Oracle Advanced Queuing

Message queuing vanuit PL/SQL

Oracle Advanced Queuing (AQ) is Oracle's implementatie van message queuing, een belangrijke technologie voor de integratie van systemen, in het bijzonder voor Enterprise Application Integration (EAI), de integratie van systemen binnen een bedrijf of organisatie. Achter de ogenschijnlijke eenvoudige interface van de (supplied) PL/SQL packages die de functionaliteit van Advanced Queuing beschikbaar stellen gaat een grote complexiteit schuil. In dit artikel geeft Erwin Groenendal een inleiding tot Advanced Queuing aan de hand van een aantal voorbeelden, en presenteert hij een kort overzicht van de overige mogelijkheden die Advanced Queuing biedt.

Bij message queuing worden berichten (*messages*) door een systeem in een wachtrij (*queue*) geplaatst en er door een ander systeem weer uitgehaald. De queue dient dus als een soort "doorgeefluik" tussen de systemen. De messages worden tijdelijk in de queue opgeslagen en kunnen er door "het andere systeem" later uitgehaald worden (om verwerkt te worden). De message queue maakt op deze manier een *asynchrone* koppeling tussen twee systemen mogelijk; de message wordt namelijk niet direct (*synchroon,* in dezelfde actie) door het andere systeem verwerkt, maar later (asynchroon, in een aparte actie). Message queuing is bij uitstek geschikt voor *event based* koppelingen tussen systemen. Hierbij worden op specifieke momenten (*events*) messages gegenereerd waarin de informatie over het event wordt doorgegeven aan één of meer andere systemen. Een goed voorbeeld van een event met bijhorende message is de wijziging van een adres (het event) en de gewijzigde adresgegevens (de message).

Grote complexiteit

Advanced Queuing is een 'feature' van de Oracle database. De implementatie van message queuing als functionaliteit van de Oracle database zelf brengt een aantal voordelen met zich mee. Het is zeer gemakkelijk om message queuing te gebruiken vanuit PL/SQL en via SQL kan op een heel eenvoudige manier *business intelligence* informatie verkregen worden over de messages die uitgewisseld zijn (en worden) tussen systemen.

Bovendien profiteert Advanced Queuing van belangrijke 'eigenschappen' van de Oracle database, zoals schaalbaarheid, betrouwbaarheid en veiligheid. Advanced queuing is ook de basis voor Oracle9iAS Integration (InterConnect en diens opvolger ProcessConnect, die beiden een op Advanced Queuing gebaseerde *hub-and-spoke* architectuur implementeren voor integratie).

Achter de ogenschijnlijke eenvoudige interface van de (*supplied*) PL/SQL packages die de functionaliteit van Advanced Queuing beschikbaar stellen – een package voor "administratieve" acties (zoals het creëren van queues) en een package voor transacties – gaat een grote complexiteit schuil. In dit artikel wordt stap voor stap aangegeven hoe een queue opgezet moet worden, hoe messages in deze queue geplaatst worden en hoe de messages er uitgehaald worden.

Stap 1: Creatie queue table

De eerste stap is het creëren van een *queue table*. Dit gebeurt met de procedure CREATE_QUEUE_TABLE in de supplied PL/SQL package DBMS_AQADM. De betreffende user moet hiervoor de role AQ_ADMINISTRATOR_ROLE hebben toegekend gekregen. Bij het aanmaken van een queue table moet het datatype van de messages aangegeven worden. Het datatype kan VARCHAR2, RAW (voor *binary* messages) of een object type zijn. De laatste maakt het mogelijk om een user defined type te maken dat precies "op maat" is voor de informatie die in de message moet kunnen worden opgeslagen. In Oracle9i Release 2 is het ook mogelijk om XMLTYPE te gebruiken als datatype van de *payload* – zoals de inhoud van een message genoemd wordt – van een message. Met XMLTYPE kan de inhoud van een message een XML document zijn.

CREATE_QUEUE_TABLE heeft twee verplichte parameters: de naam van de queue table en het datatype van de payload. Met het onderstaande "stukje" PL/SQL wordt de queue table XMLTYPE_QUEUE_TABLE aangemaakt met XMLTYPE als datatype van de payload.

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE('XMLTYPE_QUEUE_TABLE', 'SYS.XMLTYPE');
END;
```

Twee andere belangrijke parameters zijn 'sort_list', waarmee de volgorde waarin de messages uit de queue gehaald worden gestuurd kan worden (hiervan wordt later in dit artikel een voorbeeld gegeven) en 'multiple_consumers', waarmee aangegeven kan worden dat een message door meerdere systemen uit de queue gehaald kan worden (hierop wordt nog kort teruggekomen aan het eind van het artikel).

Stap 2: Creatie queue

"Binnen" een queue table kunnen vervolgens meerdere queues gecreëerd worden (die dan dus allemaal dezelfde payload hebben). Het aanmaken van een queue gebeurt met behulp van de procedure CREATE_QUEUE (in de package DBMS_AQADM):

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE('XML_MESSAGES', 'XMLTYPE_QUEUE_TABLE');
END;
```

Met de bovenstaande code wordt de queue XML_MESSAGES gecreëerd binnen de queue table XMLTYPE_QUEUE_TABLE. Andere parameters van CREATE_QUEUE_TABLE die later in dit artikel geïllustreerd worden zijn 'queue_type', 'max_retries', 'retry_delay' en 'retention_time'.

Stap 3: Starten queue

Voordat de queue gebruikt kan worden moet deze eerst gestart worden:

```
BEGIN
   DBMS_AQADM.START_QUEUE('XML_MESSAGES');
END;
```

Met de parameters 'enqueue' en 'dequeue' (beide van het datatype boolean) kan eventueel (default is true) nog afzonderlijk aangegeven worden dat er wel of geen messages in de queue geplaatst mogen worden (enqueue) en dat er wel of geen messages uit de queue gehaald mogen worden (dequeue).

Stap 4: Privileges toekennen

De queue is nu klaar voor gebruik. Echter, alleen de "eigenaar" van de queue (de user binnen wiens schema de queue is gecreëerd) kan de queue nu gebruiken. Om het gebruik van de queue door andere users mogelijk te maken moeten privileges toegekend worden via GRANT_QUEUE_PRIVILEGE. Aan een user kan het privilege toegekend worden om een message in een bericht te plaatsen (ENQUEUE), een message uit de queue te halen (DEQUEUE) of beiden (ALL). In het onderstaande PL/SQL block worden deze privileges respectievelijk toegekend aan users AQDEMO_USER1, AQDEMO_USER2 en AQDEMO_USER3.

```
BEGIN
   DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('ENQUEUE', 'XML_MESSAGES', 'AQDE-
MO_USER1');
   DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('DEQUEUE', 'XML_MESSAGES', 'AQDE-
MO_USER2');
   DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('ALL', 'XML_MESSAGES',
'AQDEMO_USER3');
END;
```

Stap 5: Enqueue

Het is nu een goed moment om wat termen nader te introduceren. Het in de queue plaatsen van een message wordt *enqueue* genoemd. Het systeem of proces dat de enqueue van message uitvoert (en de message dus 'creëert') is de *producer* van de message. De message wordt met een *dequeue* uit de queue gehaald door de *consumer* van de message.

De volgende stap is het uitvoeren van een enqueue (van een message). Naast de (juiste) privileges op een queue moet de user die de enqueue doet ook 'execute' rechten hebben op de package DBMS_AQ (en voor het gemak hiervoor een synonym hebben). De enqueue wordt uitgevoerd met de procedure ENQUEUE in DBMS_AQ. Deze procedure heeft vijf parameters: de naam van de queue, de zogenaamde *enqueue options* en *message property's*, de payload zelf (de inhoud van de message) en de OUT parameter 'msgid'. De enqueue options en message property's zijn beide parameters van een datatype dat gedefinieerd is in de DBMS_AQ package. In de parameter 'msgid' wordt de *message id* van de message teruggegeven. De onderstaande code laat zien hoe een enqueue van een message gedaan kan worden.

```
DECLARE
   l_enqueue_options       DBMS_AQ.ENQUEUE_OPTIONS_T;
   l_message_properties    DBMS_AQ.MESSAGE_PROPERTIES_T;
   l_msgid                  RAW(16);
BEGIN
   DBMS_AQ.ENQUEUE('AQDEMO.XML_MESSAGES'
   ,               l_enqueue_options
   ,               l_message_properties
   ,               XMLTYPE('<?xml version="1.0"?><data>...</data>')
   ,               l_msgid
   );
END;
```

visibility	Geeft aan of de message na een commit (ON_COMMIT, default) of direct (IMMEDIATE), zonder commit, uit de queue gehaald kan worden.
relative_msgid	Geeft, in combinatie met 'sequence_deviation' aan waar (ten opzichte van welke) de message in de queue geplaatst moet worden.
sequence_deviation	Geeft aan dat voor het plaatsten van de message afgeweken moet worden van het normale gedrag.
transformation	Geeft aan welke transformatie (van de message payload) uitgevoerd moet worden voordat de message daadwerkelijk in de queue geplaatst wordt.

Tabel 1. Enqueue options

priority	Geeft de prioriteit van de message aan (hoe lager de waarde, hoe hoger de prioriteit).
expiration	Geeft aan (in seconden) hoe lang het duurt voordat de message verlopen is (en deze niet meer uit de queue gehaald kan worden).
correlation	Geeft een identificatie aan de message waarmee de message uit de queue gehaald kan worden.
recipient_list	Alleen van toepassing voor multiple consumers queues. Geeft aan welke consumers (systemen) de message uit de queue kunnen halen.
exception_queue	Geeft de queue aan waarin de message geplaatst (overgeheveld) wordt indien het niet lukt om de message uit de queue te halen.

Tabel 2. Message property's

Zoals al eerder gezegd is er veel complexiteit verscholen bij Advanced Queuing. Twee van de 'schuilplaatsen' voor deze complexiteit zijn de enqueue options en message property's parameters. De datatypes van deze parameters zijn beide record types met een groot aantal elementen. De belangrijkste hiervan zijn opgenomen in de overzichtstabellen 1 en 2.

Stap 6: Dequeue

Als laatste stap wordt met DBMS_AQ.DEQUEUE een message uit een queue gehaald:

```
DECLARE
   l_dequeue_options       DBMS_AQ.DEQUEUE_OPTIONS_T;
   l_message_properties    DBMS_AQ.MESSAGE_PROPERTIES_T;
   l_msgid                  RAW(16);
   l_payload                XMLTYPE;
BEGIN
   l_dequeue_options.wait := DBMS_AQ.NO_WAIT;
   DBMS_AQ.DEQUEUE('AQDEMO.XML_MESSAGES'
   ,               l_dequeue_options
   ,               l_message_properties
   ,               l_payload
   ,               l_msgid
   );
   DBMS_OUTPUT.PUT_LINE(l_payload.GETSTRINGVAL());
END;
```

Net als bij ENQUEUE (in de enqueue options) kunnen een groot aantal 'opties' worden aangegeven in de *dequeue options*. De laatste drie parameters zijn OUT parameters en bevatten respectievelijk de message property's (zoals aangegeven bij de enqueue, maar bevat ook enkele waarden die betrekking hebben op de 'levensloop' van de message, zoals tijd van enqueue en het aantal keren dat al geprobeerd is de message uit de queue te halen), de payload zelf (de inhoud van de message) en de message id.

De belangrijkste elementen in het DBMS_AQ.DEQUEUE_OPTIONS_T record type (een derde "schuilplaats" van complexiteit) worden toegelicht in tabel 3.

Het uitvoeren van het bovenstaande PL/SQL block geeft het volgende resultaat:

```
<?xml version="1.0"?><data>...</data>

PL/SQL procedure successfully completed.
```

consumer_name	Alleen van toepassing voor multiple consumers queues. Geeft de naam van de consumer aan die (als wie) de message uit de queue gehaald wordt.
dequeue_mode	Geeft de mode aan waarin de dequeue wordt uitgevoerd: BROWSE om alleen de message te bekijken (en deze dus niet "echt" uit de queue te halen (te verwijderen), LOCKED om de message te bekijken en te locken (zodat deze niet meer zichtbaar is vanuit een andere sessie), REMOVE (default) om de message "echt" uit de queue te halen (te verwijderen en te kunnen verwerken) en REMOVE_NODATA om de message alleen te verwijderen.
navigation	Geeft aan hoe "genavigeerd" moet worden naar de volgende message: NEXT_MESSAGE (default) om naar de "volgende" message te gaan, NEXT_TRANSACTION is alleen relevant indien gebruik wordt gemaakt van message grouping (wat niet behandeld wordt in dit artikel) en FIRST_MESSAGE om naar de "eerste" message te gaan. Het verschil in gedrag tussen NEXT_MESSAGE en FIRST_MESSAGE speelt met name bij foutsituaties (waarbij een message "opnieuw" in de queue geplaatst wordt) en heeft ook performance aspecten.
visibility	Geeft aan of de actie (zie dequeue_mode) die op de message uitgevoerd wordt na commit (ON_COMMIT, default) "zichtbaar" is voor andere sessies of direct (IMMEDIATE).
wait	Geeft (in seconden) aan hoe lang gewacht moet worden op het beschikbaar komen van een message. Met DBMS_AQ.FOREVER kan aangeven worden dat de call (naar DBMS_AQ.DEQUEUE) geblokt moet worden totdat er een message beschikbaar is, met DBMS_AQ.NO_WAIT dat er (helemaal) niet gewacht moet worden op het beschikbaar komen van een message. Als een dequeue gedaan wordt met NO_WAIT er is geen message beschikbaar wordt er een foutmelding gegeven.
msgid	Geeft de message id aan van de message die uit de queue gehaald moet worden.
correlation	Geeft de correlation (identificatie) aan van de message die uit de queue gehaald moet worden.
deq_condition	Geeft de dequeue condition aan waarmee "gezocht" moet worden naar een message in de queue (die voldoet aan de conditie).
transformation	Geeft aan welke transformatie (van de message payload) uitgevoerd moet worden (direct) nadat de message uit de queue gehaald wordt en voordat deze als OUT parameter teruggeven wordt.

Tabel 3. De belangrijkste elementen in het DBMS_AQ.DEQUEUE_OPTIONS_T record type.

Uit de hiervoor besproken zes stappen blijkt dat de functionaliteit en gedrag van Advanced Queuing bepaald worden door:
- Hoe de queue table is aangemaakt.
- Hoe de queue is aangemaakt.
- De enqueue options.
- De message property's.
- De dequeue options.

Hieronder wordt het "samenspel" van deze onderdelen verder geïllustreerd door in te gaan op prioriteit van messages, "zichtbaarheid" (visibility) van messages en acties en foutafhandeling.

Prioriteit

De default volgorde waarin messages uit de queue gehaald worden is de volgorde waarin zij in de queue geplaatst zijn: first-in, first-out (FIFO). Het onderstaande voorbeeld laat dit zien. Eerst worden drie messages in de queue geplaatst met behulp van de procedure 'enqueue_xml_message' (gebaseerd op het eerder getoonde voorbeeld van een anoniem PL/SQL block dat een enqueue uitvoert).

```
SQL> BEGIN
  2     enqueue_xml_message(XMLTYPE('<data>Message 1</data>'));
  3     enqueue_xml_message(XMLTYPE('<data>Message 2</data>'));
  4     enqueue_xml_message(XMLTYPE('<data>Message 3</data>'));
  5  END;
  6  /

PL/SQL procedure successfully completed.

SQL> COMMIT
  2  /

Commit complete.
```

Deze messages worden er, na een commit in de sessie die de enqueue van de message heeft uitgevoerd, in dezelfde volgorde uitgehaald met de procedure 'dequeue_xml_message' (gebaseerd op het eerder getoonde voorbeeld voor het uitvoeren van een dequeue):

```
SQL> BEGIN
  2     dequeue_xml_message;
  3  END;
  4  /
<data>Message 1</data>

PL/SQL procedure successfully completed.

SQL> /
<data>Message 2</data>

PL/SQL procedure successfully completed.

SQL> /
<data>Message 3</data>

PL/SQL procedure successfully completed.
```

Van deze default volgorde kan op verschillende manieren afgeweken worden. Bij het uitvoeren van een dequeue op basis van message id of correlation bijvoorbeeld of bij gebruik van de 'sequence_deviation' en 'relative_msgid' parameters bij een enqueue. Ook kan bij het creëren van de queue table aangegeven worden dat de volgorde niet op basis van het moment van de enqueue van de message moet zijn, maar op basis van prioriteit. Hiervoor wordt gebruik gemaakt van de 'sort_list' parameter van CREATE_QUEUE_TABLE:

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE('XMLTYPE_QUEUE_TABLE'
   ,                             'SYS.XMLTYPE'
   ,                             sort_list => 'PRIORITY');
END;
```

Bij een enqueue kan in de message property 'priority' de prioriteit aangegeven worden (via een extra parameter in 'enqueue_xml_message'):

```
   l_message_properties.priority := p_priority;
```

Vervolgens worden er in het onderstaande stukje code drie messages in de queue geplaatst met verschillende prioriteit:

NIEUWS

Oracle en Yahoo! verzorgen gezamenlijk aanbod bedrijfsportals

Oracle en Yahoo! bieden bedrijven nieuwe mogelijkheden voor het verrijken van hun bedrijfsportal, die steeds meer als interface tussen medewerkers en bedrijfsapplicaties fungeert. Gebruikers krijgen toegang tot meer dan honderd kant-en-klare nieuwsfuncties, zogenaamde portlets, die informatie bevatten afkomstig van ruim 2.000 belangrijke web- en nieuwsbronnen. Deze Yahoo! portlets kunnen afzonderlijk samengesteld en gepersonaliseerd worden. Portal administrators hebben de beschikking over tools om het Oracle9i Application Server Portal framework centraal te beheren. Zij kunnen controleren welke Yahoo! portlets worden weergegeven, en via het gebruik van model accounts kunnen "lock" selecties worden gemaakt, toegespitst op functie van de werknemer of de locatie van de organisatie. De Oracle9i Application Server is vooraf geïntegreerd voor het ontvangen van content feeds via een directe connectie met Yahoo!. Dit beperkt de netwerkbelasting en bevordert betere performance, die nodig is om alle externe portal content soepel te kunnen binnenhalen. Oracle9i Application Server integreert gefragmenteerde middleware producten in een suite, waaronder volledige J2EE 1.3 ondersteuning, ingebouwde enterprise portal software, high-speed caching, business intelligence, rapid application development, applicatie en business integration en webservices. My Yahoo! Enterprise Edition is een suite van meer dan honderd web-gebaseerde portlets en tools waarmee werknemers dynamische content, die voor hun werk van groot belang is, kunnen beheren, customizen en monitoren. De combinatie van My Yahoo! Enterprise Edition en Oracle9i Application Server levert de beste elementen van de consumentenversie van My Yahoo! verbeterd met portal features die essentieel zijn voor bedrijven. Yahoo! portlets zijn voor gebruikers van Oracle9i Application Server Portals direct beschikbaar via het Oracle 9iAS Portal Center of het Oracle Technology Network (OTN). Voor gebruikers van Oracle9i Application Server biedt Yahoo! Inc. een trial abonnement op My Yahoo! Enterprise Edition content.

```
BEGIN
   enqueue_xml_message(XMLTYPE('<data>Message 1</data>'), 3);
   enqueue_xml_message(XMLTYPE('<data>Message 2</data>'), 1);
   enqueue_xml_message(XMLTYPE('<data>Message 3</data>'), 2);
END;
```

Hierna worden de messages in volgorde van prioriteit uit de queue gehaald (hoe lager de waarde hoe hoger de prioriteit).

```
SQL> BEGIN
  2     dequeue_xml_message;
  3  END;
  4  /
<data>Message 2</data>

PL/SQL-procedure is geslaagd.

SQL> /
<data>Message 3</data>

PL/SQL-procedure is geslaagd.

SQL> /
<data>Message 1</data>

PL/SQL-procedure is geslaagd.
```

Visibility

Eén van de belangrijkste aspecten van het gedrag van Advanced Queuing is het moment waarop een message die in de queue geplaatst is, uit de queue gehaald kan worden, de *visibility* van de message of de uitgevoerde actie. Zowel de enqueue options als de dequeue options hebben een 'visibility' optie. Bij beiden is de default waarde ON_COMMIT. Dit betekent dat de actie (enqueue of dequeue) na het uitvoeren van een commit "zichtbaar" is voor andere sessies. Bij een enqueue betekent dit dat de message zichtbaar (beschikbaar) is voor dequeue nadat de sessie die de enqueue uitvoert een commit gedaan heeft. Voor een dequeue betekent dit dat de uitgevoerde actie (zie de optie 'dequeue_mode') zichtbaar is (doorgevoerd wordt) na een commit. De tegenhanger van ON_COMMIT is IMMEDIATE. Bij een enqueue met visibility IMMEDIATE is de message direct beschikbaar voor een dequeue, dus ook als er een exception optreedt en er een rollback wordt uitgevoerd. Deze mogelijkheid is vooral interessant omdat hiermee de mogelijkheid geboden wordt om informatie vanuit een transactie die faalt (waarin bijvoorbeeld een unhandled exception optreedt) te bewaren (door te geven). Het onderstaande voorbeeld illustreert dit.

Allereerst passen we de 'enqueue_xml_message' procedure aan en laten deze nu een enqueue uitvoeren met visibility IMMEDIATE:

```
l_enqueue_options.visibility := DBMS_AQ.IMMEDIATE;
```

Vervolgens wordt in een transactie een enqueue gedaan en faalt de transactie daarna (met een unhandled exception nadat er door nul gedeeld wordt):

```
SQL> DECLARE
  2     l_dummy   NUMBER;
  3  BEGIN
  4     enqueue_xml_message(XMLTYPE('<data>Message sent before trans-
action rolls back.</data>'));
  5     l_dummy := 1/0;
  6  END;
  7  /
DECLARE
*
ERROR at line 1:
ORA-01476: divisor is equal to zero
ORA-06512: at line 5
```

In een andere sessie kan de verstuurde message opgehaald (ontvangen) worden:

```
SQL> BEGIN
  2     dequeue_xml_message;
  3  END;
  4  /
<data>Message sent before transaction rolls back.</data>

PL/SQL procedure successfully completed.
```

Foutafhandeling

Hierboven is behandeld wat het gedrag is van Advanced Queuing bij fouten (falende transacties) bij het uitvoeren van een enqueue: afhankelijk van de visibility is de message beschikbaar (zichtbaar) voor dequeue of niet. Hieronder wordt beschreven wat het gedrag van Advanced Queuing is bij falende transactie waarin een dequeue wordt gedaan. Hierbij wordt ook het mechanisme dat Advanced Queuing gebruikt om te garanderen dat een message (succesvol) afgeleverd wordt en dat een message niet meer dan één keer wordt afgeleverd,

De default volgorde waarin messages uit de queue gehaald worden is de volgorde waarin zij in de queue geplaatst zijn

hetgeen een vereiste is bij message queuing, duidelijk. Om de foutafhandeling goed te kunnen laten zien wordt de queue met een aantal extra parameters opnieuw gecreëerd:

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE('XML_MESSAGES'
   ,                        'XMLTYPE_QUEUE_TABLE'
   ,                       max_retries => 1
   ,                       retention_time => 60 * 60 * 24 * 10
   ,                       retry_delay => 0
   );
END;
/
```

De parameter 'max_retries' geeft aan dat slecht één keer opnieuw geprobeerd mag worden om een message uit de queue te halen (nadat een eerdere transactie waarin dit geprobeerd is gefaald is). De *retention time* wordt op tien dagen gezet. Dit betekent dat na een succesvolle dequeue de message nog tien dagen bewaard wordt (waardoor gedurende deze tijd business intelligence informatie beschikbaar blijft over de messages). Met parameter 'retry_delay' wordt aangegeven dat na een fout (falende transactie) de message direct (met een 'vertraging' van 0 seconden) beschikbaar is voor een nieuwe poging. Hierbij is het wel afhankelijk van de 'navigation' dequeue option of deze message de 'volgende' message is en direct opgehaald wordt.

Voor het goed laten functioneren van retry delays en retention times moet er minstens één queue monitor process draaien. Het opstarten van deze processen wordt gestuurd via de INIT.ORA parameter AQ_TM_PROCESSES.

Met de procedure 'enqueue_xml_message' (zonder de extra parameter voor het aangeven van de prioriteit van de message) worden weer drie messages in de queue geplaatst:

```
BEGIN
   enqueue_xml_message(XMLTYPE('<data>Message 1</data>'));
   enqueue_xml_message(XMLTYPE('<data>Message 2</data>'));
   enqueue_xml_message(XMLTYPE('<data>Message 3</data>'));
END;
```

Bij het uitvoeren van de dequeues wordt na het uit de queue halen van de eerste twee messages een rollback gedaan. Vervolgens wordt de derde message uit de queue gehaald en opnieuw de eerste message. Na deze laatste twee actie wordt een commit uitgevoerd.

```
SQL> BEGIN
  2     dequeue_xml_message;
  3  END;
  4  /
<data>Message 1</data>

PL/SQL procedure successfully completed.

SQL> /
<data>Message 2</data>

PL/SQL procedure successfully completed.

SQL> ROLLBACK
  2  /

Rollback complete.

SQL> BEGIN
  2     dequeue_xml_message;
  3  END;
  4  /
<data>Message 3</data>

PL/SQL procedure successfully completed.

SQL> /
<data>Message 1</data>

PL/SQL procedure successfully completed.

SQL> COMMIT
  2  /

Commit complete.
```

De eerste message kwam dus opnieuw beschikbaar en is toen wel succesvol verwerkt. In de view 'aq$xml_type_queue_table' (die voor iedere queue table automatisch gecreëerd wordt, met als naam 'aq$<naam van queue table>') kan gezien worden wat de status is van de drie messages.

```
SQL> SELECT user_data
  2  ,       msg_state
  3  ,       retry_count
  4  FROM    aq$xmltype_queue_table
  5  /

USER_DATA                      MSG_STATE RETRY_COUNT
_____________ _____ ______
<data>Message 1</data>         PROCESSED            1
<data>Message 2</data>         READY                1
<data>Message 3</data>         PROCESSED            0
```

Message 1 is na een niet-succesvolle eerste poging (in een tweede poging) succesvol uit de queue gehaald en bevindt zich nog in de queue omdat de retention time op tien dagen staat. Message 2 is READY (beschikbaar) na een niet-succesvolle eerste poging. Message 3 is *retained* in de queue en is in één poging succesvol uit de queue gehaald.

Vervolgens wordt er opnieuw een dequeue gedaan en wordt er weer een rollback uitgevoerd. Daarna wordt er nog een dequeue gedaan. Deze laatste resulteert in een foutmelding omdat er geen messages meer beschikbaar zijn (en de dequeue uitgevoerd wordt met de 'wait' optie op NO_WAIT).

```
SQL> BEGIN
  2     dequeue_xml_message;
  3  END;
  4  /
<data>Message 2</data>

PL/SQL-procedure is geslaagd.

SQL> ROLLBACK
  2  /

Rollback complete.

SQL> BEGIN
  2     dequeue_xml_message;
  3  END;
  4  /
BEGIN
*
ERROR at line 1:
ORA-25228: timeout or end-of-fetch during message dequeue from
AQDEMO.XML_MESSAGES
ORA-06512: at "SYS.DBMS_AQ", line 221
ORA-06512: at "AQDEMO_USER2.DEQUEUE_XML_MESSAGE", line 9
ORA-06512: at line 2
```

Er zijn nu twee niet-succesvolle pogingen gedaan om 'Message 2' uit de queue te halen. Omdat 'max_retries' op 1 stond wordt deze message nu als "niet afleverbaar" beschouwd en krijgt de message de status EXPIRED:

```
SQL> SELECT user_data
  2  ,       msg_state
  3  ,       retry_count
  4  FROM    aq$xmltype_queue_table
  5  /

USER_DATA                      MSG_STATE RETRY_COUNT
_____________ _____ ______
<data>Message 1</data>         PROCESSED            1
<data>Message 2</data>         EXPIRED              1
<data>Message 3</data>         PROCESSED            0
```

Message 2 bevindt zich nu in een *exception queue*. De exception queue waarin een message geplaatst (overgeheveld) moet worden wanneer deze niet afgeleverd kan worden, dat wil zeggen, als het aantal pogingen om een dequeue uit te voeren boven 'max_retries' (parameter van DBMS_AQADM.CREATE_QUEUE) komt, is een message property. Bij het creëren van een queue

table wordt een default exception queue aangemaakt. Met DBMS_AQADM.CREATE_QUEUE kan door de parameter 'queue_type' de waarde EXCEPTION_QUEUE te geven een "eigen" exception queue gecreëerd worden.

Door de hierboven gebruikte query op 'aq$xmltype_queue_table' iets uit te breiden is te zien dat 'Message 2' zich nu in de default exception queue bevindt:

```
SQL> SELECT queue
  2  ,       user_data
  3  ,       msg_state
  4  ,       retry_count
  5  FROM    aq$xmltype_queue_table
  6  /

QUEUE                          USER_DATA            MSG_STATE
RETRY_COUNT
------------------------------ -------------------- ---------
-
XML_MESSAGES                   <data>Message 1</data>          PROCES-
SED                            1
AQ$_XMLTYPE_QUEUE_TABLE_E       <data>Message 2</data>         EXPIRED
1
XML_MESSAGES                   <data>Message 3</data>          PROCES-
SED                            0
```

Advanced Queuing zorgt er dus voor dat een message pas uit de queue verwijderd wordt wanneer de transactie waarin de dequeue wordt uitgevoerd slaagt, dat wil zeggen, een commit doet. Dit geldt ook voor een enqueue: pas als een commit wordt uitgevoerd (en de enqueue transactie dus slaagt) is de message beschikbaar. Op deze manier wordt een message nooit meer dan één keer verstuurd en gegarandeerd afgeleverd.

Overige features

Dit artikel heeft als doel om een inleiding te geven tot Advanced Queuing. De onderstaande onderdelen van Advanced Queuing vallen buiten een inleiding, maar ik zal deze, gezien hun belang, wel kort toelichten.

- *Multiple consumers queues*
  Met multiple consumers queues is het mogelijk om een message door meerdere consumers (systemen) uit de queue te laten halen. Eén message kan dan dus aan meerdere systemen "gestuurd" worden. De producer van de message kan hierbij een lijst aangeven van de consumers (via de 'recipient_list' message property) voor *point-to-(multi-)point* communicatie of de consumers "abonneren" zich op bepaalde messages voor *publish-subscribe* communicatie.

- *Propagation*
  Met Advanced Queuing kan een message automatisch gepropageerd worden naar een andere queue, mogelijk in een andere database. Via gateways kunnen messages ook aan andere message queuing producten worden doorgegeven, zoals IBM MQ Series.

- *Notifications*
  Via OCI, Java, PL/SQL en e-mail kunnen automatisch notificaties worden gegeven wanneer er een message beschikbaar is in een queue.

- *SMTP en HTTP interfaces*
  Via SMTP (e-mail) en HTTP kunnen enkele Advanced Queuing acties uitgevoerd worden, bijvoorbeeld een enqueue van een message.

- *Java API en Java Message Service (JMS) API*
  Naast de in dit artikel gebruikte PL/SQL API voor Advanced Queuing is er ook een Java API beschikbaar. Bovendien heeft Oracle de standaard Java Message Service (JMS) J2EE API voor message queuing vanuit Java geïmplementeerd "bovenop" Oracle Advanced Queuing.

- *Transformations*
  Zoals al beschreven bij de enqueue options en dequeue options is het mogelijk om een transformatie aan te geven die moet worden uitgevoerd bij een enqueue of dequeue. Voor het uitvoeren van transformaties biedt Oracle middels de supplied PL/SQL package DBMS_TRANSFORM een aantal faciliteiten. Vanuit Advanced Queuing kan hiervan gebruik gemaakt worden. Deze transformaties zijn overigens niet gebaseerd op XSLT (wat het gebruik zou beperken tot XML messages) maar op een ander metamodel en daardoor toepasbaar voor verschillende payload datatypes.

Meer informatie

Voor meer informatie over Advanced Queuing zijn met name de volgende twee manuals van belang:
Application Developer's Guide – Advanced Queuing (A96587)
Supplied PL/SQL Packages and Types Reference (A96612): DBMS_AQ (hoofdstuk 5), DBMS_AQADM (hoofdstuk 6), DBMS_AQELM (over notifications, hoofdstuk 7), Advanced Queuing Types (hoofdstuk 106) en JMS Types (hoofdstuk 107)
Verder zijn er een aantal interessante white papers op OTN te vinden.

Erwin Groenendal
is algemeen en technisch directeur van Cumquat Information Technology (e-mail: erwin.groenendal@cumquat.nl).