarknoten, wobei in dem Knoten der Pfad $foreignkey(k, r).dimkey(r)$ zu r gespeichert ist, woraus sich sich der Pfad $foreignkey(f, k).foreignkey(k, r).dimkey(r)$ für die zu benutzende Dimension r ergibt

Die Funktionen $foreignkey, dimkey$ sind definiert durch:

$$foreignkey(a, b) = \text{ Name des Fremdschlüssels von Tabelle } a \text{ zu Tabelle } b$$

$$dimkey(x) = primarykey(x) = \text{ Name des Primärschlüssels der Tabelle } x \tag{3.4}$$

3.3 Erweiterungen

Bei der Erweiterung werden zusätzlich gewünschte Dimensionen als Parameter übergeben, welcher folgenden Aufbau hat:

- ▷ "table$_1$:column$_1$,...,table$_t$:column$_t$"

Er wird umgeformt zur Map `Dims`, wobei:

$$Dims(table) = \begin{cases} column_i & \text{,falls } table = table_i \text{ im Parameter vorhanden} \\ \text{“”} & \text{,sonst} \end{cases}$$

Die Varianten 1 und 2 unterscheiden sich in der Festlegung der Dimensionen. Bei Variante 1 werden nur die Dimensionen benutzt, die als Parameter übergeben worden, hingegen werden bei Variante 2 die übergebenen Dimensionen erweiternd zur Grundvariante benutzt.

Es ist möglich, dass eine Wunschdimension kein Blatt des Graphen ist, daher muss überprüft werden, ob die Wunschdimension ein Blatt ist. Falls sie kein Blatt ist, darf die Dimension nicht nach der Erzeugung ihrer BDC-Tabelle aus der Menge der zu bearbeitenden Knoten N genommen werden.

3.3.1 Variante 1

Das Grundverfahren wird an den aufgeführten Stellen erweitert:

- ▷ Formel 3.1 von Seite 19 wird zu:

$$D_{ext} = \{x \mid \forall x : Dims(x) \neq \text{“”}\}$$

- ▷ Formel 3.3 von Seite 19 wird zu: $\forall d_i \in D_{new}$

$$dimname(d_i) = \text{“}DIM\_\text{“} + tablename(d_i) + Dims(d_i)$$

- ▷ Formel 3.4 von Seite 21 wird erweitert durch:

$$dimkey(x) = Dims(x)$$

3.3.2 Variante 2

Das Grundverfahren wird erweitert:

- ▷ $D_{new} = \{x \mid \forall x : Dims(x) \neq \text{“”}\}$
- ▷ Formel 3.1 von Seite 19 wird zu:

$$D_{ext} = D \cup D_{new}$$

- Formel 3.3 von Seite 19 wird zu: $\forall d_i \in D_{new}$

$$dimname(d_i) = \text{“}DIM\_\text{“} + tablename(d_i) + Dims(d_i)$$

- Formel 3.4 von Seite 21 wird erweitert durch:

$$dimkey(x) = \begin{cases} Dims(x) & \text{,falls } Dims(x) \neq \text{“”} \\ primarykey(x) & \text{,sonst} \end{cases}$$

Die Variante 2 wird im folgenden Kapitel noch näher mit Algorithmen und konkreten Umsetzungen beschrieben.

Good code is short, simple, and symmetrical - the challenge is figuring out how to get there.

Sean Parent

4 Implementierung

4.1 Datenstrukturen

4.1.1 Map/ List

In den Algorithmen werden Map und List Datenstrukturen benutzt. Eine Map ist dabei eine Hashmap mit einer beliebigen Sondierungsstrategie. Die Keys der Maps werden grundsätzlich als `str` (Zeichenkette) festgelegt. Die Map wird erzeugt durch den Aufruf `new Map(Typ)`, wobei `Typ` angibt, welchen Typ die Elemente in der Map haben. Analog wird auch eine Liste erzeugt. Zugriffe auf Maps oder Listen geschehen über den [ ]-Operator . Die verwendete Map Datenstruktur besitzt die Besonderheit, dass man alle gespeicherten Elemente durchlaufen kann.

4.1.2 MGraph

In Abschnitt 3.1 auf Seite 13 wird das Konzept eines gerichteten Multigraphen beschrieben. Zur Vereinfachung wird der MultiGraph $G = (V, E)$ noch durch die Δ-, die $primarykey$- und die $foreignkey$-Funktionen erweitert. Für die Algorithmen wird nun festgelegt, wie die Komponenten des Multigraphen realisiert werden:

$G.V$ die Menge der Knoten durch `Map(str)`, der `key` ist dabei der Knotenname,

$G.E$ die Menge der Kanten durch `Map(List(str))`, der `key` setzt sich aus Startknoten a und Endknoten b zusammen, ein Zugriff $G.E[a, b]$ wird intern als $G.E[a + b]$ behandelt,

$G.\Delta$ die Pfadspeicherungsfunktion durch `Map(List(str))`, der `key` ist analog zu $G.E$

$G.primarykey(x)$ gibt den Primärschlüssel der Tabelle x als Zeichenkette zurück (intern als `Map` realisierbar)

$G.foreignkey(a, b)$ gibt den Fremdschlüssel der Tabelle a zur Tabelle b als Zeichenkette zurück

4.2 Überblick: BDC_Auto()

Algorithmus 1: bdc_auto(sc, H, Dims)

```
   Input  : SystemCatalog sc, Histogram H, Map Dims: Dimensionswünsche
1  max_cbits = 16 // Konstante
2  D = new Map(str) // Dimensionskandidaten
3  Path = new Map(List(str)) // speichert Pfade
4  G = bdc_auto_scheme_analysis_transform(sc)
   // G zyklusfrei und zusammenhängend
5  G_copy = G.copy()
6  D = bdc_auto_find_dimensions(G, Dims)
7  Dimnames = new Map(str)
8  bdc_auto_register_dims(G, D, H, Path, Dimnames, max_cbits)
   // registriere BDC Tabellen
9  L = getAllLeaves(G)
10 while L not empty do
11     foreach l in L do
12         bdc_auto_create_table(l, G, G_copy, Dimnames, H, Path)
13         G.removeAllEdgesToNode(l)
14         G.removeNode(l)
15     L = getAllLeaves(G)
```

Der BDC_Auto Algorithmus 1 besteht aus drei Phasen. In der ersten Phase wird das Datenbankschema analysiert und transformiert. Anschließend werden die initialen Dimensionen ermittelt und registriert. Am Ende werden alle Tabellen schrittweise als BDC Tabellen erzeugt. Alle Funktionen die `"bdc_auto"` als Präfix haben, werden im weiteren Kapitel beschrieben. Der beschriebene BDC_Auto() Algorithmus benutzt die im Kapitel 3.2 beschriebene Grundvariante und wurde an einigen Stellen, um die in 3.3.2 beschriebenen Erweiterungen (Variante 2), ergänzt.

getAllLeaves. Mit Formel 3.1 Seite 19 kann man die Blätter eines Graphen ermitteln. Algorithmus 2 realisiert das formal beschriebene Verfahren.

Algorithmus 2: getAllLeaves(G)

```
   Input  : Multigraph G
   Result : Blätter L des Graphen G
1  L = G.V.copy()
2  foreach (x,y) in E do
3      L.remove(x)
4  return L
```

4.2.1 Schema Analyse und Transformation

Algorithmus 3: bdc_auto_scheme_analysis_transform(sc)

```
   Input  : Syscat sc: Systemkatalog oder Datenstruktur mit Schemainformationen
   Result : G Schemagraph
 1 G = new MGraph()
   // Konstruktion von G aus dem Systemkatalog
 2 foreach t in sc.getTables() do
 3     G.V.add(t)
 4     G.primarykey(t) = sc.getPrimaryKey(t)
 5     foreach fk in sc.getForeignKeys(t) do
 6         x = sc.getReferencingTable(fk) // fk zeigt auf Tabelle x
 7         G.E[x,t].append(x, t)
 8         G.Δ[x,t].append(x, t)
 9         G.foreignkey(x,t) = fk
   // Transformation(1): Eigenkanten
10 foreach t in sc.getTables() do
11     if G.E[t,t] exists then
12         G.E.removeAll(t, t)
   // Transformation(2): Zyklen
13 bdc_auto_transform_circle(G)
14 return G
```

Algorithmus 4: bdc_auto_transform_circle(G)

```
   Input  : G Schemagraph
 1 (O,E_O) = dfs_find_circle(G) // Tiefensuche um Kreise zu finden
   // O beinhaltet die Knoten des Kreises, E_O die Kanten
 2 while O not empty do
       // transformiere Kreis
 3     if O = G.V then
 4         error("Graph ist ein kompletter Kreis, bisher nicht behandelt")
 5     foreach o in O do // durchlaufe alle Kreisknoten
 6         paths = dfs_all_paths_to_leaves(o)
 7         foreach p in paths do
 8             G.E[o,p.end].append(p)
 9             G.Δ[o,p.end].append(p)
               // p.end gibt den Endknoten des Pfades zurück
10     G.E.removeAll(E_O)
11     (O,E_O) = dfs_find_circle(G) // suche weiteren Kreis
```

In Algorithmus 3 werden aus dem Systemkatalog oder einer Datenstruktur mit Schemainformationen alle Tabellen ermittelt und als Knoten festgelegt. Anschließend werden für jede

Tabelle alle Fremdschlüssel ermittelt und neue Kanten in den Graphen eingefügt.

Die Transformation teilt sich in zwei Schritte. Im Ersten werden alle vorhandenen Eigenschleifen entfernt. Der Letzte transformiert eventuelle Kreise im Schema nach dem Verfahren, welches in 3.1 beschrieben und im Algorithmus 4 ausformuliert wurde.

Algorithmus 4 erkennt außerdem ob der Graph nur aus einem Kreis besteht und wirft eine Fehlermeldung, falls dieser Fall eintritt. Es soll auf diesen Fall in der hier beschriebenen Implementierung nicht eingegangen werden.

4.2.2 Dimensionserkennung

Algorithmus 5: bdc_auto_find_dimensions(G, Dims)

```
   Input  : Multigraph G , Dims: Wunschdimensionen
   Result : D Dimensionen
1  D = getAllLeaves(G) // .leaf für jedes Element auf true
2  foreach dk in Dims do
3      if dk not in D then
4          D.add(dk)
5          D[dk].leaf = false
6  return D
```

In der Grundvariante 3.2 sind Dimensionen die Blätter eines Graphen, falls der Benutzer nun aber Dimensionswünsche übergibt, muss überprüft werden, ob der Dimensionsvorschlag ein Blatt ist. In dem Fall, dass der Dimensionswunsch kein Blatt ist wird das Attribut `leaf` auf `False` gesetzt. Die Dimensionstabelle wird dann nach der initialen Registrierung der Dimensionen nicht aus dem Graphen gelöscht, da sonst die Pfade zu den Kindern der Wunschdimension verloren gehen würde, vergleiche dazu Abschnitt 3.3.2.

4.2.3 Dimensionsregistrierung

Algorithmus 6: bdc_auto_register_dims(G, D, H, Path, Dimnames, max_cbits)

```
   Input  : MGraph G, Map D, Histogram H, Map Path, Map Dimnames, int max_cbits
1  foreach d in D do
2      histbits = floor(log(2, H[d]) )
3      minmaxbits = floor(log(2, max(d) - min(d) +1 )
       // Parameter für das Erzeugen der Dimensionen
4      cbits = min(histbits, minmaxbits, max_cbits)
5      dimname = "DIM_"+ d + primarykey(d)
6      bdc_dim_create(dimname, d, [dimkey(d)], cbits)
       // dimkey(d) ist entweder primarykey oder durch Dims Parameter in
          bdc_auto festgelegt
7      Path[d] = "[dimkey(d)]"
8      Dimnames[d] = dimname
       // falls d ein Blatt ist: lösche Knoten d und alle Kanten zu d
9      if d.leaf then
10         G.removeAllEdgesToNode(d)
11         G.removeNode(d)
```

Für die Berechnung der `cbits` zum Erzeugen der Dimension wird die Formel 3.2.1 Seite 19 benutzt. Die Map `Path` speichert die Pfade zur Dimension, das bedeutet im initialen Fall wird nur die Dimensionsspalte gespeichert. In `Dimnames` werden zur weiteren Verarbeitung die Dimensionsnamen gespeichert.

4.2.4 BDC Tabellen erzeugen

Algorithmus 7: bdc_auto_create_table(l, G, G_{copy} , Dimnames, H, Path)

```
   Input : Blatt l, Graph G, Graph G_copy, Dimnames, Histogram H, Path
1  foreach n in G_copy.getChildrenOf(l) do // Nachbarn mit l → n
2      used_dims// Liste der Dimensionsinformationen
3      foreach pToN in G.Δ[l,n] do // jeder Pfad zum Nachbarn
4          pathToNeighbor = str(pToN)
           // Path[n] speichert alle Pfade von n zu Dimtabs
5          foreach pToDim in Path[n] do
6              dimpath = pathToNeighbor + pToDim
7              Path[l].append(dimpath)
8              dimname = Ermittle Dimensionsname // aus pToDim und Dimnames
9              used_dims.add(dimpath,dimname)
10     mask= round_robin(used_dims, iobits) // erzeugt Masken aus globaler
       cbits Information für jede Dimension
11     dimspec += dimpath+ mask // alle dimspec's werden gesammelt
12 iobits = max_iobits(l) // aus dem System für die Tabelle ermittelt
13 bdc_create("BDC"+l,l, dimspec, iobits)
```

Um die übergebene Tabelle l als BDC-Tabelle zu registrieren müssen alle Nachbarn im originalen Graphen zuerst abgearbeitet sein. Die Nachbarknoten sind abgearbeitet, da im BDC_Auto Algorithmus 1 auf Seite 25 der Graph schrittweise von den Blättern aus bearbeitet wird, anschließend werden die bearbeiteten Knoten und hinführenden Kanten zu den Blättern gelöscht. Aus diesem Grund ist der Einsatz des original Graphen erforderlich, da in G die Kinderknoten und Kanten bereits gelöscht sind.

Im ersten Schritt des Algorithmus werden alle möglichen Wege von dem Blatt l zu allen erreichbaren Dimensionen erzeugt. Anschließend wird die BDC-Tabelle mit den nötigen Informationen und einem Präfix erzeugt. Der Parameter `iobits` wird dabei aus dem System für die Tabelle ausgelesen.

Jedes Denken wird dadurch gefördert, daß es in einem bestimmten Augenblick sich nicht mehr mit Erdachtem abgeben darf, sondern durch die Wirklichkeit hindurch muß.

ALBERT EINSTEIN

5 Auswertung

5.1 TPC-H BDC-Schema

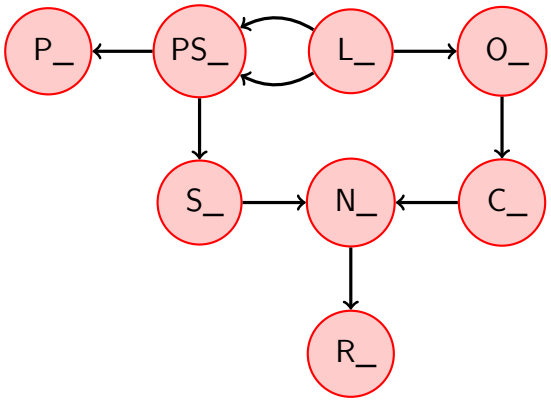


Abbildung 5.1: TPC-H Schemagraph

In Abbildung 1.4 auf Seite 4 wird das TPC-H Testdatenbankschema dargestellt. Die vereinfachte Graphendarstellung des Schemas zeigt Abbildung 5.1. Man beachte, dass die Pfeile im TPC-H Schema vom Primärschlüssel zum Fremdschlüssel zeigen, dagegen aber im Schemagraph vom Fremdschlüssel zum Primärschlüssel. Der BDC_Auto() Vorgang, ohne Wunschdimensionen, liefert auf dem TPC-H Graphen die Dimensionstabellen `P_` (`part`) und `R_` (`region`). Für den Skalierungsfaktor(SF) 1 ergibt sich dann für die Clusterung der Tabellen die folgenden Festlegungen:

Tabelle	**Datensätze bei SF**$= 1$
part	$200000 \cdot SF = 2 \cdot 10^5$
region	5

Dimensionen

Dimname	**Tabelle**	**Dimkey**	**cbits**
part_dim	part	_p_partkey	$min\{\underbrace{\lfloor \log_2(2 \cdot 10^5) \rfloor}_{=17}, \underbrace{max_{cbits}}_{16}\} = 16$
region_dim	region	_r_regionkey	$min\{\underbrace{\lfloor \log_2(5) \rfloor}_{=2}, \underbrace{max_{cbits}}_{16}\} = 2$

Die Dimensionsnamen wurden vereinfacht.

BDC-Tabellen

BDC-Tab	Tabelle	Dimspec
'_p'	'part'	:['_p_partkey']:'part_dim':'1111111111':
'_r'	'region'	:['_r_regionkey']:'region_dim':'11':
'_n'	'nation'	:'_region'['_n_regionkey']['_r_regionkey']. ['_r_regionkey']:'region_dim':'11':
'_s'	'supplier'	:'_nation'['_s_nationkey']['_n_nationkey']. '_region'['_n_regionkey']['_r_regionkey']. ['_r_regionkey']:'region_dim':'11':
'_ps'	'partsupp'	:'_part'['_ps_partkey']['_p_partkey']. ['_p_partkey']:'part_dim':'101011111111111111': :'_supplier'['_ps_suppkey']['_s_suppkey']. '_nation'['_s_nationkey']['_n_nationkey']. '_region'['_n_regionkey']['_r_regionkey']. ['_r_regionkey']:'region_dim':'010100000000000000':
'_c'	'customer'	:'_nation'['_c_nationkey']['_n_nationkey']. '_region'['_n_regionkey']['_r_regionkey']. ['_r_regionkey']:'region_dim':'11':
'_o'	'orders'	:'_customer'['_o_custkey']['_c_custkey']. '_nation'['_c_nationkey']['_n_nationkey']. '_region'['_n_regionkey']['_r_regionkey']. ['_r_regionkey']:'region_dim':'11':

Die Pfadangabe in `dimspec` Spalte unterscheidet sich von der bisher verwendeten Darstellung. In der Zeile für `supplier` bedeutet die dimspec Angabe:

- `":'_nation'['_s_nationkey']['_n_nationkey']."`
  um zur Tabelle `_nation` zu kommen, muss man von `_supplier` ausgehend, über den Fremdschlüssel `"_s_nationkey"` gehen, dabei bezieht sich der Fremdschlüssel auf den Primärschlüssel `"_n_nationkey"` der Tabelle `_nation`
- in `_nation` angekommen führt der Weg analog weiter zu `_region`
- die Tabelle `_region` ist eine Dimensionstabelle, daher folgt anschließend `"['_r_regionkey']:'region_dim':'11'"`
- dabei ist `"_r_regionkey"` die Dimensionsspalte, `"region_dim"` der Dimensionsname und `"11"` die Maske, bestehend aus zwei Bits.

5.2 TPC-H Ergebnisse

5.2.1 Testumgebung

Hardware. Als Testsystem kam ein Notebook (Samsung R560) mit:

- ▷ `CPU`: Intel(R) Core(TM)2 Duo CPU P7350, beide Kerne @ 2.00GHz
- ▷ `HDD`: 500GB Sata2
- ▷ `RAM`: 4GB DDR3-1066

zum Einsatz.

Messungen. Es wurden 5 Messungen mit 3 unterschiedlichen Einstellungen:

- ▷ `auto`: automatisch erzeugtes Schema mit BDC_Auto(), ohne Wunschdimensionen
- ▷ `manuell`: im Paper [BBS11] beschriebenes Schema
- ▷ `ohne`: Schema ohne BDC-Clusterung als Referenz

durchgeführt.

Nach jedem Messvorgang, d.h. Ausführen aller Anfragen von $Q1$ bis $Q22$, wurde der Server neu gestartet, um einen Kaltstart des Systems zu simulieren.

5.2.2 Ergebnisse

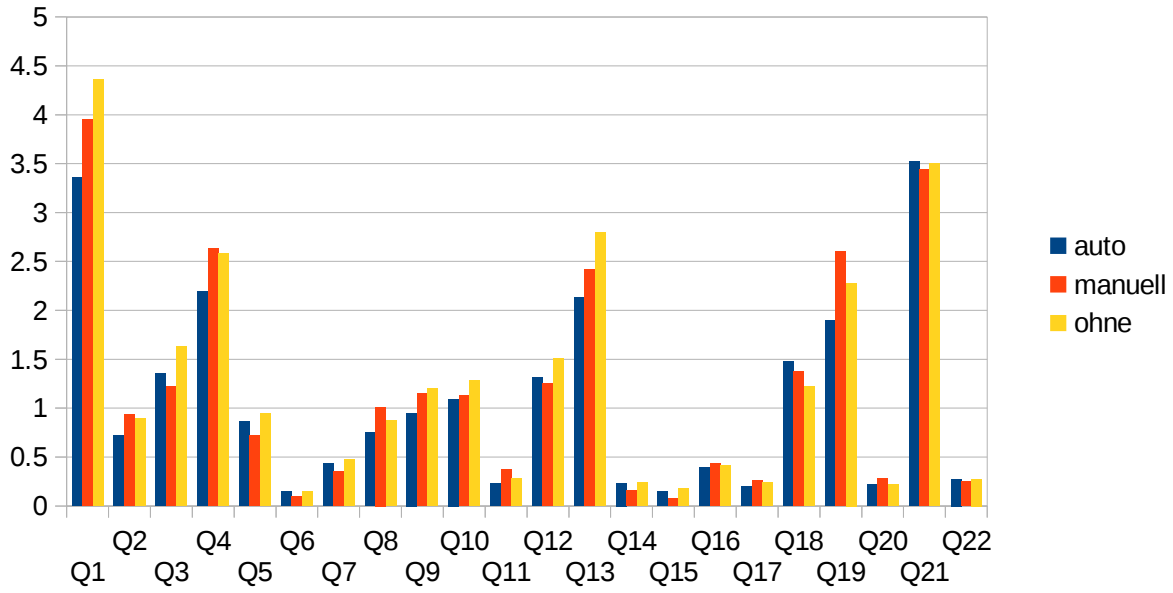


Abbildung 5.2: TPC-H: durchschnittliche Laufzeit der einzelnen Anfragen in ms

Hinweise. Die durchgeführten Messungen wurden bei einem Skalierungsfaktor $SF = 1$ ohne Optimierung der Anfragen auf das jeweilige Schema durchgeführt. Der TPC-H Test führt 22 Anfragen $Q1$ bis $Q22$ durch (siehe Abbildung 5.2), dabei werden unterschiedliche Szenarien getestet.

Auswertung. Anfrage $Q5$ listet die Umsätze der lokalen Anbieter auf. An dem Vorgang sind die Tabellen `lineitem`, `oders`, `customer`, `supplier`, `nation` und `region` nach [TPC] beteiligt. Man erkennt, dass sich `"auto"` zwischen `"manuell"` und `"ohne"` platziert.

Die Auswertung zeigt, dass `"auto"` bei 11 Anfragen schneller ist als die beiden anderen Varianten. In 9 Fällen ist `"manuell"` und in 2 Fällen ist die `"ohne"` Variante schneller. Die `"manuell"` festgelegte Variante platziert sich in vielen Fällen hinter `"auto"`, da sie für höhere Skalierungsfaktoren angelegt wurde. Auswertend ist zu sagen, dass die durchgeführten Tests zeigen, dass das automatisch erzeugte BDC-Schema ungefähr genauso gut ist, wie das Manuelle.

Verbesserungen. An einigen Stellen könnten die Test noch bessere Ergebnisse liefern. Zum Beispiel ist der Skalierungsfaktor problematisch, $SF = 1$ ist eher zum schnellen Testen geeignet. Umfangreichere Vergleichstest mit höheren SF Werten müssen durchgeführt werden. Selbst die im Internet gelisteten TPC-H Ergebnisse beginnen bei einem Skalierungsfaktor von $SF = 100$.

Auf der anderen Seite liefert die BDC-Technik auf Solid-State-Disks sehr gute Ergebnisse, SSDs standen im Umfang dieser Arbeit aber nicht zur Verfügung.

Die Anfragen sind momentan noch nicht optimiert, da die automatische Anfrage-Optimierung für ein bestehende BDC-Datenbankschema in der momentanen BDC-Implementierung noch nicht vollständig vorhanden ist.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Das multidimensionale Clustern von Datenbanken ist ein Ansatz um Data-Warehouse Szenarien effizient umzusetzen. In der Realität finden sich eine Vielzahl von unterschiedlichen Datenbank-Schemas für Data-Warehous Situationen, dabei sind die wenigsten davon klassische Stern- oder Schneeflockenschema.

Bei den üblichen Techniken sind die Parameter für die Clusterung meist von unterschiedlichen Faktoren abhängig. Die Paramaterfestlegungen erschweren das manuelle Erzeugen einer möglichst guten Clusterung.

Ein automatisiertes Verfahren erleichtert die Arbeit für die Festlegung der Parameter.

Es gibt bereits Verfahren für einige Datenbanksysteme. Die untersuchten Verfahren benutzen zur Automatisierung grundsätzlich Anfrageinformationen. Das Verfahren in dieser Arbeit benötigt keine Anfraginformation, sondern nur Schemainformationen (Tabellennamen, Beziehungen und Datenstatistiken) und ist speziell auf Bitwise Dimensional Clustering abgestimmt.

Es wurde gezeigt, dass es möglich ist ein BDC-geclustertes Datenbank-Schema automatisch zu erzeugen. Dabei wurden auch Sonderfälle wie zyklusbehaftete Datenbank-Schemas betrachtet.

Beim dem Prozess des Erzeugens werden Dimensionskandidaten anhand von Analysen, Graphentransformationen und Graphenalgortihmen gefunden und anschließend für die Clusterung benutzt. Die Clusterung aller Tabellen des Datenbankschemas verläuft komplett automatisch.

Es wurde außerdem eine Erweiterung des Verfahrens um die manuelle Angabe von Dimensionenskandidaten beschrieben. Dem Algorithmus werden dabei nicht nur die benötigten Schemainformationen übergeben, sondern auch noch Dimensionswünsche.

Abschließend wurden Messungen mit dem automatisch erzeugte Datenbankschema für den TPC-H Test durchgeführt. Dabei wurden Vergleiche zwischen dem im BDC-Paper beschriebenen Schema gezogen. Festzustellen ist, dass in einigen Situationen das erzeugte Schema bessere Ergebnisse zeigte als das manuelle Schema.

6.2 Ausblick

An einigen Stellen kann das beschriebene Verfahren noch verbessert werden, z.B. werden die Dimensionen momentan nach ihrer Größe in die Clusterung einer Tabelle eingebracht, eine genauere Analyse des Nutzens einer Dimension wäre angebracht.

Außerdem müssen ausführliche Benchmaks mit größeren Datenmengen durchgeführt werden um die Qualität des beschrieben Verfahrens zu bewerten.

Man könnte das beschriebene Verfahren dahingehend erweitern, dass die Wunschdimensionen automatisch durch Anfrageanalysen erzeugt werden.

Literaturverzeichnis

[Act11] Actian: Vectorwise Technical Whitepaper, November 2011. http://www.actian.com/products/vectorwise/overview#Resources. Version: November 2011

[BBS11] Baumann, Stephan; Boncz, Peter; Sattler, Kai-Uwe: Bitwise Dimensional Clustering in Column Stores. (2011). – to appear

[CCG+99] Chaudhuri, S.; Christensen, E.; Graefe, G.; Narasayya, V.R.; Zwilling, M.J.: Self-tuning technology in microsoft sql server. In: Data Engineering Bulletin 22 (1999), Nr. 2, S. 20–26

[CFL+07] Chen, W.-J.; Fisher, A.; Lalla, A.; McLauchlan, A.D.; Agnew, D.: Database Partitioning, Table Partitioning, and MDC for DB2 9. IBM Redbooks, 2007

[CN07] Chaudhuri, S.; Narasayya, V.: Self-tuning database systems: a decade of progress. In: Proceedings of the 33rd international conference on Very large data bases VLDB Endowment, 2007, S. 3–14

[LB04] Lightstone, S.S.; Bhattacharjee, B.: Automated Design of Multidimensional Clustering Tables for Relational Databases. (2004)

[PA04] Papadomanolakis, S.; Ailamaki, A.: Autopart: Automating schema design for large scientific databases using data partitioning. In: Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on IEEE, 2004, S. 383–392

[PBM+03] Padmanabhan, Sriram; Bhattacharjee, Bishwaranjan; Malkemus, Tim; Cranston, Leslie; Huras, Matthew: Multi-dimensional Clustering: A New Data Layout Scheme in DB2. In: SIGMOD, 2003

[TPC] TPC BENCHMARK H Specification http://www.tpc.org/tpch/spec/tpch2.14.3.pdf. – Stand: 30.01.2012

[ZB] Zukowski, M.; BV, A.N.: Integration of VectorWise with Ingres. In: SIGMOD Record

[ZBNH05] Zukowski, M.; Boncz, P. A.; Nes, N. J.; Héman, S.: MonetDB/X100 - A DBMS In The CPU Cache. In: IEEE Data Engineering Bulletin 28 (2005), June, Nr. 2, 17 - 22. http://oai.cwi.nl/oai/asset/11098/11098B.pdf

Eidesstattliche Erklärung

Ich, Steve Göring, Matrikel-Nr 43952, versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema:

"*Automatisierter Entwurf eines Schemas für Bitwise Dimensional Clustering*

(Automated Design of Bitwise Dimensional Clustering) "

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ilmenau, den

Steve Göring