Unique Identifiers

HET GEMAK EN ONGEMAK VANUIT DATABASE OOGPUNT

Veel ontwikkelaars gebruiken GUID's (Globally Unique IDentifier). Vooral omdat het makkelijk is, ze snel kunnen worden gegenereerd en ook door code. Een ID op basis van integer is minder populair. De reden daarvoor is vanuit ontwikkelaars oogpunt te begrijpen. Om het consistent te doen, moet je met een database communiceren en dat kost moeite en ontwikkeltijd.

Laten we dieper ingaan op de GUID. Een GUID, die door een Microsoft SQL Server (MS SQL) wordt gegenereerd, is niet globally unique. Deze is enkel uniek binnen een instance van MS SQL. Dit geldt ook voor een GUID die door een applicatieserver wordt aangemaakt. Als er meerdere applicatieservers zijn die allen een GUID genereren om weg te schrijven, bestaat de kans dat dubbele ID's worden aangemaakt. Dus als je een applicatie aan het bouwen bent waarvan je verwacht dat een scale out plaatsvindt om de load te verdelen, denk dan vooral 2 keer na als je GUID's wil gebruiken als unieke ID's.

Naast het mogelijk niet uniek zijn van een GUID, heeft het ook een gigantisch performance nadeel als het gaat om databases. Dit nadeel is gebaseerd op het feit dat een GUID een alfanumeriek tekstveld is. In MS SQL termen is dat dus een varchar, maar vaker zelfs een nvarchar, vanwege Unicode voorschriften.

Even een stukje MS SQL engine kennis. Om een join te maken tussen 2 tabellen, ongeacht het join type (inner join, left outer join etc. etc.), worden deze gekoppeld op een corresponderend veld. Als dit veld, vaak een ID veld, een 4 byte integer is kost elke 'row by row' vergelijking 2 CPU cycles. Dus bij een join van een tabel met 10.000 rows en een tabel van 1.000 rows vinden er theoretisch 10.000.000 regelvergelijkingen plaats. Het gebruik van indexes niet meegerekend. Wat dus inhoudt dat 20.000.000 CPU cycles nodig zijn om de join uit te voeren op basis van een integer. Als dezelfde join wordt gedaan met een varchar(4), ook 4 bytes, dan kost elke 'row by row' vergelijking 17 CPU cycles. Wat voor de join tussen de eerder genoemde tabellen 170.000.000 CPU cycles inhoudt. Voor een join op basis van een 4 byte nvarchar, heeft SQL Server 35 CPU cycles nodig voor een enkele 'row by row' vergelijking. Omgerekend dus 350.000.000 CPU cycles.

Image not readable or empty
/public_html/assets/img/body-images/_bodyCopy/CPU-Cycles.png
Cpu Cycles

Als we deze kennis toepassen op een GUID die een 36 byte varchar (of nvarchar) is, wordt duidelijk dat het verschil tussen een ID op basis van een integer of op basis van een GUID in het positiefste geval (een varchar) 153 keer meer CPU cycles kost. In het negatieve geval (een nvarchar) kost het 315 keer meer CPU cycles dan een ID op basis van een integer.

Als we uitgaan van een 3GHz processor met een enkele core, is de doorlooptijd van de join als volgt:

Integer

Image not readable or empty
//rubicon.nl/assets/img/body-images/_bodyCopy/Integer.png

Hiervoor werd al aangegeven dat indexes in de berekening niet zijn meegenomen. Dit geeft niet aan dat een index het simpele antwoord is om een GUID qua performance wel te blijven gebruiken. Als je tijdens een query meer dan 1% van de in de tabel aanwezige records als resultaat hebt, uitgaande van een tabel die groter is dan 64Kb, maakt SQL server geen gebruik van indexes, tenzij het een columnstore index is, maar stapt het over op een table scan (lees: alle records van een tabel in het geheugen).

Daarnaast is er natuurlijk ook een verschil tussen een ontwikkel- en productieserver. Als een databaseserver de hele dag niks doet, wat vaak het geval is op een ontwikkelserver, lijkt elke query snel. Maar als dezelfde query op productie los wordt gelaten, is de query opeens traag. Dit komt omdat een productieserver over het algemeen meer query's afhandelt dan alleen de query's van een enkele applicatie. Daarnaast bevat een productieserver vaak meer en complexere data dan een ontwikkelserver.

Alles bij elkaar bekeken, is het dus niet verstandig om GUID's te gebruiken als ID's in een database. Dat neemt niet weg dat het genereren van ID's of volgnummers door middel van een integer veld in een tabel omslachtig kan zijn.

Eerst moet een insert commando worden uitgevoerd en vervolgens moet het ge-inserte ID worden opgehaald. Commando's als SELECT@@IDENTITY werken niet goed als het gaat om een multi thread applicatie en er geen gebruik wordt gemaakt van transacties. Het is met SELECT@@IDENTITY mogelijk dat een ander ID wordt teruggegeven aan een thread dan het ID dat door de thread is aangemaakt. Daarnaast kost deze methode tijd, aangezien het een tabeloperatie behelst en dus IO activiteit. Ook is het vaak een vereiste om een volgnummer pas op te slaan als het gehele proces succesvol is afgerond, bij niet afgeronde processen mag het volgnummer niet worden opgeslagen. Dan is er bij het gebruik van een tabel ook een delete commando nodig.

Gelukkig is er een oplossing. Sinds SQL Server 2008 R2 is er de functie genaamd Sequence. Deze functie biedt de mogelijkheid unieke volgnummers te genereren zonder dat tabeloperatie nodig is. Een Sequence is primair een functie die enkel in memory wordt afgehandeld. Onderwater wordt de actuele waarde van een sequence wel opgeslagen, maar dat is een parallel proces.

Conclusie

Gebruik als ID-velden voor tabellen geen GUID's, maar integers. Indien nodig een 8 byte integer (BIGINT). Indien je een uniek ID of volgnummer moet genereren voordat deze kan worden weggeschreven, gebruik dan een Sequence.

Meer informatie over Sequences

Remko de Boer - senior database administrator en Business Intelligence consultant bij Rubicon met jarenlange ervaring met alle facetten van Microsoft SQL Server. Specialisaties: database en BI architectuur, Performance Tuning en opleidingen.

Lage Biezenweg 5
Postbus 169, 4130 ED Vianen
0347 35 88 00 | info@rubicon.nl | www.rubicon.nl

Gold
Microsoft Partner