Antipatterns hulp voor performance

Weinig bewust gebruikt in Oracle projecten

In elk softwareproject gaan zaken goed en zaken fout. Niet alleen zie je in meerdere projecten dezelfde dingen goed gaan, je ziet ook dat fouten worden herhaald. Je kunt patronen waarnemen. Antipatterns kunnen helpen om performanceproblemen te voorkomen. Vervolgens helpen ze bij het oplossen daarvan. Indien je voor een specifiek probleem weet met welk antipattern je te maken hebt, kun je eenvoudig nagaan waar de oorzaak ligt. Ook kun je zo eenvoudig bepalen wat mogelijke oplossingen of workarounds zijn.

Design patterns en antipatterns zijn voor software engineers geen onbekenden. Ze zijn te vinden op het niveau van code en op het niveau van ontwerp en zelfs over het hele ontwikkelproces. Binnen Oracle-projecten worden ze weinig bewust gebruikt. Er zijn specifieke performance antipatterns, op www.perfeng.com staat een aantal artikelen van Connie Smith en Lloyd Williams hierover. Veel van deze antipatterns komen ook voor in Oracle-omgevingen. Met de beschrijving van een aantal typische Oracle performance antipatterns als de Things table, Not invented here en The ramp wil ik laten zien hoe het gebruik hiervan helpt bij het oplossen en voorkomen van performanceproblemen.

Things table

De Things table komt vaak voort uit een overdreven (en onverantwoorde) behoefte aan flexibiliteit. In extreme vorm van de Things table wordt één enkele tabel de tabel voor de applicatie. Zie hiervoor bijvoorbeeld het boek Tales of the Oak Table. De Things table bestaat bij voorkeur uit twee kolommen: waarde en betekenis. Als het moet kan er een extra kolom bij als categorie of datatype. Op deze manier kan altijd alle data in die ene tabel. De klanten, adressen, medewerkers, producten, orders, facturen, noem maar op. Nooit hoeven kolommen toegevoegd, laat staan tabellen. Een iets minder extreme variant wordt teruggevonden in Oracle EBS, waarbij tabellen voorkomen met kolommen als attribute1 t/m attribute16 en global_attribute1 t/m global_attribute16.

Wat is het probleem?

De Things table heeft direct gevolgen voor het bouwen en onderhouden van een applicatie. Query's en andere code worden uitermate complex. Een datamodel met één tabel is eigenlijk geen datamodel. De Oracle-database is een relationele database. Alles in ontwerp en bouw gaat er vanuit dat de applicatie ontworpen en gemaakt is op basis van een relationeel of dimensioneel datamodel. Alles? Ja alles, ook de optimizer.

Om een query uit te voeren, maakt de optimizer een executieplan. Dit geeft aan hoe de query wordt uitgevoerd. Het geeft de joinvolgorde aan, de indexen die gebruikt worden etcetera. Om zo'n executieplan te maken gebruikt de optimizer gegevens over de data. Deze zijn terug te vinden in de metagegevens in de database. Datatypes zijn van belang, not-null constraints, foreign key constraints, unique constraints, aantallen van rijen, grootte van kolommen, noem maar op. Met een Things table beschikt Oracle over bijzonder weinig, zeg maar gerust te weinig, gegevens om een efficiënt executieplan te maken.

Daarnaast vergroot de Things table de kans dat de benodigde data op een en dezelfde plek in de database terecht komt. Hij vergroot de kans op hot spots. Vergelijk het met de IKEA. Bij de IKEA staan de artikelen logisch verspreid over de winkel. Je hebt afdelingen voor woonkamer, slaapkamer, keuken, studeerkamer, kinderkamer met daartussen brede paden voor het 'doorgaand verkeer'.

Afhankelijk van wat een klant wil, kan hij naar de juiste afdeling, en nog rustig winkelen ook. Doordat klanten naar verschillende afdeling gaan zorgt deze indeling naar kamersoort voor een redelijke verspreiding van de klanten over het winkeloppervlak. IKEA had er voor kunnen kiezen alle artikelen qua ruimte zo efficiënt mogelijk in het pand te stoppen. Dus geen brede looppaden, maar alles lekker bij elkaar en op elkaar. Kasten, bedden en banken onderop en bestek en handdoeken bovenop.

Dan kan er wel meer in de winkel, maar wordt het met meer

Om hotspots te voorkomen staan bij IKEA de artikelen logisch verspreid over de winkel. Je hebt afdelingen voor woonkamer, slaapkamer, keuken, elektra, etc. Daartussen brede paden voor het 'doorgaand verkeer'.

mensen aardig vol in die ruimte. Bovendien wordt het lastig om snel de juiste spullen te bemachtigen. Dit geldt helemaal als de voorraad bijgevuld moet worden en de nieuwe collectie binnenkomt. Iets dergelijks gebeurt ook met de Things table. Alles wordt op een grote hoop gegooid: laatste orders, nieuwe klanten, nieuwe producten. Dat betekent een grotere kans dat meerdere query's net dezelfde datablokken willen lezen of schrijven. Ofwel ze moeten op elkaar wachten.

Wat is er tegen te doen?
De oorzaak voor de Things table zit niet in de code, maar overduidelijk in het ontwerp. Daar ligt dus ook de echte oplossing: redesign. Afhankelijk van de exacte problemen en graad van Things table, kunnen workarounds worden gemaakt. De flexfields in EBS zijn niet half zo erg als de Things table uit Tales of the Oak Table. Mogelijke workarounds omvatten het hele arsenaal dat Oracle te bieden heeft: indexen, materialized views, outlines, hulptabellen, extra ijzer, enzovoorts. De workaround moet uiteraard gekozen worden op basis van feiten en cijfers.

Not invented here

Not invented here komt er op neer dat aanwezige functionaliteit niet wordt gebruikt, maar dat deze functionaliteit juist wordt nagebouwd of nagebootst. Vaak komt dit doordat architecten, ontwerpers en ontwikkelaars de bestaande functionaliteit niet kennen of soms niet willen kennen. Ook gebeurt het dat bestaande functionaliteit bewust niet wordt gebruikt, omdat de software op alle soorten databases (Oracle, MySQL, SQL Server) moet kunnen werken. Je ziet dit in Oracle onder

andere bij constraints, autorisatie, VPD, Advanced Queueing en Fine Grained Auditing. Dit zijn allemaal beschikbare features die in praktijk nogal eens nagebouwd worden.

Wat is het probleem?
Evident is dat Not invented here met name een probleem is voor beheer. Uiteindelijk is het extra code die problemen oplevert bij patches en upgrades en waar geen ondersteuning van de leverancier voor bestaat. Bovendien ontbreekt vaak adequate documentatie. Ook performanceproblemen kunnen hier hun oorsprong vinden. Dit komt omdat de extra code altijd minder efficiënt is dan de door Oracle ingebouwde code. Je kunt in een applicatie zelf de foreign key constraints bouwen. Maar dat kan alleen met complexe, want generieke, query's. Uiteindelijk betekent dit altijd extra werk voor de database. Extra werk dat voorkomen had kunnen worden.

Wat is er tegen te doen?
Als benodigde functionaliteit aanwezig is, bouw het dan niet zelf. Alleen en alleen dan, als deze functionaliteit er niet is, of het eventueel niet doet, bouw dan zelf iets. Gebruik de bestaande functionaliteit. Gebruik een sequence als je een uniek nummer nodig hebt voor een primary key. Schrijf geen functie om de hoogste waarde van die primary key kolom op te halen, om dan een hogere waarde als volgend nummer te kunnen gebruiken.

The ramp

The ramp treedt op als de performance afneemt naarmate de applicatie langer in gebruik is. Dit is helaas een bekend verschijnsel bij databases. Als de applicatie nieuw is loopt alles nog als een zonnetje en naarmate de applicatie langer in gebruik is, wordt een aantal processen of taken steeds trager.

Wat is het probleem?
Afnemende performance bij The ramp komt door de toename van de hoeveelheid data. Hoe langer de applicatie in gebruik is, hoe meer data. Telecombedrijven verwerken duizenden telefoongesprekken per dag, webwinkels verkopen honderden artikelen per dag. Tabellen worden zo zeer groot. Hierdoor kan een query veel werk hebben om alle 100 miljoen rijen te lezen van de ordertabel op zoek naar orders uit 2009 van mevrouw Hafkamp uit Losser. Een goed datamodel helpt natuurlijk wel, maar is geen voldoende voorwaarde tegen The ramp. Zonder gebruik van indexen en met slecht geschreven SQL zullen query's langer lopen als de tabellen groter worden. Efficiënt gebruik van grote hoeveelheden vereist een intelligente ordening en een goede manier om er bij te kunnen.

Wat is er tegen te doen?
De oorzaak van The ramp kan liggen in het datamodel. In die gevallen moet daar ook de oplossing worden gezocht. Zeer vaak is de oorzaak echter slechte SQL of PL/SQL. Dit is te verhelpen met SQL tuning en efficiënte PL/SQL algoritmes. Bij zeer grote hoeveelheden data kan ook worden gekeken naar de gebruikte database objecten, tabellen kunnen bijvoorbeeld worden gepartitioneerd.

Gebruik van antipatterns

Hoe kunnen antipatterns helpen bij het voorkomen of oplossen van performanceproblemen? Om te beginnen helpen ze bij het categoriseren. Een probleem is nooit in alle opzichten uniek, het past altijd in een bepaald patroon. Dit helpt bij het vinden van de oorzaak en de mogelijke oplossing. Oplossingen die werken voor andere problemen binnen die categorie zijn te vertalen naar het specifieke probleem. Daarnaast helpen antipatterns bij de communicatie met andere betrokkenen over oorzaak en oplossing.
Als een DBA kan aantonen dat een probleem het gevolg is van het Things table antipattern helpt dat om de projectleider, ontwerper en gebruiker duidelijk te maken waar de echte oplossing ligt. De DBA kan nu laten zien dat redesign de echte oplossing is en dat meer ijzer en meer indexen beperkte work-arounds zijn.
In de literatuur en op internet zijn antipatterns uitgebreid beschreven, inclusief oplossingen en alternatieven. Maak hier gebruik van. De performance antipatterns waar je in Oracle-omgevingen mee te maken hebt, komen ook voor in niet-Oracle omgevingen. Oplossingen die daar bedacht zijn en zich hebben bewezen, zijn te vertalen naar Oracle-oplossingen.

Ook kunnen zo performanceproblemen worden voorkomen. Een aantal antipatterns is al waar te nemen in de architectuur en het ontwerp, zoals Not invented here. Bij review van een ontwerp kunnen deze herkend worden en dan kan ook tijdig worden ingegrepen om ze te voorkomen of verminderen. Performance antipatterns helpen bij het voorkomen en het oplossen van performanceproblemen, juist ook in software tegen Oracle-databases.

Literatuur

- *M. Norgaard, et al., Oracle Insights: Tales of the Oak Table, Berkeley CA, Apress, 2004.*
- *C. U. Smith and L. G. Williams, Software Performance AntiPatterns, Proc. 2nd Int. Workshop on Software and Performance, Ottawa, September 2000.*
- *C. U. Smith and L. G. Williams, New Software Performance Antipatterns: More Ways to Shoot Yourself in the Foot, Proc. CMG, Reno, December 2002.*

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