**Kapitel 4: Datentypen**

Das erste Thema, das wir in Bezug auf die Performance - Optimierung von Datenbanken betrachten, sind die unterschiedlichen Datentypen und ihre Effizienzsteigerungen. Bei der Auswahl des korrekten Datentyps gibt es unterschiedliche Faktoren, die vom jeweiligen Typen abhängen. Es gibt aber auch allgemeinere Prinzipien, die auf fast alle Datentypen angewendet werden können.

Bei der Erstellung von Tabellen sollte man folgende Schritte für die Auswahl von Datentypen befolgen. Zunächst muss die allgemeine Klasse der Typen, wie beispielsweise numerisch, Zeichenketten oder zeitbezogen, festgelegt werden. Anschließend sollte der spezifische Typ ausgewählt werden. Für numerische Daten kommen beispielsweise Ganzzahlen wie INT oder Fließkommazahlen wie FLOAT und DOUBLE in Frage. Die spezifischen Typen können dieselbe Art von Daten speichern, unterscheiden sich jedoch im Bereich der Werte, die sie speichern können. Auch sind sie unterschiedlich in der Genauigkeit (precision), die sie erlauben und dem physischen Speicherplatz, den sie entweder auf der Festplatte oder im Arbeitsspeicher benötigen. Einige Datentypen haben auch spezielle Verhaltensweisen und Eigenschaften.

Allgemein gilt für Datentypen, dass kleiner besser ist, weshalb man den kleinstmöglichen Datentypen wählen sollte, den man speichern kann und der die vorhandenen Daten entsprechend repräsentieren kann. Dadurch wird zum einen weniger Speicherplatz (In-Memory und CPU-Cache) in Anspruch genommen, was meistens zu schnelleren Abfragen führt. Zum anderen spricht für die Benutzung von kleinstmöglichen Typen die einfache Typveränderung. Wenn die vorhandenen Daten falsch eingeschätzt wurden und nachträglich ein größerer Datentyp benötigt wird, kann der Typ ohne größere Probleme vergrößert werden. Eine weitere allgemeine Richtlinie ist die Einfachheit von Datentypen. Damit ist beispielsweise gemeint, dass Integer einfach zu verarbeiten ist als Character, weshalb man immer einen Integer wählen sollte, wenn man durch ihn die Daten korrekt abbilden kann. Dies liegt daran, dass weniger CPU-Zyklen benötigt werden, um Operationen auf einfacheren Datentypen zu verarbeiten. Bei dem Beispiel mit Integer und Character liegt dies an den Character Sets und Sortierregeln, die den Character-Vergleich erschweren.

Die letzte allgemeine Regel, die Performancegewinne bringt, ist die Vermeidung von NULL. Viele Tabellen enthalten NULLABLE Spalten, selbst wenn die Anwendung kein NULL (Fehlen eines Wertes) speichern muss, da dies die Standardeinstellung ist. Daher ist es am besten solche Spalten bei der Tabellenerstellung mit dem Identifier NOT NULL zu definieren. Wenn allerdings NULL-Werte gespeichert werden sollen, dann sollte der Identifier nicht genutzt werden. Für MySQL ist es dann schwieriger Abfragen zu optimieren, da dadurch Indizes, Indexstatistiken und Wertevergleiche mehr Speicherplatz benötigen und komplizierter werden. Dies liegt daran, dass indizierte nullable Spalten ein zusätzliches Byte pro Eintrag gebrauchen, was dazu führen kann, dass ein Index mit fester Größe in einen variablen Index umgewandelt wird. Allerdings fällt die Leistungssteigerung, die durch die Änderung von NULL-Spalten in NOT NULL erzielt wird, in der Regel gering aus. Besonders bei Verwendung von Indizes sollte aber darauf geachtet werden.

MySQL unterstützt auch viele Aliase, z.B. INTEGER, BOOL, NUMERIC. Diese Aliase können verwirrend sein, aber sie beeinflussen nicht die Performance. Wenn eine Tabelle mit einem aliasierten Datentyp erstellt wird und die Tabelle mit SHOW CREATE TABLE untersucht wird, fällt auf, dass statt des aliasierten Datentyps der Basistyp angezeigt wird, da der aliasierte Datentyp intern in den Basistyp umgewandelt wurde.

Bevor wir untersuchen, ob die eben beschriebene Prinzipien tatsächlich eine Einfluss auf die Performance haben, müssen die spezielle Verhaltensweisen der bekanntesten Datentypen betrachtet werden.

Für nummerische Datentypen gibt es die Wahl zwischen Ganzzahlen und Fließkommazahlen. Die spezifischen Typen unterscheiden sich nur in der Anzahl der Bits, die sie speichern können. SMALLINT kann 16 Bits speichern, während INT 32 und BIGINT 64 Bits speichern kann. Dementsprechend verändert sich auch der mögliche Wertebereiche der Zahlen, die durch den Speicherplatz abgedeckt sind. Mit den optionalen UNSIGNED - Attribute können keine negative Werte gespeichert werden können, dafür verdoppelt sich aber die obere Grenz der Positiven. Zeitgleich bleiben der Speicherplatz und die Leistung gleich. (TODO: Daniel Bereich hier). Eine Breitenangabe wie INT(11) beeinflusst nur die Anzeige und nicht den Wertebereich oder die Speicheranforderungen.

Ein spezifischer Typ für eine Festkommazahl ist DECIMAL, die auch für die Speicherung von Ganzzahlen eignet sind. Außerdem kann man bei einer Festkommazahl auch die Genauigkeit angeben, da die maximale Anzahl der Ziffern vor und nach dem Dezimalpunkt definiert werden. DECIMAL(18, 9) beispielsweise speichert neun Ziffern vor und nach dem Dezimalpunkt und benötigt dafür 9 Bytes Speicherplatz. DECIMAL speichert Zahlen in einer binären Zeichenkette (binary string) mit neun Ziffern pro vier Bytes und unterstützt bis zu 65 Ziffern insgesamt.

Zu den Fließkommazahlen gehören die FLOAT- und DOUBLE-Typen, die die standardmäßige Gleitkomma-Arithmetik verwenden und für ungefähre Berechnungen optimiert sind. FLOAT benötigt 4 Bytes, während DOUBLE 8 Bytes Speicherplatz beansprucht und eine höhere Präzision sowie einen größeren Wertebereich bietet. Die Gleitkomma-Arithmetik ist aufgrund der nativen Verarbeitung durch die CPU deutlich schneller als die präzise Berechnung mit DECIMAL, bringt jedoch einen gewissen Präzisionsverlust mit sich. Alternativ kann auch BIGINT genutzt werden, um sowohl die Ungenauigkeit von Gleitkomma-Speicherungen als auch die höheren Kosten der DECIMAL-Arithmetik zu vermeiden.

Die beiden Haupttypen für Zeichenketten sind VARCHAR und CHAR. VARCHAR speichert die Zeichenfolgen mit variabler Länge und benötigt daher weniger Speicherplatz als Typen mit fester Länge, da nur so viel Platz verwendet wird, wie tatsächlich benötigt wird. Zusätzlich werden ein oder zwei Bytes für die Speicherung der Länge der Zeichenfolge verwendet (1 Byte für < 255 Bytes Zeichenfolge). Durch diese effiziente Speichernutzung ist VARCHAR der am häufigsten verwendete Datentyp für Zeichenketten, aber es hat auch Nachteile, da Aktualisierungen an den Werten zu wachsenden Zeilen und damit auch zusätzliche Verarbeitung der Speicher - Engine erfordern kann. Und obwohl die Speicherung von „hello“ in VARCHAR(5) oder VARCHAR(200) gleich viel Speicherplatz benötigt, kann es trotzdem ineffizienter für Sortierungen oder Operationen auf temporären Tabellen sein. Deshalb sollte trotzdem immer so viel Platz reserviert werden, wie tatsächlich benötigt wird.

CHAR hingegen hat eine feste Länge und MySQL reserviert immer auch den nicht gebrauchten Platz für die angegebene Anzahl an Zeichen. Daher ist CHAR ideal für sehr kurze Strings oder Werte, die alle nahezu gleich lang sind, da VARCHAR(1) zwei Bytes aufgrund des Längen - Bytes benötigt, CHAR (1) hingegen auch. Außerdem ändert sich bei CHAR die Speicherstruktur bei Aktualisierungen nicht, weshalb dieser Datentyp besser geeignet ist, wenn die Daten häufig verändert werden. Hingegen VARCHAR eignet sich besonders, wenn die maximale Länge einer Spalte deutlich größer ist als die durchschnittliche Länge der gespeicherten Werte.

DATETIME und TIMESTAMP können dieselbe Art von Daten speichern und beide haben dabei eine Genauigkeit von einer Sekunde. TIMESTAMP benötigt aber nur halb so viel Speicherplatz, ist zeitzonenbewusst und verfügt über spezielle Auto-Update-Funktionen. Allerdings hat TIMESTAMP einen viel kleineren Bereich an erlaubten Werten und manchmal können seine speziellen Fähigkeiten ein Nachteil sein.

**Analyse der Benchmarks**

Den ersten Leitsatz, den wir untersuchen ist, dass am besten die Spalten als NOT NULL deklariert werden, wenn dies möglich ist. Für den Nachweis haben wir einmal eine Kundentabelle (\ref{TODO: Kundentabelle}) wie in dem Beispiel von oben erstellt und verglichen mit einer Tabelle, bei der alle Spalten als NOT NULL deklariert werden. Dazu haben wir passende Testdaten erstellt und kann mit der Variable „every“ frei wählen, für jeder wievielte Namen „NULL“ eingefügt werden soll. In der erstellten Grafik kann mir dann die Resultate für die Select-Befehle sehen, die sowohl einfache WHERE - Klauseln als auch Count - und Group By - Befehle enthalten. Anhand der Grafik für die Abfrageergebnisse (Read) kann man sehen, dass alle 3 Skripte sehr nah aneinander liegen, dennoch ist die Query mit der Tabelle mit den „NOT NULL“ Attributen am schnellsten.

Für den Vergleich, um zu zeigen, dass simplere Datentypen besser sind, haben wir wieder die Kundentabelle benutzt (\ref{TODO: Kundentabelle}). Diesmal haben wir einmal als Schlüsselattribut der Tabelle den Datentypen „int“ und einmal den Datentypen „char“ verwendet. An den Ergebnissen fällt auch, dass die Schreibbefehle bei beiden Skripten etwa gleich schnell sind, aber bei den Abfragen gibt es deutliche Unterschiede. Hier fällt nämlich auch, dass bei Wertevergleichen „int“ deutlich schneller (etwa 50%) ist als „char“, bei der Sortierung ist die Reihenfolge gleich, die Abstände sind aber deutlich geringer.

Als letztes wollten wir unterschiedliche Datentypen vergleichen. Dazu haben wir wieder die Kundentabelle (\ref{TODO: Kundentabelle}) benutzt und diesmal für den Datentypen unterschiedliche Numerische oder Zeichenkettentypen gewählt. Bei dem Vergleich der nummerischen Typen fällt auch, dass Decimal mit deutlichem Abstand am Langsten ist. Danach kommt wie schon vermutet mit BIGINT der nächstgrößere Datentyp. Auffällig ist, dass der Unterschied zwischen INT, MEDIUMINT und SMALLINT kleiner ist als erwartet. Dies liegt aber auch daran, dass wir nur eine Tabelle mit wenigen tausende Datensätzen vergleichen. Wenn man in der Produktion mehrere Tabellen mit Hunderttausenden oder Millionen von Datensätzen hat, ist anzunehmen, dass die Abstände zwischen den Typen größer wäre als in unserem Vergleich.

Beim Vergleich zwischen den beiden Zeichenkettentypen „char“ und „varchar“ ist unabhängig von der Länge zu erkennen, dass „varchar“ effizienter ist als „char“. Bei dem ersten Vergleich wurde immer die komplette Länge von 4 bzw. 64 Stellen ausgenutzt. Wenn wir aber einen Varchar mit der Länge von 255 haben und nur 60 Stellen einfügen, dann fällt auf, dass der Abstand zwischen den beiden noch größer wird, da „char“ die restlichen 195 Zeichen durch leere Character auffüllen muss.

TODO: Geh auf die Write Performance bei Char/Varchar ein.