WHITEPAPER:

SentryOne

Wie Ihre Datentyp-Wahl die Leistung von SQL Server-Datenbanken beeinflussen kann

EINFÜHRUNG

Warum Ihre Datentyp-Wahl wichtig ist

Wussten Sie, dass eine schlechte Datentyp-Wahl signifikante Auswirkungen auf das Design und die Leistung Ihrer Datenbanken haben kann? Entwickler und DBAs können die Datenbank-Leistung verbessern, wenn sie die von SQL Server unterstützten Datentypen und die Folgen der Auswahl anderer Datentypen verstehen. Es ist eine bewährte Praxis, die richtigen Datentypen zu erfassen, indem man für das Unternehmen relevante Fragen stellt und die Datentypen bestimmt, die am besten für die Anforderungen des Unternehmens und der Anwendung geeignet sind.

Die richtige Erfassung von Datentypen kann zu enormen Speichereinsparungen und folgerichtig zu einer besseren Datenbank-Leistung führen. Sie kann dabei helfen, andere Leistungsprobleme wie die übermäßige B-Baum-Seitenaufteilung zu mindern. Außerdem hat sie direkte Auswirkungen auf die T-SQL-Leistung in Form von impliziten Konvertierungen.

Es folgen einige Richtlinien für die richtige Erfassung Ihrer Datentypen und die Steigerung Ihrer Datenbank-Leistung.

Über den Autor

Andy Yun ist ein SentryOne Senior Solutions Engineer und ein Microsoft MVP. Er arbeitet seit 15 Jahren sowohl als Datenbank-Entwickler als auch als Administrator mit SQL Server. Er ist bestrebt, seine Kenntnisse der SQL Server Internals und seine umfassende Erfahrung in stark transaktionsintensiven Umfeldern einzusetzen, um T-SQL agiler und leistungsstärker zu machen.

Andy hat eine große Leidenschaft dafür, sein Wissen mit anderen zu teilen, und tritt oft bei User Groups, SQLSaturdays und am PASS-Summit auf. Andy ist ein Mitbegründer der Chicago SQL Association, Co-Ortsgruppenleiter der Chicago Suburban User Group und Teil des Chicago SQLSaturday Organisationskomitees. Dieses Whitepaper wurde anhand eines Vortrags erstellt, den Andy bei SQLBits XVI gehalten hat.

Andy Yun
SENIOR SOLUTIONS ENGINEER

SentryOne

Wie Ihre Datentyp-Wahl die Leistung von SQL Server-Datenbanken beeinflussen kann

VON ANDY YUN

Datentypen mit fester Breite und Speicheranforderungen	
TINYINT	1 Byte
INT	4 Bytes
DATETIME	8 Bytes
CHAR	n Bytes
DECIMAL	5-17 Bytes
UNIQUEIDENTIFIER	16 Bytes
SMALLINT	2 Bytes
BIGINT	8 Bytes
SMALLDATETIME	4 Bytes
NCHAR	(n * 2) Bytes
FLOAT	4*8 Bytes

Datentypen mit variabler Breite und Speicheranforderungen	
VARCHAR	n + 2 Bytes
NVARCHAR	(n * 2) + 2 Bytes

Diese Diagramme zeigen eine hilfreiche Übersicht der Speicheranforderungen von Datentypen mit fester Breite und variabler Länge.

Das Verständnis von SQL Server Internals hilft Entwicklern und DBAs, effizienteren Code zu schreiben

Betrachten wir zunächst einige zentrale Punkte hinsichtlich der SQL Server-Datentypen.

1. **In SQL Server verfügen Datentypen mit fester und variabler Länge über unterschiedliche Speicheranforderungen.**

   - Datentypen mit fester Breite erfordern immer dieselbe Menge an Speicherplatz, unabhängig vom Wert, der in diesen Spalten oder Variablen gespeichert ist. Das trifft sogar zu, wenn der Wert NULL ist.
   - Datentypen mit variabler Länge erfordern zwei zusätzliche Bytes an Speicherplatz. Folglich erfordert CHAR(1) hinsichtlich des Speichers nur 1 Byte, während VARCHAR(1) 2 oder 3 Bytes an Speicher benötigt.
   - NCHAR und NVARCHAR speichern Unicode-Daten, die 2 Bytes pro Zeichen benötigen. Werten Sie aus, ob Unicode benötigt wird oder nicht.

2. Das FIXVAR-Format ist das tatsächliche physische Format, das SQL Server verwendet, um Datensätze zu speichern.

Tag A	Tag B	Fsize	Fdata	Ncol	Nullbits	VarCount	VarOffset	VarData
1 Byte	1 Byte	2 Bytes	Fsize - 4	2 Bytes	Höchstgrenze (Ncol/ 8)	2 Bytes	2 x VarCount	n Bytes

Tag A und B	Status-Bits A und B
Fsize	Gesamtgröße der festen Länge
Fdata	Tatsächliche Daten mit fester Länge
Ncol	Gesamtzahl der Spalten
Nullbits	NULL-Bitmap (1 Bit pro Spalte in der Tabelle)
VarCount	Gesamtzahl der Spalten mit variabler Länge
VarOffset	Versatzreihe der variablen Spalte
VarData	Tatsächliche Daten mit variabler Länge

3. Bei Spalten mit fester Breite, kann die Speicher-Engine auf effiziente Weise spezifische Datenspalten in einem Datensatz finden, da die Spaltenbreiten konstant sind.

Wenn eine Tabelle vollständig aus Datentypen mit fester Breite besteht, erfordert ein Datensatz in dieser Tabelle immer eine bestimmte Menge an Speicherplatz.

Mitarbeiter-ID	UNIQUEIDENTIFIER
Geschlecht	INT
Sozialversicherungsnummer	NCHAR(50)
Manager-ID	UNIQUEIDENTIFIER
Einstellungsdatum	DATETIME

F-Data	Feste Daten				
144 Bytes	Mitarbeiter-ID	Geschlecht	SVN	Manager-ID	Einstellungsdatum
	16 Bytes	4 Bytes	100 Bytes	16 Bytes	8 Bytes

4. Bei Spalten mit variabler Breite, muss die Speicher-Engine zusätzliche Arbeit leisten, um spezifische Datenspalten in einem Datensatz zu finden, da die Spaltenbreite von Datensatz zu Datensatz unterschiedlich ist.

Bezeichnung	NVARCHAR(255)
E-Mail-Adresse	NVARCHAR(255)

VarCount	VarOffset *		Daten mit variabler Breite	
2	0x1C	0x2E	Entwickler	isabella@countingbytes.com
2 Bytes	2 Bytes	2 Bytes	18 Bytes	52 Bytes

VarCount	VarOffset *		Daten mit variabler Breite	
2	0x1C	0x48	Datenbank-Administrator	james@countingbytes.com
2 Bytes	2 Bytes	2 Bytes	44 Bytes	46 Bytes

**** = Hex-Adresswert***

SQL Server verwendet Hinweise, um ihr die Navigation und Suche nach Daten zu erleichtern. Diese Hinweise werden in den zusätzlichen 2 Bytes gespeichert, die alle Spalten mit variabler Breite erfordern.

5. **Eine bewährte Praxis bei der richtigen Erfassung von Datentypen ist es, zu analysieren, ob ein Datentyp der richtige Container für den Wert ist, der gespeichert werden soll.**

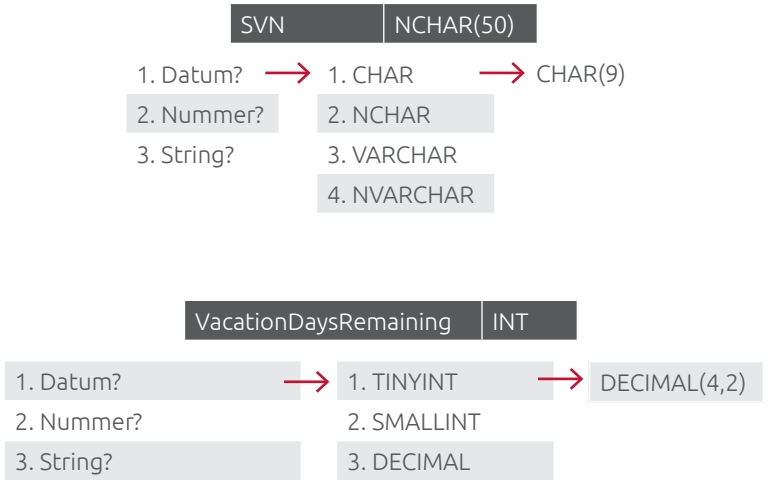
Es ist wichtig, für das Unternehmen relevante Fragen und Fragen über die zukünftige Ausrichtung des Unternehmens zu stellen. Das Ziel der richtigen Erfassung ist es, zu bestimmen, ob ein Datentyp angemessen für die Anwendung oder das Unternehmen ist.

Datentyp	Speicher	Wertbereich
TINYINT	1 Byte	0 bis 255
SMALLINT	2 Bytes	-31.768 bis 32.767
INT	4 Bytes	-2.147.483.648 bis 2.147.483.647
BIGINT	8 Bytes	-9.223.372.036.854.775.808 bis 9.223.372.036.854.755.807
DATUM	3 Bytes	0001-01-01 bis 9999-12-31
SMALLDATETIME	4 Bytes	1900-01-01 00:00:00 bis 2079-06-06 23:59:59
DATETIME	8 Bytes	1753-01-01 00:00:00.0000 bis 9999-12-31 23:59:59.997

Diese Tabelle fasst die Speicheranforderungen und Wertbereiche für diverse richtig erfasste Datentypen zusammen.

Stellen Sie sich für das erste Beispiel rechts vor, dass Sie eine Anwendung aus den USA sind, die Sozialversicherungsnummern speichert. Wenn Sie von links nach rechts gehen, welchen allgemeinen Datentyp sollten Sie verwenden? Wählen Sie einen String; welche Art von String? Wenn Sie möchten, dass Ihre Anwendung die Striche auf der Anzeigeebene wieder einfügt, könnten Sie CHAR(9) verwenden.

Im zweiten Beispiel müssen Sie die verbleibenden Urlaubstage eines Angestellten speichern. Ein INT-Datentyp ist viel zu groß (außer Sie sind ein äußerst glücklicher Mitarbeiter!). Wenn Sie von links nach rechts gehen, welchen allgemeinen Datentyp sollten Sie verwenden? Wählen Sie einen numerischen Wert aus; welche Art von nummerischem Wert? Weist Ihr Unternehmen Urlaubstage nur in Inkrementen von einzelnen Tagen zu? Kann man einen halben Tag frei nehmen? Was ist, wenn Sie für ein Unternehmen arbeiten, bei dem Sie 15 Urlaubstage im Jahr erhalten und 1,25 Tage pro Monat Ihrer Beschäftigung ansammeln? Aus diesem Grund ist es wichtig, für das Unternehmen relevante Fragen zu stellen – denn die Antwort ist „das kommt darauf an!“.



Zwei Beispiele von richtig erfassten Datentypen für diverse Situationen.

SentryOne

6. Die richtige Erfassung einer Tabelle kann in enormen Speichereinsparungen resultieren.

Im unteren Beispiel beträgt die minimale Datensatz-Größe 1.304 Bytes. Vergleichen Sie das mit einer richtig erfassten Mitarbeiter-Tabelle. Ein einzelner Datensatz erfordert ein Minimum von nur 104 Bytes.

```
CREATE TABLE dbo.Employee_Large (
        EmployeeID              UNIQUEIDENTIFIER PRIMARY KEY CLUSTERED,
        FirstName               NCHAR(255),
        LastName                NCHAR(255),
        Gender                  INT,
        SocialSecurityNumber    NCHAR(50),
        Title                   NVARCHAR(255),
        ManagerID               UNIQUEIDENTIFIER,
        DateHired               DATETIME,
        Salary                  FLOAT,
        EmailAddress            NVARCHAR(255),
        HomeOfficeLocationCode  NVARCHAR(4000),
        VacationDaysRemaining   INT,
        RecordCreatedDate       DATETIME,
        RecordCreatedBy         UNIQUEIDENTIFIER,
        RecordLastModifiedDate  DATETIME,
        RecordLastModifiedBy    UNIQUEIDENTIFIER
)
```

Große Mitarbeiter-Tabelle. Mindestgröße von 1.304 Bytes.

```
CREATE TABLE dbo.Employee_Narrow (
        EmployeeID              INT IDENTITY(1, 1) PRIMARY KEY CLUSTERED,
        FirstName               VARCHAR(100),
        LastName                VARCHAR(100),
        Gender                  CHAR(1),
        SocialSecurityNumber    CHAR(9),
        Title                   VARCHAR(40),
        ManagerID               INT,
        DateHired               DATE,
        Salary                  DECIMAL(10, 2),
        EmailAddress            VARCHAR(100),
        HomeOfficeLocationCode  TINYINT,
        VacationDaysRemaining   DECIMAL(4, 2),
        RecordCreatedDate       SMALLDATETIME,
        RecordCreatedBy         INT,
        RecordLastModifiedDate  SMALLDATETIME,
        RecordLastModifiedBy    INT
)
```

Richtig erfasste Mitarbeiter-Tabelle. Mindestgröße von 104 Bytes.

7. SQL Server verwendet seinen eigenen Container, der als Datenseite bezeichnet wird, um Datensätze zu speichern.

Es gibt drei Arten von Datenseiten: 1) In-Zeilen-Datenseiten, 2) zeilenübergreifende Datenseiten und 3) LOB-Datenseiten. Alle Seiten verfügen über eine feste Größe von 8 KB oder 8.192 Bytes.

Datenseiten bestehen aus drei Komponenten: 1) einem 96-Byte Seitenheader, 2) Datensätzen und 3) einem Datensatz- oder Slot-Array, das 2 Bytes pro Datensatz verbraucht.

Entwickler und DBAs müssen sich über folgendes im Klaren sein:

- Das 8 KB Limit wirkt sich darauf aus, was in eine Datenseite passt. Normalerweise werden Datensätze in einer einzelnen In-Zeilen-Datenseite gespeichert. Wenn ein Datensatz allerdings mehr als 8 KB beträgt, stellt das ein Problem dar. Ein einzelner Datensatz kann sich nicht auf mehrere In-Zeilen-Datenseiten erstrecken. Folglich müssen in diesem Fall zeilenübergreifende oder LOB-Datenseiten verwendet werden.
- VARCHAR(8000) und NVARCHAR(4000) verwenden zeilenübergreifende Datenseiten.
- VARCHAR(MAX) und NVARCHAR(MAX) verwenden LOB-Datenseiten.

Vergessen Sie nicht, dass SQL Server innerhalb der 8 KB-Datenseitenbeschränkung arbeitet, da dies signifikanten Einfluss auf die Leistungsanpassung hat.

Eine schlechte Datentyp-Auswahl kann zu einer Unmenge unbeabsichtigter Leistungsauswirkungen führen

Die Transaktionsprotokollierung ist eine Aufgabe, die SQL Server erledigen muss.

Bei der Leistungsanpassung ist es wichtig, alle I/O-Operationen zu beachten, die SQL Server ausführen muss. Die Transaktionsprotokollierung ist ein Faktor, der nur selten die Aufmerksamkeit bekommt, die er verdient.

SQL Server schreibt jedes Mal im Transaktionsprotokoll, wenn ein INSERT-, UPDATE- oder DELETE-Statement erfolgt. Abfragen können erst vollständig eingehen und abgeschlossen werden, wenn die Schreibaktivität im Transaktionsprotokoll abgeschlossen ist. Die Verwendung unangemessen großer Datentypen führt dazu, dass bei ansonsten identischen DML-Statements mehr Transaktionsprotokoll-Daten verfasst werden müssen.

Diese Beeinträchtigung der Leistung wirkt sich auch auf andere Bereiche aus. Das Volumen des Transaktionsprotokolls wirkt sich auch auf die Backup-/Wiederherstellungs-, Replikations-, Protokollversand-, Spiegelungs- und Verfügbarkeitsgruppen aus. Wenn Sie sich die Zeit nehmen, Ihre Datentypen richtig zu erfassen, reduzieren Sie auch die Größe des Transaktionsprotokolls.

Unnötige Seitenaufteilung ist kostspielig.

Wenn eine SQL Server-Tabelle über einen geclusterten Index verfügt, können unnötige Seitenaufteilungen entstehen.

Eine Seitenaufteilung tritt auf, wenn ein neuer Datensatz in eine In-Zeilen-Datenseite eingefügt wird und nicht genügend freier Speicherplatz verfügbar ist. SQL Server weist eine neue In-Zeilen-Datenseite zu, nimmt die Hälfte der Datensätze der ursprünglichen Datenseite, verschiebt sie zu einer neuen Seite und versucht den neuen Datensatz erneut einzufügen. Wenn der neue Datensatz noch immer nicht hineinpasst, erstellt SQL Server eine weitere Seitenaufteilung. Diese Operationen können weiter und weiter laufen. Seitenaufteilungen sind vollständig protokollierte Operationen, die eine enorme Menge an I/O-Aufwand erzeugen.

Seitenaufteilungen finden nur in geclusterten Tabellen und nur bei INSERT-Operationen statt. Wenn eine UPDATE-Operation in einer Spalte mit variabler Breite durchgeführt wird und der Datensatz nicht in derselben Position aktualisiert oder an eine andere Position auf der gleichen Seite verschoben werden kann, wird aus der UPDATE-Operation eine DELETE- und INSERT-Operation. Die INSERT-Operation löst dann eine Seitenaufteilung aus.

Das ist relevant für die Auswahl von Datentypen, da viele sich dafür entscheiden, GUIDs für Clustering-Schlüssel zu verwenden. Allerdings sollten Sie dabei die entsprechende Auslastung einer Tabelle beachten. GUIDs sind randomisierte Werte; wenn die Auslastung also aus vielen einzelnen INSERTS besteht und eine GUID für einen Clustering-Schlüssel verwendet wird, ist es sehr wahrscheinlich, dass die Seite bei fast jedem INSERT aufgeteilt wird. Und in Anbetracht des erzeugten I/O-Aufwands, ist eine übermäßige Seitenaufteilung alles andere als ideal, besonders bei einer transaktionsintensiven Datenbank.

Ich empfehle, Clustering-Schlüssel zu erstellen, die vor allem folgendes sind:

1. **Schmal.** Das spart Platz.
2. **Einzigartig.** Ein Clustering-Schlüssel sollte sich nicht wiederholen.
3. **Statisch.** Der Schlüssel für einen bestimmten Datensatz sollte immer gleich bleiben.
4. **Ständig wachsend.** Wenn ein Clustering-Schlüssel ständig wachsend ist, befindet sich der Hotspot am Ende der Clustering-Schlüssel-Sequenz, sodass Seitenaufteilungen nicht inmitten der Sequenz stattfinden.

Die automatisch inkrementierenden INT und BIGINT erfüllen diese vier Kriterien und sind in der Regel eine hervorragende Wahl.

Einige könnten auch NEWSQUENTIALID() vorschlagen. Das führt zu einem automatisch inkrementierenden GUID. Allerdings erhält NEWSQUENTIALID() seine Werte vom Windows-Betriebssystem. Wenn der zugrundeliegende Windows-Server also neugestartet wird, wird der Startpunkt von NEWSEQUENTIALID() auf einen zufälligen Wert zurückgesetzt. Das ist sehr riskant, da der neue Wert vor oder inmitten der bestehenden GUID-Sequenz liegen könnte. Wenn ein Team die Nutzung von GUIDs bevorzugt, ist die Erstellung eines INT/BIGINT Clustering-Schlüssels und eines GUID-Primärschlüssels ein möglicher Kompromiss.

Implizite Konvertierungen in SQL Server können zu einer Minderung der Leistung führen.

Wenn zwei Felder in SQL Server über unterschiedliche Datentypen verfügen, sind ihre Werte nicht vergleichbar, selbst wenn sie einem außenstehenden Beobachter identisch erscheinen. Daher müssen die Datentypen zunächst konvertiert werden, um übereinzustimmen, bevor die Werte verglichen werden können. Die Konvertierung von Werten in SQL Server erfolgt basierend auf vordefinierten Prioritätsregeln. Datentypen, die im Prioritätsdiagramm kleiner sind, werden immer hochkonvertiert, um mit den entsprechenden größeren Dateitypen übereinzustimmen. Anschließend kann SQL Server diese Werte vergleichen. Das hat Leistungsauswirkungen auf den T-SQL Code. Auf der rechten Seite ist eine Tabelle, die die Prioritätsregeln schildert.

Es folgen einige weitere Punkte, die bei impliziten Konvertierungen beachtet werden sollten:

- Nicht-übereinstimmende Datentypen verhindern Index-Suchoperationen. Die Tabelle / der Index mit dem kleineren Datentyp muss vollständig gescannt werden, um konvertiert zu werden und mit dem höheren Datentyp übereinzustimmen. Das wirkt sich sowohl auf das JOIN- als auch auf das WHERE-Prädikat aus.
- Mehrere Konvertierungen sind kostenlos. Konvertierungen zwischen CHAR und VARCHAR sowie Konvertierungen zwischen NCHAR und NVARCHAR sind kostenlos. Allerdings sind Konvertierungen zwischen dem UNICODE- und nicht-UNICODE-String nicht kostenlos.
- Es gibt Tools, um implizite Konvertierungen zu finden. Jonathan Kehayias von SQLskills.com hat einen Code geschrieben, mit dem Sie den Cache Ihres Ausführungsplans scannen und Code finden können, der an impliziten Konvertierungen leidet.

Gekürzte Prioritäten
1. DATETIME
2. SMALLDATETIME
3. DATUM
4. DECIMAL
5. BIGINT
6. INT
7. SMALLINT
8. NVARCHAR
9. NCHAR
10. VARCHAR
11. CHAR

Die Konvertierung von Werten erfolgt basierend auf vordefinierten Prioritätsregeln. Kleinere Datentypen werden immer zu größeren Datentypen hochkonvertiert.

Die Größe von Datentypen wirkt sich auch auf Ausführungspläne aus.

Die Verwendung von übermäßig großen Datentypen kann sich negativ auf die Effizienz eines Ausführungsplans auswirken. Wenn SQL Server einen Ausführungsplan erstellt, berechnet es die geschätzten Kosten anhand mehrerer Faktoren, einschließlich der durchschnittlichen Zeilengröße. Bei einem Datentyp mit variabler Breite, kennt SQL Server die durchschnittliche Größe der einzelnen Zeilen nicht und verwendet daher die Datentyp-Definition.

Stellen Sie sich nun vor, dass eine Tabelle über zahlreiche NVARCHAR(500) oder andere ähnlich arbiträr große String-Datentypen mit variabler Breite verfügt und die tatsächlichen Durchschnittswerte 50 Zeichen oder weniger betragen. Unabhängig von den einzelnen gespeicherten Werten, wird SQL Server seine Ausführungsplan-Schätzungen basierend auf der Größe von NVARCHAR(250) erstellen. Das wird zu einem großen Teil zur Erstellung suboptimaler Ausführungspläne beitragen.

UNICODE oder nicht-UNICODE für String-Datentypen.

Wenn Ihr System keine UNICODE-Daten speichern muss, können Sie sich für die Verwendung von nicht-UNICODE String-Datentypen entscheiden. Der Hauptvorteil davon sind Speichereinsparungen. Was allerdings, wenn Ihre Anwendung sich in Zukunft verändert und Sie UNICODE benötigen werden? Eine Refaktorierung könnte einen nicht-trivialen Aufwand bedeuten. Würden Sie sich dafür entscheiden, Ihre Datenarchitektur auf Kosten von Speicher und I/O zukunftssicher zu machen? Es gibt keine richtige Antwort, außer die Kompromisse zu verstehen, die Sie bei jeder der Optionen eingehen müssen, um die geeignetste Auswahl zu treffen.

Schnelle Datenbankleistung beginnt mit der Wahl des richtigen Datentyps

Die Wichtigkeit der Auswahl eines Datentyps wird oft unterbewertet. Doch diese fundamentalen Entscheidungen haben diverse Konsequenzen, die sich auf viele Weisen auf die Leistung von SQL Server auswirken. Investieren Sie die Zeit im Voraus, stellen Sie die richtigen Fragen und treffen Sie informierte Entscheidungen, andernfalls wird Ihre Anwendung in Zukunft von Leistungsproblemen geplagt sein.

Wie Ihre Datentyp-Wahl die Leistung von SQL Server-Datenbanken beeinflussen kann

VON ANDY YUN

SentryOne