Franz Färber, Bernhard Jäcksch, Christian Lemke, Philipp Große, Wolfgang Lehner

Hybride Datenbankarchitekturen am Beispiel der neuen SAP In-Memory-Technologie

Erstveröffentlichung in / First published in:

Datenbank-Spektrum. 2010, 10 (10), S. 81-92 {Zugriff am: 20.01.2023}. Springer. ISSN 1610-1995.

DOI: http://dx.doi.org/10.1007/s13222-010-0020-8

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

Hybride Datenbankarchitekturen am Beispiel der neuen SAP In-Memory-Technologie

Franz Färber · Bernhard Jäcksch · Christian Lemke · Philipp Große · Wolfgang Lehner

Zusammenfassung Die Verfügbarkeit neuer Technologien wie Multi-Core, SSD oder große Hauptspeicherkapazitäten bieten eine Gelegenheit, die klassischen Architekturansätze von Datenbanksystemen zu überdenken und an bestimmten Stellen zu korrigieren. In diesem Beitrag stellen wir die Grobstruktur der neuen hauptspeicherzentrierten SAP Technologie als einen Ansatz einer kommerziellen Umsetzung moderner Architekturkonzepte vor. Zentrales Design-Kriterium ist dabei ein hybrider Ansatz, um eine möglichst hohe Anzahl von Anforderungsvarianten optimal zu unterstützen.

Nach einer Einleitung führt der Artikel durch die wichtigsten Architekturkomponenten und illustriert den grundsätzlichen Aufbau des Systems. Für einen „deep dive“ werden zwei Bereiche in Teil 3 und 4 des Artikels im Detail diskutiert. Dabei greift der Artikel zum einen den Aspekt der physischen Optimierung im Kontext eines hauptspeicherzentrierten Systems auf und diskutiert unterschiedliche Komprimierungs- und Sortierungskriterien, wie sie im klassischen disk-zentrierten Ansatz nicht zu finden sind. Zum anderen wird die Unterstützung von Planungsanwendungen skizziert, wodurch ein Einblick in die spezifische Unterstützung einer Anwendungsdomäne („business planning“) und die prinzipiellen Erweiterungen für komplexe Operationen zur direkten Unterstützung von darauf aufbauender Planungsfunktionalität gezeigt werden.

Schlüsselwörter Hybride Datenbankarchitektur · Column Store · Planning · Kompression

F. Färber · B. Jäcksch (✉) · C. Lemke · P. Große
SAP AG, Walldorf, Deutschland
e-mail: b.jaecksch@sap.com

F. Färber
e-mail: franz.faerber@sap.com

C. Lemke
e-mail: c.lemke@sap.com

P. Große
e-mail: philipp.grosse@sap.com

W. Lehner
TU Dresden, Dresden, Deutschland
e-mail: wolfgang.lehner@tu-dresden.de

1 Einleitung und Rahmenbedingungen

Kommerzielle Datenbank-Management-Systeme sind im Wesentlichen von einer Architektur geprägt, die sich an blockbasierten Zugriffen mit Blick auf klassische Magnetplatten orientiert. Entwicklungen im Bereich der Hardware auf der einen Seite und auch im Umfeld von Basis-Software auf der anderen Seite bieten eine „gute Gelegenheit“ Veränderungen im Aufbau von Datenbanksystemen zu untersuchen und zu einer Data-Management-Plattform weiter zu entwickeln [17]. Aggressiver könnte die Feststellung auch formuliert werden: Systeme, die erfolgreich in großen Szenarien eingesetzt werden sollen und auch mittelfristig Grundlage von Anwendungssoftware sein wollen, müssen sich an neuen bzw. veränderten Randbedingungen orientieren. In diesem Artikel geben wir einen Überblick über die Architektur der neuen SAP In-Memory-Technologie (SAP IM-Technologie), die den Anspruch besitzt nicht nur moderne Hardware-Gegebenheiten [8] optimal auszunutzen, sondern eine effiziente und dynamisch skalierbare Grundlage für komplexe Geschäftsanwendungen zu sein.

Die Entwicklung der SAP IM-Technologie ist aus diesem Grund nicht ausschließlich technologisch motiviert. So fordern gleichwohl moderne Geschäftsanwendungen den Zugriff auf eine große Datenbasis in einer Art und Weise, wie es bisher nur Spezialsystemen im Rahmen von

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

Data-Warehouse-Infrastrukturen vorbehalten war. Analytische Auswertungen kristallisieren sich mehr und mehr als der Treiber des operationalen Geschäfts heraus bzw. werden zum integralen Bestandteil eines operativen Geschäftsprozesses. Die darunterliegende Datenbankinfrastruktur sieht sich somit vor die Herausforderung gestellt, mit einem Mix aus klassisch transaktionalen und analytisch-orientierten Anfragen umgehen zu können.

Neben diesen quantitativ bewertbaren Eigenschaften im Umfeld von Performanz, Transaktionsdurchsatz, Star-Query Ausführung etc. spielen darüber hinaus weiche Faktoren eine zentrale Rolle bei der Entwicklung der neuen SAP IM-Technologie. So muss eine Datenbank-Plattform zum skalierbaren Betrieb von Anwendungssoftware in der Lage sein, Mandantenfähigkeit nativ zu unterstützen und entsprechende Technologie zum Umgang mit mehreren Tausend Mandanten (und insbesondere deren individuellen Erweiterungen) innerhalb einer einzelnen System-Instanz bereitzustellen. Darauf aufbauend muss das System Datenbankdienste anbieten, die effizient und möglichst unterbrechungsfrei Schemaänderungen beispielsweise beim Release-Wechsel der Anwendungssoftware ermöglichen.

Als zentrales Design-Kriterium wurde für die neue SAP IM-Technologie ein hybrider Architekturansatz gewählt. Dies bedeutet, dass hier nicht ein Spezialsystem für eine spezifische Anwendung mit einer spezifischen Technologie realisiert wurde, sondern unterschiedliche und teilweise sogar funktional redundante Komponenten so orchestriert werden, dass für entsprechende Anforderungen jeweils spezifische Subsysteme bzw. Strategien gewählt werden können [5]. Der hybride Ansatz zeigt sich beispielsweise in folgenden Bereichen:

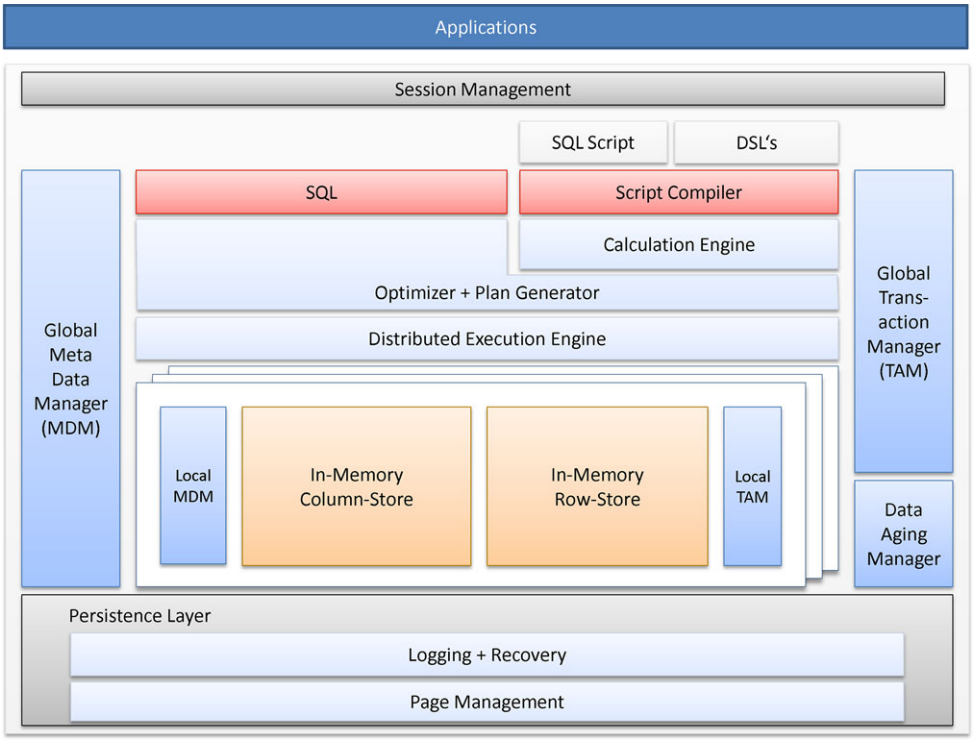
- *Main memory versus disk-basiert*: Die Grundprämisse des Systementwurfs liegt darin, dass alle (relevanten) Daten im Hauptspeicher liegen und direkt für eine Auswertung zur Verfügung stehen. So entfällt im Gegensatz zu disk-zentrierten Systemen zunächst der Zwang sich an Blockstrukturen zu orientieren und darauf aufbauende Indexstrukturen zu realisieren. Der hybride Ansatz wird dadurch sichtbar, dass auf Ebene der Anwendungsplattform entschieden werden kann, welche Daten vor dem Hintergrund der Anwendungssemantik aus dem Hauptspeicher auf langsameren Disk/SSD Speicher ausgelagert und zur Anfragelaufzeit in den Hauptspeicher wieder eingelesen werden sollen. Obwohl prinzipiell hauptspeicherzentriert, können also aus Sicht der Anwendung und nicht aus Sicht eines Systempuffers weniger relevante Datenbestände grobgranular auf Externspeicher verwaltet werden.
- *Column versus row-basiert*: In Abhängigkeit von der Arbeitslast haben sich unterschiedliche physische Datenorganisationen als praktikabel erwiesen [7]. Sind Zugriffe über eine Vielzahl von Tupel aber nur wenige ausgewählte Attribute vorherrschend, erscheint eine spaltenbasierte Ablage sinnvoll; wird häufig auf einzelne aber ganze Tupel zugegriffen, ist eine row-basierte Ablage zu bevorzugen. Die neue SAP IM-Technologie ermöglicht mit einer Kombination wiederum die Ablage und Interpretation in beiden Formen, wobei die Art der Organisation durch die Anwendung vorgegeben werden kann.
- *Analytisch versus transaktionaler Zugriff*: Wie bereits erwähnt, erfordern moderne Geschäftsanwendungen einen direkten analytischen Zugriff auf die operationale Datenbasis, so dass auf Ebene der Systemarchitektur ein Mischbetrieb zu realisieren ist, der neben dem „Verbuchen“ von Geschäftstransaktionen im Sinne von Schreibvorgängen auf die Datenbasis uneingeschränkte analytisch-geprägte (Aggregations-)Anfragen ermöglicht. Auch dieser hybride Charakter wird von der neuen SAP IM-Technologie durch entsprechende Entkopplung von Leser und Schreiber mit transaktionalem Schutz gewährleistet.
- *Scale-Out versus Scale-Up*: Die Konfiguration eines Systems zur Unterstützung großer Anwendungssoftware muss in der Lage sein, die spezifischen Zugriffscharakteristika mit Blick auf eine Skalierung zu reflektieren; die neue SAP IM-Technologie ist dabei so entworfen, dass überwiegend transaktionale Lasten durch einen Scale-Up auf Basis von SMP-Architekturen, eher analytische Anfragen durch eine Scale-Out-Konfiguration optimiert werden können. Die neue SAP IM-Technologie verfolgt auch auf dieser Ebene einen hybriden Ansatz, um sich den Anforderungen gegenüber flexibel und anpassbar zu gestalten.
- *SQL versus DSL*: Kommunizieren klassischerweise Anwendungen über SQL mit dem zugrundeliegenden Datenbanksystem, so bietet die neue SAP IM-Technologie neben einem klassischen SQL-Zugang über übliche Protokolle wie JDBC/ODBC auch native Unterstützung für domänenspezifische Sprachen. Das System ist dabei so entworfen, dass ein einheitliches semantisches Modell die direkte Integration unterschiedlicher Anfrage- und Programmiersprachkonzepte in die neue SAP IM-Technologie ermöglicht. Ziel ist es dabei nicht nur anwendungsspezifische Operationen als Grundbausteine im Datenbanksystem für eine effiziente Ausführung direkt an den Daten zu realisieren, sondern Programmiereffizienz der darüberliegenden Schichten zu steigern und leichtgewichtige Applikationen möglichst gut aus Sicht der Daten-Management-Plattform zu unterstützen.

1.1 Gliederung des Beitrags

Der Beitrag gliedert sich im Folgenden in drei Bereiche. Im nachfolgenden Abschn. 2 wird ein Überblick über den aktuellen Stand der neuen SAP IM-Technologie und deren Architektur gegeben. Dabei werden einzelne spezifische Eigenschaften, welche die neue SAP IM-Technologie von ei-

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

Abb. 1 Architekturüberblick



nem klassischen Datenbanksystem unterscheidet, ausführlicher diskutiert. Da ein „deep dive“ nicht in jeder Komponente möglich ist, greifen sich die beiden folgenden Kapitel jeweils einen konkreten Aspekt heraus. Im Abschn. 3 wird ein Aspekt der physischen Organisation der Daten im spaltenorientierten Kontext herausgegriffen; im Detail werden unterschiedliche Komprimierungs- und Sortierungskriterien diskutiert. Abschn. 4 greift die Unterstützung einer spezifischen Klasse von Geschäftsanwendungen im Umfeld der Planung auf; es wird skizziert, wie eine Planungssprache auf das zugrundeliegende Verarbeitungsmodell abgebildet wird. Dadurch werden dem Leser auf der einen Seite ein breiter Überblick, auf der anderen Seite punktuell zwei spezifische Teilaspekte präsentiert.

2 Architekturüberblick

Im diesem Kapitel wird in Anlehnung an die schematische Darstellung aus Abb. 1 ein Überblick über die grundsätzliche Architektur der neuen SAP IM-Technologie gegeben. Dabei werden insbesondere jene Komponenten und Konzepte in Szene gesetzt, welche die neue SAP IM-Technologie von herkömmlichen Datenbanksystemen bewusst unterscheidet.

2.1 Schnittstellen

Auf der obersten Ebene der SAP IM-Technologie übernimmt ein *Session Manager* den Aufbau und die Organisation von Verbindungen zur Ebene der Applikationssoftware. Dabei wird für jede Session ein Satz von Parametern, wie etwa das zu verwendende Transaktionsisolationslevel, verwaltet. Sobald eine Verbindung aufgebaut wurde, kann der Client mit Hilfe von SQL, SQL-Script oder domänenspezifischen Sprachen mit der Datenbank kommunizieren.

Jede Kommunikation mit der Anwendung wird dabei im Kontext einer Transaktion ausgeführt und neue Sessions implizit einer neuen Transaktion zugeordnet, wobei der *Transaction-Manager* die Isolation, sowie die Verwaltung der verschiedenen Transaktionen übernimmt. Wird eine Transaktion abgeschlossen, teilt der globale Transaction-Manager dies den betroffenen Engines (row\column, main-memory\from disk) mit, so dass diese den Vorgang in ihrem lokalen Transaktionsmanagement berücksichtigen können. Außerdem speichert und organisiert der Transaction-Manager die Transaktionsnummern (TID), welche vom Column- bzw. Row-Store, für die Umsetzung der Multi Version Concurrency Control (MVCC) verwendet wird [4, 9]. Obwohl die MVCC in den jeweiligen Engines zum Teil unterschiedlich umgesetzt ist, sieht das grundsätzliche Konzept der MVCC vor, dass Einträge in der Datenbank nicht überschrieben werden, um auch ohne Sperren konkurrierende Zugriffe auf konsistenten Daten zu ermöglichen. Aktualisierte Einträge werden wie neue Einträge mit einer aufsteigenden TID hinzugefügt. Bei einer Leseoperation wird dann entsprechend auf den aktuellsten Datenausschnitt zugegriffen, in dem die vom Transaction-Manager geführten TIDs als zusätzlicher Filter herangezogen werden.

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

Die Tatsache, dass für alle Änderungen kontinuierlich aufsteigende TIDs verwendet werden, vereinfacht auch die Implementierung zwei weiterer Konzepte der neuen SAP IM-Technologie. Zum einen helfen die TIDs ein Data Aging zu definieren, wobei Dateneinträge nach einer von der Anwendung vorgegebenen Zeit an die disk-basierte Engine übertragen werden. Zum anderen ermöglichen die TIDs, dass mit dem sogenannten *Time Traveling* jederzeit auch vergangene Zustände der Datenbank rekonstruiert werden können.

2.2 Anfrageverarbeitung

Eine von der Anwendung abgesetzte SQL Nachricht wird vom *SQL Processor* in einen logischen Ausführungsplan überführt, der anschließend durch algebraische Transformationen optimiert wird. Der so entstandene logische Ausführungsplan wird dann in Abhängigkeit von der in der Anfrage beteiligten Tabellen und Tabellentypen vom *Plan Generator* ein weiteres Mal optimiert und schließlich in einen tatsächlichen physischen Ausführungsplan überführt. Dieser physische Ausführungsplan enthält schließlich, die von den jeweiligen Engines (Column-/Row-Store) bereitgestellten Operationen, wie etwa ein hauptspeicherbasierter „B-Tree Index Join“ oder „Table Scan“. Die neue SAP IM-Technologie erlaubt dabei auch explizit Anfragen auf Tabellen mit verschiedenen Speicher Layout in nur einem Statement, wie etwa ein Join über eine Column- und eine Row-Tabelle. Da jedoch Cross-Engine Operationen zeitaufwendiger sind als Operationen innerhalb einer Engine, versucht der Plan Generator im Ausführungsplan den Austausch von Zwischenergebnissen verschiedener Engines so gering wie möglich zu halten.

Neben Standard SQL bietet die SAP IM-Technologie, wie in Abschn. 4 noch ausführlicher dargestellt wird, mit *SQL-Script* und/oder mit *Domänenspezifischen Sprachen* (DSLs) die Möglichkeit anwendungsspezifische Logik in die Datenbank einzubringen. In beiden Fällen wird die jeweilige Sprache intern in ein *Calculation-Modell* überführt. Dieses Calculation-Modell beschreibt dabei einen Datenflussgraph, wobei neben klassischen Datenbankoperationen wie Join oder Aggregation auch flexible anpassbare Skripte oder selbst definierte Operationen als Knoten innerhalb des Datenflussgraphs unter Einbeziehung der Optimierung integriert werden können. Diese Flexibilität der zugrundeliegenden Calculation Engine erlaubt es SQL-Script wiederum auch fremde Skriptsprachen, wie etwa R, als spezielle Funktionen innerhalb des SQL-Scripts einzubinden. Ausgehend von der semantischen Beschreibung des Calculation-Modells wird vom Plan Generator anschließend, ähnlich wie bei einem Standard SQL Statement, wiederum ein physischer Ausführungsplan generiert.

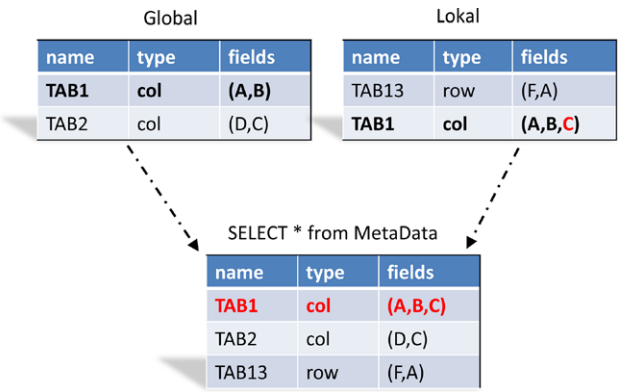


Abb. 2 Tabellen Schemata aus Sicht eines Mandantens

Der SQL Processor, der Script Compiler und auch der Plan Generator benötigen dabei jeweils Zugang zu Metadaten, wie etwa Schemainformationen der Tabellen. Diese werden vom *Metadata Manager* verwaltet, wobei bei einem Multimandanten System [3] zwischen global verfügbaren Metadaten und lokal für den jeweiligen Mandanten verfügbaren Metadaten unterschieden wird. Wie in Abb. 2 dargestellt, setzt sich die Sicht des jeweiligen Mandanten aus den global verfügbaren Metadaten und den nur für ihn sichtbaren lokalen Metadaten zusammen. Diese Tatsache erlaubt nicht nur mandantenspezifische Tabellen einzuführen, sondern auch Schemaänderungen an global verfügbaren Tabellen nur lokal sichtbar für einzelne Mandanten vorzunehmen.

Der physische Ausführungsplan, der vom Plan Generator erstellt wurde, wird schließlich der *Execution Engine* übergeben. Sie übernimmt dabei die Aufgabe der Orchestrierung der verschiedenen Planoperationen, sowie die Übergabe der Zwischenergebnisse über die verschiedenen Engines (z.B. Column-/Row-Store) hinweg. Insofern ist die Execution Engine auch für die Organisation der parallelen Ausführung verantwortlich, sei es nun im Kontext mehrerer CPUs auf einem lokalen Rechner oder über die Knoten eines verteilten Rechnerclusters hinweg (Threadbasierte/Knotenbasierte Parallelisierung).

2.3 Speicherung der Daten

Zur Verwirklichung der in der Einleitung skizzierten Eigenschaften spielt die Möglichkeit Tabellen nicht nur klassisch zeilenorientiert (Abb. 3 oben) sondern auch spaltenorientiert (Abb. 3 unten) im Hauptspeicher ablegen zu können eine zentrale Rolle. Daher spiegelt der hybride hauptspeicherbasierte Column-/Row-Store einen wichtigen Bestandteil der neuen SAP IM-Technologie wider. Auf Grund der bekannten Stärken des Column-Stores wurde das Konzept in der Vergangenheit häufig als Ergänzung und zur Beschleunigung traditioneller zeilenorientierter Datenbank

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

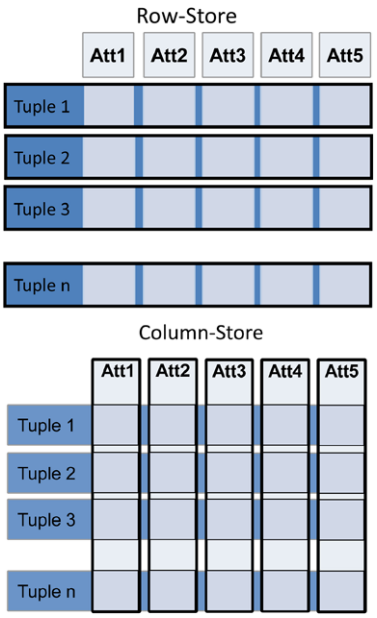


Abb. 3 Layout von Column und Row-Store

Systeme genutzt. Der hybride Column-/Row-Store der neuen SAP IM-Technologie führt diesen Gedankengang insofern konsequent fort, indem der Column-Store hier schließlich gleichberechtigter Bestandteil wird.

Neben den beiden hauptspeicherbasierten Engines beinhaltet das Hybridkonzept auch eine klassische, disk-basierte Engine, die vor allem zum Zweck der Archivierung und Auslagerung weniger relevanter Einträge zum Einsatz kommt. Außerdem teilen sich die beiden In-Memory Engines die disk-basierte *Persistenz*, die als Abstraktion oberhalb der zu persistierenden Speicherseite virtuelle Dateien anbietet. Diese virtuellen Dateien werden von den beiden In-Memory Engines unter anderem auch zum Protokollieren von Schreiboperationen genutzt, wobei dies aufgrund des unterschiedlichen Aufbaus der Engines strukturell individuell, aber operational konsistent erfolgt.

So werden beim Row-Store [6] Schreiboperationen zunächst im Hauptspeicher vollzogen, jedoch auch asynchron als XOR Abbild der Speicherseite in Logdateien persistiert. Die Architektur der Row-Store Logs ist dabei durch die Verwendung redundanter Logdateien vor allem dafür ausgelegt, beim Systemstart die Daten möglichst schnell in den Hauptspeicher übertragen zu können.

Der Column-Store [18] hingegen arbeitet bei Schreibtransaktionen mit sogenannten Delta Tabellen, welche an Stelle der leseoptimierten Tabellen die neuen Einträge führen. Ein *Consistency View Manager*, der auch im Zusammenhang mit der MVCC zum Einsatz kommt, übernimmt im Column-Store dabei die Aufgabe für Leseoperationen eine konsistente Sicht zwischen den leseoptimierten Tabellen und den entsprechenden Delta Tabellen herzustellen. Da die Delta Tabellen jedoch nur schreiboptimiert sind, werden nach Möglichkeit in regelmäßigen Abständen sogenannte *Delta Merges* vollzogen, wobei die Einträge aus der Delta Tabelle in die leseoptimierte Tabelle verschoben werden [12]. Um auch vor dem Delta Merge und der damit verbundenen Speicherung der aktualisierten Tabellen auf der Persistenz Ausfallsicherheit gewähren zu können, wird zusätzlich mit *Redo Delta Logs* gearbeitet. Diese Redo Delta Logs speichern im Gegensatz zu den Delta Tabellen nicht die neuen Einträge, sondern lediglich die Befehle, die zu der Veränderung geführt haben. Für die Wiederherstellung setzt der Column-Store im Gegensatz zum Row-Store, der stets alle Daten lädt, auf ein Recovery on Demand Konzept, wobei in den Metadaten hinterlegt werden kann, ob eine Tabelle zwingend zum Systemstart geladen werden soll, oder erst zum Zeitpunkt der ersten Anfrage. Im Fall eines Ausfalls müssen nach dem Laden der Tabellen die Befehle aus dem Redo Delta Log erneut ausgeführt werden, um die Tabellen wieder auf den aktuellen Stand zu bringen.

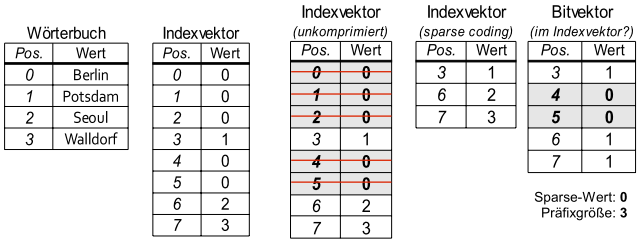
2.4 Zusammenfassung

Zusammenfassend bleibt festzuhalten, dass mit der neuen SAP In-Memory-Technologie versucht wird, die Ideen und Herausforderungen einer hybriden Datenbankarchitektur umzusetzen, um für die Anforderungen bereits existierender aber auch kommender Geschäftsanwendungen gerüstet zu sein. Um einen besseren Einblick geben zu können, werden im Folgenden zwei spezifische Themengebiete aus zwei unterschiedlichen Schichten des Architekturstacks ausführlich diskutiert – die Optimierung physischer Datenstrukturen im Column-Store und die Anwendungsintegration am Beispiel von Planungsanwendungen.

3 Detailbetrachtung 1: Physisches Design von In-Memory Column-Store-Strukturen

Im gleichen Umfang, wie die zu verarbeitenden Datenmengen im OLAP-Umfeld von Jahr zu Jahr ansteigen, wächst auch die Nachfrage nach Ad-hoc- und Echtzeitanalysen. Um nun im knappen und schnellen Hauptspeicher so viele Daten wie möglich verarbeiten zu können, bietet sich die spaltenweise Kompression an. Die daraus folgende größere Datenlokalität erlaubt zusätzlich eine Beschleunigung der Anfragebearbeitung durch eine bessere Cache-Ausnutzung. Damit die Kompression vorhandene Redundanzen besser ausnutzen und ein optimales Ergebnis erzielen kann, müssen die Daten vorher geeignet umsortiert werden. Da eine Dekompression einige der erzielten Vorteile zunichte macht, sollten auch möglichst viele Operatoren direkt auf den komprimierten Daten arbeiten. In den folgenden Abschnitten wird auf die einzelnen Punkte im Detail eingegangen.

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8



(a) Domain Coding (b) Sparse Coding

Abb. 4 Beispiel für Domain und Sparse Coding

3.1 Kompressionsverfahren

Die Hauptziele der in diesem Abschnitt vorgestellten Kompressionsverfahren sind eine hohe Kompressionsrate und ein effizienter Zugriff sowohl auf Einzelwerte als auch auf Blöcke von Werten. Als Basis wird das *Domain Coding* [1, 19, 21] verwendet, bei dem alle vorkommenden Werte lexikographisch sortiert in einem Wörterbuch abgelegt und nur noch bit-komprimierte Verweise darauf in einem Indexvektor gespeichert werden. Neben dem Wörterbuch wird so für die Speicherung von n Zeilen mit u unterschiedlichen Werten genau $n\lceil\log_2 u\rceil$ Bits benötigt. Abb. 4(a) zeigt das Domain Coding für die Beispieldaten ‚*Berlin, Berlin, Berlin, Potsdam, Berlin, Berlin, Seoul, Walldorf*', wobei jeder Wert durch zwei Bits repräsentiert wird und die *Position*-Spalte nur logisch vorhanden ist. Die Verwendung von Ganzzahlen bietet neben der Platzersparnis eine höhere Geschwindigkeit durch die Ausnutzung von Hardwareoptimierungen und der SIMD Parallelisierung [20].

Bei dem *Prefix Coding* als einfachstes Verfahren, um den Indexvektor durch Beseitigung von Redundanzen weiter zu komprimieren, wird das wiederholte Vorkommen des gleichen Wertes am Anfang zusammengefasst. Falls der häufigste Wert nicht nur am Anfang, sondern verstreut in der ganzen Spalte auftritt, ist das *Sparse Coding* am besten geeignet. Hier werden alle Positionen des häufigsten Wertes in einem Bitvektor gespeichert und die Vorkommen aus dem Indexvektor gelöscht. Für den Bitvektor kann wieder das Prefix Coding angewendet werden, was bei einen großen Präfix die Kompression zusätzlich verbessert. Das Verfahren ist in Abb. 4(b) für das laufende Beispiel dargestellt.

Die folgenden Kompressionsverfahren arbeiten auf Datenblöcken mit möglichst wenig verschiedenen Werten, um eine gute Kompressionsrate zu erzielen. So wird im *Cluster Coding* für Blöcke mit nur einem unterschiedlichen Wert der jeweilige Wert gespeichert und in einem Bitvektor festgehalten, welche Blöcke komprimiert wurden. Falls in einer Spalte pro Block nur wenig verschiedene Werte auftreten, kann *Indirect Coding* eingesetzt werden. Hier wird für jeden geeigneten Block das bereits beschriebene Domain Coding angewendet, wodurch eine zusätzliche Indirektionsstufe entsteht. Um die Anzahl und den Speicherbedarf der Block-Wörterbücher zu reduzieren, benutzen aufeinander folgende Blöcke das gleiche Wörterbuch, solange sich durch neue Einträge die Anzahl der zur Speicherung benötigten Bits nicht erhöht. Bei einer Spalte mit u_{col} verschiedenen Werten lohnt sich Indirect Coding für einen Block der Größe k mit u_{block} verschiedenen Werten genau dann, wenn das Wörterbuch und die Referenzen kleiner als die Daten mit angewendeten Domain Coding sind:

$$u_{block}\lceil\log_2 u_{col}\rceil + k\lceil\log_2 u_{block}\rceil < k\lceil\log_2 u_{col}\rceil$$

Schließlich steht noch ein leicht modifiziertes *Run Length Coding* zur Verfügung, welches eine optimale Kompression bei großen und variablen Blöcken mit nur einem unterschiedlichen Wert erreichen kann. Es werden dabei aufeinander folgende Vorkommen des gleichen Wertes zusammengefasst, indem die jeweilige Startposition und der dazugehörige Wert in jeweils einem Indexvektor abgespeichert werden.

3.2 Optimale Sortierung

Bevor die gerade vorgestellten Verfahren ihre maximale Kompressionsrate erzielen können, müssen die Daten in geeigneter Weise sortiert werden. Falls es in einer Spalte einen Wert gibt, der besonders häufig auftritt, so lohnt es sich dessen Vorkommen an den Anfang zu verschieben, damit dieser Präfix bei der weiteren Kodierung nicht berücksichtigt werden muss. Alle auf Blöcken basierenden Verfahren profitieren weiterhin davon, dass große zusammenhängende oder viele Blöcke mit möglichst wenig verschieden Werten entstehen. Da jedoch für eine ideale Sortierung der Lösungsraum exponentiell mit der Anzahl der Spalten ansteigt, ist es praktisch unmöglich eine optimale Lösung für das Problem zu finden. So wird in [2] gezeigt, dass eine optimale Sortierung unter Verwendung von Run Length Coding ein NP-vollständiges Problem ist. Aus diesem Grund wurden einige Greedy-Algorithmen entwickelt, die versuchen einen Kompromiss zwischen Laufzeit und Platzersparnis zu finden. Die einzelnen Spalten können nicht unabhängig voneinander sortiert werden, weil die Position eines Wertes die Zugehörigkeit zu einer Zeile angibt und ansonsten ein teures Mapping notwendig wäre. Durch das Verschieben des häufigsten Wertes an den Anfang und das Sortieren von Bereichen, entstehen Restbereiche, die in anderen Spalten weiter sortiert werden können ohne eine existierende Ordnung zu zerstören (siehe Abb. 5). Um möglichst große zusammenhängende Blöcke zu erhalten, werden in den Restbereichen die Werte nach ihrer Häufigkeit sortiert.

Für die Bestimmung der Reihenfolge, in der die Spalten sortiert werden, gibt es verschiedene Strategien. In der *Reversed* Strategie werden die Restbereiche umgekehrt zur

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

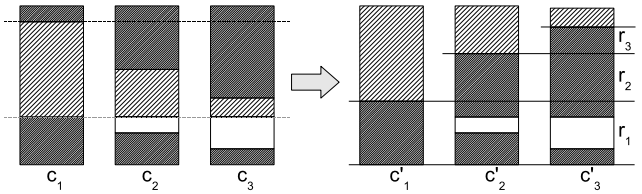


Abb. 5 Entstehung von Restbereichen durch Sortieren

reihenfolge sortiert, in welcher der häufigste Wert nach oben verschoben wurde. Dadurch werden die Spalten vorrangig behandelt, die noch am wenigsten optimiert wurden. Die Strategie *Single Value Blocks* (Maß: Blöcke mit nur einem unterschiedlichen Wert) versucht die Kompression unter Verwendung des Cluster Codings zu verbessern, wohingegen *Average Distinct Values* (Maß: durchschnittliche Anzahl von Werten pro Block) und *Valuable Blocks* (Maß: Anzahl lohnenswerter Blöcke) sich auf das Indirect Coding konzentrieren.

Die Sortierung einer Spalte ist so implementiert, dass nicht die Werte in allen Spalten getauscht werden, sondern nur die Einträge in einem Mapping, das die neue Reihenfolge der Zeilen bestimmt. Nach einer solchen Umsortierung einer Spalte können die Daten nicht mehr blockweise verarbeitet werden, sondern es sind viele teurere Einzelzugriffe nötig. Deshalb werden die Maße für die Strategien nicht in jedem Schritt neu berechnet, sondern nur einmal initial, was nach [13] völlig ausreichend ist. In der Arbeit ist auch eine weiterreichende Auswertung und tiefere Einblicke in das Themengebiet zu finden. Zum Abschluss der Sortierung wird mit Hilfe des Mappings die neue Reihenfolge materialisiert und persistiert.

Ein anderer Ansatz, bei dem die Spalten nicht einzeln, sondern mehrere abhängige Spalten zusammen komprimiert werden, wurde erstmalig von [16] beschrieben. Da in den meisten Fällen die Kenntnis (schwach) funktionaler Abhängigkeiten nicht explizit gegeben ist, werden diese mit Hilfe geschätzter Entropiewerte einzelner Spalten und Spaltenpaare bestimmt. Um die Berechnung zu beschleunigen, kommt darin ein blockweises Sampling und zusätzliche Pruningheuristiken zum Einsatz, um die Anzahl der zu betrachteten Spaltenpaare zu reduzieren. Die so gewonnenen Abhängigkeitsinformationen können weiterhin zur Bestimmung einer besseren Sortierung genutzt werden.

3.3 Operatoren auf komprimierten Daten

Weil die Dekompression in den meisten Fällen ein nicht unerheblicher Kostenfaktor ist, wird in diesem Abschnitt auf einen wichtigen Basisoperator *Scan* eingegangen und anhand des Sparse Codings beschrieben, wie er direkt auf den komprimierten Datenstrukturen arbeitet. Die Geschwindigkeit der Operatoren hängt von der für die Materialisierung der Ergebnisse verwendeten Datenstrukturen (Bitvektor oder Vektor von Ganzzahlen), den Eigenschaften der Kompressionstechnik, der Selektivität und vielen anderen Faktoren ab. Wie mögliche Optimierungen für andere Kompressionstechniken und Operatoren (wie zum Beispiel Aggregation) aussehen, wird von [14] im Detail behandelt. Aus der ausführlichen Auswertung werden im nächsten Abschnitt auch einige Ergebnisse kurz vorgestellt.

Wie bereits im Abschn. 3.1 erwähnt, wird beim Sparse Coding ein Bitvektor B verwendet, in dem für eine Zeile aus dem Indexvektor I das Bit gesetzt ist, falls an der Position *nicht* der häufigste Wert steht. Der Präfixoffset p ist die erste Zeile, deren Wert sich vom Wert im Präfix unterscheidet. Im ersten Schritt des Scan-Operators wird überprüft, ob einer der gesuchten Werte im Präfix vorkommt. Wenn der häufigste Wert angefordert wird, dann lassen sich die dazugehörigen Zeilen leicht über B ermitteln. Falls erforderlich wird abschließend in I noch nach den gewünschten Werten gesucht und die Zeilen durch Zählen der nicht gesetzten Bits in B bestimmt:

1. Abschätzung der aktuellen Zeile z unter der Annahme, dass keine Werte in I gelöscht wurden.
2. Bestimmung der Anzahl der nicht gelöschten Werte n in diesem Bereich.
3. Wiederhole folgende Schritte, solange n größer als z ist:
   (a) Inkrementiere z.
   (b) Falls der aktuelle Wert nicht der häufigste Wert ist, inkrementiere n.
4. Die Ergebniszeile ergibt sich aus $p + z$.

Um die Berechnung der gesetzten Bits zu beschleunigen, wird eine Datenstruktur aufgebaut, die für einen Block die Summe der in den vorherigen Blöcken gesetzten Bits speichert.

Eine weitere Möglichkeit zum Beschleunigen von Zugriffen auf einzelne Werte sind geblockte, invertierte Indexe. Hier wird pro Wert gespeichert, in welchen Blöcken er vorkommt, wodurch bei einem Scan im Idealfall nur einige wenige Blöcke betrachtet werden müssen.

3.4 Experimentelle Auswertung

Abschließend sollen in diesem Abschnitt einige Ergebnisse der Experimente von [14] vorgestellt werden, in denen für verschiedene synthetische Datenverteilungen und Kompressionstechniken der erzielte Speicherbedarf und die Zeiten für die Suche nach einem Wert untersucht wurden. Die Daten bestehen aus einer Spalte mit 10 Millionen Zeilen und 4472 unterschiedlichen Werten (Ganzzahlen). In der *Linear Skew* Verteilung kommt der i-te Wert zusammenhängend $i + 1$ mal vor. Die Werte in den *Sparse*-Daten werden aufeinanderfolgend durchnummeriert und es wird ein sehr häufiger (Sparse) Wert am Anfang (top) eingefügt, der 66 % der Daten ausmacht.

Die Ergebnisse sind immer relativ zu den Daten mit angewandtem Domain Coding (Indexvektor) angegeben, um

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

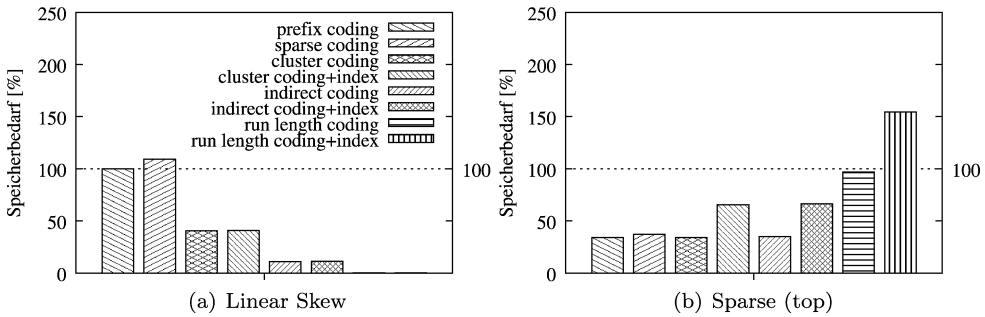


Abb. 6 Speicherbedarf relativ zu nur Domain Coding

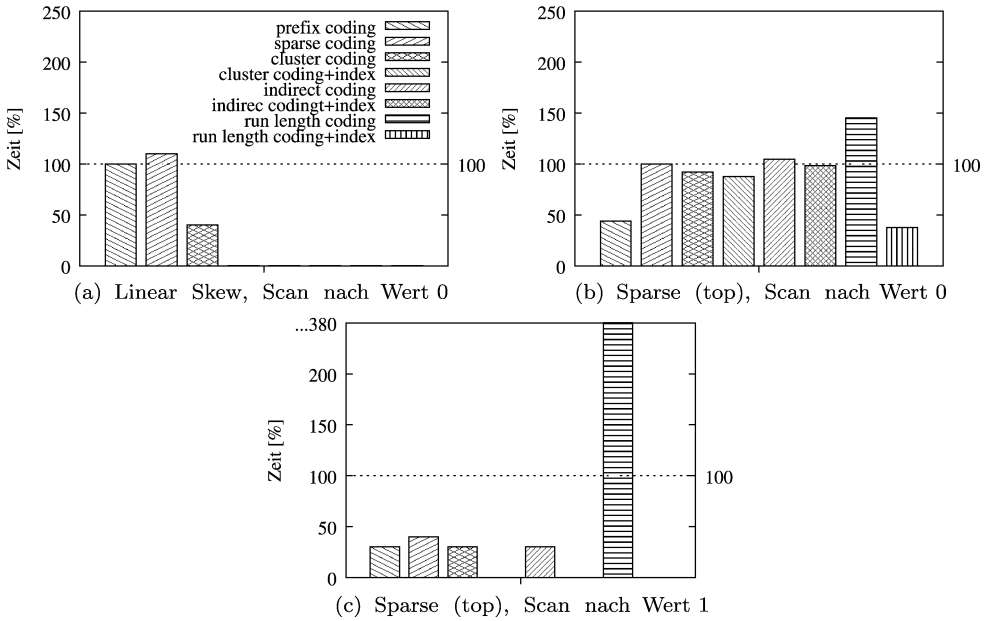


Abb. 7 Zeit für Scan nach Einzelwerten relativ zu nur Domain Coding

Effekte bezüglich bestimmter Datentypen zu umgehen. Weiterhin setzt die verwendete Implementierung von Cluster Coding und Indirect Coding eine Blockgröße von 1024 Werten ein. Alle Experimente wurden auf einem Intel® Xeon® X5650 Prozessor mit 2,67 Ghz durchgeführt.

Abb. 6 zeigt den Speicherbedarf der eben beschriebenen Datenverteilungen, wobei auch die Größen unter Verwendung des invertierten Indexes enthalten sind. Anhand der *Linear Skew* Verteilung in der Abb. 6(a) kann man gut sehen, dass die zusätzlichen Kosten für den Bitvektor im Sparse Coding zu einem höheren Speicherbedarf führen, wenn sich die Anzahl des häufigsten Wertes unter einer bestimmten Schwelle befindet. Das Run Length Coding passt sich am besten an die verschieden große Blöcke an, wodurch die beste Kompressionsrate erzielt werden kann. Weil durch die zusammenhängende Generierung ein Wert nur in wenigen Blöcken vorkommt, ist der zusätzliche Speicherbedarf für die invertierten Indexe nur minimal.

Bei den *Sparse*-Daten mit dem häufigsten Wert am Anfang (Abb. 6(b)) profitieren alle Kompressionstechniken davon, den Präfix nicht mit kodieren zu müssen. Jedoch ist das Run Length Coding durch die vielen Werteänderungen ungeeignet.

Um den möglichen Geschwindigkeitsvorteil durch das Arbeiten auf den komprimierten Datenstrukturen zu zeigen, wurden Scans nach den einzelnen Werten 0 und 1 durchgeführt und die Ergebnisse in Abb. 7 dargestellt. Der Wert 0 entspricht bei den Sparse-Daten dem häufigsten Wert.

Für die *Linear Skew* Verteilung zeigen die Ergebnisse in der Abb. 7(a), dass von allen Kompressionstechniken, die auf Blöcken arbeiten, nur Cluster Coding sich nicht gut an die kleinen Blöcke mit einem unterschiedlichen Wert anpassen und somit der optimierte Scan-Operator nicht seine volle Leistung bringen kann. Weiterhin korreliert die Geschwindigkeit der Anfrage mit der Größe der Datenstrukturen, wenn man von den Zeiten mit invertierten Index absieht. Dann steigt der Speicherbedarf durch die Indexstrukturen zwar an, jedoch müssen immer nur wenige Blöcke des invertierten Indexes betrachtet werden, was zu einer sehr schnellen Anfragebearbeitung führt.

Mit den *Sparse*-Daten in Abb. 7(b) ist am Beispiel des Sparse Codings gut zu erkennen, dass ein geringerer Speicherverbrauch nicht immer eine schnellere Ausführung im-

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

pliziert. So muss auch der Overhead betrachtet werden, der durch die Rekonstruktion der Zeilenpositionen entsteht, besonders wenn das Anfrageergebnis groß wird. Falls die Ergebnisgröße klein ist, wie bei den Scans in der Abb. 7(c), dann sind im Vergleich zu den Ersparnissen die Kosten für Rekonstruktion und für das Vergrößern des Ergebnisvektors zu vernachlässigen. Weil es keine großen Blöcke nur mit dem Wert 1 gibt, sind die häufigen Zugriffe auf einzelne Werte der beiden Indexvektoren, die im Run Length Coding verwendet werden, zu teuer. Das ist auch der Fall, wenn durch einen großen Präfix die Datenmenge reduziert wurde.

Alle Messungen zeigen, dass beim Einsatz von invertierten Indexen die Scans nach einzelnen Werten immer schneller sind. Run length coding lohnt sich nur, wenn es wenige aber große Blöcke mit nur einem unterschiedlichem Wert gibt und ist im Zusammenhang mit dem invertiertem Index die schnellste Technik. Auf der anderen Seite bieten Cluster Coding und Indirect Coding allgemein betrachtet den besten Kompromiss zwischen Geschwindigkeit und Speicherverbrauch.

Nachdem die Optimierung physischer Datenstrukturen im Column-Store einen tieferen Einblick in die unteren Schichten der Architektur der SAP In-Memory-Technologie gegeben hat, folgt ein Blick von oben aus Sicht der Anwendung und zeigt am Beispiel Planung, wie eine Anwendungsintegration mit der SAP In-Memory-Technologie aussehen kann.

4 Detailbetrachtung 2: Unterstützung von Planungsapplikationen

Wie bereits erwähnt, ist ein wichtiger Aspekt der neuen SAP IM-Technologie die enge Integration mit der Anwendungsschicht. Dadurch sollen Anwendungen in die Lage versetzt werden, mit geringem Aufwand die Vorteile effizienter und schneller Verarbeitung großer Datenmengen zu nutzen. Dafür bietet SAP's IM-Technologie eine Reihe von anwendungsspezifischen Funktionsprimitiven an, die durch die Anwendung aufgerufen und miteinander kombiniert werden können, um komplexe Anwendungslogik zu formulieren. Das ermöglicht leichtgewichtige Anwendungen, die sich hauptsächlich um die Interaktion mit dem Benutzer kümmern und datenintensive Prozesse innerhalb der SAP IM-Technologie kontrollieren. Dieses Konzept soll am Beispiel von Planungsanwendungen aus dem Bereich Geschäftsplanung (Controlling) im Folgenden ausführlich erläutert werden.

4.1 Planungsprimitive

Geschäftsplanung weist per se eine enge Verwandtschaft zu Reporting Aufgaben und damit auch zu analytischen Anfragen und dem zugrundeliegenden mehrdimensionalen Datenmodell auf. Beim klassischem Berichtswesen stehen historische Daten aus dem vergangenen Zeitraum im Fokus. Bei der Planung dagegen sind sowohl die aktuellen Zahlen der laufenden Geschäftsperiode von Interesse, als auch die Erstellung von Plan-Zielen für zukünftige Perioden. Diese Zielvorgaben dienen nachfolgend dazu, den Geschäftserfolg mittels einen Soll-Ist Vergleich zu messen.

Beispiel *Ein Controller soll Umsätze für Produkte in verschiedenen Regionen für das Jahr 2010 planen. Dafür bedient er sich der Daten aus dem vorangegangenen Jahr. Er erwartet ein generelles Umsatzwachstum von 5 % und aufgrund saisonaler Trends einen besonderen Umsatzanstieg von Produkten in der Kategorie HD-Fernseher.*

Aus Sicht der Anwendung ergeben sich damit eine Menge von Grundfunktionen, die für die Geschäftsplanung benötigt werden. Zum einen müssen Plandaten beschafft oder erzeugt werden. So werden beispielsweise als Grundlage der Planung die Daten des Vorjahres verwendet (*Kopieren*). Weiterhin verändern sich Dimensionswerte, wenn zum Beispiel die Daten aus dem Vorjahr über nommen werden, muss die Zeitdimension entsprechend angepasst werden (*Kopieren/Umbuchen*). Außerdem werden Kennzahlen manipuliert sowie neue Fakten erzeugt (*Umwerten/Werte setzen*) wie im Beispiel die Erhöhung des erwarteten Umsatzes. Weiterhin kann ein Zielumsatz für eine gesamte Produktkategorie – im Beispiel HD-Fernseher – eingegeben werden, welcher dann anteilig auf einzelne Produkte zu verteilen ist (*Verteilen*). Danach können Detailwerte eventuell weitere Anpassungen erfahren, etwa wenn bestimmte Produkte nicht mehr oder in begrenzten Stückzahlen erhältlich sind. An diesem Beispiel wird auch deutlich, dass Planung auf verschiedenen Verdichtungsebenen stattfindet. Neben den genannten Standardfunktionen wie *Kopieren, Umbuchen, Umwerten, Verteilen* und *Werte setzen* gibt es weitere, wie zum Beispiel *Löschen* oder auch *Prognosefunktionen*. Darüber hinausgehende Funktionen sind häufig benutzerspezifisch und werden über die Möglichkeit zur Eingabe von Formeln abgedeckt. Häufig bietet die Anwendung für diesen Zweck eine auf die Anwendungsdomäne zugeschnittene Sprache an (Domänenspezifische-Sprache). Der Planungsprozess kann nun ganz abstrakt als eine langlaufende Transaktion gesehen werden, die eine Abfolge von Anfragen und Aufrufen von Planungsfunktions-Primitiven enthält und am Ende der Transaktion die erzeugten Daten in eine Zieltabelle schreibt. Abb. 8 stellt diesen Prozess schematisch dar.

Wie bereits erwähnt, findet Planung auf unterschiedlichen Verdichtungsebenen statt. Häufig zeigt die Anwendung dem Nutzer aus Gründen der Übersichtlichkeit eine aggregierte Sicht der Daten. Die Planungsfunktionen wirken sich dann allerdings auf die wesentlich größere Menge der nichtaggregierten Daten aus. Beispielsweise wenn entlang einer

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

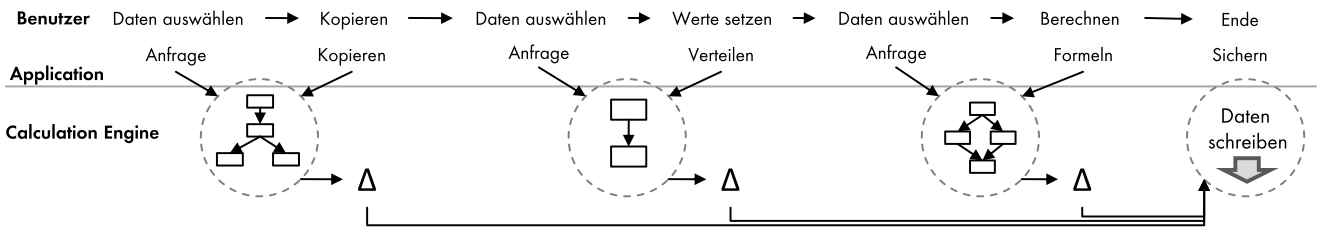


Abb. 8 Planung als Abfolge von Planungsfunktionen innerhalb einer langlaufenden Transaktion

komplexen Hierarchie mit vielen 1000 Gruppen, Untergruppen und Details eine Summe auf die Details verteilt wird. Für solche Funktionen bietet es sich deshalb an, sie innerhalb der Datenbank auszuführen und der Anwendung Funktionsprimitive anzubieten, durch die sich diese Funktionalität ausdrücken lässt. Die neue SAP In-Memory-Technologie stellt der Anwendung solche Planungsprimitive zur Verfügung, also unter anderem das *Kopieren* von Daten, das *Verteilen* oder *Umwerten*. Intern werden diese Funktionen jedoch nicht direkt implementiert, sondern auf die logische Ausführungsschicht der Calculation Engine abgebildet. Diese bietet eine Reihe datenorientierter Datenbankprimitive wie Join, Union, Projektion und Selektion an. Damit wird die Menge an tatsächlich vorhandenen Primitiven in der Datenbank möglichst minimal gehalten und durch geschickte Kombination der vorhandenen Primitive kann relativ einfach und generisch neue Funktionalität angeboten werden (siehe [10]). Ein zusätzlicher Vorteil ist, dass durch die interne Abbildung auf bestehende Operationen bereits vorhandene Optimierungen ausgenutzt werden können. Zusätzlich kann es aus funktionaler Sicht nötig sein einige neue Planoperationsprimitive einzuführen, wie es zum Beispiel für die korrekte Verteilung von Restwerten bei der Verteilungsfunktion der Fall ist oder um eine höhere Ausführungsgeschwindigkeit zu erzielen. In diesem Fall muss die Menge der Datenbankprimitive behutsam erweitert werden, so dass die Erweiterung generisch ist und ein möglichst breites Spektrum an Funktionalität abdeckt.

4.2 Abbildung der Planungsprimitive auf logische Ausführungspläne

Die Calculation-Engine der SAP IM-Technologie bietet die Möglichkeit einzelne Teilmodelle sukzessive übereinander zu stapeln. Das wird bei der Abbildung der Planungsprimitive ausgenutzt, um automatisch eine Art Historie der Planungssitzung zu erstellen. Damit kann zum Beispiel sehr einfach das rückgängig machen eines Planungsschrittes realisiert werden, indem die entsprechenden Operatoren wieder aus dem Modell, welches die Historie widerspiegelt, entfernt werden. Weiterhin können die gespeicherten Modellinformationen von der Anwendung zu Zwecken der Nachvollziehbarkeit verwendet werden.

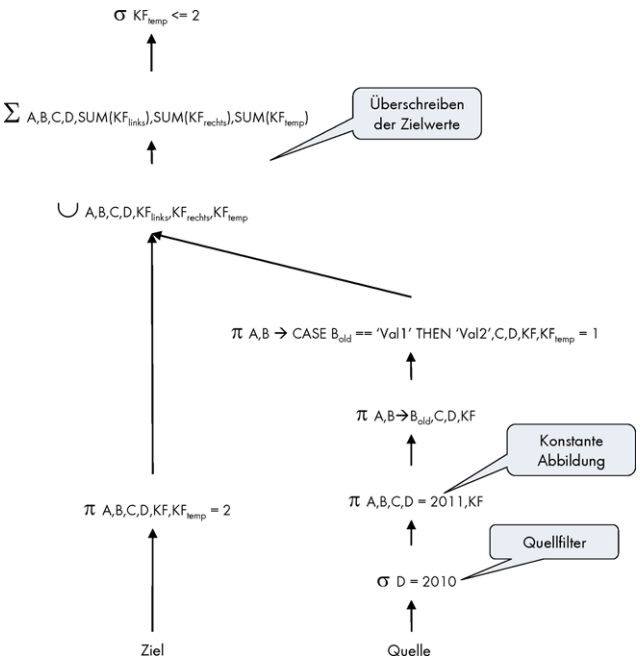


Abb. 9 Abbildung einer Kopieren Funktion mit konstanten und variablen Wertetransformationen auf ein Calculation-Engine Modell

Das folgende Beispiel soll die Abbildung eines Planungsprimitives auf ein Calculation-Modell verdeutlichen. Das abzubildende Planungsprimitiv ist in diesem Fall das Kopieren. Prinzipiell sollen dabei Daten aus einer Quelle in ein Ziel kopiert werden. Die zu kopierenden Daten können eingeschränkt werden und außerdem können mit Hilfe von Abbildungsregeln Quellwerte in neue Zielwerte transformiert werden. Beispielsweise wird das Jahr in den Quelldaten auf 2010 eingeschränkt und soll im Ziel auf 2011 abgebildet werden. Abb. 9 zeigt einen logischen Datenflussgraphen, der eine mögliche Kopierfunktion repräsentiert. Die Kopierfunktion ist in diesem Fall die Vereinigung der existierenden Zieldaten mit den gefilterten Quelldaten. Die konstante Wertetransformation kann im Falle des Beispiels mittels einer Konstanten für die Dimension *D* innerhalb der Projektion, die auf das Zielschema abbildet, umgesetzt werden. Zusätzlich gibt es eine variable Wertetransformation für die Dimension B – in der Abbildung nur beispielhaft gezeigt, die durch einen CASE-Ausdruck formuliert wird. Wird eine große Zahl an Wertetransformationen

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

für eine Dimension definiert, so wird dies über einen Join mit einer temporären Tabelle abgebildet, welche die entsprechenden Wertepaarungen enthält. Sollen wie im Beispiel die Quellwerte existierende Zielwerte überschreiben, müssen diese aus dem Ziel herausgelöscht werden. Das geschieht durch einen Union und eine nachgeschaltete Aggregation über eine Hilfskennzahl, so dass als Ergebnis die Menge der Quellwerte ohne die in der Quelle enthaltenen Zielwerte vereinigt mit den neuen Zielwerten übrig bleibt. Nach diesem Prinzip lassen sich alle Planungsfunktionsprimitive auf Datenflussgraphen der Calculation Engine abbilden.

4.3 Abbildung von prozeduralen Formeln

Ein weiterer wesentlicher Teil einer Planungssitzung besteht aus spezifischer Geschäftslogik, die nicht über die Standard Planungsprimitive abgebildet werden kann, sondern sich der Möglichkeit von Formeln und einer prozeduralen Domänenspezifischen Sprache (DSL) bedient. Auf den ersten Blick scheint es hier nicht ohne weiteres möglich, die prozedurale Logik auf einen deklarativen Plan abzubilden, wie dies für die Planungsprimitive der Fall ist. Bei genauerer Betrachtung gibt es jedoch diverse prozedurale Konstrukte, die sich in vielen Fällen auf einen deklarativen Plan abbilden lassen. Seiteneffektfreie Schleifen, die über Eingangsdaten iterieren, lassen sich ausrollen und mittels Projektionen und berechneten Attributen abbilden. Berechnungen mit Elementen aus verschiedenen Zellen des unterliegenden multidimensionalen Datenwürfels lassen sich durch Joins abbilden. Weiterhin lassen sich Bedingungen in einem deklarativen Datenflussgraph mit Hilfe von Filtern realisieren. Neben diesen „gutartigen“ prozeduralen Konstrukten gibt es solche, die kein Äquivalent in der deklarativen Welt haben oder sich nur schwer abbilden lassen. Aus diesem Grund bietet die Calculation Engine die Möglichkeit hybrider Pläne an. In einem hybriden Plan lassen sich Datenflussoperatoren mit prozeduralen Skriptblöcken mischen. Ein Skriptoperator definiert das Schema seiner Ein- und Ausgaben und agiert ansonsten als eine Black-Box, die ihre Eingangsdaten in beliebiger Weise manipuliert. Durch die definierte Schnittstelle lässt sich der Operator in einen Datenflussgraphen einbinden. Der Nachteil ist allerdings, dass mögliche Optimierungen, wie zum Beispiel der Push-Down von Filtern, durch den Skriptoperator blockiert werden, da sein Verhalten dem Optimierer unbekannt ist.

Basierend auf diesem Prinzip ist das Ziel benutzerspezifischer Planungslogik in Form prozeduraler Skripte, möglichst weitgehend auf die Datenflussgraphen der Calculation Engine abzubilden. Eine Variante besteht darin, alles durch einen einzelnen Skriptoperator auszudrücken, der das komplette prozedurale Skript enthält. Dies würde aber zur

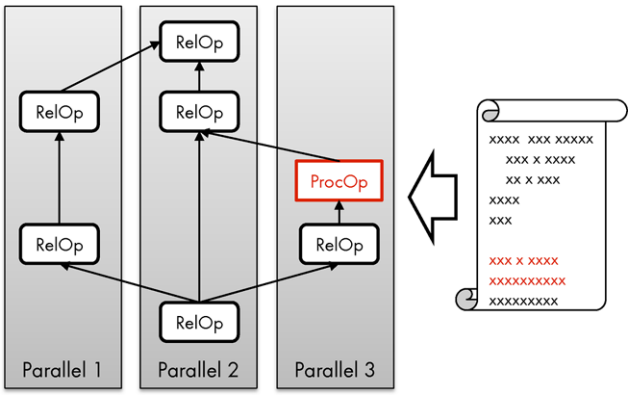


Abb. 10 Schematische Abbildung eines hybriden, verteilten physischen Ausführungsplans

Listing 1 Beispiel Skript

```
FOREACH tupel IN dataset:
IF tupel['Kategorie'] == 'Kat. 1':
tupel['Erlös'] = tupel['Erlös'] * 1.1;
ELSE IF tupel['Kategorie'] == 'Kat. 2':
tupel['Erlös'] = tupel['Erlös'] * 1.2;
ELSE IF ...
ELSE
tupel['Erlös'] = 0;
```

Ausführungszeit jegliche Möglichkeit zur Optimierung oder Parallelisierung verhindern. Die andere Variante besteht darin, ein prozedurales Skript vollständig auf ein Calculation-Modell abzubilden [11]. Typischerweise entsteht jedoch ein hybrider Plan, in dem der Optimizer versucht die Balance zwischen beiden Varianten zu finden (Abb. 10). Das daraus abgeleitete Optimierungsziel ist die Minimierung der Skriptknoten im Modell sowie damit einhergehend die Minimierung der Skriptanteile, die prozedural übersetzt werden. Um dieses Ziel zu erreichen, werden Transformationsregeln definiert, die aus einer Quell- und Zielrepräsentation bestehen. Als Quelle dient jeweils ein Muster für ein prozedurales Sprachkonstrukt, welches sich in ein äquivalentes deklaratives Teilmodell – beschrieben durch das Ziel einer Regel – überführen lässt. Die Menge dieser Regeln wird auf das Eingangsskript angewendet, um damit alle Teile des Skripts auszuwählen und zu transformierten, für die eine Übersetzung möglich ist. Die verbleibenden Teile des Skriptes werden in Skriptoperatoren transformiert. Ein einfaches Beispiel für eine solche Transformationsregel soll hier kurz erläutert werden. Über eine Formel soll die Iteration über eine Menge bestimmter Fakten ausgedrückt werden, wobei in jeder Iteration, abhängig von dem gerade gültigen Wert einer Dimension, eine Kennzahl mit einem bestimmten Faktor multipliziert wird (siehe Listing 1). Die prozedurale Formel enthält eine Schleife über die Eingangsdatenmenge. Innerhalb der Schleife wird mittels einer IF/ELSEIF-Bedingung jeweils die entsprechende Berechnung ausgeführt und der

Final edited form was published in "Datenbank-Spektrum" 10 (10), S.81-92. ISSN: 1610-1995
http://dx.doi.org/10.1007/s13222-010-0020-8

Wert der zu ändernden Kennzahl zugewiesen. Diese Schleife ist seiteneffektfrei, da während einer Iteration nur die aktuelle Zeile der Eingangsdatenmenge verändert wird. Außerdem werden zur Berechnung keine Zwischenergebnisse aus der vorherigen Iteration referenziert. Für diesen Fall existiert eine Regel, welche die Schleife in eine Projektion übersetzt, die als Ergebnis genau die Eingangsdaten liefert. Für die IF-Bedingung greift eine weitere Regel, welche das Ergebnis der Projektion in disjunkte Blöcke entsprechend der Zahl der Bedingungen aufspaltet. Jede Bedingung wird auf eine berechnete Dimension abgebildet, auf welche gefiltert werden kann. Die eigentliche Berechnung innerhalb der Bedingungen wird dann mittels einer weiteren Regel durch eine berechnete Kennzahl formuliert. Am Ende werden die Blöcke über eine Union-Operation vereinigt, um wieder die Gesamtdatenmenge zu erhalten. Im Beispiel lässt sich das prozedurale Skript komplett auf ein äquivalentes deklaratives Modell abbilden. Zusätzlich hat diese Abbildung den Vorteil, dass die einzelnen Verzweigungen des IF-Befehls parallel berechnet werden können, da sie unabhängig voneinander sind. Damit auch eventuell verbleibende Skriptoperatoren in einem hybriden Plan schnell ausgeführt werden können, ist dafür eine Ausführungsinfrastruktur mit Hilfe der Low-Level-Virtual-Machine [15] eingeführt worden. Damit lassen sich die prozeduralen Knoten zur Erstellungszeit des Calcengine-Modells in native, ausführbare Codeblöcke kompilieren, welche dann wesentlich effizienter ausgeführt werden können als beispielsweise eine interpretierte Sprache.

5 Zusammenfassung

Ziel des Beitrags ist es, die Anforderungen an eine hybride Datenbankarchitektur zu skizzieren und deren Realisierung am Beispiel der SAP In-Memory-Technologie zu illustrieren. Die Realisierung des Systems ist dabei motiviert durch technische Entwicklungen insbesondere im Bereich der Hardware (Multi-Cores, Verfügbarkeit von Hauptspeicher) auf der einen Seite und getrieben durch Anforderungen aus dem Bereich der Entwicklung moderner und leichtgewichtiger Geschäftsanwendungen. Neben der allgemeinen Architektur greift der Beitrag zwei Themengebiete heraus und bespricht diese im Detail. Das erste Themengebiet demonstriert Probleme und Lösungen im Kontext der spaltenbasierten Datenorganisation, wie sie in klassischen Datenbanksystemen nicht anzutreffen sind. Weiterhin existiert ein reiches Metadatenmodell, welches eine direkte Abbildung komplexer Geschäftsvorgänge auf das interne Verarbeitungsmodell erlaubt, um hier eine möglichst enge Kopplung von Anwendung und Daten-Management Plattform zu realisieren.

Literatur

1. Abadi DJ, Madden SR, Ferreira MC (2006) Integrating compression and execution in column-oriented database systems. In: Proc SIGMOD, S 671–682
2. Alsberg PA (1975) Space and time savings through large data base compression and dynamic restructuring. Proc IEEE 63(8):1114–1122
3. Aulbach S, Grust T, Jacobs D, Kemper A, Rittinger J (2008) Multi-tenant databases for software as a service: schema-mapping techniques. In: SIGMOD
4. Bernstein PA, Hadzilacos V, Goodman N (1987) Concurrency control and recovery in database systems. Addison-Wesley, Reading
5. Boncz P, Manegold S, Kersten ML (2009) Database architecture evolution: mammals flourished long before dinosaurs becaume exinct. In: PVLDB
6. Cha SK, Song C (2004) P*time: Highly scalable oltp dbms for managing update-intensive stream workload. In: VLDB
7. French CD (1995) „One size fits all“ database architectures do not work for dds. In: SIGMOD
8. Gray J (2006) Tape is dead. disk is tape. flash is disk, ram locality is king. In: Storage guru Gong show
9. Gray J, Reuter A (1993) Transaction processing: concepts and techniques. Kaufmann, Los Altos
10. Jaecksch B, Lehner W, Faerber F (2010a) A plan for OLAP. In: EDBT, S 681–686
11. Jaecksch B, Lehner W, Faerber F (2010b) Cherry picking in database languages. In: Proc 10th IDEAS
12. Krueger J, Grund M, Tinnefeld C, Plattner H, Zeier A, Faerber F (2010) Database systems for advanced applications: optimizing write performance for read optimized databases. Springer, Berlin, S 291–305
13. Lemke C, Sattler KU, Färber F (2009) Kompressionstechniken für spaltenorientierte bi-accelerator-lösungen. In: BTW, S 486–497
14. Lemke C, Sattler KU, Färber F, Zeier A (2010) Speeding up queries in column stores: a case for compression. In: Proc 12th DaWaK
15. LLVM (2010) http://www.llvm.org/
16. Paradies M, Lemke C, Plattner H, Lehner W, Sattler KU, Zeier A, Krüger J (2010) How to juggle columns: an entropy-based approach for table compression. In: Proc 10th IDEAS
17. Plattner H (2009) A common database approach for oltp and olap using an in-memory column database. In: SIGMOD
18. Ross JA (2008) SAP NetWeaver BI accelerator. SAP Press, Dedham
19. Westmann T, Kossmann D, Helmer S, Moerkotte G (2000) The implementation and performance of compressed databases. SIGMOD Rec 29(3):55–67
20. Willhalm T, Popovici N, Boshmaf Y, Plattner H, Zeier A, Schaffner J (2009) Simd-scan: ultra fast in-memory table scan using on-chip vector processing units. PVLDB 2(1):385–394
21. Zukowski M, Héman S, Nes N, Boncz P (2006) Super-scalar ram-cpu cache compression. In: Proc 22nd ICDE, S 59