‘Eindgebruikers zijn olifanten’

Driedaags seminar door Ton Kyte

Op 8, 9 en 10 februari verzamelden 65 zeer ervaren Oracle professionals zich in het Domstad College te Utrecht om Tom Kyte te horen spreken. Dit seminar bewoog zich op het grensvlak van de DBA, de ontwikkelaar en de architect. Belangrijke thema’s waren performance, applicatie-architectuur en database design, waardevolle nieuwe features in 10g. De leidraad vormde het boek Effective Oracle by Design van Tom Kyte.

Het seminar was georganiseerd door Lex de Haan, Oracle-veteraan en lid van het OakTable Network (www.oaktable.net). Dit is een genootschap van deels zelfbenoemde, maar wijd en zijd ook als zodanig erkende Oracle-goeroes en -profeten. Lex heeft het voornemen de topsprekers van dit OakTable genootschap naar Nederland te halen om zodoende een serie geavanceerde seminars aan te bieden. Deze verrijking voor het aanbod op de Nederlandse markt lijkt erg aan te slaan: Oracle-professionals die meer presentaties en trainingen geven dan volgen en met over het algemeen vele jaren Oracle-kennis op zak hebben kwamen zelfs vanuit België, Duitsland en Engeland naar dit eerste seminar. De meerwaarde zit hem dan ook niet alleen in de spreker maar ook in de contacten met ‘soortgenoten’ gedurende de pauzes. Het volgende OakTable seminar zal plaatsvinden op 18, 19 en 20 oktober, met Steve Adams - Oakie en top-DBA uit Australië.

Oracle-goeroe Tom Kyte is vice president core technologies voor Oracle government, education & healthcare.

Tom Kyte is zo’n ‘knight of the Oak Table’, of kortweg Oakie: hij is vice president core technologies voor Oracle government, education & healthcare. Hij heeft zeventien jaar Oracle-ervaring - sinds versie 5.1.5c. Tom maakt deel uit van de Oracle Sales organisatie maar is allereerst techneut en databasefreak. Zijn AskTom! (asktom.oracle.com) website trekt meer dan een miljoen bezoekjes van meer dan 25.000 Oracle ontwikkelaars per maand.

Question authority

Met zo’n visitekaartje ben je wellicht snel geneigd een spreker zeer serieus te nemen. Kyte opende het seminar echter met het motto: *question authority*. Vraag altijd door, vraag naar het waarom achter een stelling en laat het bewijzen. Geloof nooit iets of iemand zonder meer, niet je boeken, niet je collega’s en zeker jezelf niet.

Waarheden uit het verleden kunnen met nieuwe versies allang achterhaald zijn. Baseer je oordeel niet op een enkele, anekdotische ervaring. Stel dat ‘aliens’ alles wat ze weten over mensen van televisie leren - hebben ze dan een goed beeld van onze wereld? Probeer zelf dingen uit, test ze, overtuig jezelf op een wetenschappelijke manier van de correctheid van keuzes, verwachtingen en acties. Het is ook heel belangrijk, om eerst alles in een testomgeving te testen, alvorens de productie-omgeving en dus de eindgebruikers met nieuwe dingen te confronteren. Volgens Tom zijn eindgebruikers net olifanten: ze onthouden heel goed en heel lang wat je allemaal fout hebt gedaan. Hij noemde daarvan een aantal sterke voorbeelden: klanten die

hun recovery-procedure pas probeerden toen er een recovery nodig was, of de gemeente Chicago die op vrijdag vroeg of er nog aandachtspunten waren rondom 10g, want in het weekend zou er een (niet geteste) upgrade van een 9i database gaan plaatsvinden. Een groot deel van de zaal gaf toe wel eens een patch te hebben doorgevoerd, bijvoorbeeld op suggestie van Oracle Support, zonder deze eerst in een testomgeving uit te proberen.

Mistige mythen

Er doen veel richtlijnen de ronde die keer op keer herhaald worden. 'De export en import van een database moet regelmatig plaatsvinden om de organisatie van je database te verbeteren en geeft altijd een betere performance' is er zo één. Of: 'je indexen moet je frequent rebuilden'. Of: 'je moet nooit een SELECT INTO gebruiken maar altijd een expliciete Cursor: OPEN, FETCH INTO, CLOSE'.

Tom Kyte is allergisch voor dat soort uitspraken. Niet alleen omdat deze drie allemaal onzin zijn, maar vooral ook vanwege de vanzelfsprekendheid en onnadenkendheid waarmee dit soort quasi-waarheden worden geciteerd. Ook al zijn stellige uitspraken ooit waar geweest, iedere nieuwe databaseversie geeft een nieuwe waarheid.

Het SELECT INTO-verhaal heb ik ook vaak gehoord. Misschien heb ik het zelf wel doorverteld. Bij mijn weten staat het zelfs in de Oracle CDM-standaarden. Het heeft alleen nooit gegolden voor Stored PL/SQL, maar uitsluitend voor client side PL/SQL in Forms 2.3 en 3.0. Er zijn nog meer verhalen die de ronde doen en niet (meer) kloppen:

- (NOT) IN mag nooit en moet vervangen worden door een constructie met (NOT) EXISTS. Ooit was dit waar, tegenwoordig doet de Optimizer dat zelf en komt in beide gevallen met een anti-join stap in het execution plan.
- `Select * from table where nvl(:bindvariable, column) = column` is zeer ongunstig. Bepaal in de applicatie in een if-then-else of bindvariable NULL is of een waarde heeft en voer dan één van beide query's uit: `select * from table of select * from table where column=:bindvariable`. Dat zou je denken, maar de 10g CBO denkt dat ook en maakt zelf een execution-plan met twee taken waarvan er maar één wordt uitgevoerd.
- De buffer cache hit-ratio moet groter zijn dan 99% - dat is vergelijkbaar met wanneer je zegt, dat je auto terug moet naar de garage, wanneer die niet op 99% van het maximale vermogen rijdt. Ratio's zijn sowieso alleen indicatoren, nooit een doel op zich. Kyte is eigenlijk alleen echt geïnteresseerd in de parse ratio's.
- De meest selectieve kolom moet voorop staan in de index - ook dat is allang niet meer het geval. "Where a=:a and b=:b" wordt precies hetzelfde uitgevoerd voor index (A,B) als voor index (B,A), onafhankelijk van selectiviteit. In de meeste gevallen worden rijen vaker via minder selectieve kolommen benaderd dan via de meest selectieve, dus ook daarvoor moeten de minst selectieve vooraan staan. Sterker nog, om COMPRESSION effectief te gebruiken zou de meest selectieve kolom, met de meeste verschillende waarden, juist achteraan in de index definitie behoren te staan!
- `Select count(1)` is sneller dan `select count(*)` - is niet waar en is ook nooit waar geweest. In 8i werd `select count(1)` zelfs geoptimaliseerd tot `select count(*)`!
- Full table scans zijn slecht en het gebruik van indexen is altijd goed: te onzinnig voor woorden.
- Je moet vaak committen en je transacties niet te groot maken. Deze stelling is zeer beperkt houdbaar: een commit betekent een redo log synch - de transactie gegevens moeten door de redo-log file geschreven worden en dat kost tijd; vaak committen gaat vrijwel zeker ten koste van de performance. Bovendien wordt er meer redo en undo log geschreven. Belangrijker misschien nog: de logische samenhang van de transactie wordt misschien doorbroken. Transaction level integrity kan niet langer geïmplementeerd worden als de logische transactie met meerdere commits wordt opgesplitst.
- Je moet aparte tablespaces hebben voor tabellen en indexen - complete flauwekul, behalve misschien in het geval van Transportable Tablespaces waar je alleen de data wilt transporteren en de indexen beter opnieuw kunt opbouwen. Het heeft wel zin, met name met het oog op nieuwe read-ahead schijflees-technieken, om onderscheid te maken tussen tablespaces waaruit vooral multiblock read (full table scans) plaatsvinden en tablespaces waaruit vooral single block reads worden gedaan - tabellen die via indexen worden benaderd.

Doorgewinterde Oracle-professionals kwamen zelfs vanuit België, Duitsland en Engeland naar dit eerste OakTable-seminar.

• Als je in PL/SQL meerdere rijen uit een cursor-resultaatset wilt gaan verwerken moet je "FETCH cursor BULK COLLECT INTO collection" gebruiken in plaats van een gewone for-loop. In 8i en 9i is dat hartstikke waar, maar in 10g wordt, achter de schermen en automatisch, een bulk collect uitgevoerd voor iedere *for cursor loop*. Deze is zelfs iets efficiënter dan een bulk-collect die je zelf expliciet programmeert.

De trend die we zien in 10g is dat we weer meer kunnen gaan zeggen wát we willen van de database in plaats van hóe de database het moet gaan doen. We zeggen gewoon SELECT INTO of we zeggen gewoon NOT IN, en de database optimaliseert voor ons hoe hij het antwoord verkrijgt.

'Whoops' management

Eén van de termen die sterk met de 9i en 10g releases van de database wordt geassocieerd is Flashback. In versie 9i dook de Flashback Query op, in 10g werd dit uitgebreid met Flashback Table, Flashback Table to before DROP (UNDROP) en Flashback Database. Al deze features hebben te maken met 'in het verleden kijken' of zelfs naar het verleden terugkeren. Die mogelijkheid kan erg waardevol zijn voor 'whoops' management: in situaties waar de ontwikkelaar of DBA iets doet en direct inziet: "whoops... dat had ik niet moeten doen".
De Flashback Query heeft een heel eenvoudige syntax in 9iR2 en 10g. Om de data in een tabel te zien zoals die er gisteren uitzagen (vergeet even alle DML-operaties op die tabel die sindsdien hebben plaatsgevonden) luidt de syntax:

```
Select * from Table as of Timestamp sysdate -1

Met behulp van Flashback Query kunnen ongewenste
DML-operaties selectief ongedaan gemaakt worden:

Update emp
Set sal = sal * 1.+ (0.5* dbms_random.value)
Where job='CLERK'
/
Commit;
-- WHOOPS - dat hadden we niet moeten doen.

Update emp
Set sal = ( select sal from emp emp_old AS OF timestamp sysdate
- 5/60/24 where emp.empno = emp_old.empno)
Where job = 'CLERK'
/
Commit;
--- pffffff
```

Mensen die bezwaren hebben tegen Flashback Query's - heel vaak DBA's die veronderstellen dat het wel heel veel overhead zal opleveren of resources zal kosten om de voor flashback noodzakelijke informatie bij te houden - zien een nogal belangrijk punt over het hoofd: *iedere* query die je sinds Oracle 4 in de database hebt uitgevoerd, is een flashback query geweest. `Select * from emp` wordt door de database uitgevoerd als `select * from emp as` of <SCN op het moment van de start van deze query>. Dat is zo ongeveer de definitie van read-consistency!

De meerwaarde van het seminar bestond onder meer uit de informele contacten met 'soortgenoten'.

Flashback Table is evenals de Flashback Query gebaseerd op de UNDO-data die ook voor normale read consistency wordt gebruikt. De afstand in tijd die met flashback kan worden overbrugd, is afhankelijk van de hoeveelheid UNDO-data die beschikbaar zijn. Je kunt in 10g een *retention guarantee* instellen die ervoor zorgt, dat de gewenste *retention period* - het aantal dagen dat je met Flashback kunt teruggaan - wordt afgedwongen. Dit kan er overigens wel voor zorgen dat in het geval de UNDO-tablespace vol is, er geen transacties meer kunnen worden verwerkt door de database tot de oudste undo data buiten de retention period vallen. Flashback wordt door vrijwel alle Alter Table-statements geblokkeerd - je kunt niet flashbacken tot voor het meest recente Alter Table-statement.
Met Flashback Table kan een tabel worden teruggebracht in de situatie van het aangegeven moment. De Oracle 10g gaat bij een Flashback Table de compenserende SQL genereren op basis van Flashback Query's. Flashback Table genereert en executeert UNDO SQL - je kunt zelfs DML-triggers laten afgaan tijdens de Flashback-operatie.
Flashback Table to before Drop is een heel ander mechanisme. Het stelt ons in staat om - vanaf Oracle 10g - een tabel die is gedropped te 'undroppen'. Dit kan zolang de extents die door de tabel werden gebruikt nog niet zijn hergebruikt voor een andere tabel. Dat kan heel kort zijn, maar ook heel erg lang.

Flashback Table to before Undrop hangt samen met de 10g Recycle Bin. Deze bin kost geen extra ruimte! Het enige dat in 10g gebeurt, is dat tabellen die worden gedropt wel onthouden worden en dat hun extents als laatste uit de freespace worden gebruikt om andere tabellen te bedienen.

Weer een ander mechanisme is Flashback Database, gebaseerd op de Flashback Recovery Area, een file area waar ondermeer Flashback Logs worden bijgehouden met de oude versies van gewijzigde datablokken. De Flashback Logs in combinatie met Redo Log files maken een snelle Point in Time recovery mogelijk: de oude datablokken worden teruggezet en met behulp van de Redo Log files wordt naar een exact tijdstip vooruit gerold. Flashback Database geeft de mogelijkheid om een desastreuze 'truncate table' vrij eenvoudig te corrigeren - mits de database in archive log mode is en flashback logging aan staat:

- Shutdown abort.
- Mount database, Flashback database to <just before truncate> en `ALTER DATABASE OPEN RESETLOGS;`
- Exporteer de tabel die is ge-truncate.
- Shutdown immediate.
- Mount Database, Flash Forward (dat wil zeggen: flashback database <naar nu>), en `ALTER DATABASE OPEN RESETLOGS;`
- Importeer de tabel; de database is nu ook weer beschikbaar voor alle andere operaties.

Junior DBA in je database

De vraag komt vaak op of het vak van DBA gaat verdwijnen. Dit seminar geeft daar een oorverdovend 'nee' als antwoord op. Ja, het vak van de DBA gaat veranderen. En het goede nieuws: het wordt er alleen maar interessanter op. Zoals Kyte het stelt: de 10g database bevat als het ware een junior DBA. Routinematige taken worden automatisch uitgevoerd. Basale taken die in het verleden het grootste deel van de tijd van DBA's in beslag namen (tablespace en datafile management, user management, rechttoe rechtaan back-up en recovery procedures, tunen van slechte SQL statements) worden nu door de database zelf gedaan.

De DBA gaat zich bezig houden met de interessantere zaken: optimale beschikbaarheid door inzet van Flashback Database, Stand-by Database en Data Guard of zelfs Grid, het tunen van algoritmes in plaats van query's, het ontwerpen van optimale opslagstructuren als Index Organized Tables, Sorted Hash Clusters en Partities en het testen van upgrades, patches en nieuwe releases. Dit vereist van DBA's hoogwaardige kennis van geavanceerde faciliteiten in de database. Het is niet ondenkbaar dat er een schifting zal optreden - junior DBA's worden door 10g weggeautomatiseerd en senior DBA's kunnen steeds meer en steeds grotere databases op een steeds uitdagender wijze gaan administreren.

Is tuning dead?

De aandacht bij tunen verschuift van statement tuning naar het tunen van algoritmes: als een query honderdduizend keer wordt uitgevoerd omdat een ontwikkelaar een procedurele loop gebruikt waar een enkele SQL Set operatie had volstaan, is er maar weinig winst te halen met optimalisatie van die query. De winst zit in het vervangen van de procedurele logica door SQL Sets-gewijze operaties.
Kyte hamerde er op - zoals hij dat ook doet in zijn boeken - dat alles wat je in SQL kunt oplossen, in PL/SQL vrijwel altijd langzamer en duurder tot veel langzamer en onuitvoerbaar zal zijn. Declaratieve constraints verdienen altijd de voorkeur boven client-side of procedurele server-side oplossingen. Gebruikmaken van specialistische functionaliteit die in de database aanwezig is, biedt altijd voordelen boven eigen procedurele logica. Hij gaf een voorbeeld van een consultant in Canada die uit een tabel met locaties een selectie wilde maken van alle locaties binnen een straal van vijftien mijl om een ziekenhuis. Met een gewone relationele query zou deze query een join van de tabel hebben vereist; het query-resultaat zou jaren op zich laten wachten. Met inzet van Oracle Spatial werd dit tot minuten-werk teruggebracht.
Wat Kyte betreft zijn hints alleen maar waardevol om de Cost Based Optimizer informatie te geven - niet om de CBO te forceren een bepaalde methode te hanteren. Waardevolle hints zijn bijvoorbeeld FIRST_ROWS(n) (vooral bij OLTP) en ALL_ROWS (met name voor batches), CHOOSE, (NO)REWRITE, DRIVING_SITE (voor gedistribueerde query's over database links), (NO)PARALLEL, (NO)APPEND en CARDINALITY. Deze laatste is niet gedocumenteerd en kan worden gebruikt om voor WITH clauses, Temporary Tables of Table Functions een indicatie aan de CBO te geven hoeveel rijen ongeveer te verwachten zijn. Dit kan grote invloed hebben op het Execution Plan, zoals te zien is in het volgende voorbeeld:

```
WITH good_customers as
( select c.* from many_many_many_customers c, debts d where c.id =
d.ctr_id and d.amount < 5000)
select /*+ CARDINALITY(100) */...
from  t1, t2, t3, good_customers
```

Aan diverse stokpaardjes van Kyte ben ik in dit verslag niet eens toegekomen:

- Instrumentatie in je code - zorg dat debug en trace in je applicatie ingebouwd is en dat je deze altijd kunt aanzetten van buiten je applicatie, juist in productie-omgevingen!
- Probeer niet database-onafhankelijk te bouwen - je code zal op elke database middelmatig tot slecht performen. Bouw een API-laag per database, tussen applicatie en database, die voor iedere database geoptimaliseerd is.

Read-consistency en multiversioning

De tweede dag van het Tom Kyte seminar stond in het teken van read-consistency en multiversioning van de data. Dit is sinds release 4 een fundamenteel element in de architectuur van de Oracle database en één van de wezenlijke verschillen tussen Oracle en concurrenten als DB2 en SQLServer. Read consistency zorgt ervoor dat de resultaatset van een query consistent is. Alle rijen in de resultaatset zie je zoals ze bestonden op het moment dat de query werd gestart. Als de query een uur loopt en in dat uur worden er vele wijzigingen in de data aangebracht, dan nog zie je in het query-resultaat een consistente set zoals die bestond bij het begin van de query. Deze faciliteit vereist uiteraard dat meerdere versies van gewijzigde rijen moeten worden bewaard door de database - de UNDO informatie.

Voor het geval iemand denkt dat dit geen goed idee is: doe het volgende gedachte-experiment in een database *zonder* read-consistency - bijvoorbeeld DB2 of SQLServer. Er is een grote tabel met bankrekeningen met kolommen KlantId, RekeningNummer en Saldo. Een klant met id = 5 heeft twee rekeningen, A en B, de één met 1000 euro, de ander met 2000 euro. We starten een query die alle saldi per klant gaat sommeren. Vlak nadat de query gestart is en de eerste rekening van klant 5 heeft verwerkt wordt een overmakingsopdracht van deze klant verwerkt: 500 euro van rekening A naar rekening B. Als de query rekening B verwerkt staat daar inmiddels geen 2000 maar 2500 euro. Het totaalresultaat voor klant 5 komt daarmee op 3500 euro, hetgeen overduidelijk niet correct is (zie tabel 1).

Read-consistency leidt tot een merkwaardig fenomeen bij update operaties: update-statements updaten soms meer rijen dan ze updaten. Dit fenomeen bestaat al sinds Oracle 4, maar Tom Kyte werd zich dat pas bewust in 2003. Neem de volgende situatie in gedachten: Tabel T met kolommen I en V, respectievelijk int en varchar2 (zie tabel 2). Op deze tabel is een before-row update trigger gedefinieerd:

```
create or replace trigger t_bru
before update on t
for each row
begin
  dbms_output.put_line('BRUpd trigger on T fires for i='||:old.
i||','||:new.i);
end;
```

Nu vindt het scenario plaats, dat beschreven staat in tabel 3. De uitkomst was bijzonder verrassend en voor dit toch tamelijk Oracle-geletterde publiek nogal spectaculair. In sommige omstandigheden kan dit effect - dat gezien Oracle's benadering van read-consistency volledig correct is - tot vervelende en lastig herkenbare consequenties leiden.

Als de row-level update-trigger[1] een niet-transactie gebonden operatie uitvoert, zoals het versturen van een e-mail, het aanroepen van een webservice of het schrijven naar een pipe of een file, zal deze actie mogelijk dubbel of eventueel ten onrechte worden uitgevoerd. Row-level update en delete triggers worden wel eens gebruikt als oplossing voor het mutating-table probleem. In de row-level trigger wordt het row-id in een PL/SQL Collection geplaatst die in een after statement level trigger wordt uitgelezen en verwerkt. Door de herstart van het update-statement kunnen rijen dubbel in deze collection terecht komen. De logica in de Statement-Level trigger zou daar op moeten anticiperen. Het is ook mogelijk dat een grote batch-operatie, die altijd prima in zijn batch-window paste, door een ogenschijnlijk heel kleine verandering in de omgeving opeens royaal over zijn tijdvenster heenschiet. Stel dat op negentig procent van een zware update in deze batch een rij wordt aangetroffen, die door een ander proces is geüpdate na het begin van het update-statement, dan kan het zijn dat de zware update van voor af aan moet beginnen.

[1] *Dit kan ook een delete-trigger zijn, want voor delete operaties geldt hetzelfde effect.*

Tabel 1

Tijd	(tussentijds) Query-resultaat	Rekening A	Rekening B
		1000	2000
De query start met lezen en leest ondermeer rekening A: 1000 euro.	1000	1000	2000
De klant maakt 500 euro over van A naar B.	1000	500	2500
De query leest de rij voor rekening B; in een niet read-consistent read levert dat inmiddels 2500 euro op en deze wordt in het query-resultaat verwerkt.	3500	500	2500

Tabel 2

I	V
1	A
2	B
3	C

Tabel 3

	Sessie 1	Sessie 2
		Update t Set i = 6 Where i = 3
De update in sessie 1 loopt tegen het lock van sessie 2 op de rij voor i= 3 aan. De updates op rijen 1 en 2 zijn gedaan.	Update t Set i=i+1 Where i<10 BRUpd trigger on T fires for i=1,2 BRUpd trigger on T fires for i=2,3	
Sessie 2 geeft het lock vrij. Sessie 1 constateert dat de kolom i – waaraan wordt gerefereerd in de where-clause van dit update statement – is gewijzigd; de updates die al gedaan zijn worden teruggerold. Er wordt een select for update uitgevoerd om alle benodigde rijen voor de update met lock te verkrijgen.		Commit; BRUpd trigger on T fires for i=2,6 1 row updated.
Als alle locks verkregen zijn wordt de update opnieuw uitgevoerd. Uiteraard gaan nu de row-level triggers opnieuw af.	BRUpd trigger on T fires for i=1,2 BRUpd trigger on T fires for i=6,7 BRUpd trigger on T fires for i=3,4 3 rows updated.	

- Doe niet buiten de database wat je in de database kunt doen - vooral de data-integriteit niet!
- Probeer niet slimmer te zijn dan de database - zeg wat je wilt, maar niet hoe de CBO het moet doen.
- Gebruik bind-variabelen - het (hard-)parsen van query-statements is de duurste operatie die je de database kunt opleggen.
- Ontwerp voor performance - probeer niet om pas als de applicatie al af is, de performance nog eens aan te verbeteren. Performance en schaalbaarheid moeten ontwerpdoelen zijn, vanaf het prille begin van het ontwerp. Je kunt niet substantieel performance verbeteren als de applicatie-architectuur niet is afgestemd op de eisen voor wat betreft responstijd, batch-window en schaalbaarheid.

Conclusie

Zo denk je dat je een heleboel van Oracle weet, en zo weet je dat er een heleboel is dat je eigenlijk nog helemaal niet zo goed wist. Ik kan niet spreken voor de andere 64 deelnemers aan dit seminar, maar ik weet wel dat dit ongeveer mijn gevoel was toen ik na drie dagen stevig gemangeld weer buiten stond. Kyte is een zeer ervaren Oracle-ontwikkelaar en architect. Hij noemt zichzelf nadrukkelijk geen DBA. Hij kan ogenschijnlijk eenvoudige situaties analyseren en dan nog een keer ontleden en doorspitten, zodat je het gevoel krijgt dat je tenslotte dan echt begrijpt waarom de CBO een bepaald plan maakt of waarom een bepaald verschijnsel optreedt. Zijn verhaal had niet altijd een duidelijke lijn en structuur, maar vrijwel ieder onderdeel was de moeite waard en bracht nieuwe inzichten.
Kyte en Java blijft een wat ongemakkelijke combinatie. Wat mij betreft zou hij zich daar ook maar niet aan moeten wagen. Hij zit er niet voldoende in om met doorwrochte argumenten te komen over de situaties waarin je al dan niet Java zou moeten inzetten. Hij komt niet verder dan een vrij goedkope ridiculisering van Java en al wat daarmee samenhangt. Ik vermoed dat een deel van het publiek dit ook graag hoort, aangezien Java wel eens als nieuw en daardoor bedreigend wordt gezien.
Seminars als deze zijn een verademing voor de gevorderde Oracle-professional, zowel door de kennis en ervaring van de spreker, gevoegd bij een zeer gemotiveerd publiek in combinatie met uitstekende faciliteiten - zoals voor iedere deelnemer een gratis exemplaar van het boek 'Effective Oracle by Design' en een memory stick met de scripts van alle getoonde demo's. Ik kijk uit naar het volgende Oak Table seminar.

Lucas Jellema (jellema@amis.nl) is sinds 2002 werkzaam bij AMIS Service in Nieuwegein, als Expertise Manager Technologie en Technisch Consultant. Daarvoor werkte hij ruim acht jaar bij Oracle, ondermeer binnen het iDevelopment Center of Excellence. Hij houdt zich onder meer bezig met Java, XML/XSLT en andere webtechnologie als ook de Oracle database en tools voor applicatie ontwikkeling.