This article shows you how to exchange information between Oracle and WebSphere MQ applications using two different approaches: Oracle Advanced Queuing (AQ) to MQ propagation, and Java clients for AQ.

Introduction

This article shows you how to integrate an Oracle database with IBM® WebSphere® MQ applications, using Oracle Advanced Queuing (AQ), which is the messaging engine provided with the Oracle database. AQ leverages the functions of the Oracle database so that messages can be stored persistently, propagated between queues on different machines and databases, and transmitted using Oracle Net Services, HTTP(S), and SMTP. Since Oracle AQ is implemented in database tables, all the operational benefits of high availability, scalability, and reliability apply to the queue data. Standard database features such as recovery, restart, and security are supported in AQ.

AQ capabilities depend on the following AQ components:

Queue

The repository of messages. A queue contains message for asynchronous communication between applications.

Queue table

A table in the Oracle database that hosts the message data for all the queues defined within it. It contains information such as message id, put time, and user data for the message. Every queue belongs to one queue table, and one queue table may have multiple queues within it.

Oracle AQ provides messaging capabilities like other messaging providers (including WebSphere MQ). These capabilities are built over the database engine, and the messages are persisted in the Oracle database.

This article describes two integration approaches: using Oracle AQ and the Oracle Messaging Gateway, and and using Java clients for AQ.
Oracle Messaging Gateway

Oracle Messaging Gateway (OMG) provides direct propagation of AQ messages to WebSphere MQ queues. An Oracle AQ queue can be linked to a WebSphere MQ queue and as soon as messages are put in the AQ queue, OMG propagates them to WebSphere MQ queues. Here are the components of OMG:

Messaging System Link
The link between AQ and MQ. A reference to the link is used by subsequent operations of configuration.

Foreign Queue
Every MQ queue which is to be linked with AQ queue needs to be registered as foreign queue in Oracle AQ.

Subscriber
A subscriber is required to link AQ queue with a foreign queue. Subscriber also holds the information Whether the propagation is outbound or inbound.

Propagation Schedule
A subscriber is required to be scheduled for propagating the messages between AQ and MQ. The schedule defines the interval at which messages will be propagated between AQ and MQ.

Configuration steps
First, OMG should be configured on the database instance where you are trying to create the queues. Instructions to set up the OMG are available in the product documentation. To check OMG status, use the following command:

```sql
select AGENT_STATUS,AGENT_PING,LAST_ERROR_MSG from MGW_GATEWAY
```

The output should be something like:

```sql
begin
  dbms_mgwadm.startup;
end;
```

Take a single queue for our sample and name it OMG_SAMPLE_QUEUE. The queue table hosts a queue. For this sample, create the queue table and queue with the names SAMPLE_QUEUE_TABLE and OMG_SAMPLE_QUEUE respectively:

```sql
begin
  DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'SAMPLE_QUEUE_TABLE',
  multiple_consumers=>TRUE,Queue_payload_type => 'RAW',compatible => '8.1');
  DBMS_AQADM.CREATE_QUEUE (Queue_name => 'OMG_SAMPLE_QUEUE',
  Queue_table => 'SAMPLE_QUEUE_TABLE');
end;
```

Start the queue so that it is available to the applications:

```sql
begin
  DBMS_AQADM.START_QUEUE (Queue_name => 'OMG_SAMPLE_QUEUE');
end;
```
Now that AQ queue is created and started, create the messaging link to the WebSphere MQ provider:

```sql
declare
    v_options sys.mgw_properties;
    v_prop sys.mgw_mqseries_properties;
begin
    v_prop := sys.mgw_mqseries_properties.construct();
    v_prop.max_connections := 1;
    v_prop.interface_type := DBMS_MGWADM.JMS_QUEUE_CONNECTION;
    v_prop.username := null;
    v_prop.password := null;
    v_prop.hostname := '10.13.41.124';
    v_prop.port := 5421;
    v_prop.channel := 'SYSTEM.DEF.SVRCONN';
    v_prop.queue_manager := 'QMEIAS1';
    v_prop.outbound_log_queue := 'OMG_OUT_LOG_QUEUE';
    v_prop.inbound_log_queue := 'OMG_IN_LOG_QUEUE';
    dbms_mgwadm.create_msgsystem_link(
        linkname => 'OMG_LINK', properties => v_prop, options => v_options );
end;
```

The next step is to register the MQ queue as a foreign queue:

```sql
begin
    dbms_mgwadm.register_foreign_queue(
        name => 'OMG_MQ_QUEUE',
        linkname => 'OMG_LINK',
        provider_queue => 'OMG_MQ_QUEUE',
        domain => dbms_mgwadm.DOMAIN_QUEUE);
end;
```

Create a subscriber for outbound propagation on this queue:

```sql
begin
    dbms_mgwadm.add_subscriber(
        subscriber_id => 'SUB_OMG_SAMPLE_QUEUE',
        propagation_type => dbms_mgwadm.outbound_propagation,
        queue_name => 'apps.OMG_SAMPLE_QUEUE',
        destination => 'OMG_MQ_QUEUE@OMG_LINK');
end;
```

As discussed above, a propagation schedule is necessary for transferring messages from AQ to MQ. To create it:

```sql
begin
    dbms_mgwadm.schedule_propagation(
        schedule_id => 'SCH_OMG_SAMPLE_QUEUE',
        propagation_type => dbms_mgwadm.outbound_propagation,
        source => 'apps.OMG_SAMPLE_QUEUE',
        destination => 'OMG_MQ_QUEUE@OMG_LINK',
        latency => 10);
end;
```

Enable the propagation schedule so that automatic message transfer from AQ to MQ is enabled:
begin
dbms_mgwadm.enable_propagation_schedule('SCH_OMG_SAMPLE_QUEUE');
end;

**Testing the propagation**

AQ to MQ propagation is ready. If you have successfully configured it using the above steps, you are ready to test the propagation. Check the depth of the MQ queue to ensure that queue is empty before a message is transferred from AQ:

runmqsc QMEIAS1
display ql(OMG_MQ_QUEUE) curdepth

Output should look like this:

runmqsc QMEIAS1
display ql(OMG_MQ_QUEUE) curdepth
1 : display ql(OMG_MQ_QUEUE) curdepth
AMQ8409: Display Queue details.
QUEUE(OMG_MQ_QUEUE) CURDEPTH(0)

It shows that there are no messages in the MQ queue. Put a sample message into the AQ queue and see it transferred to the MQ queue:

```sql
DECLARE
  queue_options  DBMS_AQ.ENQUEUE_OPTIONS_T;
  message_properties DBMS_AQ.MESSAGE_PROPERTIES_T;
  message_id   RAW(16);
  p_xmlstring   varchar2(3000);
  queue_name_val  varchar2(100);
  v_agent    sys.aq$_agent := sys.aq$_agent(' ', null, 0);
  v_jms_message  sys.aq$_jms_text_message;
  enqueue_options  dbms_aq.enqueue_options_t;
  msgid    raw(16);
BEGIN
  queue_name_val := 'OMG_SAMPLE_QUEUE';
  v_jms_message := sys.aq$_jms_text_message.construct;
  v_jms_message.set_replyto(v_agent);
  v_jms_message.set_type('mcd://xmlns');
  v_jms_message.set_text(p_xmlstring);
  DBMS_AQ.ENQUEUE(
    queue_name => queue_name_val,
    enqueue_options => queue_options,
    message_properties => message_properties,
    payload => v_jms_message,
    msgid => message_id);
END;
```

Once again, check the depth of the MQ queue. It should show a depth of 1:

runmqsc QMEIAS1
display ql(OMG_MQ_QUEUE) curdepth
1 : display ql(OMG_MQ_QUEUE) curdepth
AMQ8409: Display Queue details.
QUEUE(OMG_MQ_QUEUE) CURDEPTH(1)

Now that you have transferred the messages using AQ-MQ propagation, here is the other alternative.
Java client
In this approach, use a Java API-based client to transfer the messages from AQ to MQ directly. This approach is useful when additional processing must be applied to the message, such as when an MQ application requires that additional headers be sent along with the message. For example, in Oracle 9i, you cannot send user-defined headers with messages.

Java APIs for AQ
Oracle’s JMS interface to AQ provides Java-based APIs for interacting with AQ.

Components of the JMS interface to AQ:

Queue Connection Factory
Required for creating and accessing queues. It is accessed by JDBC using a connection string consisting of host name, port, SID, username, and password.

Queue connection
Required to create the queue session.

Queue session
Required to create and access the queues. It contains two parameters: transaction behavior and client acknowledge. This means that session will be transactional and client will issue commit or rollback.

Queue receiver object
Required to receive messages from AQ. It is associated with a queue and listens for incoming messages when instructed.

Configuration steps
Configure the OMG as described above. Continue creating the rest of the objects as shown below. Take a single queue and name it JMS_SAMPLE_QUEUE. The queue table hosts a queue. For this sample, create the queue table with the name SAMPLE_QUEUE_TABLE:

```
BEGIN
    DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'SAMPLE_QUEUE_TABLE', queue_payload_type => 'SYS.AQ$_JMS_TEXT_MESSAGE', multiple_consumers => FALSE, compatible => '8.1');
    DBMS_AQADM.CREATE_QUEUE (Queue_name => 'JMS_SAMPLE_QUEUE', Queue_table => 'SAMPLE_QUEUE_TABLE');
END;
```

Start the AQ queue so that it is available to the applications:

```
BEGIN
    DBMS_AQADM.START_QUEUE (Queue_name => 'JMS_SAMPLE_QUEUE');
END;
```

Configuring the Java client
If you have worked with Java clients, this part should be easy. If not, each step is described below. Start by getting the queue connection factory. This factory is used for creating sessions with the Java provider, which in our case is Oracle AQ.

```
QueueConnectionFactory qc_fact = AQjmsFactory.getQueueConnectionFactory(host, ora_sid, port, driver);
```
where:

**host**
Hostname or IP of the Oracle AQ server

**ora_sid**
SID of the Oracle database instance which hosts the queues

**port**
Numeric port number of database listener

**driver**
Type of driver used for connecting to Oracle database. We will use thin driver.

Create the queue connection:

```java
q_conn = qc_fact.createQueueConnection(user, pwd);
```

where:

**user**
Username for connecting to the database

**pwd**
Password of the user

Create and start the queue session from queue connection:

```java
q_sess = q_conn.createQueueSession(true, Session.CLIENT_ACKNOWLEDGE);
qu_conn.start();
```

Get the reference and start the AQ queue

```java
queue = ((AQjmsSession) q_sess).getQueue(schema, "JMS_SAMPLE_QUEUE");
((AQjmsDestination) queue).start(q_sess, true, true);
```

where

**schema**
Schema name of Oracle in which queue exists -- usually the username with which the queue is created

Create a queue receiver which will read the message from AQ queue:

```java
q_recvr = q_sess.createReceiver(queue);
```

Read the message from queue using the receiver:

```java
TextMessage obj_message = (TextMessage) q_recvr.receive(10);
```

Print the message text from the AQ queue:

```java
if (obj_message != null)
    System.out.println("Message recieved is : " + obj_message.getText());
```
Commit the session and close the objects:

```java
q_sess.commit();
q_recrv.close();
q_sess.close();
q_conn.close();
```

Now that you have the text message, you can use the JMS APIs to put into MQ queue. The complete code is below.

### Testing the Java client

If you have successfully configured the above steps, you are ready to test the Java client. First check the depth of the MQ queue to ensure that the queue is empty before a message is transferred from AQ.

```sh
cmrqsc QMEIAS1
display ql(JMS_MQ_QUEUE) curdepth
```

The output should look like this:

```
display ql(JMS_MQ_QUEUE) curdepth
   1 : display ql(JMS_MQ_QUEUE) curdepth
AMQ8409: Display Queue details.
   QUEUE(JMS_MQ_QUEUE) CURDEPTH(0)
```

It shows that there are no messages in the MQ queue. Put a sample message into the AQ queue:

```java
DECLARE
    queue_options  DBMS_AQ.ENQUEUE_OPTIONS_T;
    message_properties DBMS_AQ.MESSAGE_PROPERTIES_T;
    message_id  RAW(16);
    p_xmlstring  varchar2(3000);
    queue_name_val  varchar2(100);
    v_agent   sys.aq$_agent := sys.aq$_agent(' ', null, 0);
    v_jms_message  sys.aq$_jms_text_message;
    enqueue_options  dbms_aq.enqueue_options_t;
    msgid              raw(16);
BEGIN
    queue_name_val := 'JMS_SAMPLE_QUEUE';
    v_jms_message := sys.aq$_jms_text_message.construct;
    v_jms_message.set_replyto(v_agent);
    v_jms_message.set_type('mcd://xmlns');
    v_jms_message.set_text(p_xmlstring);
    DBMS_AQ.ENQUEUE(
        queue_name => queue_name_val,
        enqueue_options => enqueue_options,
        message_properties => message_properties,
        payload => v_jms_message,
        msgid => message_id);
END;
```

Now that message is lying in the AQ queue, run the Java client to dequeue the message from the AQ queue (JMS_SAMPLE_QUEUE) and enqueue it into the MQ queue (JMS_MQ_QUEUE):

```java
java SamplQAqToMq
```
Once again, check the depth of the MQ queue. It should show the current depth as 1:

```
display ql(JMS_MQ_QUEUE) curdepth
1 : display ql(JMS_MQ_QUEUE) curdepth
AMQ8409: Display Queue details.
  QUEUE(JMS_MQ_QUEUE)       CURDEPTH(1)
```

**Conclusion**

This article described Oracle Advanced Queuing, and showed you how to use automatic propagation of messages from AQ to MQ. If you prefer Java clients, it also described the Java APIs for transferring the messages from AQ to MQ. If you want to read more about AQ, see the Reference section below.
Related topics

- **WebSphere MQ developer resources page**
  Technical resources to help you design, develop, and deploy messaging middleware with WebSphere MQ to integrate applications, Web services, and transactions on almost any platform.

- **WebSphere MQ product page**
  Product descriptions, product news, training information, support information, and more.

- **WebSphere MQ V6 trial download**
  A no-charge trial download of WebSphere MQ V6. Includes limited online support for Windows® and Linux® installations at no charge during the trial period.

- **WebSphere MQ V6 information center**
  A single Eclipse-based Web portal to all WebSphere MQ V6 documentation, with conceptual, task, and reference information on installing, configuring, and using your WebSphere MQ environment.

- **WebSphere MQ documentation library**
  WebSphere MQ product manuals.

- **WebSphere MQ SupportPacs**
  Downloadable code, documentation, and performance reports for the WebSphere MQ family of products.

- **WebSphere MQ public newsgroup**
  A non-IBM forum where you can get answers to your WebSphere MQ technical questions and share your WebSphere MQ knowledge with other users.

- **Introduction to Oracle Advanced Queuing**
  From the Oracle9i Application Developer’s Guide

- **Managing Oracle Advanced Queuing**
  From the Oracle9i Application Developer’s Guide

- **Creating an application using JMS**
  From the Oracle9i Application Developer’s Guide

- **Oracle Messaging Gateway**
  From the Oracle9i Application Developer’s Guide

- **developerWorks WebSphere Business Integration zone**
  For developers, access to WebSphere Business Integration how-to articles, downloads, tutorials, education, product info, and more.

- **WebSphere Business Integration products page**
  For both business and technical users, a handy overview of all WebSphere Business Integration products

- **Most popular WebSphere trial downloads**
  No-charge trial downloads for key WebSphere products.