**Kapitel 4: Datentypen**

**Einfluss der Auswahl von Datentypen**

Bei der Auswahl von Datentypen gibt es viele Faktoren, die Einfluss auf die Performance haben können. Einige von diesen Faktoren sind abhängig vom jeweils gewählten Datentyp, andere sind dafür genereller anwendbar. Zu den allgemeinen Einflussfaktoren kommen wir zunächst und anschließend betrachten wir die einzelnen Typen wie Ganzzahlen oder Strings im Detail an.

Allgemein gilt bei 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 weniger Speicherplatz (In-Memory und CPU-Cache) in Anspruch genommen, weshalb die Abfragen meistens schneller sind. Ein weiterer Vorteil, der für die Benutzung von kleinstmöglichen Typen spricht, ist die einfache Typveränderung, wenn man die vorhandenen Daten falsch eingeschätzt hat und nachträglich ein größerer Datentyp benötigt wird. Ein weiterer allgemeiner Leitsatz ist, dass ein einfacherer Datentyp gut ist, denn es werden weniger CPU-Zyklen benötigt, um Operationen auf einfacheren Datentypen zu verarbeiten. Beispielweise ist Integer einfacher zu verarbeiten als Character, da Character Sets und Sortierregeln den Character-Vergleich erschweren.

Generell bringt es auch Performancegewinne, wenn man NULL vermeidet, wenn es möglich ist. Viele Tabellen enthalten NULLABLE Spalten, selbst wenn die Anwendung kein NULL (Fehlen eines Wertes) speichern muss, da dies die Standardeinstellung ist. Daher ist am besten solche Spalten bei der Tabellenerstellung mit dem Identifier NOT NULL zu definieren. Wenn allerdings NULL-Werte gespeichert werden soll, dann sollte der Identifier nicht genutzt werden und für MySQL ist es dann schwieriger Abfragen zu optimieren, da durch Indizes, Indexstatistiken und Wertevergleiche komplizierter werden. Dadurch benötigen sie auch mehr Speicherplatz und erfordern eine spezielle Verarbeitung innerhalb von MySQL. Das liegt daran, dass indizierte nullable Spalten ein zusätzliches Byte pro Eintrag gebrauchen und das kann dazu führen, dass ein Index mit fester Größe in einen variablen Index umgewandelt wird. Die Leistungsverbesserung (TODO(Daniel): check with benchmark), die durch die Änderung von NULL-Spalten in NOT NULL erzielt wird, ist in der Regel gering, aber bei der Verwendung von Indizes sollte besonders darauf geachtet werden.

Wenn man bei der Erstellung von Tabellen nicht sicher ist, welchen Datentypen für die Spalten geeignet sind, bieten sich folgende Schritte für die Auswahl von Datentypen an. Als Erstes sollte die allgemeine Klasse von Typen, z.B. numerisch, Zeichenketten oder zeitbezogen, bestimmt werden. Daraufhin sollte der spezifische Typ gewählt werden. Viele 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.

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. (TODO (Daniel): add rest oft ext here).

MySQL unterstützt auch viele Aliase, z.B. INTEGER, BOOL, NUMERIC. Diese Aliase können verwirrend sein, sie beeinflussen aber nicht die Performance. Erkennen kann das an, wenn man eine Tabelle mit einem aliasierten Datentyp erstellt und dann mit SHOW CREATE TABLE die Tabelle untersucht. Denn tatsächlich würde man anstelle des aliasierten Datentypen den Basistyp angezeigt bekommen, da intern der aliasierte Datentyp gegen diesen umgewandelt wird.

**Ganze Zahlen**

Es gibt zwei Arten von Zahlen: ganze Zahlen und reelle Zahlen. Zum Speichern ganzer Zahlen bietet MySQL die Integer-Typen TINYINT, SMALLINT, MEDIUMINT, INT und BIGINT, die 8, 16, 24, 32 bzw. 64 Bits Speicherplatz benötigen. Sie können Werte von −2(N−1)−2(N−1) bis 2(N−1)−12(N−1)−1 speichern, wobei N die Anzahl der Bits ist.

Mit dem optionalen UNSIGNED-Attribut können negative Werte ausgeschlossen und die obere Grenze der positiven Werte verdoppelt werden (z. B. bei TINYINT von [0;255][0;255] statt [−128;127][−128;127]). Signed- und unsigned-Typen benötigen denselben Speicherplatz und sind in der Leistung gleich.

Berechnungen mit Integer-Werten verwenden meist den 64-Bit-Typ BIGINT, auch auf 32-Bit-Systemen, außer bei Aggregatfunktionen, die DECIMAL oder DOUBLE nutzen. Die Breitenangabe wie INT(11) beeinflusst nur die Anzeige, nicht den Wertebereich oder die Speicheranforderungen.

**Reelle Zahlen**

Reelle Zahlen sind Zahlen mit einem Bruchteil. Für ihre Speicherung bietet MySQL verschiedene Typen an, die je nach Anwendungsfall unterschiedliche Eigenschaften besitzen. Der DECIMAL-Typ eignet sich für die exakte Speicherung von Zahlen, einschließlich solcher, die für BIGINT zu groß sind. Zudem erlaubt er die Angabe der Genauigkeit, indem die maximale Anzahl der Ziffern vor und nach dem Dezimalpunkt definiert wird. DECIMAL(18, 9) beispielsweise speichert neun Ziffern vor und nach dem Dezimalpunkt und benötigt dafür 9 Bytes Speicherplatz.

FLOAT- und DOUBLE-Typen hingegen verwenden standardmäßige Gleitkomma-Arithmetik (floating-point math) und sind für ungefähre Berechnungen ausgelegt. FLOAT benötigt 4 Bytes, DOUBLE 8 Bytes Speicherplatz und bietet höhere Präzision sowie einen größeren Wertebereich als FLOAT. 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.

DECIMAL speichert Zahlen in einer binären Zeichenkette (binary string) mit neun Ziffern pro vier Bytes und unterstützt bis zu 65 Ziffern insgesamt. Gleitkomma-Typen benötigen in der Regel weniger Speicherplatz, um denselben Wertebereich abzudecken. MySQL verwendet bei internen Berechnungen mit Gleitkomma-Typen den DOUBLE-Typ.

Für Finanzdaten empfiehlt sich DECIMAL, um exakte Ergebnisse zu gewährleisten. Alternativ kann auch BIGINTgenutzt werden, um sowohl die Ungenauigkeit von Gleitkomma-Speicherungen als auch die höheren Kosten der DECIMAL-Arithmetik zu vermeiden. Wie bei ganzen Zahlen wählt man auch hier den passenden Speicher-Typ je nach Anforderungen an Präzision und Performance.

**String-Typen**

Seit MySQL 4.1 kann jede String-Spalte ihre eigene Zeichencodierung (character set) und ein zugehöriges Sortierungsregelset (collation) verwenden. Die beiden Haupttypen für Strings sind VARCHAR und CHAR, die Zeichenfolgen speichern, wobei die Speicherung im Detail von der verwendeten Speicher-Engine abhängt.

VARCHAR speichert Zeichenfolgen mit variabler Länge und ist der am häufigsten verwendete String-Datentyp. Er benötigt weniger Speicherplatz als Typen mit fester Länge, da nur so viel Platz verwendet wird, wie tatsächlich benötigt wird. Zusätzlich werden 1 oder 2 Bytes zur Speicherung der Länge der Zeichenfolge verwendet (1 Byte für Werte unter 255 Bytes). Da VARCHAR Werte variabler Länge speichert, können Aktualisierungen dazu führen, dass Zeilen wachsen und je nach Speicher-Engine zusätzliche Verarbeitung erfordern. VARCHAR eignet sich besonders, wenn die maximale Länge einer Spalte deutlich größer ist als die durchschnittliche Länge der gespeicherten Werte.

CHAR hingegen hat eine feste Länge, wobei MySQL immer genügend Platz für die angegebene Anzahl von Zeichen reserviert. Dies macht CHAR ideal für sehr kurze Strings oder für Werte, die alle nahezu gleich lang sind. Ein Vorteil von CHAR ist, dass er Fragmentierung vermeidet, was ihn besser geeignet macht für Daten, die häufig geändert werden. Außerdem ist CHAR bei sehr kurzen Strings effizienter als VARCHAR; beispielsweise benötigt CHAR(1) nur ein Byte, während VARCHAR(1) aufgrund des Längen-Bytes zwei Bytes benötigt.

Die verwandten Typen BINARY und VARBINARY speichern binäre Strings anstelle von Zeichen. Sie sind nützlich für die Speicherung von Binärdaten und führen byteweise Vergleiche durch, die schneller und einfacher als Zeichenvergleiche sein können. BINARY füllt Werte mit Null-Bytes (\0) auf, und diese Auffüllung bleibt beim Abruf erhalten. Daher sollte Vorsicht geboten sein, wenn der Wert unverändert bleiben muss.

Es ist wichtig zu beachten, dass VARCHAR unabhängig von der definierten maximalen Länge denselben Speicherplatz für einen Wert benötigt, z. B. speichert 'hello' in VARCHAR(5) und VARCHAR(200) gleich viel. Dies kann jedoch ineffizient für Sortierungen oder Operationen mit temporären Tabellen sein. Es sollte immer nur so viel Platz reserviert werden, wie tatsächlich benötigt wird.

**BLOB -und TEXT-Typen**

BLOB- und TEXT-Typen sind String-Datentypen, die für die Speicherung großer Datenmengen entwickelt wurden. Sie sind in zwei Familien unterteilt: BLOB-Typen für binäre Daten und TEXT-Typen für Zeichen-Strings. Zu den BLOB-Typen gehören TINYBLOB, SMALLBLOB, BLOB, MEDIUMBLOB und LONGBLOB, während TEXT-Typen als TINYTEXT, SMALLTEXT, TEXT, MEDIUMTEXT und LONGTEXT verfügbar sind. MySQL behandelt jeden BLOB- und TEXT-Wert als ein Objekt mit eigener Identität.

Der Hauptunterschied zwischen den beiden Typen besteht darin, dass BLOB binäre Daten ohne Sortierung (collation) oder Zeichencodierung (character set) speichert, während TEXT eine Zeichencodierung und Sortierung unterstützt. Beim Sortieren von BLOB- oder TEXT-Spalten berücksichtigt MySQL standardmäßig nur die ersten max\_sort\_lengthZeichen. Dies kann durch Anpassung der Servervariablen oder Verwendung von ORDER BY SUBSTRING(column, length) beeinflusst werden.

Da Abfragen mit BLOB- oder TEXT-Spalten oft temporäre Tabellen erfordern, können sie die Leistung erheblich beeinträchtigen. Es wird empfohlen, diese Typen nur dann zu verwenden, wenn es unbedingt notwendig ist. Falls unvermeidlich, kann der Einsatz von SUBSTRING(column, length) hilfreich sein, um die Werte in Zeichen-Strings zu konvertieren, sodass temporäre Tabellen im Speicher genutzt werden können. Dabei sollte darauf geachtet werden, dass das Substring kurz genug ist, um die Grenzen von max\_heap\_table\_size oder tmp\_table\_size nicht zu überschreiten.

**ENUM**

ENUM-Typen können manchmal als Alternative zu herkömmlichen String-Typen verwendet werden. Eine ENUM-Spalte speichert eine vordefinierte Menge eindeutiger Werte und ist dabei äußerst platzsparend, da sie intern als Ganzzahlen (integers) repräsentiert werden, die ihre Position in der Definitionsliste widerspiegeln. Je nach Anzahl der definierten Werte werden die Daten in ein oder zwei Bytes gespeichert. Die Zuordnung von Zahlen zu Strings erfolgt über eine "Nachschlagetabelle" (lookup table), die in der .frm-Datei der Tabelle gespeichert ist.

Ein Nachteil dieser Dualität ist, dass die interne Speicherung als Integer zu Verwirrung führen kann, insbesondere wenn numerische Strings wie '1', '2', '3' in der ENUM-Definition verwendet werden. Solche Konstruktionen sollten vermieden werden. ENUM-Werte werden nach ihren internen Integer-Werten sortiert und nicht nach den Strings selbst. Daher sollten Sie die ENUM-Mitglieder in der gewünschten Sortierreihenfolge definieren oder in Abfragen die Funktion FIELD()verwenden, um eine benutzerdefinierte Sortierung zu erzwingen. Letzteres verhindert allerdings, dass MySQL Indexe für die Sortierung nutzen kann.

Der größte Nachteil von ENUM besteht darin, dass die Liste der erlaubten Strings fest ist. Änderungen wie das Hinzufügen oder Entfernen von Werten erfordern ein ALTER TABLE, was aufwendig sein kann. Zudem kann das Verbinden von ENUM-Spalten mit CHAR- oder VARCHAR-Spalten langsamer sein als der direkte Vergleich zwischen CHAR- oder VARCHAR-Spalten.

Eine Analyse zeigt jedoch, dass das Konvertieren von VARCHAR-Spalten in ENUM Vorteile bietet: Joins zwischen ENUM-Spalten sind schneller, die Tabelle wird kleiner (etwa ein Drittel laut SHOW TABLE STATUS) und die Primärschlüssel sowie die Indizes benötigen weniger Speicherplatz. Diese Vorteile treten besonders dann hervor, wenn keine regelmäßigen Verbindungen mit VARCHAR-Spalten notwendig sind. Andernfalls ist es eine gängige Praxis, für solche Zwecke Nachschlagetabellen mit ganzzahligen Primärschlüsseln zu verwenden, um stringbasierte Joins zu vermeiden.

**Datums- und Zeit-Typen**

**BIT-Typ**

Der BIT-Typ wird nicht empfohlen, um Wahr/Falsch-Werte (true/false values) in einer Spalte zu speichern. Stattdessen ist es sinnvoller, für solche Szenarien MySQLs nativen SET-Datentyp zu nutzen, der mehrere Wahr/Falsch-Werte in einer einzigen Spalte kombinieren kann. Allerdings hat der SET-Typ Einschränkungen: Änderungen an der Spaltendefinition sind nur über ALTER TABLE möglich, was aufwendig ist und Indizes für diese Spalten nicht unterstützt werden.

Alternativ können Integer-Spalten für bitweise Operationen (bitwise operations) verwendet werden, wenn ein gutes Verständnis von bitweiser Manipulation vorliegt. Diese Methode erlaubt flexible Änderungen durch Anpassung der „Aufzählung“ (enumeration), erfordert jedoch sorgfältige Planung und präzise Implementierung.