entwickler.press
shortcuts

SQL-Server-Indizes

Schlüssel zur optimalen Datenbankperformance

Robert Panther

Robert Panther

SQL-Server-Indizes

Schlüssel zur optimalen Datenbankperformance

ISBN: 978-3-86802-590-3

Ein Imprint der Software & Support Media GmbH

1 Klassische Indexvarianten

Dieser shortcut bietet eine Einführung in die verschiedenen mit SQL Server verfügbaren Indexformen mit deren individuellen Besonderheiten. Es wird erklärt, wann welche Indexvariante am besten einzusetzen ist, aber auch, was bei der Definition und Verwendung zu beachten ist, damit diese möglichst effizient genutzt werden können.

Nach wie vor nutzen die meisten ernsthaften Businessanwendungen relationale Datenbanksysteme wie den SQL Server zur Datenspeicherung. Damit auf die dort gespeicherten Daten auch performant zugegriffen werden kann, sind – angefangen vom Datenbankdesign bis hin zur Abfrage – Indizes ein ganz entscheidender Faktor. Jedoch bietet MS SQL Server mittlerweile eine ganze Reihe von verschiedenen Indexformen: Neben den klassischen gruppierten und nicht gruppierten Indizes gibt es Indizes für spezielle Einsatzgebiete wie Volltextindizes, XML-Indizes und räumliche Indizes. Dazu kamen später die ursprünglich für Data Warehouses entworfenen Columnstore Indizes, die spätestens in der dritten Generation auch gut für andere Bereiche sinnvoll nutzbar werden.

Allgemeine Funktionsweise von Indizes

Um die grundlegende Funktionsweise von Indizes besser verstehen zu können, ist es hilfreich zu wissen, wie SQL Server die Daten auf dem Datenträger ablegt. Die Daten werden in Dateien mit der Endung *.mdf* oder *.ndf* abgelegt. Diese sind in Speicherseiten mit einer Größe von jeweils 8 KB aufgeteilt, was auch die kleinste Einheit ist, in der SQL Server Daten liest oder schreibt. Jeweils acht aufeinanderfolgende Speicherseiten (Pages) bilden ein so genanntes Extent. Während eine Speicherseite aber nur Daten einer Tabelle enthält, gibt es bei den Extents sowohl einheitliche als auch gemischte, die

Speicherseiten von verschiedenen Tabellen beinhalten. Wenn eine Abfrage beispielsweise alle Personen mit einem bestimmten Nachnamen aus einer Tabelle liest, für die kein Index existiert, müssen alle Speicherseiten gelesen werden, die Daten zu dieser Tabelle enthalten.

Um diese Abfragen effektiver ausführen zu können, sind entsprechende Strukturen hilfreich, die dafür sorgen, dass die relevanten Speicherseiten schneller gefunden werden können und somit nicht mehr alle Speicherseiten zu lesen sind. Genau diese Aufgabe übernehmen Indizes.

Gruppierte (engl. Clustered) Indizes

Im einfachsten Fall ist die Tabelle selbst nach einer bestimmten Spalte sortiert (im Fall der Personentabelle beispielsweise nach dem Nachnamen). Genau das ist es, was ein gruppierter Index bewirkt. Dies geschieht allerdings nicht in Form einer einfachen Liste, sondern in Form eines binären Baums, bei dem jeder Knoten des Baums eine eigene Speicherseite verwendet, die eine alphabetisch sortierte Liste von Namen enthält, ergänzt um jeweils einen Verweis auf die Speicherseite, in der weiter nach diesem Namen (oder darauf alphabetisch folgenden Namen) zu suchen ist. Dies kann sich – je nach Anzahl der Datensätze in der Tabelle – mehrstufig fortsetzen, bis schließlich auf die Speicherseite verwiesen wird, in der die kompletten Personendaten abgelegt sind. Somit müssen nur wenige Datenseiten gelesen werden, bis schließlich der oder die passenden Tabellenzeile(n) gefunden sind.

Während ein Nachname in einer Personentabelle sicherlich mehrfach vorkommen kann, nutzt man in der Praxis für einen gruppierten Index meist eine eindeutige Spalte, die häufig dem Primärschlüssel der Tabelle entspricht, also dem Schlüssel, über den alle Zeilen einer Tabelle eindeutig identifiziert werden können. Daher wird bei Erstellung eines Primärschlüssels auch implizit ein gruppierter Index dazu erzeugt, sofern nicht

explizit das Schlüsselwort *NONCLUSTERED* angegeben wird (Listing 1.1).

```
-- explizit
CREATE CLUSTERED INDEX PK_Person_BusinessEntityID
ON Person.Person (BusinessEntityID ASC)

-- implizit durch Erstellung eines Primärschlüssels
ALTER TABLE Person.Person
ADD CONSTRAINT PK_Person_BusinessEntityID
PRIMARY KEY CLUSTERED (BusinessEntityID ASC)
```

Listing 1.1: Varianten zur Erstellung eines gruppierten Index

Die gängigste Variante für einen gruppierten Index ist eine *Id*-Spalte vom Typ *int* (oder *bigint* bei großen Tabellen) mit einer Identitätsspezifikation, die dafür sorgt, dass die Spalte automatisch mit eindeutigen Nummern versehen wird. Als Alternative werden oft auch Spalten vom Typ *uniqueidentifier* verwendet, die über die Funktion *NewId()* als Default Constraint ebenfalls mit eindeutigen – aber nicht fortlaufenden – Werten versorgt werden. Dies ergibt vor allem in verteilten Umgebungen Sinn, wo neue Zeilen in verschiedenen Datenbankkopien erstellt werden, die dann später zusammengefügt werden, ohne dass bei den Primärschlüsseln Überschneidungen auftreten.

Ein großer Nachteil der gruppierten Indizes liegt darin, dass die Sortierung nur in einer Reihenfolge vorliegen kann. Eine Abfrage, die anhand einer anderen Spalte (z. B. dem Vornamen) sucht, wird von einem Index auf der Spalte *Nachname* nicht profitieren. Um dies zu lösen, werden n nicht gruppierte Indizes verwendet.

Anzeige von Ausführungsplänen

Um die Verwendung von Indizes in Abfragen zu überprüfen, bietet das SQL Server Management Studio die Möglichkeit, sowohl die voraussichtlichen als auch die tatsächlichen Ausführungspläne von Abfragen anzeigen zu lassen. Beide Optionen lassen sich am einfachsten über die entsprechenden Optionen