XML-opslag en indexeringsopties in Oracle 11g

Een use case gebaseerde vergelijking

In 'XML en Oracle11g RDBMS' van Jan Vissers (nummer 4, september 2007) is de theorie van binary XML-opslag en het indextype XMLIndex al behandeld. In het artikel dat u nu voor zich heeft, wordt deze nieuwe functionaliteit in de praktijk aan een eerste test onderworpen, en vergeleken met andere opslag- en indexeringsopties voor XML data.

De tests zijn vereenvoudigde use cases uit een demotoepassing gebouwd met het op de Oracle XML-database gebaseerde contentmanagementsysteem Tangelo. Het is een eerste test in 'default omstandigheden'. Er is geen tuning gedaan door middel van fysiek databaseontwerp en/of hints in de queries.

Opslagopties

In Oracle 11g zijn er vier opslagopties voor XML data:

CLOB	de XML-data worden opgeslagen als een CLOB
objectrelationeel	de XML-data worden 'shredded' opgeslagen in een door de database op basis van een XML Schema-definitie gegenereerde object-relationele datastructuur
hybrid	de XML-data worden deels opgeslagen in een objectrelationele datastructuur en deels als CLOB-waarden (gestuurd door annotaties in de XML Schema-definitie)
binary XML	nieuw in 11g, opslag in een binair *post-parse* formaat (zie 'XML en Oracle11g RDBMS, Optimize 4, september 2007)

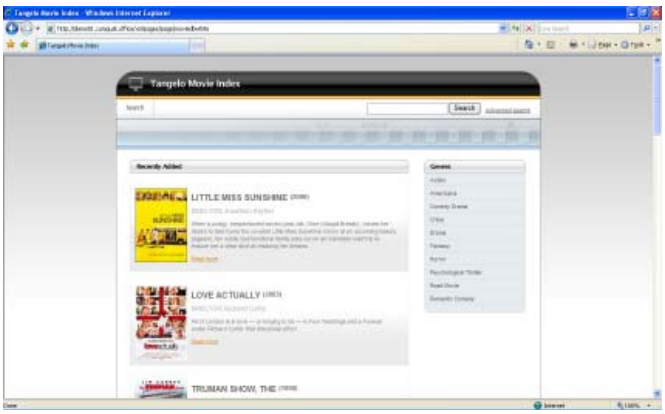
Indexeringsopties

De opties voor indexering in Oracle 11g voor XML data die getest zullen worden zijn:

- geen index; als "0-meeting"
- function based index
- (algemene) XMLIndex (nieuw in Oracle 11g)
- XMLIndex met secondary index
- XMLIndex voor specifieke XPath expressies
- Oracle Text context index voor full text search

Movie DB demo

De tests in dit artikel zijn gebaseerd op vereenvoudigde use cases uit de Movie DB demo. Deze toepassing is gebouwd met Tangelo XML Documents en Tangelo XML Pages en is een *dynamische* website waarop informatie over films gepubliceerd wordt. De content wordt opgeslagen als XML data in de Oracle XML database. De informatie over een film bestaat uit een overview met een aantal *gestructureerde* gegevens zoals titel, regisseur, genres en lengte. En een *ongestructureerde* samenvatting ("plot synopsis") van de film. De tests in dit artikel richten zich op het opslaan en bevragen van de overview. Hieronder is hier een voorbeeld van te zien.



Afbeelding 1. De 'witte versie' van de Movie DB demo.

```
<movie year="2006" runTime="99" AMGrating="40" MPAArating="R">
   <title>Little Miss Sunshine</title>
   <director id="83"/>
   <genres>
      <genre id="25"/>
      <genre id="23"/>
      <genre id="40"/>
      <genre id="18"/>
   </genres>
   <plotSynopsis>
      <p>
```

```
         <summary>When a pudgy, bespectacled seven-year-old, Olive
(<person
         id="88">Abigail Breslin</person>), voices her desire to take
home the

coveted Little Miss Sunshine crown at an upcoming beauty pageant, her
         wildly dysfunctional family sets out on an interstate road
trip to
         ensure her a clear shot at realizing her dreams.</summary>
            In former music video directorial team <person
id="83">Jonathan
         ...
      </p>
      <p>
         ...
      </p>
      ...
   </plotSynopsis>
</movie>
```

Voorbeeld XML data.

Afbeelding 2. De 'zwarte versie' van de Movie DB demo. Zelfde XML data, maar een andere presentatie door gebruikmaking van XSLT.

Een aantal grote documenten (0,98MB, 5,87MB en 31,8MB) wordt als testdata gebruikt met overviews van films en grote aantallen kleine, door een stored function gegenereerde 'documenten'.

XML Schema-definities registreren

Om bij het aanmaken van de tabellen te kunnen verwijzen naar de XML Schema-definitie moet deze eerst geregistreerd worden in de database. Dit wordt gedaan met de REGISTERSCHEMA-procedure in de DBMS_XMLSCHEMA-package. Het registreren van een XML Schema-definitie voor gebruik in combinatie met binary XML-opslag of objectrelationele opslag verschilt. Voor binary XML-opslag moet REGISTERSCHEMA aangeroepen worden met de waarde FALSE voor parameter 'gentypes'. Bovendien moet via parameter 'options' aangegeven worden dat de XML Schema-definitie voor binary XML-opslag gebruikt gaat worden. Om onderscheid te kunnen maken wordt dezelfde XML Schema-definitie (in directory MOVIES) onder twee verschillende URL's geregistreerd: 'http://www.cumquat.nl/xsd/movie' voor objectrelationele opslag en 'http://www.cumquat.nl/xsd/movie_binary' voor binary XML-opslag. Bij het registreren van een XML Schema-definitie vanuit een directory moet de karakterset gespecificeerd worden.

```
BEGIN
   DBMS_XMLSCHEMA.REGISTERSCHEMA('http://www.cumquat.nl/xsd/movie'
   ,                             BFILENAME('MOVIES', 'movie.xsd')
   ,                             csid => NLS_CHARSET_ID('AL32UTF8')
   );
END;
Registereren XML Schema-definitie voor objectrelationele opslag.

BEGIN
   DBMS_XMLSCHEMA.REGISTERSCHEMA('http://www.cumquat.nl/xsd/movie_bina-
ry'
   ,                             BFILENAME('MOVIES', 'movie.xsd')
   ,                             csid => NLS_CHARSET_ID('AL32UTF8')
   ,                             options => DBMS_XMLSCHEMA.REGISTER_
BINARYXML
   ,                             gentypes => FALSE
   );
END;
```

Registeren XML Schema-definitie voor binary XML-opslag.

Tabellen aanmaken

Na het registeren van de XML Schema-definities kunnen de tabellen aangemaakt worden. Voor CLOB-opslag wordt de kolom 'movie_info' als een XMLTYPE-kolom gedefinieerd en wordt met de XMLTYPE COLUMN-clause aangegeven dat deze als een CLOB moet worden opgeslagen. Er wordt niet verwezen naar een geregistreerde XML Schema-definitie.

```
CREATE TABLE xml_movies_clob
( id            NUMBER(10,0)
, movie_info    XMLTYPE
)
  XMLTYPE COLUMN movie_info STORE AS CLOB
Aanmaken tabel voor CLOB-opslag.
```

Het aanmaken van een tabel waarbij de XML-informatie objectrelationeel wordt opgeslagen, is vrijwel identiek. Nu wordt echter OBJECT RELATIONAL als opslagoptie gespecificeerd en wordt verwezen naar de (eerste) geregistreerde XML Schema-definitie.

```
CREATE TABLE xml_movies_object_relational
( id            NUMBER(10,0)
, movie_info    XMLTYPE
)
```

```
  XMLTYPE COLUMN movie_info STORE AS OBJECT RELATIONAL
  XMLSCHEMA "http://www.cumquat.nl/xsd/movie" ELEMENT "movie"
```

Aanmaken tabel voor objectrelationele opslag.

Ten slotte wordt een tabel aangemaakt voor binary XML-opslag. BINARY XML wordt aangegeven in de XMLTYPE COLUMN-clause en er wordt verwezen naar de voor binary XML-opslag geregistreerde (tweede) XML Schema-definitie.

```
CREATE TABLE xml_movies_binary
( id            NUMBER(10,0)
, movie_info    XMLTYPE
)
  XMLTYPE COLUMN movie_info STORE AS BINARY XML
  XMLSCHEMA "http://www.cumquat.nl/xsd/movie_binary" ELEMENT "movie"
```

Aanmaken tabel voor binary XML-opslag.

Laadtijd

Voordat de performance van diverse queries gemeten wordt, is het interessant om te kijken of er verschillen in laadtijd zijn bij de verschillende opslagopties. In theorie wordt de laadtijd beïnvloed door: 'shredden' (bij objectrelationele opslag), parsen (bij binary XML-opslag), en bijwerken index(en).
De verwachting is dat de verschillen alleen meetbaar zijn bij het laden van *grote aantallen* (kleine documenten) en *grote documenten*. In theorie zou CLOB-opslag zonder index(en) het snelste moeten zijn.
Bij het laden van één klein document (180 Kbytes) (vanuit een directory) is er geen meetbaar verschil; alle documenten laden binnen 0,1 seconde.
Bij het laden van een groot aantal kleine (met een stored function gegenereerde) 'documenten' zijn er wel verschillen meetbaar. In de volgende tabel staan de meetresultaten: de gemiddelde tijd (van enkele metingen) en de 'factor' die het verschil aangeeft in verhouding tot de snelste opslagoptie.

	CLOB	*objectrelationeel*	*binary XML*
laden van 1.000 files	*0,76 sec.* *1*	*1,23 sec.* *1,6*	*2,03 sec.* *2,7*
laden van 10.000 files	*6,17 sec.* *1*	*10,93 sec.* *1,8*	*17,86 sec.* *2,9*

Meetresultaten laden van grote aantallen documenten in tabel zonder index.

Ook bij een groot document (6.013 Kbytes, 5,87MB) zijn er duidelijke verschillen in laadtijd. Bij een nog groter document (32.567 Kbytes, 31,8 MB) worden de verschillen nog groter

	CLOB	*objectrelationeel*	*binary XML*
laden file van 5,87MB	*5,73 sec.* *2,9*	*8,35 sec.* *4,3*	*1,95 sec.* *1*
laden file van 31,8MB	*1 minuut 40 sec.* *8,2*	*4 minuten 18 sec.* *21,1*	*12,23 sec.* *1*

Meetresultaten laden een groot document in tabel zonder index.

Opmerkelijk is de korte laadtijd voor binary XML van een groot document, terwijl bij het laden van een groot aantal kleine documenten binary XML de meeste tijd nodig heeft. Bovendien valt het op dat het laden van een groot document bij CLOB-opslag erg lang duurt.
Vervolgens is gemeten of de verhoudingen van de laadtijd anders komen te liggen als er een XMLIndex op de tabel gedefinieerd is; zie 'Zoeken op XML data', later in dit artikel.

	CLOB	*objectrelationeel*	*binary XML*
laden van 1.000 files	*3,07 sec.* *1*	*XMLIndex niet mogelijk*	*7,76 sec.* *2,5*
laden van 10.000 files	*1 minuut 34 sec.* *1*		*1 minuut 42 sec.* *1,1*

Meetresultaten laden van grote aantallen documenten in tabel met index.

Uit de metingen blijkt dat de verhouding bij grote aantallen anders komt te liggen, waarschijnlijk omdat het bijwerken van de index relatief (ten opzichte van de insert) veel tijd kost en niet veel verschilt per opslagoptie.

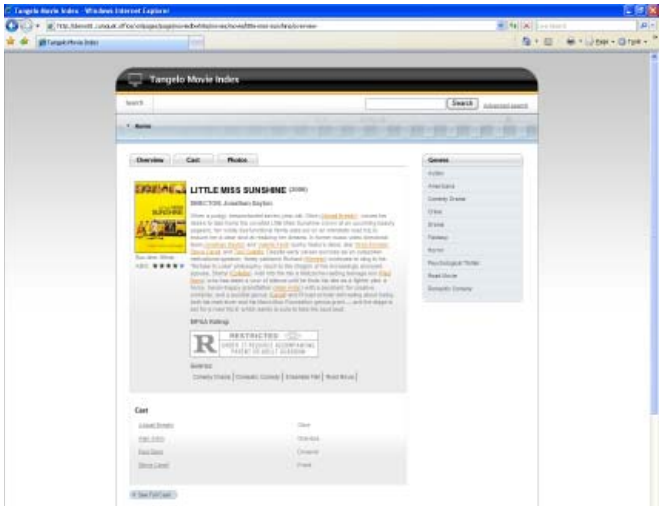
	CLOB	*objectrelationeel*	*binary XML*
laden file van 5,87MB	*56,57 sec.* *1,3*	*XMLIndex niet mogelijk*	*43,06 sec.* *1*
laden file van 31,8MB	*Afgebroken (1)* *n.v.t.*		*3 minuten 39 sec.* *1*

Meetresultaten laden een groot document in tabel met index.

Bij het laden van grote documenten lijkt dezelfde 'nivellering' op te treden als bij grote aantallen kleine documenten door de overhead van het bijwerken van de XMLIndex.
Het is mogelijk een XMLIndex *asynchroon* bij te werken. Via de paramaters clause van het CREATE INDEX-commando kan dit aangegeven worden. De index wordt dan niet *synchroon* (direct) bijgewerkt bij het uitvoeren van het DML-commando, maar bij een commit of om de zoveel tijd. De index kan dan weliswaar tijdelijk stale zijn, maar DML-commando's worden sneller uitgevoerd.

XMLType-waarde toekennen

Het toekennen van een XMLType-waarde aan een variabele is een van de basisacties die gedaan wordt in een toepassing. In de Movie DB democase gebeurt dit ondermeer bij het tonen van de informatie van een bepaalde film.



Afbeelding 3. Tonen informatie van een bepaalde film.

Om de (eventuele) verschillen te meten bij het toekennen van een XMLType-waarde wordt als eerste de volgende test gebruikt. Merk op dat in de where-clause van de query de XMLType-kolom niet voorkomt. Op deze manier wordt alleen het (eventuele) verschil van de toekenning gemeten tussen de verschillende opslagopties (het 'vinden' van de waarde zou ongeacht de opslagoptie hetzelfde moeten zijn).

```
DECLARE
   l_movie_info   XMLTYPE;
BEGIN
   SELECT movie_info
   INTO   l_movie_info
   FROM   xml_movies_binary
   WHERE  id = 1;
END;
```

Test voor toekennen XMLType-waarde.

De resultaten zijn als volgt.

	CLOB	*objectrelationeel*	*binary XML*
select into 0,98 MB	*0,03 sec.* *1*	*0,03 sec.* *1*	*0,03 sec.* *1*
select into 5,87 MB	*0,07 sec.* *1*	*0,40 sec.* *5,7*	*0,07 sec.* *1*
select into 31,8 MB	*0,07 sec.* *1*	*7,92 sec.* *113*	*0,07 sec.* *1*

Meetresultaten toekennen XMLType-waarde.

Opmerkelijk is de goede performance van CLOB, gelijk aan die van binary XML. Bij objectrelationele opslag is duidelijk te zien dat het opbouwen van de XMLType-waarde op basis van de objectrelationele waarden aanzienlijk veel tijd kost bij grote documenten. In bovenstaande test wordt geen operatie uitgevoerd op de XMLType-waarde. Interessant is om te onderzoeken of het uitvoeren van een operatie verschil uitmaakt. In theorie moet er bij binary XML-opslag geprofiteerd kunnen worden van de opslag in *post-parse* formaat, waardoor er tijd wordt bespaard bij het parsen van een XMLType-waarde voordat er operaties op uitgevoerd kunnen worden.
Om dit te meten wordt als eerste de onderstaande test gebruikt, waarbij een extract wordt gedaan op de XMLType-waarde.

```
DECLARE
   l_movie_info   XMLTYPE;
BEGIN
   SELECT extract(movie_info, '/movie/genres/genre[1]/@id')
   INTO   l_movie_info
   FROM   xml_movies_binary
   WHERE  id = 1;
END;
```

Test extract uitvoeren op XMLType-waarde bij toekenning.

	CLOB	*objectrelationeel*	*binary XML*
select into 0,98 MB	*1,14 sec.* *38*	*0,03 sec.* *1*	*0,26 sec.* *8,7*
select into 5,87 MB	*7,51 sec.* *107*	*0,07 sec.* *1*	*3,79 sec.* *54*
select into 31,8 MB	*1 minuut 23 sec.* *1.186*	*0,07 sec.* *1*	*1 minuut 58 sec.* *1.686*

Meetresultaten extract uitvoeren op XMLType-waarde bij toekenning.

Bij objectrelationele opslag wordt overduidelijk geprofiteerd van de mogelijkheid om de XPath-expressie (ook in select list) om te schrijven naar een query op de onderliggende objectrelationele datastructuur. Opmerkelijk is de slechte performance bij binary XML-opslag om een extract uit te voeren op een groot document, terwijl verwacht werd dat geprofiteerd kan worden van het post-parse formaat van binary XML. In theorie kan een XMLIndex de performance van een extractoperatie versnellen. Dit was echter niet meetbaar/zichtbaar bij een test met het document van 5,87 MB.
Na de extractoperatie is er nog een tweede test gedaan. Namelijk het uitvoeren van een XSLT-transformatie bij de toekenning. Op de XMLType-waarde wordt een zogenaamde 'identity transformatie' uitgevoerd, deze transformatie 'raakt' alle delen van het XML-document en levert in feite een kopie op.

```
DECLARE
   l_movie_info   XMLTYPE;
   l_stylesheet   XMLTYPE;
BEGIN
   l_stylesheet :=
      XMLTYPE(  '<xsl:stylesheet version="1.0" '
```

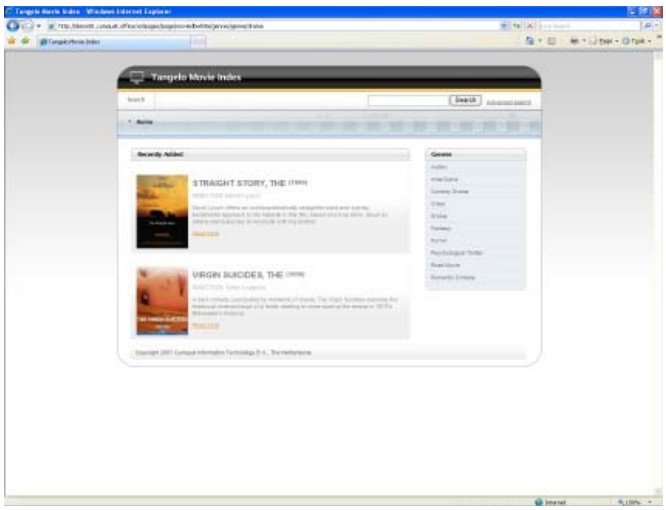
```
                ||'xmlns:xsl="http://www.w3.org/1999/XSL/Transform">'
                ||'<xsl:template match="node()">'
                ||'<xsl:copy>'
                ||'<xsl:copy-of select="@*"/>'
                ||'<xsl:apply-templates select="node()"/>'
                ||'</xsl:copy>'
                ||'</xsl:template>'
                ||'</xsl:stylesheet>'
              );
    SELECT m.movie_info.transform(l_stylesheet)
    INTO   l_movie_info
    FROM   xml_movies_binary m
    WHERE  id = 1;
END;
```

Test uitvoeren van XSLT-transformatie op XMLType-waarde bij toekenning.

	CLOB	*objectrelationeel*	*binary XML*
select into 0,98 MB	*1,21 sec. 3,9*	*0,31 sec. 1*	*0,51 sec. 1,6*
select into 5,87 MB	*9,26 sec. 2,0*	*4,59 sec. 1*	*6,23 sec. 1,4*
select into 31,8 MB	*2 minuten 14 sec. 1*	*2 minuten 55 sec. 1,3*	*2 minuten 55 sec. 1,3*

Meetresultaten XSLT transformatie uitvoeren op XSLT transformatie bij toekenning.

Opmerkelijk is dat objectrelationele opslag de binary XML-opslag 'verslaat' bij de eerste twee documenten. De meetresultaten komen dicht bij elkaar te liggen bij het grootste document. De overhead van de XSLT-transformatie lijkt dan de meeste tijd in beslag te nemen en nauwelijks te verschillen per opslagoptie. Bovenstaande test is nog herhaald met een XSLT-stylesheet met één template die met geen enkel element matcht. De tijden komen dan wel lager uit, maar de verhoudingen tussen de verschillende opslagopties komen niet anders te liggen.

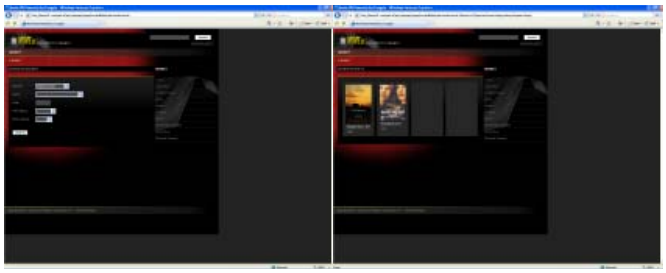


Afbeelding 4. Selectie van films van een bepaald genre.

Zoeken op XML data

Eerder in dit artikel is gekeken naar de invloed van indexen op laadtijd. Nu is het tijd om na te gaan welke invloed de verschillende indexen hebben op queries op XMLType-waarden. In de Movie DB democase worden dit soort queries ondermeer gebruikt om films van een bepaald genre te kunnen selecteren.

Via uitgebreid zoeken (advanced search) kunnen films van een bepaalde regisseur gevonden worden.



Afbeelding 5. Vinden van films van een bepaalde regisseur via uitgebreid zoeken.

```
SELECT count(id)
FROM   xml_movies_binary
WHERE  EXISTSNODE(movie_info, '/movie/genres/genre[@id=1]') > 0

SELECT count(id)
FROM   xml_movies_binary
WHERE  EXTRACTVALUE(movie_info, '/movie/director/@id') = 1
Test queries op XML-data
```

Zonder index leiden de bovenstaande queries tot de volgende meetresultaten.

	CLOB	*objectrelationeel*	*binary XML*
100	*0,17 sec. 1,2 0,15 sec. 1,25*	*0,14 sec. 1 0,12 sec. 1*	*0,20 sec. 1,4 0,12 sec. 1*
10000	*0,56 sec. 3,7 0,32 sec. 2,7*	*0,15 sec. 1 0,12 sec. 1*	*0,28 sec. 1,9 0,15 sec. 1,25*
10.000	*21,93 sec. 14,2 3,10 sec. 10*	*1,54 sec. 1 0,31 sec. 1*	*1,85 sec. 1,2 0,51 sec. 1,6*
25.000	*2 minuten 39 sec. 92 6,17 sec. 12*	*1,73 sec. 1 0,51 sec. 1*	*3,21 sec. 1,8 0,92 sec. 1,8*
50.000	*n.v.t. (1) 10,2 sec. 11,3*	*2,31 sec. 1 0,90 sec. 1*	*4,92 sec. 2,1 1,73 sec. 1,9*

Meetresultaten queries op XML data zonder index. (1) niet uitgevoerd

CLOB-opslag gaat bij de eerste query flink onderuit bij grote aantallen records. Opvallend is dat de performance van beide queries bij objectrelationele opslag heel redelijk blijft bij grote aantallen. Binary XML blijft daar een factor 1,5 a 2,0 bij achter, maar schaalt ook redelijk goed zonder index.

Function based index

De eerste uitgeprobeerde index, is een function based index. Verschillende XML-functies kunnen gebruikt worden bij dit type index, waaronder EXTRACTVALUE en EXISTSNODE. Om een function based index te kunnen gebruiken, moet in de query exact dezelfde functieaanroep gedaan worden. Bovendien moet de functie een *singleton* (enkele) waarde opleveren. Dit maakt het onmogelijk om een function based index te gebruiken voor de eerste query. In deze query wordt namelijk de volgende clause gebruikt:

```
EXISTSNODE(movie_info, '/movie/genres/genre[@id=1]') > 0
```

Eigenlijk zou er dan voor ieder genre een aparte index gemaakt moeten worden (met '[@id=1]', '[@id=2]' et cetera), maar dat is uiteraard niet praktisch. Herschrijven van de query geeft ook geen oplossing voor het gebruik van een function based index. Uitgaande van maximaal drie genres in de XML-data zou de query bijvoorbeeld als volgt herschreven kunnen worden:

```
   EXTRACTVALUE(movie_info, '/movie/genres/genre[1]/@id') = 1
OR EXTRACTVALUE(movie_info, '/movie/genres/genre[2]/@id') = 1
OR EXTRACTVALUE(movie_info, '/movie/genres/genre[3]/@id') = 1
```

Het is echter niet toegestaan om een function based index te maken op

```
EXTRACTVALUE(movie_info, '/movie/genres/genre/@id')
```

omdat deze functie geen singleton oplevert. Dit betekent dat alleen bij de tweede query een function based index gebruikt kan worden.

```
CREATE INDEX movie_director_idx_bin
ON xml_movies_binary(EXTRACTVALUE(movie_info, '/movie/director/@id'))
```

Aanmaken function based index op director.

Hieronder is het effect van de function based index te zien. Aangegeven is hoeveel keer sneller de query is met index in vergelijk met de meting zonder index.

	CLOB	*objectrelationeel*	*binary XML*
50.000	*index wordt niet gebruikt*	*0,03 sec.* *30 x sneller*	*0,06 sec.* *29 x sneller*

Meetresultaten function based index.

De function based index geeft een uitstekende optimalisatie bij objectrelationele en binary XML-opslag. Overigens wordt bij objectrelationele opslag de function based index *herschreven* naar een B-tree index op de onderliggende objectrelationele datastructuur. Bij CLOB-opslag wordt de index niet gebruikt; ondanks het toevoegen van een hint in de query.

XMLIndex

Een function based index kan dus goed gebruikt worden om bepaalde specifieke queries (met *specifieke* XPath-expressies) te optimaliseren. De in 11g geïntroduceerde XMLIndex kan gebruikt worden om queries op *alle* XPath-expressies te verbeteren.

```
CREATE INDEX movie_idx_bin
ON xml_movies_binary (movie_info)
INDEXTYPE IS XDB.XMLINDEX
```

Aanmaken XMLIndex.

Het gebruik van de XMLIndex geeft verrassende resultaten:

	CLOB	*objectrelationeel*	*binary XML*
10.000	*6,60 sec.* *3,3 x sneller* *0,81 sec.* *3,8 x sneller*		*0,17 sec.* *11 x sneller* *0,03 sec.* *17 x sneller*
25.000	*8,64 sec.* *18 x sneller* *3,79 sec.* *1,6 x sneller*	*XMLIndex niet mogelijk (1)*	*0,85 sec.* *3,7 x sneller* *0,40 sec.* *2,3 x sneller*
50.000	*7,51 sec.* *n.v.t.* *3,72 sec.* *2,7 x sneller*		*7,58 sec.* *1,5 x **langzamer*** *4,66 sec.* *2,7 x **langzamer***

Meetresultaten XMLIndex.

(1) Bij objectrelationele opslag is het niet mogelijk om een XMLIndex aan te maken, bij hybrid opslag kan dit wel op de delen die als CLOB worden opgeslagen.

De grote verbeteringen bij CLOB-opslag waren te verwachten. De eerste query op 25.000 records wordt van meer dan 2 minuten teruggebracht naar minder dan 10 seconden; een verbetering met een factor 18. De absolute tijden bij zowel CLOB als binary XML vallen echter nog behoorlijk tegen. Het terugbrengen naar *sub second* responstijden zoals met de function based index, lukt niet met een XMLIndex. Bij binary XML-opslag verslechtert de performance zelfs bij grote aan-

tallen. De responstijden van CLOB en binary XML lijken naar elkaar toe te groeien.

Secondary index op numerieke waarden

Standaard indexeert een XMLIndex alle element- en attribuutwaarden in de XML data als *string*-waarden. Het is echter mogelijk om *secundaire* indexen te definiëren op numerieke en datumwaarden. Met de onderstaande code wordt een secundaire index op numerieke waarden aangemaakt.

```
BEGIN
   DBMS_XMLINDEX.CREATENUMBERINDEX('XDBTEST'
   ,                               'MOVIE_IDX_BIN'
   ,                               'MOVIE_IDX_BIN_SEC_NUM');
END;
```

Aanmaken secundaire index op numerieke waarden.

Deze secundaire index leverde echter geen meetbare verbeteringen op.

XMLIndex op specifieke XPath-expressies

Omdat een XMLIndex standaard alle element- en attribuutwaarden indexeert, om een verbetering te kunnen geven bij alle XPath-expressies, kan de index snel erg groot worden. Dit kan het effect van de index nadelig beïnvloeden. Dit is (waarschijnlijk) ook de reden dat in de vorige test de verbetering van de performance afneemt bij grotere aantallen records. De index kan kleiner gehouden worden door bepaalde XPath-expressies uit te sluiten. Of alle uit te sluiten en bepalen welke XPath-expressies juist wel geïndexeerd moeten worden. Dit wordt gedaan met de PATHS-optie in de PARAMETERS-clause van het CREATE INDEX-commando.
Er wordt een XMLIndex aangemaakt die alleen de waarden die van belang zijn voor de testqueries indexeert.

```
CREATE INDEX movie_idx_bin
ON xml_movies_binary (movie_info)
INDEXTYPE IS XDB.XMLINDEX
PARAMETERS ('PATHS (INCLUDE (/movie/director/@id /movie/genres/genre/@
id))')
```

Aanmaken XMLIndex voor specifieke XPath-expressies.

In de volgende tabel zijn de resultaten van deze test te zien.

	CLOB	*objectrelationeel*	*binary XML*
10.000	*2 minuten en 26 sec.* *6,7 x langzamer* *1,17 sec.* *2,6 x sneller*	*XMLIndex niet mogelijk*	*0,75 sec.* *2,5 x sneller* *0,04 sec.* *13 x sneller*
50.000	*afgebroken*		*3,95 sec.* *1,25 x sneller* *0,76 sec.* *2,3 x sneller*

Meetresultaten XMLIndex voor specifieke XPath-expressies.

Het effect is bij binary XML-opslag duidelijk te zien. Een algemene XMLIndex verslechterde de performance bij 50.000 records, maar een specifieke XMLIndex geeft wel een verbetering. Zeer opmerkelijk is dat hetzelfde effect bij CLOB-opslag niet waarneembaar is. Daar geeft de specifieke XMLIndex zelfs een dramatische achteruitgang bij de eerste query. Al bij 10.000 records loopt de responstijd op naar boven de twee minuten; veel langzamer dan zonder index. De query op 50.000 records is afgebroken na minutenlang wachten. Bij binary XML-opslag is de versnelling van de specifieke XMLIndex minder groot dan de algemene XMLIndex. Het lijkt dus zinvol om bij grote aantallen records, indien dit mogelijk is, een XMLIndex aan te maken voor specifieke XPath-expressies. Belangrijk hierbij is, dat aan een bestaande index XPath-expressies toegevoegd kunnen worden via het ALTER INDEX-commando.

Updaten van XML data

Naast het laden van XML-data en het uitvoeren van queries op XML-data kan in bepaalde toepassingen ook het updaten van XML-data aan de orde zijn. Bij CLOB-opslag zullen alle XML-data overschreven moeten worden. Bij objectrelationele en binary XML-opslag is het echter mogelijk om een zogenaamde *piecewise update* te doen, waarbij alleen een deel van de XML-data aangepast wordt. Dit zou bij grote documenten aanzienlijk sneller moeten gaan dan het overschrijven van alle XML-data.

Het onderstaande statement wordt gebruikt als test.

```
update xml_movies_binary
set    movie_info = updatexml(movie_info
                      ,         '/movie/director/@id'
                      ,         1
                      )
where  id = 1
```

Test voor updaten van XML-data.

	CLOB	objectrelationeel	binary XML
updaten van 0,98 MB	1,82 sec. 26	0,07 sec. 1	3,86 sec. 55
updaten van 5,87 MB	11,32 sec. 162	0,07 sec. 1	3,86 sec. 55

Meetresultaten updaten van XML-data (zonder index).

Zoals verwacht is het updaten van XML-data bij CLOB-opslag het langzaamste. Objectrelationele opslag geeft een optimale performance. Opvallend is dat de performance van binary XML (weer) tegenvalt. In theorie zou een XMLIndex het updaten van XML-data gunstig beïnvloeden. Hieronder is in de meetresultaten echter het *tegendeel* te zien.

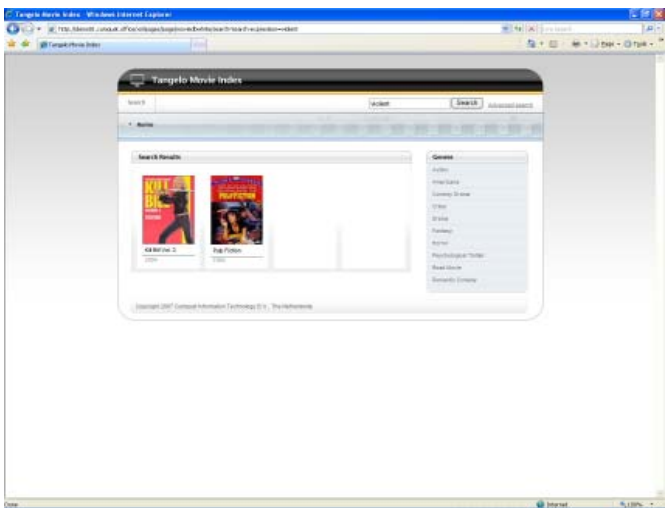
	CLOB	objectrelationeel	binary XML
updaten van 0,98 MB	12,82 sec. 7 x langzamer	0,07 sec. geen verschil	15,31 sec. 26 x langzamer
updaten van 5,87 MB	1 minuut 53 sec. 10 x langzamer	0,07 sec. geen verschil	1 minuut 15 sec. 19 x langzamer

Meetresultaten updaten van XML-data met XMLIndex.

Deze meetresultaten zijn onverklaarbaar. De respons van de update is zelfs vele malen langzamer dan het indexeren van de XML-data.

Full text search

Het zoeken naar woorden die voorkomen in een tekst wordt *full text search* genoemd. In de Movie DB demo case wordt deze zoekfunctionaliteit geboden.



Afbeelding 6. Full text search in de Movie DB democase.

In 11g is de Oracle Text CTXXPATH-index komen te vervallen en vervangen door de XMLIndex. Voor full text search wordt ook in 11g nog steeds gebruik gemaakt van de CONTEXT-index van Oracle Text. Dit type kan aangemaakt worden voor alle opslagopties van XMLType-kolommen.

```
CREATE INDEX movie_text_idx_bin
ON xml_movies_binary (movie_info)
INDEXTYPE IS CTXSYS.CONTEXT
```

Aanmaken Oracle Text index voor full text search.

De tijd die benodigd is voor het aanmaken van de index verschilt (iets) per opslagoptie. CLOB is het snelste. Waarschijnlijk omdat die data direct geïndexeerd kunnen worden, terwijl bij objectrelationeel en binary XML de data eerst 'teruggevormd' moet worden naar tekst.

	CLOB	objectrelationeel	binary XML
indexeren 10.000	15,11 sec. 1	17,18 sec. 1,1	19,46 sec. 1,3

Meetresultaten aanmaken contextindex.

De performance van een full text search query is vervolgens getest met onderstaande query. De CONTAINS-operator wordt hierin gebruikt om op het voorkomen van het woord 'sunshine' te zoeken.

```
SELECT id
FROM   xml_movies_binary
WHERE  CONTAINS(movie_info, 'sunshine') > 0
```

Test full text search.

	CLOB	objectrelationeel	binary XML
query	0,15 sec.	0,15 sec.	0,14 sec.

Meetresultaten full text search.

Zoals te verwachten zijn er geen verschillen. Er wordt immers gezocht via de contextindex die ongeacht de opslagstructuur dezelfde is.

Opslagruimte

Voor dit artikel is niet gekeken naar de ruimte die wordt ingenomen door de XML-data bij de diverse opslagopties. In de Oracle XDB Developer's Guide wordt wel een dergelijke vergelijking gedaan.

Functionele verschillen

Alvorens conclusies te trekken over welke opslagopties en indexeringsopties in welke situaties de voorkeur hebben op basis van *performance*, is het goed om enkele *functionele* verschillen tussen de besproken opties te noemen. Alleen bij CLOB-opslag is *document fidelity* gegarandeerd. Hetgeen betekent dat het XML-document exact hetzelfde (met spaties en

regelovergangen) is bij opslaan en ophalen. Bij de andere opslagopties is er sprake van *DOM fidelity*.
Objectrelationele opslag is (veel) minder flexibel dan CLOB en binary XML-opslag. Wanneer de XML Schema-definitie wijzigt moet de objectrelationele structuur aangepast worden. Hiervoor biedt Oracle 11g naast het al bestaande *copy evolve*-ook in *place evolve*-functionaliteit. Hoewel de opslagstructuur hiermee onder bepaalde voorwaarden eenvoudig(er) aan te passen is, is objectrelationele opslag nog altijd bewerkelijk(er). Bij objectrelationele opslag kunnen constraints (met wat moeite) gedefinieerd worden op de onderliggende objectrelationele datastructuur. Bij binary XML kan, met beperkingen, gebruik worden gemaakt van virtuele kolommen voor het aanmaken van constraints. CLOB-opslag biedt geen mogelijkheden om constraints te gebruiken.

Conclusie

Zoals al in de inleiding opgemerkt zijn de metingen verricht in 'default omstandigheden'. Er is geen tuning gedaan en de resultaten betreffen dus (een gedegen) 'eerste indruk'. Bij de hierboven beschreven tests is al een aantal vaststellingen gemaakt. In een aantal situaties valt de performance (erg) tegen:

- extract uitvoeren op een groot XML-document opgeslagen als binary XML
- laden en toekennen grote XML-documenten bij objectrelationele opslag
- function based index bij CLOB-opslag
- XMLIndex voor specifieke XPath-expressies bij CLOB-opslag
- queries zonder XML Index bij grote aantallen en CLOB-opslag
- piecewise update van grote XML-documenten bij gebruik van een XMLIndex.

Deze situaties vergen extra aandacht. Belangrijker echter is de vaststelling dat een XMLIndex niet zonder meer *sub-second* responstijden geeft. Met een function based index wordt dit wel bereikt.

Voorlopige richtlijnen op basis van de meetresultaten zijn:

- gebruik binary XML met XMLIndex, specificeer bij grote aantallen records (indien mogelijk) welke XPath-expressies geïndexeerd moeten worden of gebruik (of vul aan met) function based indexen.
- bij kleine XML-documenten met een structuur die niet zal veranderen is objectrelationeel zeker ook een optie.
- wanneer de XML-data alleen maar opgeslagen en opgehaald hoeft te worden, zonder dat deze doorzocht hoeft te worden, kan CLOB-opslag gebruikt worden.

Een aantal punten dat in dit artikel naar voren is gekomen, verdienen nog nader onderzoek. Hiervan zal verslag worden gedaan in de 'technology corner' (http://www.cumquat.nl/technology.html) op de website van Cumquat.

Noten

(1) Het laden van een document van 31,8MB bij CLOB-opslag is na meer dan tien minuten wachten afgebroken. Het insert-proces had toen een enorm stuk temporary tablespace-ruimte in beslag genomen.

Erwin Groenendal is productmanager van de Tangelo productsuite bij Cumquat Information Technology. Voor vragen of opmerkingen kan contact met Erwin worden opgenomen via erwin@cumquat.nl.