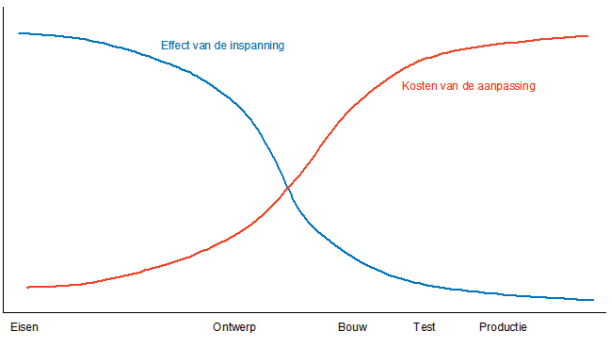
Oracle performance tijdens het ontwikkelen

Zeker ook issue voor ontwikkelaars

Veel boeken en artikelen over Oracle performance gaan over SQL tuning en databasetuning. Dit is niet voor niets, het zijn belangrijke onderwerpen en zijn nodig voor succesvolle Oracle applicaties. Door de grote aandacht voor SQL tuning en databasetuning ontstaat bij sommige ontwikkelaars ten onrechte de indruk dat performance met name een vraagstuk voor de DBAs is en dat performance geen issue is voor ontwikkelaars.

De enige uitzondering betreft dan wellicht SQL tuning, maar daar wordt meestal pas aandacht aan besteed helemaal aan het eind van het ontwikkeltraject, of nog erger: pas in productie. Dit terwijl geen ontwikkelaar het in haar of zijn hoofd zal halen om te ontkennen dat een Ferrari veel harder rijdt dan een Lada vanwege het ontwerp. Niemand zal beweren dat het louter het werk van de chauffeur is dat er voor zorgt dat de Ferrari harder rijdt. Waarom dan toch die geringe aandacht voor performance tijdens het ontwikkelen van Oracle software? In de OGh Visie van december 2008 heeft Toine van Beckhoven een begin gemaakt om performance bij Oracle-ontwikkelaars onder de aandacht te krijgen. Ik wil laten zien dat een ontwikkelaar gedurende het hele traject invloed kan uitoefenen op de performance. Het is niet de bedoeling hierin uitputtend of volledig te zijn, maar wel om te tonen dat en waar een ontwikkelaar de performance kan beïnvloeden.

Ook voor performance geldt dat voorkomen beter is dan genezen. Hoe later in de levenscyclus van een applicatie performanceproblemen opgelost moeten worden, hoe groter de kosten om die problemen op te lossen, en vaak hoe minder de mogelijkheden die je hebt om ze op te lossen. Aan de tekentafel kan ik nog besluiten welke vorm en materialen ik ga gebruiken voor mijn racewagen. Als ik eenmaal de Ford Transit op de oprit heb staan in plaats van de Ford GT, kan ik wel de banden op spanning brengen en er een krachtigere motor in zetten, maar verder kan ik weinig meer doen. Door eerder aandacht te besteden aan performance, kan deze vaak tegen een kleine inspanning zeer effectief worden beïnvloed. En juist omdat de gevolgen van verkeerde beslissingen naarmate het traject vordert, niet of steeds slechter kunnen worden bijgesteld, is het dus van belang dat vroegtijdig aandacht wordt besteed aan performance bij het ontwikkelen van software.



Hoe en wanneer heeft een ontwikkelaar dan invloed op de performance van een applicatie? In elke fase in de levenscyclus van een applicatie worden er keuzes gemaakt die van invloed zijn op de performance. Hoe later in de levenscyclus van een applicatie performanceproblemen opgelost moeten worden, hoe duurder de oplossing is. Ontwikkelaars hebben te maken met performance bij de eisen, bij het functioneel en het technisch ontwerp, bij de bouw en het schrijven van SQL statements en bij het testen.

Fasen in proces

Eisen

De eerste stap die een ontwikkelaar moet nemen, is het achterhalen van eisen, ook met betrekking tot performance. Aan welke eisen moet worden voldaan? En vooral: aan welke performance-eisen moet worden voldaan? Als je niet weet wat je moet bouwen, kun je net zo goed niets bouwen. Elke informaticus weet dat alle uitspraken over een lege verzameling waar zijn. Anders gezegd: zijn er geen eisen, dan is altijd aan de eisen voldaan. Een probleem is vaak dat aanwezige performance-eisen niet expliciet gemaakt zijn. Door ze niet expliciet te maken, is

er vaak onduidelijkheid over performance-eisen en zijn ze niet eenduidig. Een ontwerper die een nieuwe fiets moet maken, zal willen weten wat voor soort fiets het wordt en hoe snel die moet kunnen, naast veel andere eisen. Zelfs als het een racefiets moet worden, is er een verschil tussen het gebruik door een professionele wielrenner tijdens wedstrijden en recreatief gebruik.

Zodra je als ontwikkelaar aan een nieuwe klus begint, moet je navragen wat de eisen voor de performance zijn. Hoe snel moeten gegevens op een scherm verschijnen? Hoe lang mag een rapport er over doen? Wat is de maximale doorlooptijd van een laadprocedure? Maar ook: hoeveel gebruikers werken tegelijkertijd, en welke functionaliteit gebruiken ze dan? Is het het invoeren van orders, of het opvragen van rapporten? Welke processen lopen op hetzelfde moment? En om hoeveel data gaat het en met welke frequentie? Hoeveel orders worden per dag ingevoerd? Moeten bestanden dagelijks of wekelijks worden ingelezen? Deze en vergelijkbare vragen kunnen en moeten gesteld worden.

Bij het achterhalen van de eisen, hoort ook het verifiëren van de eisen. Hoe realistisch en zinnig zijn de eisen? Toen mijn vrouw en ik vorig jaar op zoek waren naar een nieuw huis wilden wij een huis op loopafstand van de binnenstad, op fietsafstand van ons werk, met uitzicht op zee en aan de rand van het bos en dat voor een maximaal 932 Euro netto in de maand. Uiteindelijk bleek aan twee eisen voldaan te kunnen worden.

Vraag jezelf of het wel mogelijk is om die 1600 XML-bestanden van elk zeker 12 MB in 20 minuten in te lezen tegelijk met het draaien van de maandrapportage. Er zijn omgevingen waar dit mogelijk is, maar is het ook mogelijk in deze specifieke omgeving? Sommige eisen lijken alleszins haalbaar. Maar is het een redelijke eis dat alle rapporten binnen 30 seconden klaar moeten zijn? Moet dit gelden voor zowel het rapport van beschikbare hotelkamers voor de eerstkomende nacht in Emmeloord, als voor het rapport over de hotelkamers in Zuid Europa die de komende zes weken drie aaneengesloten nachten beschikbaar zijn gelegen op een afstand van maximaal 100 kilometer van het dichtstbijzijnde vliegveld gesorteerd op prijs per kamer per persoon? Het is niet onwaarschijnlijk dat voor de ene categorie rapporten andere eisen gelden dan voor een andere categorie. Soms worden performance-eisen technisch geformuleerd, bijvoorbeeld: 50% van de queries is binnen 2 seconden klaar, 75% binnen 5 seconden en 100% binnen 10 seconden. Maar wat zegt het dat elke query in de database binnen 2 seconden afgerond is? Het verversen van een scherm kan wel 40 queries op zijn geweten hebben, maar mag de gebruiker dan serieus 80 seconden wachten? Performance-eisen kunnen beter op gebruikersniveau geformuleerd worden dan op serverniveau.

Gebruikershandelingen zijn niet één op één te relateren aan SQL queries of lees- en schrijfacties op een harde schijf. De eisen moeten wel realistisch en zinnig zijn.

Eenduidige, realistische en zinnige performance eisen kunnen worden ondergraven door andere eisen. Soms wil je niet slechts racen over het asfalt, maar ook door de modder, dat vergt een aanpassing aan je motor. Dat is niet per definitie erg. Het wordt een probleem als om een grote mate van generiekheid en flexibiliteit wordt gevraagd. Voor elke soort weg is wel een snelste motor te maken. Maar er is geen motor te verzinnen die altijd het snelst is of je nu op of naast de weg rijdt en die geschikt is voor één tot vier personen en waar je als het nodig is 500 liter bagage mee kunt vervoeren. Als een rapport dynamisch samen te stellen is door ad hoc tabellen, kolommen en zoekcriteria te kunnen aanpassen, dan kun je voor dat rapport niks verwachten van de performance.

Functioneel en technisch ontwerp

Het ontwerp moet aansluiten op de eisen, de eisen moeten op basis van het ontwerp gehaald kunnen worden. Als de eis is om een vervoermiddel te maken dat een gemiddelde snelheid van 100 kilometer per uur kan halen over een afstand van 500 kilometer, dan volstaat een gewone auto. Een fiets voldoet niet en een straaljager is behoorlijk overdreven. Ofwel, kun je met dit ontwerp voldoen aan de eisen?

Bij het functioneel ontwerp is de vraag relevant of de processen efficiënt zijn ingericht. Voor inefficiënte processen valt geen efficiënte code te produceren. Als ik voor een gezelschap van tien man eten moet maken en ik ga voor elke ingrediënt apart op en neer naar de winkel en vervolgens bereid ik de maaltijden één voor één, dan schiet dat natuurlijk niet op. Ik kan er ook voor kiezen om in één keer alle boodschappen te doen en vervolgens één flink maal voor 10 man klaar te maken.
Voor Oracle applicaties is het van belang dat er een goed relationeel datamodel aanwezig is. Het hart van een Oracle applicatie is de relationele database. Dat betekent dat alle data wordt opgehaald met behulp van SQL. Met een slecht datamodel zijn SQL queries nauwelijks goed te schrijven en uit te voeren.

Zijn de processen zelf eenmaal goed ingericht, hoe is dan de technische uitwerking? Is die wel efficiënt? Worden ook de juiste tools en technologieën gebruikt? Als ik data kopieer van de ene Oracle database naar de andere Oracle database binnen hetzelfde netwerk, moet ik dan wel regel voor regel de data naar een XML-bestand wegschrijven, dat bestand kopiëren en vervolgens dat bestand in de andere database weer inlezen? Dit kan ook met een databaselink bijvoorbeeld. Maar ook, als de 100.000 orders van een webshop verspreid over de dag binnenkomen, moet je dan slechts eenmaal per dag die informatie

doorsluizen naar de financiële administratie, in een beperkt tijdsraam van een half uur? Waarom dan niet geleidelijk per order of een aantal keren per uur de gegevens overzetten?

Voor synchronisatie van mijn basisregistratie in Oracle met een ander systeem kan ik databasetriggers aanmaken die de wijzigingen opslaan in een tabel en vervolgens een PL/SQL procedure aanroepen die de inhoud van die tabel overhevelt naar het andere systeem. Oracle Streams kan dat efficiënter, die leest de wijzigingen uit de redologfiles en die kun je dan overzetten naar welk systeem je maar wilt. Ander voorbeeld: met Apex kun je eenvoudig webapplicaties bouwen. Je hebt in een handomdraai een applicatie klaar om plaatjes te uploaden in een tabel en die in een rapportje te laten zien. Op die manier kun je een tabel maken met één rij met daarin het meest recente beeld van je webcam, om zo de beelden van je webcam op je webpagina op te nemen. Je kunt natuurlijk ook meerdere webcams op deze manier aanbieden, met voor elk een update van een BLOB in de database om de paar seconden. Maar, als er niets bewaard hoeft te worden waarom dan de data in een database stoppen met een heel transactiemechanisme er omheen? Een verkeerde technische uitwerking voorkomt dat efficiënt ingerichte processen vlot doorlopen kunnen worden. Met een raceauto van gietijzer kom je niet snel aan de finish. Dit is ook het moment om na te denken over het redundant opslaan van data en over processen voor archivering, historie en opschonen van data. Op basis van kennis over de omvang van data, frequentie van opvragen kun je door redundantie in te voeren ervoor zorgen dat de vereiste responstijden gehaald worden. Ook het redundant opslaan moet natuurlijk tegen zo laag mogelijke kosten gebeuren, dus dat wil je zo vroeg mogelijk uitvoeren dan wel plannen. Het is eenvoudiger om in een nieuwe woonwijk bij het maken van de plannen al rekening te houden met een extra school, sportvoorzieningen en winkels, in plaats van te wachten tot er voldoende gezinnen wonen en dan geen ruimte meer te hebben voor een school, sporthal of winkelcentrum, laat staan voor de benodigde parkeerplaatsen etc.

Ook bij het ontwerp is weer het gevaar dat oplossingen generiek en flexibel moeten zijn, een bekend punt is de database onafhankelijkheid. Dan weet je dat je van de voordelen van, in ons geval, Oracle geen gebruik kunt maken. Voor je het weet moet je dan zelf code gaan schrijven ter vervanging van aanwezige Oracle mogelijkheden. Overigens is dat iets dat ook zonder vraag naar generieke oplossingen voorkomt. Te vaak kom je zelfgeschreven functionaliteit tegen voor bijvoorbeeld locking, foreign keys of row-level security. Dat heeft Oracle al lang voor ons gebouwd, en bijna altijd volledieger en efficiënter. Gebruik de aanwezige technologie waarvoor het geschikt en bedoeld is.

Bouw

Ook dat wat gebouwd wordt, moet voldoen aan de performance-eisen, uiteraard op basis van het ontwerp. Bij de bouw spelen algoritmes en SQL statements een grote rol. Worden efficiënte algoritmes gebruikt en worden de juiste queries geschreven? Verkeerde algoritmes en slechte queries kunnen een perfect ontworpen applicatie om zeep helpen. Om met een eenvoudige voorbeeld te beginnen. Er blijken veel manieren te zijn om na te gaan of een kolom van een tabel een bepaalde waarde bevat. Veel meer dan ik in het dagelijks leven zou vermoeden. Als ik op zaterdagochtend bij de bakker bij mij op de hoek vraag of hij nog croissants heeft, kijkt hij waar ze liggen, bij ons is dat in een mand in de vitrine. Ziet hij daar een croissant, dan is het antwoord 'Ja'. Wat hij nooit, maar dan ook echt nooit, doet, is ze eerst allemaal gaan tellen en vervolgens als dat aantal groter dan nul is, pas 'Ja' zeggen. Toch is dat iets dat in Oraclecode wel gebeurt, regelmatig kom je namelijk een stuk code tegen als:

```
...
SELECT COUNT(*)
INTO   l_aantal_broodjes
FROM   broodjes
WHERE  naam = 'CROISSANT';

IF l_aantal_broodjes > 0
THEN
  <doe iets>
END IF;
...
```

Soms wordt er zelfs door geteld tot over de paar miljoen, alleen maar om te weten of er misschien één rij is aan de voorwaarden voldoet. Dat hoeft dus niet, er zijn zelfs meerdere alternatieven. In eerste instantie verwacht je iets als:

```
DECLARE
  CURSOR c_brd
  IS
  SELECT naam
  FROM   broodjes
  WHERE  naam = 'CROISSANT';
  r_brd c_brd%ROWTYPE;
...
BEGIN
...
  OPEN c_brd;
  FETCH c_brd INTO r_brd;
  IF c_brd%FOUND
  THEN
    <doe iets>
  END IF;
  CLOSE c_brd;
...
```

Die extra regels code en het gebruik van cursoren blijkt voor sommige ontwikkelaars lastig. Maar een mooi alternatief is deze query op DUAL met een EXISTS, een constructie die in Headstart veel gebruikt is.

```
...
SELECT COUNT(*)
INTO   l_aantal_van_dual
FROM   dual
WHERE  EXISTS (SELECT 1
               FROM   broodjes
               WHERE  naam = 'CROISSANT');

IF l_aantal_van_dual = 1
THEN
  <doe iets>
END IF;
...
```

Veel uitwerkingen van algoritmes in PL/SQL bevatten overbodig werk. Zo komt het voor dat berekeningen of functies binnen een loop herhaald worden uitgevoerd, terwijl ze slechts eenmalig vooraf aan de loop hadden kunnen worden uitgevoerd. Wat ook voorkomt is het uitvoeren van meerdere queries aan het begin van een PL/SQL blok, terwijl vanwege de logica in het blok zelf een aantal queries misschien helemaal niet uitgevoerd hoeft te worden. Ook elders in de applicatie kun je een teveel aan werk tegenkomen. Niet zelden wordt voor een scherm eerst alle data uit de database opgehaald om vervolgens in de applicatie een selectie te maken van de paar rijen die de gebruiker wil zien. Waarom dan niet eerst de selectie opgeven en op basis daarvan de veel kleinere verzameling aan data ophalen? Betrekkelijk nieuwe en mooie mogelijkheden in SQL als de LAG en LEAD functie en de CUBE en ROLLUP operatoren verleidden sommige ontwikkelaars tot complexe SQL om in een enkel statement alles aan data en lay-out te regelen, terwijl tools over het algemeen heel efficiënt column breaks kunnen uitvoeren en (sub)totalen kunnen berekenen.

In het gebruik van Virtual Private Database (VPD) komt ook regelmatig teveel werk voor, uiteindelijk een vorm van inefficiënte algoritmes. Met VPD is row-level security te regelen in Oracle. Heel kort door de bocht wordt een policy gelegd op een tabel, die policy regelt voor de tabel wat specifieke gebruikers wel en niet mogen met de rijen van die tabel. Bijvoorbeeld elke medewerker mag alleen zijn eigen rijen zien en een manager mag vervolgens alleen wijzigingen uitvoeren op rijen van zijn ondergeschikten. De policy wordt uitgevoerd met behulp van een PL/SQL functie. Wat je ziet is dat die functie per rij

bepaalt wat de functie is van de huidige gebruiker en wie zijn eventuele ondergeschikten zijn. Een gebruiker verandert gedurende een sessie niet van identiteit en de organisatie wijzigt ook niet zo vaak. Meestal is het voldoende om eenmalig, bijvoorbeeld bij het inloggen, te bepalen wat de functie van de gebruiker is. Dan wordt zo'n stuk code per dag misschien honderd keer per dag uitgevoerd, in plaats van honderd keer per minuut.

Bij het schrijven van SQL en PL/SQL is het nuttig te bedenken wat in SQL kan en moet en wat in PL/SQL. Ontwikkelaars die gedrild zijn in de traditionele Oracle ontwikkeltools als Designer en Forms hebben vaak de neiging alle selecties met cursoren op te willen lossen. Voor DML op meerdere rijen is dat zelden efficiënt. Dan is een eenvoudig update of delete statement minder werk voor Oracle. Met cursoren worden vaak eerst de benodigde rijen één voor één opgehaald en wordt vervolgens nog eens per rij een DML statement uitgevoerd, in plaats van één enkel DML statement voor de verzameling rijen waar het om gaat. Zo'n werkwijze met cursoren betekent extra werk.
Naast algoritmes zijn SQL statements zelf natuurlijk van groot belang. In een Oracle database gebeurt al het bewerken en ophalen van data middels SQL. Omdat over het algemeen in projecten er wel enige aandacht voor SQL tuning is, wil ik daar hier nu niet te diep op ingaan. Wel handig is het om te beseffen dat SQL tuning op zich ook weer een breed onderwerp is. Het gaat er uiteindelijk om met de aanwezige Oraclefeatures en SQL mogelijkheden het statement zo te formuleren dat het zo efficiënt mogelijk kan worden uitgevoerd. Voor elk SQL statement zijn meerdere alternatieven te verzinnen die hetzelfde resultaat geven, soms doelmatiger maar niet altijd.

Testen

Alleen ontwerpen en bouwen is niet voldoende. De performance zal getest moeten worden. Wordt daadwerkelijk aan de performance-eisen voldaan en waar zitten eventuele bottle-

necks? Voor de performancetest moet uiteraard van een realistische dataverzameling gebruik gemaakt worden, en van realistische testscenario's. De resources mogen tijdens het testen wel iets minder zijn dan tijdens productie, dat maakt de pijnpunten beter zichtbaar. Omdat alleen de gebruiker de performance ervaart, is het het best om de eisen te formuleren op basis van activiteiten van de gebruiker. Het is niet zondermeer duidelijk wat een specifieke activiteit van een gebruiker, bijvoorbeeld tijdschrijven, technisch tot gevolg heeft op de databaseserver en in de database. Je kunt zelden exact aangeven welke PL/SQL modules en SQL statements uitgevoerd zijn vanwege bepaalde functionaliteit. Om die relatie te kunnen leggen is het belangrijk om code meetbaar en traceerbaar te maken.

Oracle heeft zelf met de Oracle Wait Interface een belangrijke stap gezet met instrumentatie van de database, dit heeft databasetuning veel geholpen. Daarnaast biedt Oracle veel functionaliteit aan om Oraclecode te instrumenteren. Toine van Beckhoven gaat hier in zijn artikel Effective performance for Developers and Users ook op in. Door instrumentatie in te bouwen in de applicatie is deze meetbaar voor performancetesten en geschikt voor monitoring, ook niet onbelangrijk. Uiteindelijk moet de applicatie een x aantal jaar mee (met x>0) en gedurende die tijd willen beheerders wel weten hoe het met de applicatie gaat, ook qua performance.

Performance tijdens het ontwikkelen

In Oracle Insights: Tales of the Oak Table beschrijven leden van het Oak Table network wat er gebeurt als ontwikkelaars tijdens eisen, ontwerp en bouw geen rekening houden met performance. De praktijk wijst uit dat het ook anders kan. Het vakgebied Performance Engineering houdt zich hier mee bezig, met het zodanig ontwikkelen van software dat aan de performance-eisen voldaan kan worden. Voor het hele proces zijn en worden best-practices, design patterns en anti-patterns verzameld en beschreven. Performance Engineering beperkt zich niet tot Oracle, specifiek voor Oracle zijn natuurlijk ook weer veel best-practices beschreven, onder andere door Tom Kyte op zijn website Ask Tom, in zijn artikelen en in zijn boeken. Projecten waar tijdens ontwerp en bouw bewust rekening gehouden wordt met de gevraagde performance, leveren minder performanceproblemen op en bovendien problemen die eenvoudiger zijn op te lossen. Het maakt hierbij niet uit met welke methode de software ontwikkeld wordt, of het nu CDM, RUP of Scrum of methode X is, uiteindelijk komen deze stappen op een of andere manier wel terug. Ook gaat het op voor zowel grote als kleine applicaties. Het geldt ook voor wijzigingen aan bestaande systemen, ook al staat veel vast, de nieuwe functionaliteit juist nog niet. Een wijziging is weer een klein project. Ontwikkelaars kunnen tijdens het hele ontwikkeltraject de performance beïnvloeden, hoe eerder ze dat doen, hoe meer effect dat heeft en hoe eenvoudiger het is uit te voeren.

Referenties

- Toine van Beckhoven, Effective performance for Developers and Users, OGh Visie Winter 2008, Jaargang 13, Nummer 2, 2008
- Mogens Norgaard, et al., Oracle Insights: Tales of the Oak Table, Berkeley CA, Apress, 2004.
- Thomas Kyte, Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solutions, Berkeley CA, Apress, 2005

Jan Lucas van der Ploeg is database en software engineer bij Ordina.