Kritisch kijken naar de performance van het RDBMS

Database-tuning in een notendop

Peter Sap

Database-tuning is een veelomvattend gebied waar een brede vakkennis voor nodig is. Inzicht in het gebruikte Relationeel Database Management Systeem (RDBMS) is natuurlijk een eerste vereiste. In dit artikel worden enkele componenten van een RDBMS voor het voetlicht gebracht en worden de eigenschappen die van invloed zijn op de performance toegelicht.

Voor een database-ontwikkelaar worden er wellicht nieuwe begrippen geïntroduceerd, waardoor een beter inzicht in de onderliggende werking van een RDBMS wordt verkregen. Voor een ervaren DBA is dit artikel hopelijk een opfrisser of een aanleiding om nog eens kritisch naar de performance van een applicatie te kijken. De technieken en begrippen die worden besproken zijn van toepassing op de gangbare RDBMS'en zoals Oracle, DB2 UDB en Sybase.

In het algemeen zullen verbeteringen in de performance het grootst zijn door aanpassingen in programmatuur en datamodel. Denk hierbij aan zo'n 80 tot 90 procent van de totaal te behalen winst. Na optimalisatie van de code zal er nog extra winst te behalen zijn door de server te tunen maar dat zal beduidend minder zijn.

HET BELANG VAN METEN

Door regelmatig een performance-meting uit te voeren kunnen trends worden gesignaleerd. Zo'n meting kan het beste worden uitgevoerd met de gereedschappen die bij het RDBMS zijn meegeleverd. Ook worden hiervoor gereedschappen van andere leveranciers of zelf ontwikkelde software gebruikt. Als aanvulling hierop moeten ook bepaalde metingen op het niveau van het besturingssysteem worden uitgevoerd. Een database server moet bijvoorbeeld genoeg geheugen toegewezen krijgen, maar zeker niet te veel omdat daardoor paging of swapping kan ontstaan. Met het unix commando vmstat kan zo'n situatie worden gesignaleerd.

Op het moment dat er een performance-probleem ontstaat kan er een vergelijking worden gemaakt tussen de metingen van bijvoorbeeld een dag ervoor en op het moment dat het probleem zich voordoet. Uiteraard zal de vraag moeten worden gesteld wat er anders is aan de applicatie dan toen het probleem er nog niet was.

Performance-problemen kunnen ook geleidelijk ontstaan, bijvoorbeeld door een groei van het aantal gebruikers en de data. Met metingen kan dan worden bepaald welk deel van het RDBMS het zwaarst wordt belast of welk deel juist onderbelast is. Ook kunnen taken in een RDBMS, zoals auto's, in de file staan. Een voorbeeld hiervan vormen langlopende transacties die locks vasthouden, waardoor andere transacties worden geblokkeerd. Goede gereedschappen kunnen zo'n situatie detecteren en met de kennis over het gebruikte RDBMS kan het probleem verder worden geanalyseerd en een oplossing worden gevonden.

In de logfile van het RDBMS zelf, kunnen meldingen voorkomen waarop actie moet worden ondernomen. Een geregelde inspectie van dit logfile, bij voorkeur dagelijks, kan mogelijke problemen tijdig aan het licht brengen.

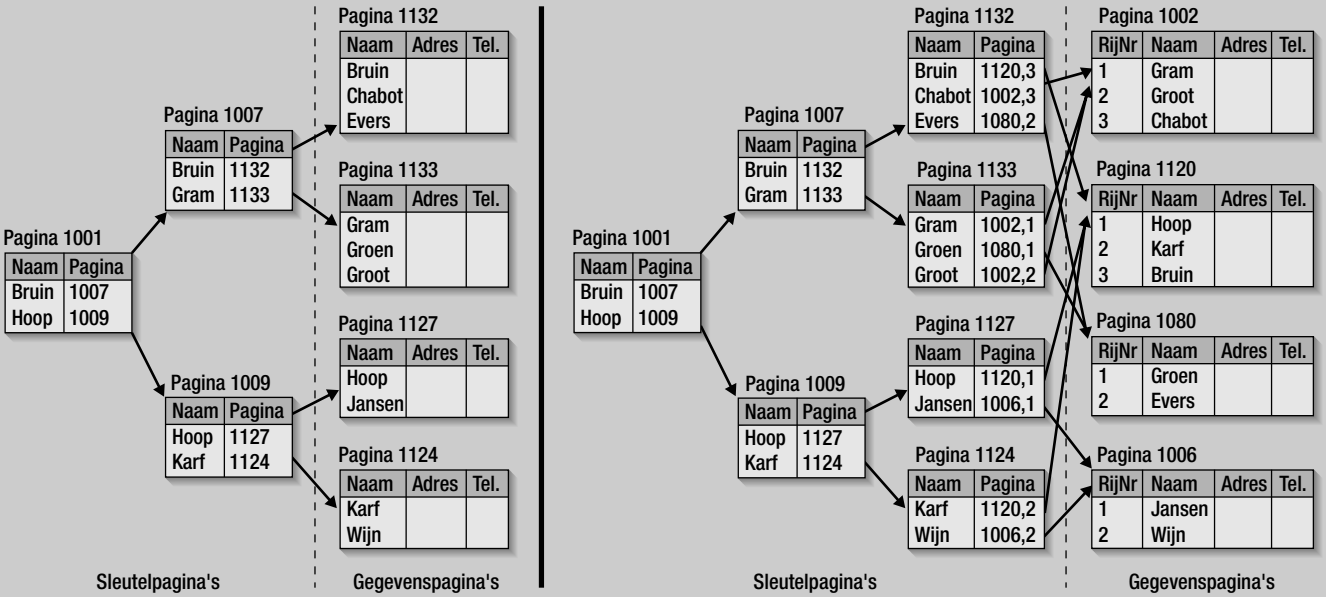
OEFENING BAART KUNST

Met de hierboven genoemde gereedschappen kan vaak ook slecht presterende SQL-code worden opgespoord. Niet elk RDBMS biedt die informatie overzichtelijk of compleet aan, maar de gebruiker is dan een perfecte graadmeter. Opmerkingen als "Na het starten van dat programma gaan we koffie drinken" of "Bij dat overzicht loopt het hele systeem vast" zijn de beste indicatie waar de problemen zitten.

Als een stuk SQL-code wordt beoordeeld op performance, wordt meestal eerst naar het toegangspad (queryplan) gekeken. Dit is vrij secuur werk en hier geldt zeker dat oefening kunst baart. Als een toegangspad niet goed is kunnen daar verschillende oorzaken voor zijn: geen (of niet juiste) sleutel, het meeselecteren van overbodige attributen, selecteren op een view in plaats van rechtstreek op de tabel, het zondigen tegen bepaalde regeltjes die afhankelijk van het DBMS zijn of het verkeerd (of niet) gebruik van bepaalde opties die de programmeur tot zijn beschikking heeft, om er slechts enkele te noemen. Doordat bij het schrijven van SQL er vrij ongemerkt zoveel fout kan gaan, kan de winst op de performance enorm zijn.

Daarnaast moeten de verbruikte cpu-tijd en lees/schrijf-acties in een redelijke verhouding staan met de geselecteerde resultaten.

Een vergelijking van twee soorten sleutels



Afbeelding 1: clustered index (links) en normale index (rechts).

CLUSTERED INDEX OP HET ATTRIBUUT "NAAM" VAN DE TABEL "KLANT"

De vereenvoudigde structuur van de clustered index van de tabel "klant" bestaat een toppagina (nr. 1001) en de onderliggende pagina's, waarbij de pagina's steeds naar elkaar verwijzen. In de verwijzigingen wordt naast het paginanummer ook de naam opgenomen die als eerste voorkomt in de pagina waar naar wordt verwezen. Omdat de sleutel gesorteerd is opgeslagen, kan er zo gemakkelijk door de structuur worden genavigeerd.

Bij het selecteren van de gegevens van een klant met de naam "Groen" wordt allereerst de top-pagina gelezen. De pagina wordt doorzocht naar een verwijzing waarbij de naam "Groen" gelijk is aan of wat betreft sorteervolgorde voor de naam ligt waar naar wordt verwezen. In dit geval moet dan pagina 1007 worden opgehaald, immers "Hoop" komt na "Groen" en "Bruin" ligt voor "Groen". Vanuit pagina 1007 blijkt dat nu pagina 1133 moet worden opgehaald. In pagina 1133, dat geen sleutelpagina is maar een een gegevenspagina, treffen we de klant met de gezochte naam aan. Omdat we hier met een clustered index te maken hebben, zijn nu direct alle gegevens van de klant beschikbaar.

INDEX OP HET ATTRIBUUT "NAAM" VAN DE TABEL "KLANT"

Bij een normale indexstructuur kan hetzelfde zoekalgoritme worden gebruikt. Bij het zoeken naar een klant met de naam "Groen" worden opnieuw de pagina's 1001, 1007 en 1133 gelezen. Als alleen de naam uit de tabel wordt geselecteerd is het niet nodig om nu verder te gaan. Bij een selectie van andere attributen, zoals adres, moet ook pagina 1080 worden opgehaald. Hieruit blijkt dat het opvragen van overbodige attributen ten nadele van de performance kan komen.

Een selectie die slechts enkele rijen toont maar er duizend leesacties voor nodig heeft, zal wellicht kunnen worden verbeterd. Een selectie herschrijven of opdelen in stukken kan helpen. Een kritische blik op het gebruik van 'distinct' kan een interne sortering besparen. In sommige gevallen kan een sortering voor een 'order by' worden voorkomen door een sleutel op de tabel te plaatsen.

Als de code onderdeel is van een stored procedure of trigger, is het zinvol om zoveel mogelijk andere code in die procedure te beoordelen. Slecht geprogrammeerde algoritmen en overbodig of verkeerd gebruik van tijdelijke tabellen komen vrij veel voor en dat kan dan in een keer worden opgelost. Andere stukken code die door dezelfde programmeur zijn ontwikkeld hebben vaak dezelfde eigenschappen. Bij de analyse van de gebruikte toegangspaden spelen de sleutels op een tabel een belangrijke rol. Dit is het mechanisme om snel en efficiënt toegang tot gegevens te krijgen.

TWEE DOELEN

In een RDBMS hebben sleutels (indexen) twee doelen: het waarborgen van uniciteit en het bieden van een efficiënt toegangspad voor een selectie. Om alle mogelijke selecties een optimaal toegangspad te bieden, zouden alle attributen van een tabel in een sleutel moeten worden opgenomen. Dit heeft echter als nadeel dat

door een wijziging in de gegevens van de tabel veel sleutels moeten worden bijgewerkt, en dat is slecht voor de performance. Er is dus een juiste balans nodig in het bepalen van de sleutels.

Een sleutel kan uit meerdere attributen van een tabel bestaan, dit is een samengestelde sleutel. Een selectie zal een samengestelde sleutel pas goed kunnen gebruiken als minimaal via de beginnende attributen van de sleutel toegang tot de gegevens wordt gevraagd. Een samengestelde sleutel op de attributen A en B zal een selectie op alleen kolom B niet veel helpen. Sommige RDBMS-en zullen echter toch via de sleutel gaan zoeken als dat minder leesacties kost.

Er zijn twee soorten sleutels, namelijk clustered (Oracle: index-organized) en non-clustered. Per tabel kan er maar een clustered index aanwezig zijn. Als vuistregel kan worden aangehouden dat de clustered index het beste kan worden gekozen ter ondersteuning van selecties waarbij veel attributen uit de tabel benodigd zijn, er worden meerdere rijen opgevraagd en in de 'where' clausule van de selectie worden in ieder geval die attributen genoemd waar de sleutel ook mee begint. Een kenmerk van een clustered index is dat de rijen in gesorteerde volgorde van de sleutelattributen worden opgeslagen. Een hash-index komt soms ook nog voor. Hierbij wordt via een formule de plaats van de rij berekend.

Sleutels moeten ook worden onderhouden. Dit kan gebeuren door de statistische gegevens die in het RDBMS worden bijgehouden weer te verfrissen. Deze statistieken spelen een belangrijke rol bij het bepalen van de juiste toegangpaden. Onderhoud moet ook plaatsvinden door sleutels te reorganiseren of te verwijderen en dan opnieuw aan te maken. Tabellen die een hoge mutatiegraad kennen zijn daar vaak bij gebaat.

Omdat sleutels een cruciale rol spelen in de performance van een systeem is het belangrijk om de eigenschappen en mogelijkheden die het RDBMS op dit vlak biedt, goed te kennen.

WINST IN PERFORMANCE

Wanneer een applicatie om informatie aan het RDBMS vraagt, moet dit worden opgezocht. Indien mogelijk zal dat via een sleutel gebeuren, maar kan dat niet dan moet de tabel in zijn geheel worden doorzocht. Als de sleutels en/of rijen worden gelezen vanaf de harde schijf, dan worden ze in het interne geheugen geplaatst en als de selectiecriteria kloppen met de rij, doorgegeven aan de applicatie. Zowel de gelezen sleutels als de rijen worden nog een tijd in het geheugen bewaard en als de applicatie er opnieuw om vraagt kan het direct worden doorgegeven, zonder opnieuw de harde schijf te doorzoeken. Dit mechanisme wordt *buffering* of *caching* genoemd. Omdat het lezen van gegevens van de harde schijf veel langzamer is dan het lezen uit het interne geheugen zijn hier belangrijke winsten in de performance te halen.

Na verloop van tijd zal het interne geheugen vol zijn en moet er ruimte worden vrijgemaakt om nieuwe gegevens vanaf de harde schijf te kunnen inlezen. Een bepaald algoritme zal de minst gebruikte informatie het eerst uit het geheugen verwijderen en de meest gebruikte nog even laten staan. Dit heet de *replacement policy*. Soms kan er via een aparte instelling voor worden gezorgd dat de sleutelinformatie wat langer bewaard blijft dan de rij zelf of kan de replacement policy anderszins worden bijgesteld. Veelgebruikte tabellen, sleutels of zelfs gehele databases/tablespaces kunnen in een apart deel van de buffer worden geplaatst om er voor te zorgen dat de gegevens zoveel mogelijk in het geheugen blijven.

Een belangrijk getal uit de metingen van het RDBMS is de cache hit ratio. Dit is een percentage dat aangeeft hoeveel procent van de gevraagde informatie rechtstreeks uit de buffer komt en niet van de harde schijf. Met een goed afgestelde server zal dit getal 90 procent of hoger moeten zijn. Bij zeer grote databases of bij een toepassing waarbij steeds volstrekt andere delen van een tabel worden gelezen mag dit lager zijn. De ratio mag pas worden gemeten als de server al een tijd in gebruik is, aangezien bij het starten van de server de ratio nul procent is.

Een geregelde inspectie van een logfile kan mogelijke problemen tijdig aan het licht brengen

Intern werkt een RDBMS altijd met pagina's (pages/blocks) om rijen of sleutels op te slaan. Op een pagina kunnen meerdere rijen voorkomen of, als de rij erg lang is, slechts één rij of een deel ervan. Ook de sleutels worden in een aantal pagina's opgeslagen. Bij het opvragen van een rij wordt eerst de betreffende pagina opgezocht en vervolgens worden de attributen van de rij daar uit gelezen.

Als een pagina van de harde schijf wordt gelezen komt deze in zijn geheel in de buffer. Er kan zelfs voor gezorgd worden dat er niet een maar meerdere pagina's per keer in de buffer worden geplaatst. Dit heeft voordelen indien er verschillende op elkaar aansluitende gegevens moeten worden gelezen. Een voorbeeld hiervan is een inkooporder waarvan alle orderregels moeten worden opgehaald. De orderregels kunnen op verschillende pagina's staan maar kunnen zo toch met slechts een leesactie van de harde schijf worden opgehaald. Dit kan men bereiken door de orderregels een clustered index te geven op de juiste sleutelattributen. Immers, bij een clustered index worden de gegevens gesorteerd opgeslagen. Het mechanisme om meerdere pagina's per keer van de harde schijf te lezen heet *pre-fetching*.

De gegevens die op een pagina staan, zijn te beïnvloeden door wijzigingen op de tabel. Bij een wijziging van een rij is het mogelijk dat de rij langer wordt en niet meer binnen de pagina past. Een andere mogelijkheid is het toevoegen van een rij aan een tabel met een clustered index. De clustered index wordt gesorteerd opgeslagen en de rij moet worden ingepast in een pagina om de sorteervolgorde niet te verstoren. Afhankelijk van het RDBMS is er een overloopgebied of maakt het een nieuwe pagina aan. Dit is een vrij bewerkelijke actie en dat gaat ten koste van de performance.

Een oplossing hiervoor is de bezettingsgraad (*usage* of *fillfactor*). Hiermee kan men aangeven dat een hoeveelheid ruimte in de pagina gereserveerd is voor een rij die langer wordt of het tussenvoegen van een nieuwe. Bij tabellen die statisch zijn en niet worden gewijzigd kan men de bezettingsgraad op de hoogst mogelijke waarde instellen. Zo kunnen er minder leesacties voor nodig zijn als de tabel wordt geraadpleegd.

In de configuratie van een RDBMS kan men de paginagrootte opgeven. Bij OLTP-toepassingen wordt vaak voor een wat kleinere pagina gekozen, terwijl bij een datawarehouse grotere pagina's beter blijken te werken. Als vuistregel zijn hiervoor pagina's van respectievelijk 2 tot 4 kB en 16 kB gebruikelijk. Door het kiezen van een grotere pagina wordt eigenlijk al een vorm van pre-fetching gerealiseerd.

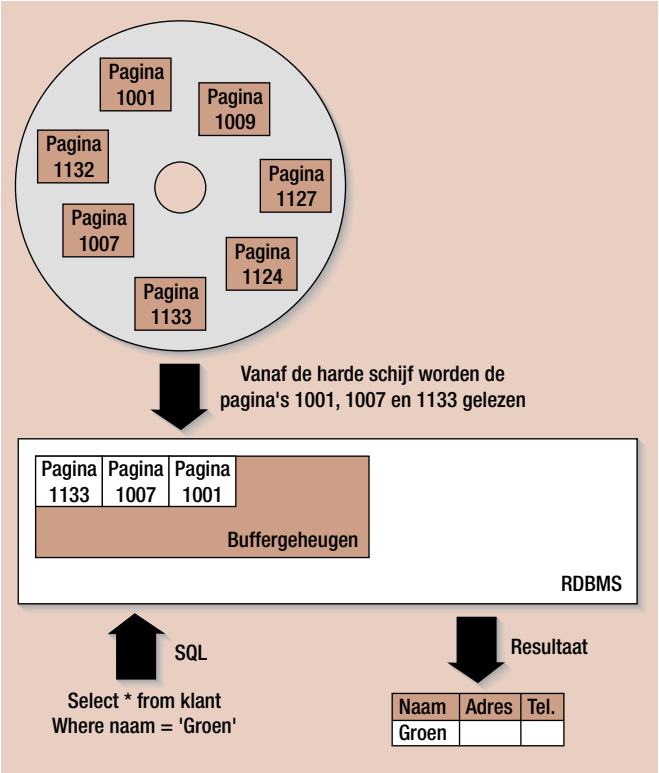
KEUZE VAN DE SCHIJFINDELING

Meestal zullen er meerdere harde schijven (harddisks) beschikbaar zijn voor een RDBMS. Het zorgvuldig indelen van de beschikbare ruimte op de schijven kan bijdragen tot een goede performance. Alle wijzigingen in een database worden in de transactielog bijgehouden en, omdat de log onderdeel uitmaakt van het recovery mechanisme, worden aanvullingen op de transactielog altijd zo snel mogelijk door het RDBMS weggeschreven naar schijf. Hieruit blijkt dat de log een veelgebruikt onderdeel is en dat verstoringen een goede performance in de weg staan. Om die redenen wordt de log bij voorkeur op een aparte schijf geplaatst en, bij verschillende typen schijf, op de snelste die voorhanden is. De databases worden dan op de overige schijven geplaatst.

De belasting over de verschillende schijven die aan een database zijn toegekend, kan men sturen door bepaalde tabellen of tablespaces op een specifieke schijf te plaatsen. Een andere mogelijkheid is het splitsen van een tabel door de rijen te verdelen over twee of meer schijven. Zo ontstaat een horizontaal gepartitioneerde tabel. Een tabel waarbij een aantal kolommen naar een tweede tabel in een 1-op-1 relatie wordt verplaatst, is een verticaal gepartitioneerde tabel. Deze techniek wordt echter eerder ingezet om bepaalde selecties te versnellen of het snel vollopen van pagina's tegen te gaan.

Een andere optie is de niet-clustered sleutels op een andere schijf te plaatsen dan waar de data staat. Door deze splitsing wordt het gebruik van de tabel in enige mate verdeeld over de beschikbare schijven. In de praktijk zullen de gereedschappen van het RDBMS moeten worden ingezet om te bepalen welke schijf over- of onderbelast is. Zo kan men ook uitzoeken of de zojuist beschreven partitionering en scheiding van index en data, de performance ten goede kan komen.

De keuze van de totale schijfindeling wordt vaak samen met de beheerders van de hardware gemaakt. Zij hebben de kennis in huis omtrent de eigenschappen van de schijven en het inzicht in de belasting, die andere processen die op dezelfde computer actief zijn met zich meebrengen. Overigens speelt de doorvoersnelheid van gegevens van de harde schijf naar het RDBMS ook een rol bij de keuze van een harde schijf.



Afbeelding 2: van schijf naar resultaat. Schematische weergave van de afhandeling van een SQL-commando door een RDBMS. Het SQL-commando wordt ontvangen door het RDBMS en de benodigde pagina's worden van de harde schijf gelezen en in het buffergeheugen geplaatst. Nadat de gezochte attributen zijn gevonden, worden de resultaten in tabelvorm teruggestuurd.

SAMENWERKING

Performance-problemen kunnen goed worden voorkomen door in een team te werken. Samenwerking tussen ontwerpers, (database) ontwikkelaars en DBA's, zal er toe bijdragen dat kennis op het gebied van ontwerp, bouw en de specifieke eigenschappen van RDBMS gemakkelijk wordt gedeeld. Zo zullen ontwerpers en DBA's nauw moeten samenwerken om een transformatie van logisch naar fysiek datamodel te maken. Ontwikkelaars kunnen de DBA informatie geven over veelgebruikte tabellen en toegangspaden. Omgekeerd kan een DBA de ontwikkelaars ondersteunen met kennis over het RDBMS. Zo zal blijken dat door samenwerking en communicatie betere toepassingen, met minder performance-problemen, worden gebouwd. ●

Aanbevolen literatuur: Database Tuning van Dennis Shasha en Philippe Bonnet, ISBN 1-55860-753-6.

Peter Sap (peter@petersap.nl) is senior database ontwikkelaar/DBA.