Zusammenfassung des Vortrages „Column Stores“

Einleitung

Der Begriff „Column Stores“ (deutsch: spaltenorientierte Datenbank) beschreibt die Art und Weise, in der Daten auf dem verwendeten persistenten Speicher abgelegt werden. In traditionellen RDBMS liegen alle Attribute einer Zeile hintereinander. Daher wird auch der Begriff „zeilenorientierte Datenbank“ verwendet. Zeilenorientierte Datenbanken werden bevorzugt, wenn in einer Transaktion wenig komplette Tupel gelesen oder geschrieben werden. Bei einer spaltenorientierten Datenbank kann jede Spalte in einer eigenen Datei liegen. Auf den Wert eines Attributes folgt in Lese-Reihenfolge nicht der nächste Attributwert, sondern der gleiche Attributwert des nächsten Tupels. Viele Anfragen benötigen nur wenige der verfügbaren Attribute, zum Beispiel:

```
SELECT cust.name, cust.address
FROM CUSTOMER cust
WHERE cust.region=Leipzig
```

Es werden nur drei Attribute der CUSTOMER-Relation benötigt, aber die Relation kann deutlich mehr als drei Attribute haben. Spaltenorientierte Datenbanken sind bezüglich des I/O-Overheads bei Leseoperationen effizienter als zeilenorientierte Datenbanken. Aus der Spaltenorientierung resultiert ein OLAP-Vorteil, da meistens nur wenige Attribute eines Tupels gelesen werden. Online Analytical Processing (OLAP) ist ein effizientes Instrument zur analytischen Bearbeitung von multidimensional aufbereiteten Daten. Im Folgen werden die Vor- und Nachteile von spalten- und zeilenorientierten Datenbanken am Beispiel OLAP dargestellt.

OLAP: Zeilenorientierte Datenbank

Eine zeilenorientierte Datenbank wird bei OLTP-Anwendungen mit kurzen Transaktionen eingesetzt. Die Tupel werden in das DBMS abgebildet, d.h. die Speicherung erfolgt sequentiell. Bei OLAP-Anwendungen werden nur die Daten einzelner Spalten ausgewertet. Bei der Anwendung einer Aggregatsfunktion entsteht bei Tupel-orientierter Speicherung einer hoher I/O-Overhead. Zeilenorientiere Datenbank sind daher nicht für OLAP-Anwendungen geeignet.

OLAP: Spaltenorientierte Datenbank

Unter der Verwendung einer spaltenorientierten Datenbank erfolgt die spaltenweise Partitionierung der Tupel. Die Spalten werden sequentiell gespeichert. Aggregatsfunktionen können direkt auf den Spalten arbeiten, daher werden nur die Daten eingelesen, die für die Auswertung relevant sind.

Kompression

Einige Systeme verbessern die Kompression durch das Sortieren der Spalten (Vertica). In Verbindung mit Bitmap-Indizes kann das Sortieren die Kompression um eine Größenordnung verbessern. Um die Kompression der lexikografischen Ordnung bei der Lauflängenkodierung zu verbessern, werden Spalten kleiner Kardinalität als Suchschlüssel eingesetzt. Angenommen es existiert eine Tabelle mit Name, Geschlecht und Alter, ist es zu empfehlen erst nach Geschlecht, dann nach Alter und zum Schluss nach Namen zu sortieren.
Bei einer spaltenorientierten Datenbank kann jede Spalte separat komprimiert werden. Die Spaltenreihenfolge hat keinen Einfluss auf die Komprimierung, kann aber bei zusammengesetzten Indizes zu besseren Kompressionsraten führen. Beim Umsortieren kann der Vorteil eines Indexes verloren gehen. Umfasst der Index über alle Mitarbeiter zum Beispiel Name und Werk, lässt sich die Kompression steigern, wenn er nach Werk und Name umsortiert wird. Für die Suche nach dem Namen ist der Index nicht mehr zu einsatzfähig.
Die Kompression der Spalten führt zur Reduzierung des Speicherplatzverbrauches auf Kosten der Lesegeschwindigkeit. Sämtliche Daten einer Spalte lassen sich leichter und effizienter lesen, wenn die Daten an der gleichen Stelle abgespeichert sind (zeilenorientierte Datenbank). Mit zunehmender Kompression wird der Zugriff auf die Daten erschwert. Es müssen erst große Datenmengen dekomprimiert werden, um einen Datensatz lesen zu können. Spaltenorientierte Datenbanken werden oft mit zusätzlichen Mechanismen bereichert, um notwendige Zugriffe auf komprimierte Daten zu minimieren.

Partitionierung

Die Partitionierung ermöglicht eine effizientere Organisation der Daten innerhalb einer Datenbank. Die Partitionierung bietet die Aufteilung einer umfangreichen Masterrelation (Master-Tabelle) in einzelne kleinere Teilrelationen (Partitionen) an.

Diese Teilrelationen können unabhängig voneinander gelesen und/oder geschrieben werden. Die Masterrelation R wird dabei vollständig in mehrere paarweise disjunkte Teilrelationen R_1,R_2,…,R_n aufgeteilt.

$$R = R_1 \cup R_2 \cup \ldots \cup R_n \; mit \; i \neq j \; und \; R_i \cap R_j = \emptyset$$

Das Ziel der Partitionierung ist die Steigerung der Performanz in den folgenden Bereichen:

- Management großer Tabellen: Das Hinzufügen und Löschen einzelner Partitionen durch DDL-Befehle ist ohne Änderung des Datenbestands möglich.
- Überspringen von Partitionen: Unterliegt eine Anfrage einer Restriktion, die als Partitionierungskriterium definiert ist, so wird zur Beantwortung der Anfrage nur die Partition ausgewertet und nicht die gesamte Tabelle.
- Ausnutzung paralleler Datenbank- und Systemarchitektur: Die Partitionierung erlaubt weitere systemtechnische Optimierungspotenziale, zum Beispiel können die einzelnen Partitionen auf verschiedene Festplatten verteilt werden. Dadurch kann bei der Auswertung parallel auf diese zugegriffen werden.

Bei der Partitionierung wird zwischen horizontaler und vertikaler Partitionierung unterschieden.

Horizontale Partitionierung

Bei der horizontalen Partitionierung werden die Datensätze einer Tabelle auf paarweise disjunkte Teiltabellen aufgeteilt. Die Attribute aller Partitionen stimmen mit denen in der Master-Tabelle überein. Die Partitionen sind über Restriktionen der Master-Tabelle definiert. Die Master-Tabelle wird über die Vereinigung der Partitionen rekonstruiert.

Vertikale Partitionierung

Bei der vertikalen Partitionierung werden die Attribute von der Master-Relation getrennt. Dadurch können selten referenzierte Attribute separat in einer Teiltabelle gehalten werden. Die einzelnen Partitionen sind über Projektionen der Master-Tabelle definiert. Die Primärschlüssel in den Partitionen sind die gleichen wie in der Master-Tabelle. Die Anzahl der Datensätze in den Partitionen ist gleich groß. Zur Identifizierung der Datensätze dürfen nur Schlüsselattribute verwendet werden. Bei der vertikalen Partitionierung wird die Master-Tabelle durch den Join von Partitionen rekonstruiert.

Partitionierungsmethoden

Range Partitionierung

Die Range Partitionierung ist die Methode, die am meisten verwendet wird. Die Werte oder Wertintervalle eines Tabellenattributes werden als Partitionierungskriterium verwendet. Die Aufteilung der Datensätze auf die Partitionen wird durch Überprüfen und Vergleichen der Attributwerte vorgenommen.

List Partitionierung

Eine diskrete Werteliste dient als Partitionierungskriterium. Anhand dieser Werteliste wird die Partition bestimmt und definiert, auf die die Datensätze verteilt sind.

Hash Partitionierung

Hier erfolgt die Aufteilung unter Verwendung einer Hash-Funktion. Inhaltliche Kriterien und die Semantik spielen hierbei keine Rolle. Die Hash-Funktion wird auf jeden Datensatz der Master-Tabelle angewendet. Das Resultat bestimmt in welche Partition der Datensatz eingeordnet wird.

Kombinierte Range-Hash-Partitionierung

Bei der kombinierten Range-Hash-Partitionierung wird zunächst eine Range-Partitionierung durchgeführt, die dann durch eine Hash-Funktion verfeinert wird.

Kombinierte List-Hash-Partitionierung

Bei der kombinierten List-Hash-Partitionierung wird zunächst eine List-Partitionierung vorgenommen, die dann durch eine Hash-Funktion verfeinert wird.

Anfrageverarbeitung

Die Spaltenorientierung reicht nicht aus, um große Fortschritte bei der Anfragebearbeitung zu erreichen. Es wurden in der Vergangenheit neue Strategien entwickelt, um auf komprimierten Spalten operieren zu können. Die Benutzerschnittstelle SQL bleibt weiterhin erhalten, aber Ausführungsplanung und Optimierung ändern sich. Bei analytischen Anfragen (OLAP) werden oft viele Datensätze benötigt, um Werte zu verdichten. Das sequentielle Lesen der Daten ist aus technischen Gründen meistens schneller, eventuelle Kompression erschwert die wahlfreien Zugriffe. Eine effektive Strategie ist daher das Scannen der Spalten parallel

durchzuführen. Die benötigten Attribute befinden sich somit im Hauptspeicher. Operationen werden nicht als Iteratoren über einzelne Tupel realisiert, sondern als Iterationen über ganze Blöcke von Werten. Weiterhin ändern sich einige Optimierungsziele, wenn ein spaltenorientiertes Datenmodell zugrunde gelegt wird. Bei klassischen RDBMS gilt, dass die Projektionsoperationen, die die benötigten Attribute aus allen verfügbaren herausprojizieren, möglichst früh auszuführen sind. Bei einer spaltenorientierten Datenbank wird späte Materialisierung angestrebt, da die benötigten Attribute erst so spät wie möglich hinzu geladen werden. Nicht nur die Spaltenorientierung sollte solange wie möglich während der Ausführung beibehalten werden, sondern auch die Kompression. Untersuchungen haben belegt, dass eine spaltenorientierte Datenbank, die komprimierte Spalten liest und sofort zu unkomprimierten Relationen zusammensetzt, sich ähnlich wie eine zeilenorientierte Datenbank verhält.

Materialisierung

Frühe Materialisierung

Die Anfrageverarbeitung ist sehr nah an Row Stores orientiert. Aggregatsfunktionen werden nur auf einzelnen Spalten ausgeführt und die Tupelrekonstruktion erfolgt sobald es verwendet wird. Die frühe Materialisierung kommt bei tupel-orientierter Anfrageverarbeitung zum Einsatz.

Späte Materialisierung

Ziel der späten Materialisierung ist es, solange wie möglich auf den Spalten zu arbeiten, dabei kann der Zugriff auf Basistabellen und/oder Zwischenergebnisse mehrfach erfolgen. Die Folge daraus ist, dass der Anfrageplan kein Baum mehr ist, aber die gleichzeitige Bearbeitung auf komprimierten und unkomprimierten Daten ist möglich. Die späte Materialisierung wird für effektive spaltenorientierte Anfrageverarbeitung benötigt.

Quellen

- http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/SpaltenorientierteDatenbank (Letzter Zugriff am: 25.05.14)
- https://cookbook.experiencesaphana.com/crm/what-is-crm-on-hana/technology-innovation/row-vs-column-based/ (Letzter Zugriff am: 26.05.14)

- http://www.gi.de/service/informatiklexikon/detailansicht/article/spaltenorientierte-datenbanken.html (Letzter Zugriff am: 24.05.14)
- https://epic.hpi.uni-potsdam.de/pub/Home/TuKLecture2009/slides_materialization_strategies.pdf (Letzter Zugriff am: 23.05.14)
- http://db.lcs.mit.edu/projects/cstore/abadiicde2007.pdf (Letzter Zugriff am: 26.05.14)
- http://db.csail.mit.edu/projects/cstore/abadi-sigmod08.pdf (Letzter Zugriff am: 22.05.14)