Messaging with Oracle

Oracle bundles a messaging platform with the database. I consider this to be one of the more powerful technologies Oracle provides and it is also included in the price for the database. This messaging platform is widely used, but it's surprising how many Oracle shops there are that still are unaware that they have already paid for a massaging platform. AQ may be more suited for their needs than the solution they have built or, even worse, bought. This article is for those that want to know a little about it, I will not dig deep into the rich feature set Oracle provides.

You can read more about Advanced Queuing (AQ) at http://www.oracle.com/technology/products/aq/index.html.

When and why would one use AQ

Messaging is a concept often used to build processing streams or to perform part of the processing later. The most common use of messaging is to use asynchronous messages. In this mode, one process will leave a message for another process. That message could be picked up one second later or one week later. The first process doesn't need to know what the second process does with the message. There is usually no feedback of status. The first process is only responsible for leaving a message for the second process, once that is done the first process has completed its part of the work.

Consider a situation where you want to set up your processing such that you have one process to receive requests, one to process the request and add data from your customer data, and one process to store the data after it has been processed. This processing stream would be a perfect candidate for messaging. You may need just one instance of the first one to handle to workload, while three are needed of the second and seven of the last process. Maybe you want to process data all day long and add the data to the database after business hours.

You could also use messaging to allow implementation of processes in different languages. The example above could have the first process implemented in Perl, the second in Java, and the last in C++.

Oracle AQ provides all the usual benefits of messaging, and it provides one benefit that is hard to get with any other messaging platform. AQ messages are stored in the database, and your messages are synchronized together with your data. A common problem is how you get messages and data to be persistent and committed at the same time. Oracle has solved this by giving us an API that we can access via the normal database connection. The result is that you can commit DML and messages with one commit.

Overview

This article will show how a message is written by one process and later picked up by another. This is what you want from asynchronous producer-consumer messaging. There are other modes of messaging with AQ, but this is what most messaging solutions need.

Every Technical Architect ought to have a working understanding for the architecture and features provided by AQ. Building effective process streams and ensuring synchronization of messages and data is a key task for an Architect.

This article will not attempt to provide code that is ready for production use. Rather, I have removed error checking for brevity. The examples are shown in PL*SQL, as it is the language most Oracle specialists have in common.

Setup

Before messages can be created, we need to set up the message queue. There are five steps required. Each step is performed with a DDL statement or a call to a stored procedure

The schema that will own the database objects must have execute rights on dbms_aq and dbms_aqadm.

Create message payload

The payload is the information one application wishes to pass to another application. The payload can be of two types, you can use RAW, or a TYPE. RAW is the easiest as it only requires use of the normal RAW data type. A TYPE is the normal Oracle TYPE used in object relational features.

Since this article is focused on messaging basics, we will use a RAW payload. Using a TYPE based message is not hard at all, but it is not needed to show messaging basics.

Create a queue table

A message is written to a queue and stored in a queue table. A queue table can be the store for many message queues. Every queue in a queue table will have the same payload definition.

This statement will create a queue table "MY_QT" that holds messages with a RAW payload that are sorted by the time the message was written (enqueued).

```
exec dbms_aqadm.create_queue_table
       (queue_table        => 'my_qt'
       ,sort_list          => 'ENQ_TIME'
       ,queue_payload_type => 'RAW'
       ,message_grouping   => DBMS_AQADM.NONE)
```

Creating a queue

Messages are accessed via a queue. Each kind of processing request will usually have its own queue. A message queue is a way to logically group messages based on what kind of processing they will result in.

This statement creates a queue MY_Q in the queue table MY_QT. The messages in this queue will be kept for on hour (3600 seconds) after the message was processed successfully. The message is moved to an exception queue if the processing application rolls back 5 times, and it will be returned to the queue 15 seconds after an attempt to process it has been rolled back.

```
exec dbms_aqadm.create_queue
       (queue_name     => 'my_q'
       ,queue_table    => 'my_qt'
       ,retention_time => 3600
       ,max_retries    => 5
       ,retry_delay    => 15)
```

Start the message queue

When a queue is created it is stopped, so before we can use it we have to start the queue we just created.

This statement starts the queue MY_Q so messages can be enqueued or dequeued.

```
exec dbms_aqadm.start_queue(queue_name=>'my_q')
```

Granting privileges on the queue

Just as with tables, other database users than the owner has to be given explicit rights to the queue.

This statement will give the user MY_Q_USER both ENQUEUE and DEQUEUE rights to the queue MY_Q.

```
exec dbms_aqadm.grant_queue_privelege
       (privilege  => ‘ALL’
       ,queue_name => 'my_q'
       ,grantee    => ‘my_q_user’)
```

Producing messages

Producer is the messaging term for an application that writes a message. The following code will produce a message that contains the string “This is a Message!”.

A message that is written is said to have been enqueued, and the message is sitting in a queue waiting to be picked up by a consumer.

```
declare
eo dbms_aq.enqueue_options_t;
mp dbms_aq.message_properties_t;
pl RAW(12) := utl_raw.cast_to_raw('This is a message!');
mh RAW(16);

begin
```

```
dbms_aq.enqueue(
queue_name              => 'my_q',
enqueue_options         => eo,
message_properties      => mp,
payload                 => pl,
msgid                   => mh);

commit;

end;
/
```

The code enqueues a message to the MY_Q queue. The payload contains the actual request. Payload is another messaging term, it is the information you want to pass to another application.

Now that the producer has left the message, it can go on doing other things without even caring if the message is ever picked up. That is all there is to producing a message. The queue can contain thousands of messages that are waiting for a consumer to pick them up.

Consuming a message

Consumer is the messaging term for the process that reads and processes a message. The following code will consume the message we just enqueued.

```
set servoutput 1000;
declare
do dbms_aq.dequeue_options_t;
mp dbms_aq.message_properties_t;
mh RAW(16);
pl RAW(20);

begin

dbms_aq.dequeue(
queue_name              => 'myQ',
dequeue_options         => do,
message_properties      => mp,
payload                 => pl,
msgid                   => mh);

commit;
dbms_output.put_line('Payload="'||utl_raw.cast_to_varchar2(pl)||'"');

end;
/
```

The code enqueues a message to the MY_Q queue. As you can see, the code is very similar to the enqueing code.

Note that the consumer processed the message without requiring the producer to wait.

The commit marks the message as consumed. If the transaction would roll back, the message would be returned to the queue.

Controlling messages

Messages can be controlled with three AQ types, the enqueue, dequeue, and message property type. Following is a brief overview of each of those types.

Enqueue Options

The enqueue options type is one of the arguments for the enqueue call. This argument has to be of type dbms_aq.enqueue_options_t. This type has four fields that allow you to control dequeue order, transformation, and visibility. In most cases you'll not use any of these fields. The transformation and dequeue order fields are for very advanced messaging needs. Visibility controls when the message becomes visible to consumers. The default is ON_COMMIT which means that it can be seen by the consumers when you commit the transaction where the message was enqueued. The visibility can be set to IMMEDIATE, which means that it will be seen immediately by the consumers.

Message Properties

The message properties type is one of the arguments for the enqueue and dequeue calls. This argument has to be of type dbms_aq.message_properties_t. This type contains 13 fields that can be used to describe and control the message. Most messaging solutions will not have to change any of the defaults. The most commonly used field is probably the priority which allows you to enqueue a message that should be processed before other messages in the queue. The delay field can be used for setting a number of seconds the message should be invisible to the consumer applications. Expiration lets you configure a time limit after which the message expires and can no longer be dequeued. There is even a field that allows an application to provide its own message properties.

Dequeue Options

The dequeue options type is one of the arguments for the dequeue call. This argument has to be of type dbms_aq.dequeue_options_t. This type contains nine fields. Navigation is used to specify if you want to get the next message or to get the first message in the queue. This is similar to continue fetching or reopen a cursor. There is a visibility field for dequeuing too, and it works the same way as with enqueuing. The time to wait for a message (if the queue is empty) can be set to no wait, wait forever, or a number of seconds to wait. A dequeue condition can be specified to filter which messages to pick a message from. As with enqueuing, messages can be transformed in the dequeue operation.

Summary

You have now seen how easy it is to use Oracle AQ to set up a processing stream where each process can work on its part of the stream independent from other parts. There are many competing platforms for messaging, the benefits with Oracle is that the price is right, data and messages are synchronized, and AQ is a feature rich messaging platform. Did I mention that the price is right?

AQ is also surprisingly fast; it will probably meet your needs unless you are looking for real time messaging. My experience is that AQ can be up to ten times faster than home grown file based messaging solutions.

Author Bio

Mathias Magnusson is a senior performance and database architect with 16 years experience in building and tuning high performance solutions.