Presentatie Tapio Lahdenmäki op RedDatabase Symposium

'Van 3-ster indexen tot Analytics'

Door Toine van Beckhoven, Jom IT

Het NH Hotel in Den Haag was 10,11 en 12 maart jl. het toneel van het eerste RedDatabase Symposium in Nederland, geheel gericht op de Oracle database server en zowel voor DBA's als ontwikkelaars. Organisatoren Rick van der Lans en Klaas Brant van RedStone Innovations hadden een reeks excellente sprekers uit of naar Nederland gehaald. Verdeeld over de drie dagen waren er in totaal acht sessies en elke spreker hield een presentatie van een volle dag.

Als deelnemer aan twee van de drie dagen doe ik graag verslag van dit symposium, waarvan ik een vervolg zeker kan aanbevelen. Het gebeurt niet vaak dat we in Nederland de keus hebben uit zo'n select gezelschap sprekers. De algehele indruk was goed: niet alleen de presentaties, maar ook de randvoorwaarden - locatie, koffie en thee, de uitgebreide lunch, het presentatiemateriaal en parkeerkaarten - waren dik in orde. Tijdens de pauzes en lunch was er uitgebreide mogelijkheid om te praten met de andere deelnemers en sprekers. Een greep uit de presentaties tijdens de drie dagen: 'Analytics and Data Warehousing using Oracle11g and Oracle BIEE' door Mark Rittman, onze eigen Lucas Jellema die een prima presentatie hield met als titel 'Advanced Application Development with the Oracle Database' en het zeer goed ontvangen 'Query Optimizer and SQL Tuning' door performance expert Christian Antognini. In dit verslag op één van de andere gevolgde sessies, een presentatie van Tapio Lahdenmäki. In de volgende OGh Visie volgt een verslag van de presentatie van Daniel Fink ('Effective performance for Developers and Users'), alsmede een uitgebreid artikel over Graham Wood, wiens presentatie tijdens het symposium niet door kon gaan, maar die op 10 april voor de OGh presentatie heeft gegeven over DBTime Oracle Performance Tuning (zie voor een korte impressie pagina 31).

Educatieve waarde

Het laatste jaar ben ik zelf heel erg veel bezig geweest met Oracle Tuning en troubleshooting. In die hoedanigheid was ik naar Den Haag gekomen voor de presentaties van Tapio Lahdenmäki en Daniel Fink. Persoonlijk voelde ik bevestigd dat ik me op de juiste boeken en artikelen heb gestort en de juiste focus heb gelegd het afgelopen jaar, want met name Daniel Fink's presentatie was een feest der herkenning. Heb ik dan kostbare tijd weggegooid die dag? Neen, integendeel: ik kon mezelf een volle dag toetsen aan wat een groot specialist op Tuning gebied te zeggen heeft over Oracle en vragen stellen die ik eerder niet naar tevredenheid beantwoord had gezien. De presentatie van Tapio Lahdenmäki's had een grote educatieve waarde. Gedurende de dag werden we een aantal malen uitgedaagd met opdrachten en vrijwel iedereen had opvallend grote moeite met het vinden van de juiste antwoorden. Dit gaf aan hoe het meest bekende performance 'handvat' dat we hebben binnen de database, indexen, voor zelfs heel ervaren DBA's nog voor hoofdbrekens kan zorgen.

Index design: ondergewaardeerd

Tapio Lahdenmäki is een 62-jarige Fin met een lange staat van dienst. Zijn expertise richt zich niet op de Oracle database alleen, maar op databases in het algemeen en dan specifiek indexering. Kennis van het goed ontwerpen van indexen is belangrijk voor elk type database. Het is zoals gezegd misschien wel het eerste wat je te binnen schiet als je denkt aan het optimaliseren van SQL statements: 'De juiste indexen aanmaken!'. Helaas is dat in de praktijk niet zo simpel en de kennis ervan ontoereikend. Veel applicaties zijn sterk overgeindexeerd (uit huidige praktijkervaring wil ik hier zeker Siebel noemen: dat heeft tabellen met meer dan 80 indexen, met vaak 80-90% nooit gebruikte!); andere applicaties zijn te weinig geïndexeerd en draaien al jaren met te zware SQL statements die met de juiste indexen in een turbo zouden veranderen.
Veel mensen zijn zich niet bewust van de negatieve bijwerkingen van teveel indexen en zeker zoveel mensen maken suboptimale indexen. Tapio begon zijn presentatie dan ook met het noemen van oorzaken van te langzame programma's. Naast problemen met indexen (missende indexen, belangrijke columns die missen in de index en een verkeerde columnvolgorde) zijn dat onnodige tabel opdelingen of gemis aan denormalisatie (denormalisatie is vaak noodzakelijk voor acceptabele performance). Daarnaast onjuiste optimizer kostencalculaties of wel en niet

ees_1_semifat op [department_id, email, hire_date] levert bij bovenstaand SQL statement het volgende Explain plan op (Oracle 9.2.0.8 database):

Plan

```
SELECT STATEMENT

2 TABLE ACCESS BY INDEX ROWID HR.EMPLOYEES

1 INDEX RANGE SCAN NON-UNIQUE HR.IDX_EMPLOYEES_1_SEMIFAT
Access Predicates: 'EMPLOYEES'.'DEPARTMENT_ID'=TO_NUMBER(:v_dept_id) AND 'EMPLOYEES'.'HIRE_DATE'>TO_DATE(:v_hire_date,'dd-mm-yyyy') AND 'EMPLOYEES'.'EMAIL' LIKE :v_email
Filter Predicates: 'EMPLOYEES'.'EMAIL' LIKE :v_email
```

Ofwel, zoals gezegd bij de derde bullet: department_id en hire_date zijn matching columns (deze worden alleen genoemd in de Access Predicates van het index toegangspad, maar Email wordt als enige column genoemd in de Filter predicates tijdens de index toegang. Omdat hire_date een range predicate in de WHERE clause bevat, is email geen matching column meer, maar een screening column.

Welke index levert nu een ster op voor het eerste criterium? Met welke columns aan het begin van de index is de te lezen index slice nu zo klein mogelijk?

Een index op alleen id_department? Nee, het is weliswaar een matching column, maar er zijn nog twee andere predicaten die simpel genoeg zijn om te refereren aan een matching column: op email en hire_date. Een index beginnende met id_department en als tweede column email OF als tweede column hire_date kunnen beide een ster voor dit eerste criterium opleveren. Welke van beide het allerbeste is, hangt weer af van de filter factor (FF) van beide: waarschijnlijk heeft e-mail hier een hogere filter factor (is selectiever) dan het predicate op hire_date, maar alleen onderzoek hiernaar kan dit uitwijzen. Een regel die Tapio hierop noemt is dat je altijd moet uitgaan van de 'worst case input': wat is de selectiviteit van het predicate met de slechtst denkbare inputwaardes?

De tweede ster is te verdienen met het elimineren van een sorteeroperatie door het gebruik van de index. Van nature zijn records in een tabel ongesorteerd, maar rowid's in een index staan op volgorde van de waardes van de indexcolumns. Het is dus mogelijk om gebruik te maken van dit fenomeen en een mogelijk dure sorteeroperatie te vermijden door een juiste index. Dit kan met name in OLTP achtige applicaties waar een scherm zo snel mogelijk gevuld moet zijn met de eerste n records, maar waarbij die records tevens gesorteerd moeten zijn op een bepaalde (set van) columnwaarde(s), enorm van pas komen en de gebruikersperceptie van zo'n scherm danig verbeteren.

Moderne RDBMS-en als Oracle hebben ingebouwde optimalisering om de kosten van dure random reads te verminderen: Data block prefetching in Oracle verzamelt de index entries waarmee de tabel bezocht moet worden en start indien mogelijk meerdere parallelle I/O's om de tabel te benaderen. Dit heeft echter als bijwerking dat een gebruikte 'tweede ster' index om een sortering te vermijden toch een SORT ORDER BY te zien kan geven. Verwarrend, maar goed om te weten.
In bovenstaand voorbeeld is maar één set van leading columns in de index die een tweede ster verdient voor dit criterium en dat is een index beginnend met [department_id, hire_date]: records zullen uit deze index op volgorde van hire_date terugkomen gegeven een department_id (en aangezien er een equality predicate staat op department_id, is er maar 1 department_id dat terugkomt. Een index beginnend met [department_id, email] levert dus weliswaar wel een ster op voor het eerste criterium, maar geen voor het vermijden van een sortering. Vergelijk maar eens de volgende Explain plans:

Met SORT ORDER BY:

Plan

```
SELECT STATEMENT CHOOSE

3 SORT ORDER BY

2 TABLE ACCESS BY INDEX ROWID HR.EMPLOYEES
Filter Predicates: 'EMPLOYEES'.'HIRE_DATE'>TO_DATE(:v_hire_date,'dd-mm-yyyy')

1 INDEX RANGE SCAN NON-UNIQUE HR.IDX_EMPLOYEES_1_DEPTIDEMAIL
Access Predicates: 'EMPLOYEES'.'DEPARTMENT_ID'=TO_NUMBER(:v_dept_id) AND 'EMPLOYEES'.'EMAIL' LIKE :v_email
Filter Predicates: 'EMPLOYEES'.'EMAIL' LIKE :v_email
```

En zonder SORT ORDER BY:

Plan

```
SELECT STATEMENT CHOOSE

2 TABLE ACCESS BY INDEX ROWID HR.EMPLOYEES
Filter Predicates: 'EMPLOYEES'.'EMAIL' LIKE :v_email
1 INDEX RANGE SCAN NON-UNI-
```

gebruiken van bind variabelen. Ook gaf hij aan dat er wel heel vreemde 'best practices' genoemd worden in literatuur of aanwezig zijn in de hoofden van veel mensen: 'Er mogen maximaal 5 indexen per tabel worden aangemaakt' of 'columns moeten in de index gesorteerd worden op volgorde van cardinality'. Deze en andere wijsheden houden heden ten dage vaak geen stand (meer) omdat er andere belangrijke overwegingen meespelen.

Drie sterren indexen

De presentatie, genoemd 'Better database performance with well designed indexes' ging vervolgens in op de verschillen tussen random reads (single block reads) en sequential reads (multiblock reads), die door Tapio in het kader van de rest van de presentatie Random Touches (TR) en Sequential Touches (TS) werden genoemd. Wellicht opmerkelijk is dat fabrikanten van storage systemen erin geslaagd zijn om de snelheid van sequential reads de laatste jaren aanmerkelijk omhoog te krijgen, maar dat random reads grofweg even snel (of traag) zijn gebleven en onder meer door de steeds maar groter wordende disks zelfs nog trager zijn geworden of gaan worden.

Aangezien we bij het gebruik van indexen vaak te maken hebben met random reads, zeker als we een table access doen via rowid's verkregen vanuit een index scan, kunnen we ons wellicht beter voorstellen dat het heden ten dage duurder is om indexen te gebruiken in vergelijking met 'vroeger'. De cost based optimizer, mits voorzien van alle informatie (denk zeker ook aan system statistics, geïntroduceerd in 9i), is zich hiervan bewust.

De rule based optimizer niet! Maar uiteraard, indexen die optimaal zijn voor een gegeven statement, zijn nog altijd onmisbaar voor de snelste verwerking. Wat maakt echter een index optimaal voor een gegeven statement? Ontmoet de 'drie sterren index' van Tapio: een index krijgt een ster voor elk van de volgende criteria gegeven een statement:

- om alle van belang zijnde records te lezen moet het indexdeel ('slice') dat gescand wordt zo dun mogelijk zijn
- de van belang zijnde records komen in de juiste volgorde terug zodat een sort wordt vermeden
- het aantal index columns is voldoende: geen noodzaak meer om de tabel te benaderen

Noot vooraf: het is niet altijd mogelijk een 3-sterren index te ontwerpen voor een gegeven query!

Wat betekent het dat het indexdeel dat gescand wordt zo klein mogelijk is? In feite is dat het geval als zoveel mogelijk in de predicates aanwezige columns 'Matching columns' zijn. Tapio maakt onderscheid tussen Matching columns en Screening columns in de predicates van een query. Dit onderscheid is het best uit te leggen aan de hand van een voorbeeld:

We nemen de Employees tabel van het Oracle HR demoschema en een query als volgt:

```
SELECT    first_name,
          last_name,
          job_id
     FROM hr.employees
   WHERE department_id = :v_
dept_id
     AND hire_date > TO_DATE
(:v_hiredate, 'dd-mm-yyyy')
        AND email like :v_
email
ORDER BY hire_date;
```

- Een index op deze tabel met [departement_id] als enige column levert voor deze query 1 matching column op: departement_id
- Een index op deze tabel met in volgorde de columns [hire_date, departement_id] levert voor de query 1 matching (hire_date) en 1 screening (departement_id) column op
- Een index op deze tabel met in volgorde de columns [department_id, hire_date, email] levert 2 matching columns op (departement_id, hire_date) en 1 screening column op (email)

Dat dit verwarrend is, is goed te begrijpen: waarom is departement_id in de tweede index een screening column en in de eerste en derde index een matching column? Dit heeft alles te maken met een aantal regels en deze zijn als volgt:

Ga de indexcolumns na van links naar rechts (van eerste naar laatste column):

1. Is er in de WHERE clause minimaal 1 voldoende eenvoudig predicate dat ernaar refereert? Zo ja, dan is de column een matching column. Zo nee, dan is deze column, maar ook alle erop volgende columns, geen matching column
2. Als het predicate een range predicate is (>, <, >=, <=, BETWEEN, LIKE), dan zijn de volgende columns geen matching columns
3. Iedere column na de laatste matching column is een screening column zolang er maar een voldoende simpel predicate is dat refereert aan die column.

Bij het lezen van een Explain plan is het wellicht al eens opgevallen dat er zoiets is als Access Predicates en Filter Predicates. Hieraan zie je ook het verschil tussen matching en screening columns:

Een index genaamd idx_employ-

```
QUE HR.IDX_EMPLOYEES_1_DEP-
TIDHIRE
Access Predicates: 'EMPLOY-
EES'.'DEPARTMENT_ID'=TO_
NUMBER(:v_dept_id)      AND
'EMPLOYEES'.'HIRE_DATE'>TO_
DATE(:v_hire_date,'dd-mm-
yyyy')
```

Maar beter is nog om ook screening columns in de index op te nemen: het aantal (dure) random reads naar de tabel wordt daarmee gereduceerd. Dit verkrijgen we door de index genoemd in het allereerste Explain plan in dit artikel: idx_employees_1_semifat op [department_id, hire_date, email]. De naam semifat heeft te maken met het feit dat alle in het predicate aanwezige columns in de index aanwezig zijn, maar niet alle columns in het hele SQL statement. Voor dit laatste belanden we bij criterium drie voor de 3 sterren index.
Die laatste ster is te verdienen als de index dusdanig is dat de tabel niet meer bezocht hoeft te worden: alle informatie die de query nodig heeft komt uit de index. Dit voordeel is in de meeste gevallen het grootst van de drie 3-sterren criteria: het aantal random reads wordt erdoor beperkt.

Een index waarmee een tabel bezoek vermeden wordt, noemt Tapio een 'fat' index: zo'n index bevat relatief veel columns. In bovenstaand voorbeeld zou de tot nu twee sterren opleverende index op [id_department, hire_date, email] uitgebreid moeten worden met alle columns gebruikt in de query: [..., last_name, first_name, job_id] waarmee de 3 sterren index voor deze query wordt:

```
CREATE INDEX idx_employees_1_
fat ON employees(department_id,
hire_date, email, last_name, first_
name, job_id);
```

Het explain plan, zonder SORT ORDER BY en zonder TABLE ACCESS BY INDEX ROWID (random reads):

Plan

```
SELECT STATEMENT CHOOSE

1 INDEX RANGE SCAN NON-UNI-
QUE HR.IDX_EMPLOYEES_1_FAT
Access Predicates: 'EMPLOY-
EES'.'DEPARTMENT_ID'=TO_
NUMBER(:v_dept_id)      AND
'EMPLOYEES'.'HIRE_DATE'>TO_
DATE(:v_hire_date,'dd-
mm-yyyy')   AND   'EMPLOY-
EES'.'EMAIL' LIKE :v_email
Filter Predicates: 'EMPLOY-
EES'.'EMAIL' LIKE :v_email
```

In zijn presentatie behandelde Tapio nog veel meer aspecten van index design, zoals indexen in een query met joins, de bijwerkingen van extra indexen of extra columns in een bestaande index (op inserts/updates/deletes), free space management en de gunstige effecten van denormalisatie. Hij ging uitgebreid in op de QUBE methode, wat staat voor Quick Upper Bound Estimate om de kosten van een zekere index voor een gegeven query te berekenen in termen van doorlooptijd en CPU verbruik. Het is teveel en te complex om in een artikel als dit te behandelen. In dat kader kan ik het boek dat Tapio samen met Michael Leach heeft geschreven, aanbevelen (zie onderaan artikel). Het geeft veel achtergrond informatie over dit belangrijke onderwerp.

Resumerend

Het RedDatabase symposium voldeed voor mij zeer aan de verwachting en voorzag in een behoefte aan presentaties door 's werelds beste Oracle experts. Het heeft mijn toch al niet geringe interesse in Oracle Performance Tuning nog verder aangewakkerd en me op boeken en artikelen gewezen waar ik eerder ten onrechte geen aandacht aan zou hebben besteed. Het Oracle RDBMS is een complex maar ontzettend krachtig en intelligent stuk software en dagen als deze doen je er nog meer waardering voor en kennis van krijgen. In de volgende OGh Visie zoals beloofd een tweede artikel naar aanleiding van dit symposium, want Daniel Fink had ook heel wat mooie en interessante dingen te vertellen.

Link: http://www.tapio1.com
Boek: 'Relational Database Index Design and the Optimizers', Tapio Lahdenmäki and Michael Leach, 2005

Toine van Beckhoven (http://www.jom-it.nl/) is zelfstandig en allround Oracle specialist. Oracle Performance Tuning en PL/SQL hebben zijn grootste interesse.