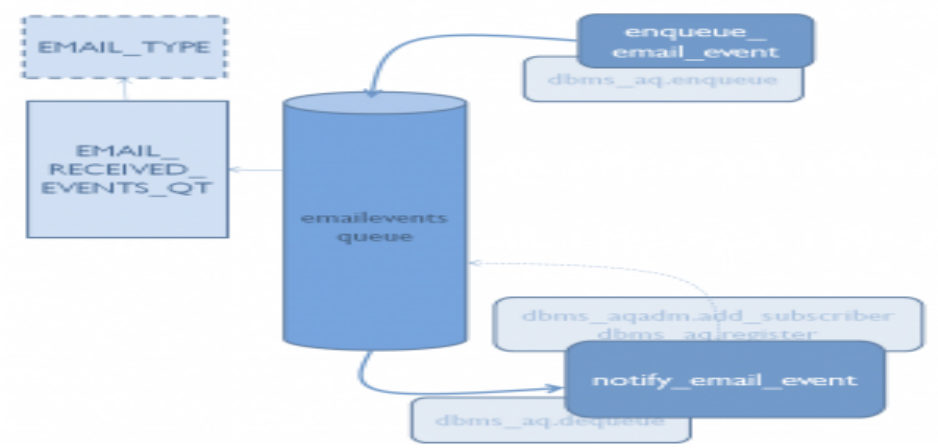
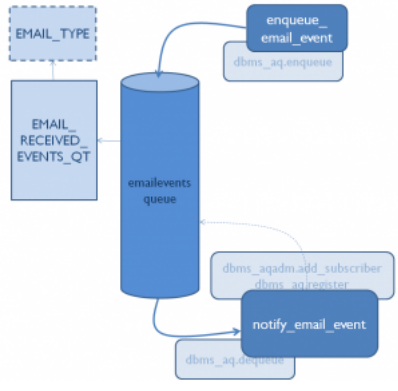


Event based Decoupling in PL/SQL – your first Advanced Queue adventure

1

BY LUCAS JELLEMA ON SEPTEMBER 17, 2010 DATABASE, DATABASES, ORACLE, PL/SQL, SOA

Back in 2004 when we started with the AMIS Technology Blog, my main objective was to record the things I infrequently do in order to have notes describing the steps to go through whenever I needed to do the thing again. I was own primary audience, so to say. Over the years, the articles have increased in complexity and sometimes in absurdity too. This one is back to that original intention. This article is not fancy at all – even though it touches upon a powerful (and underrated) subject: the Advanced Queue in the Oracle RDBMS. For the 1000s of database developers and architects that make frequent use of AQ, I am not going to add anything: this article merely shows the steps for creating an Advanced Queue, how to register a listener on the queue and how to publish a message on the queue. That is not fancy at all, obviously.



What *is* extremely fancy and powerful – and not nearly used enough – is the architectural pattern that AQ allows us to introduce. Queues are a key concept for achieving *decoupling*. Decoupling itself is like the holy grail of architects – because it allows agility and reuse. Through a Queue, a publisher (or discoverer) of events can make them available to consumers that may or may not exist and that may or may not be available at the time of publishing the event. The publisher of the event does its duty by putting the event on the queue and is then free to go. The queue infrastructure will take it from there.

This article is not about the decoupling that can be achieved, nor does it discuss Event Driven Architecture, even though queues are a foundational component for EDA. It simply shows the syntax for creating a queue, a publisher and a consumer. So whenever I next need to set this up, I remember how to do it.

In this article I will first create a queue, then register a listener on it and finally publish a message on it. Note that messages can of course be published to a queue, even in the absence of subscribers.

Create the Advanced Queue

Steps in creating a queue through Advanced Queuing

1. Define an Object Type to represent the message on the queue

ABOUT AUTHOR

Lucas Jellema

Lucas Jellema, active in IT (and with Oracle) since 1994. Oracle ACE Dire and Oracle Developer Champion. Solution architect developer on diverse areas including SQL, JavaScri Kubernetes & Docker, Machine Learning, Java, SO and microservices, events in various shapes and for and many other things. Author of the Oracle Press b Oracle SOA Suite 12c Handbook. Frequent presente on user groups and community events and conferer such as JavaOne, Oracle Code, CodeOne, NLJUG and Oracle OpenWorld.

View all posts

2. Create a Queue Table (as the messages on the Advanced Queue are persisted to a database table); the payload type specified for the Queue Table is the Object Type
3. Create the Queue itself, associated with the Queue Table
4. Start the Queue – activating it for receiving messages

First, define the payload of the messages that will be sent through the queue. We use an Object Type for this. In this example, the events represent Emails that have been received. The Object Type is called EMAIL_TYPE and it contains attributes for the essential aspects of an email:

```
create type email_type as object
( sent_date      timestamp(6)
, retrieved_date timestamp(6)
, subject        varchar2(1000)
, contents       varchar2(4000)
, from_sender    varchar2(1000)
, to_adressee    varchar2(1000)
, from_address   varchar2(1000)
);
```

Next, create the queue table, based on the payload type created above.

```
BEGIN
  dbms_aqadm.create_queue_table
  ( queue_table Â  Â  Â  Â => 'email_received_events_qt'
  , queue_payload_type  => 'email_type'
  , multiple_consumers  => true
  , comment Â  Â  Â  Â  Â  Â  => 'Email Received Events Notification Queue'
  );
END;
```

Then create the queue itself, based on the queue table (and thereby indirectly associated with the payload type email_type):

```
BEGIN
  dbms_aqadm.create_queue
  ( queue_name Â => 'emaileventsqueue'
  , queue_table => 'email_received_events_qt'
  );
END;
```

Finally, after creating the queue we have to formally get it going: just creating it is not enough to enable it to start receiving messages:

```
begin
  dbms_aqadm.start_queue( queue_name => 'emaileventsqueue');
end;
```

With this, the AQ queue has been set up. It is ready for anything that is coming.

Creating and Subscribing a Queue Consumer

A Consumer of messages from an Advanced Queue is a PL/SQL procedure with a predefined signature. It has to accept five input parameters that the AQ infrastructure provides. Using these parameters, the consumer can get its hands on the message itself. The message is effectively dequeued through a call to dbms_aq.dequeue:

```
create or replace
procedure notify_email_event
( context Â in raw
, reginfo Â in sys.aq$_reg_info
, descr Â  Â in sys.aq$_descriptor
, payload Â in raw
, payloadl in number)
as
  dequeue_options dbms_aq.dequeue_options_t;
  message_properties dbms_aq.message_properties_t;
  message_handle raw(16);
  message email_type;
begin
  dequeue_options.msgid := descr.msg_id;
  dequeue_options.consumer_name := descr.consumer_name;
```

Jun 27, 2

AMIS, stand out in advanced IT!
@AMISnl

New Article : Test Automation Days 2019 ift.tt/2FBPvyI by frederique.retsemaamis.nl

Test Automation Days 2019 - AMIS Oracle and ...
Verslag van drie presentaties op de tweede dag van de Test Automation Days 2019.
technology.amis.nl

Jun 27, 2

AMIS, stand out in advanced IT! Retweeted

Robert van Mölken
@robertvanmolken

Our @AMISnl #technology #blog turned 15 years. Hooray 😃 technology.amis.nl

Jun 25, 2

AMIS, stand out in advanced IT! Retweeted

Lucas Jellema
@lucasjellema

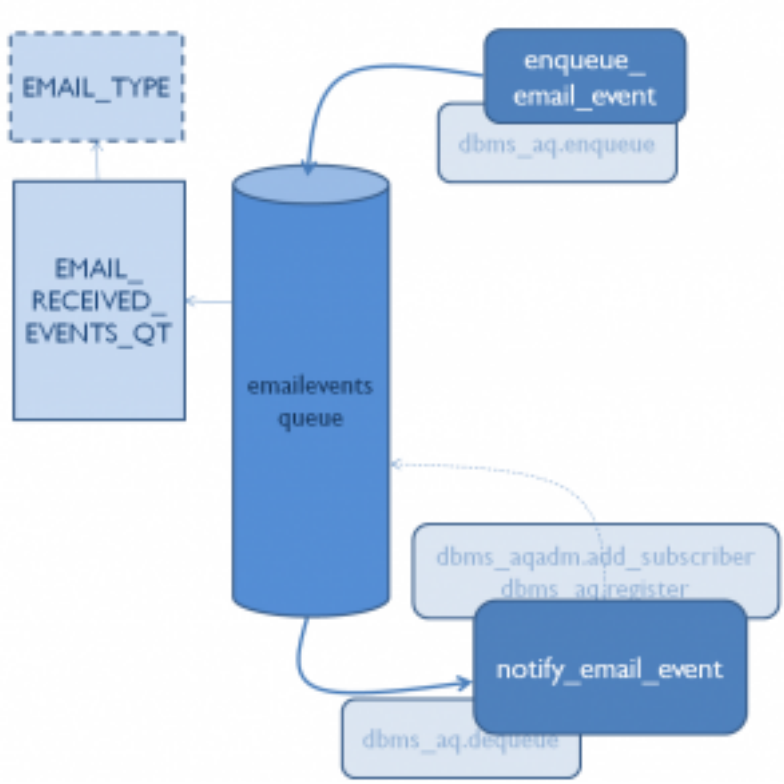
AMIS Academy is underway. Quarterly gathering with general session and 9 breakout sessions on OpenAPI DBaaS, Marketing, CQRS, Front Developnent , Data Analytics, REAL (@rexpertalliance), a customer case Ansible/Terraform/Cloudformation @AMISnl

```
dbms_aq.dequeue
( queue_name Â Â Â Â => descr.queue_name
, dequeue_options Â Â => dequeue_options
, message_properties => message_properties
, payload Â Â Â Â Â Â => message
, msgid Â Â Â Â Â Â Â => message_handle
);
-- take an action based on the message that was received from the queue
email_service_package.send
( p_from Â Â => 'maggie@stmatthews.com'
, p_to Â Â Â => message.from_address
, p_subject => 'response to: '||message.subject
, p_body Â Â => 'Your email was received and will be processed'
);
end;
```

When the PL/SQL procedure has been created according to the predefined signature, it can be registered with the queue as a consumer for its messages. With the following call to dbms_aqadm.add_subscriber, this procedure is identified as a recipient for messages that appear on the *emaileventsqueue*. Note that the add_subscriber procedure takes an optional parameter *rule* that can be used to specify a condition that messages should satisfy in order to be handed to this particular subscriber. It also accepts an optional parameter that may specify a transformation (PL/SQL function) to be performed on the message before delivery.

```
declare
  l_queue_name varchar2(30):='emaileventsqueue';
begin
  dbms_aqadm.add_subscriber
  ( queue_name => l_queue_name
  , subscriber => sys.aq$_agent( 'recipient', null, null )
  );
  dbms_aq.register
  ( sys.aq$_reg_info_list
    ( sys.aq$_reg_info
      ( l_queue_name||':recipient'
      , dbms_aq.namespace_aq
      , 'plsql://notify_email_event' -- name of the PL/SQL procedure that should be invoked to handle the message
      , hextoraw('ff')
      )
    )
  , 1
  );
end;
```

Publishing a message on the queue



Messages are published to an Advanced Queue from PL/SQL. Note that the publication of a message is is transactional: it occurs inside a database transaction and is only complete when the transaction is committed. This is unlike message sent through a database pipe (with dbms_pipe).

```
create or replace
procedure enqueue_email_event
( p_email in email_type)
```

Jun 25, 2(

AMIS, stand out in advanced IT!
@AMISnl

🙏 Samen! Een keer per kwartaal organiseert @AMIS de AMIS Academy. We komen samen, blikken terug, kijken vooruit en delen kennis. En vergeet het voortreffelijk diner niet. Met dit keer ijs als toetje! #AMISAcademy @Conclusion

Jun 25, 2(

Embed View on Twit

META

Log in

Entries RSS

Comments RSS

WordPress.org

```
is
  queue_options dbms_aq.enqueue_options_t;
  message_properties dbms_aq.message_properties_t;
  message_id raw(16);
begin
  dbms_aq.enqueue
  ( queue_name Â Â Â Â => 'emaileventsqueue'
  , enqueue_options Â Â => queue_options
  , message_properties => message_properties
  , payload Â Â Â Â Â Â => p_email
  , msgid Â Â Â Â Â Â Â => message_id
  );
  commit;
end;
```

A simple message can be send via the queue through a call to enqueue_email_event:

```
begin
  enqueue_email_event( p_email => email_type( systimestamp
                                , null
                                , 'email subject'
                                , 'some query '
                                , 'frank'
                                , 'maggie@stmatthews.com'
                                , 'frank@stmatthews.com'
                                )
                     );
end;
```

Miscellaneous

Queues can be enabled for single or multiple (publish/subscribe pattern)consumers – much like Queues and Topics in JMS. Consumers can be registered – as we have seen above – and invoked by the AQ infrastructure. Alternatively, consumers can actively inspect the queue for new messages to dequeue. Messages can be retained on the queue (for a specified period of time). Delivery of messages for which the delivery has failed can be automatically retried (after a specified period of time).

Messages can also be scheduled for automatic propagation to another queue. This propagation can be done conditionally – only when the payload of the message satisfies a specific condition – and can include a transformation – that turns the orginal payload into a different type according to a PL/SQL function that must be specified. The propagation can be controlled, for example to throttle the message flow, or to only propagate messages in specific time windows.

For documentation on AQ in 11gR2, see: http://download.oracle.com/docs/cd/E11882_01/server.112/e11013/aq_admin.htm

Related posts:

1. **Enqueuing AQ JMS Text Message from PL/SQL on Oracle XE**
2. **Extreme Decoupling in SQL – Select and DML against a table that is not a table at all**
3. **Oracle Advanced Queuing and JMS – bridging from AQ to JMS and vice versa**
4. **Scribble on a Pull-turns-push architecture based on http – registering web applications as event listeners**
5. **Interacting with JMS Queue and Topic from Java SE**

aq | decoupling | dequeue | listener | plsql | publish/subscribe | queue

ABOUT AUTHOR

LUCAS JELLEMA

Lucas Jellema, active in IT (and with Oracle) since 1994. Oracle ACE Director and Oracle Developer Champion. Solution architect and developer on diverse areas including SQL, JavaScript, Kubernetes & Docker, Machine Learning, Java, SOA and microservices, events in various shapes and forms and many other things. Author of the Oracle Press book Oracle SOA Suite 12c Handbook. Frequent presenter on user groups and community events and conferences such as JavaOne, Oracle Code, CodeOne, NLJUG JFall and Oracle OpenWorld.

RELATED POSTS

JUNE 17, 2019 0

Upgrade Failure as Blessing in Disguise?

APRIL 19, 2019 0

Creating a Report of available ASM candidate disks from the OEM Repository

APRIL 14, 2019 0

Extremely convenient way to run free Oracle Database 18c on your laptop – or anywhere else – using Vagrant & Virtual Box

1 COMMENT

Pingback: frank