30 Oracle Advanced Queuing

Oracle Advanced Queuing (AQ) provides database-integrated message queuing functionality, which is built on top of Oracle Streams.

It optimizes the functions of Oracle Database, so that messages can be stored persistently, propagated between queues on different computers and databases, and transmitted using Oracle Net Services, HTTP, and HTTPS. Oracle AQ is implemented in database tables, so all operational benefits of high availability, scalability, and reliability are also applicable to queue data. This chapter provides information about the Java interface to Oracle AQ.

Note:

- Oracle Advanced Queuing (AQ) is a feature of the Oracle JDBC Thin driver and is not supported by JDBC OCI driver.
- In Oracle Database 21c Release, support for JSON queues was added. Till Oracle Database 19c Release, supported payload types were `RAW`, `ADT`, `ANYDATA` and `XMLType`. In Oracle Database 23ai, you can also use JSON payload type with array enqueue and dequeue operations.

See Also:

Oracle Database Advanced Queuing User's Guide

This chapter covers the following topics:

- Functionality and Framework of Oracle Advanced Queuing
- Making Changes to the Database
- AQ Asynchronous Event Notification
- About Creating Messages
- Enqueuing Messages
- Dequeuing Messages
- Examples: Enqueuing and Dequeuing

30.1 Functionality and Framework of Oracle Advanced Queuing

The Oracle JDBC package `oracle.jdbc.aq` provides a fast Java interface to AQ. This package contains the following:

- Classes
  - `AQDequeueOptions`

    Specifies the options available for the dequeue operation

  – `AQEnqueueOptions`

    Specifies the options available for the enqueue operation

  – `AQFactory`

    Is a factory class for AQ

  – `AQNotificationEvent`

    Is created whenever a new message is enqueued in a queue for which you have registered your interest

- Interfaces
  – `AQAgent`

    Used to represent and identify a user of the queue or a producer or consumer of the message

  – `AQMessage`

    Represents a message that is enqueued or dequeued

  – `AQMessageProperties`

    Contains message properties such as Correlation, Sender, Delay and Expiration, Recipients, and Priority and Ordering

  – `AQNotificationListener`

    Is a listener interface for receiving AQ notification events

  – `AQNotificationRegistration`

    Represents your interest in being notified when a new message is enqueued in a particular queue

These classes and interfaces enable you to access an existing queue, create messages, and enqueue and dequeue messages.

Note:

Oracle JDBC drivers do *not* provide any API to create a queue. Queues must be created through the `DBMS_AQADM` PL/SQL package.

See Also:

For more information about the APIs, refer to *Oracle Database JDBC Java API Reference*.

30.2 Making Changes to the Database

The code snippets used in this chapter assume that user `HR` is connecting to the database. Therefore, in the database, you must grant the following privileges to `HR`:

```
GRANT EXECUTE ON DBMS_AQ to HR;
GRANT EXECUTE ON DBMS_AQADM to HR;
GRANT AQ_ADMINISTRATOR_ROLE TO HR;
GRANT ADMINISTER DATABASE TRIGGER TO HR;
```

Before you start enqueuing and dequeuing messages, you must have queues in the Database. For this, you must perform the following:

1. Create a queue table in the following way:

```
BEGIN
    DBMS_AQADM.CREATE_QUEUE_TABLE(
            QUEUE_TABLE =>'HR.RAW_SINGLE_QUEUE_TABLE',
            QUEUE_PAYLOAD_TYPE =>'RAW',
            COMPATIBLE => '10.0');
END;
```

2. Create a queue in the following way:

```
BEGIN
    DBMS_AQADM.CREATE_QUEUE(
            QUEUE_NAME =>'HR.RAW_SINGLE_QUEUE',
            QUEUE_TABLE =>'HR.RAW_SINGLE_QUEUE_TABLE',
END;
```

3. Start the queue in the following way:

```
BEGIN
    DBMS_AQADM.START_QUEUE(
 'HR.RAW_SINGLE_QUEUE',
END;
```

It is a good practice to stop the queue and remove the queue tables from the database. You can perform this in the following way:

1. Stop the queue in the following way:

```
BEGIN
    DBMS_AQADM.STOP_QUEUE(
 HR.RAW_SINGLE_QUEUE',
END;
```

2. Remove the queue tables from the database in the following way:

```
BEGIN
    DBMS_AQADM.DROP_QUEUE_TABLE(
            QUEUE_TABLE =>'HR.RAW_SINGLE_QUEUE_TABLE',
            FORCE => TRUE
END;
```

30.3 AQ Asynchronous Event Notification

A JDBC application can do the following:

- Register to the AQ namespace and receive notification when an enqueue occurs. This can be performed in the following way:

```
 public AQNotificationRegistration registerForAQEvents(
    OracleConnection conn,
    String queueName) throws SQLException
  {
    Properties globalOptions = new Properties();
    String[] queueNameArr = new String[1];
    queueNameArr[0] = queueName;
    Properties[] opt = new Properties[1];
    opt[0] = new Properties();
    opt[0].setProperty(OracleConnection.NTF_AQ_PAYLOAD,"true");
    AQNotificationRegistration[] regArr =
conn.registerAQNotification(queueNameArr,opt,globalOptions);
    AQNotificationRegistration reg = regArr[0];
    return reg;
  }
```

- Register subscriptions to database events and receive notifications when the events are triggered

  Registered clients are notified asynchronously when events are triggered or on an explicit AQ enqueue (or a new message is enqueued in a queue for which you have registered your interest). Clients do not need to be connected to a database.

  The following code snippet shows how to subscribe to database events and receive notifications when the events are triggered:

```
class DemoAQRawQueueListener implements AQNotificationListener
{
  OracleConnection conn;
  String queueName;
  String typeName;
  int eventsCount = 0;

  public DemoAQRawQueueListener(String _queueName, String _typeName)
   throws SQLException
  {
   queueName = _queueName;
   typeName = _typeName;
   conn = (OracleConnection)DriverManager.getConnection
     (DemoAQRawQueue.URL, DemoAQRawQueue.USERNAME, DemoAQRawQueue.PASSWORD);
  }

  public void onAQNotification(AQNotificationEvent e)
  {
    try
    {
     AQDequeueOptions deqopt = new AQDequeueOptions();
     deqopt.setRetrieveMessageId(true);
     if(e.getConsumerName() != null)
       deqopt.setConsumerName(e.getConsumerName());
     if((e.getMessageProperties()).getDeliveryMode()
        == AQMessageProperties.DeliveryMode.BUFFERED)
     {
       deqopt.setDeliveryMode(AQDequeueOptions.DEQUEUE_BUFFERED);
       deqopt.setVisibility(AQDequeueOptions.DEQUEUE_IMMEDIATE);
     }
     AQMessage msg = conn.dequeue(queueName,deqopt,typeName);
     byte[] msgId = msg.getMessageId();
     if(msgId != null)
     {
       String mesgIdStr = DemoAQRawQueue.byteBufferToHexString(msgId,20);
       System.out.println("ID of message dequeued = "+mesgIdStr);
     }
     System.out.println(msg.getMessageProperties().toString());
     byte[] payload = msg.getPayload();
     if(typeName.equals("RAW"))
     {
       String payloadStr = new String(payload,0,10);
       System.out.println("payload.length="+payload.length+", value="+payloadStr);
     }
```

```
      }
      catch(SQLException sqlex)
      {
       System.out.println(sqlex.getMessage());
      }
      eventsCount++;
    }
    public int getEventsCount()
    {
      return eventsCount;
    }
    public void closeConnection() throws SQLException
    {
      conn.close();
    }
  }
```

- Register to the listener in the following way:

  ```
  AQNotificationRegistration reg = registerForAQEvents(conn,queueName+":BLUE");
  DemoAQRawQueueListener demo_li = new DemoAQRawQueueListener(queueName,queueType);
  reg.addListener(demo_li);
  ```

30.4 About Creating Messages

This section describes the following concepts:

- Creating Messages
- AQ Message Properties
- AQ Message Payload

30.4.1 Creating Messages

Before you enqueue a message, you must create the message. An instance of a class implementing the `AQMessage` interface represents an AQ message. An AQ message contains properties (metadata) and a payload (data). Perform the following to create an AQ message:

1. Create an instance of `AQMessageProperties` in the following way:

   ```
   AQMessageProperties msgprop = AQFactory.createAQMessageProperties();
   ```

2. Set the property attributes in the following way:

   ```
   msgprop.setCorrelation("mycorrelation");
   msgprop.setExceptionQueue("MY_EXCEPTION_QUEUE");
   msgprop.setExpiration(0);
   msgprop.setPriority(1);
   ```

3. Create the AQ message using the `AQMessageProperties` object in the following way:

   ```
   AQMessage mesg = AQFactory.createAQMessage(msgprop);
   ```

4. Set the payload in the following way:

   ```
   byte[] rawPayload = "Example_Payload".getBytes();
   mesg.setPayload(new oracle.sql.RAW(rawPayload));
   ```

30.4.2 AQ Message Properties

The properties of the AQ message are represented by an instance of the `AQMessageProperties` interface. You can set or get the following message properties:

- Dequeue Attempts Count: Specifies the number of attempts that have been made to dequeue the message. This property cannot be set.
- Correlation: Is an identifier supplied by the producer of the message at the time of enqueuing the message.
- Delay: Is the number of seconds for which the message is in the `WAITING` state. After the specified delay, the message is in the `READY` state and available for dequeuing. Dequeuing a message by using the message ID (msgid) overrides the delay specification.

  > **Note:**
  >
  > Delay is not supported with buffered messaging.

- Delivery Mode: Specifies whether the message is a buffered message or a persistent message. This property cannot be set.
- Enqueue Time: Specifies the time at which the message was enqueued. This value is determined by the system and cannot be set by the user.
- Exception Queue: Specifies the name of the queue into which the message is moved if it cannot be processed successfully. Messages are moved in two cases:
  - The number of unsuccessful dequeue attempts has exceeded `max_retries`.
  - The message has expired.
- Expiration: Is the number of seconds during which the message is available for dequeuing, starting from when the message reaches the `READY` state. If the message is not dequeued before it expires, then it is moved to the exception queue in the `EXPIRED` state.
- Message State: Specifies the state of the message at the time of dequeuing the message. This property cannot be set.
- Previous Queue Message ID: Is the ID of the message in the last queue that generated the current message. When a message is propagated from one queue to another, this attribute identifies the ID of the queue from which it was last propagated. This property cannot be set.
- Priority: Specifies the priority of the message. It can be any integer including negative integers; the smaller the value, the higher the priority.
- Recipient list: Is a list of `AQAgent` objects that represent the recipients. The default recipients are the queue subscribers. This parameter is valid only for multiple-consumer queues.
- Sender: Is an identifier specified by the producer at the time of enqueuing the message. It is an instance of `AQAgent`.
- Transaction group: Specifies the transaction group of the message for transaction-grouped queues. It is set after a successful call to the `dequeueArray` method.

30.4.3 AQ Message Payload

Depending on the type of the queue, the payload of the AQ message can be specified using the `setPayload` method of the `AQMessage` interface. The following code snippet illustrates how to set the payload:

```
...
byte[] rawPayload = "Example_Payload".getBytes();
```

```
mesg.setPayload(new oracle.sql.RAW(rawPayload));
...
```

You can retrieve the payload of an AQ message using the `getPayload` method or the appropriate `get`*`XXX`*`Payload` method in the following way:

```
byte[] payload = mesg.getPayload();
```

These methods are defined in the `AQMessage` interface.

30.5 Example: Creating a Message and Setting a Payload

This section provides an example that illustrates how to create a message and set a payload.

Example 30-1 Creating a Message and Setting a Payload

This example shows how to Create an instance of `AQMessageProperties`, set the property attributes, create the AQ message, and set the payload.

```
 AQMessageProperties msgprop = AQFactory.createAQMessageProperties();
    msgprop.setCorrelation("mycorrelation");
    msgprop.setExceptionQueue("MY_EXCEPTION_QUEUE");
    AQAgent ag = AQFactory.createAQAgent();
    ag.setName("MY_SENDER_AGENT_NAME");
    ag.setAddress("MY_SENDER_AGENT_ADDRESS");
    msgprop.setSender(ag);
    // handle multi consumer case:
    if(recipients != null)
      msgprop.setRecipientList(recipients);
    System.out.println(msgprop.toString());
    AQMessage mesg = AQFactory.createAQMessage(msgprop);
byte[] rawPayload = "Example_Payload".getBytes();
mesg.setPayload(new oracle.sql.RAW(rawPayload));
```

30.6 Enqueuing Messages

After you create a message and set the message properties and payload, you can enqueue the message using the `enqueue` method of the `OracleConnection` interface. Before you enqueue the message, you can specify some enqueue options. The `AQEnqueueOptions` class enables you to specify the following enqueue options:

- Delivery mode: Specifies the delivery mode. Delivery mode can be set to either persistent (`ENQUEUE_PERSISTENT`) or buffered (`ENQUEUE_BUFFERED`).
- Retrieve Message ID: Specifies whether or not the message ID has to be retrieved from the server when the message has been enqueued. By default, the message ID is not retrieved.
- Transformation: Specifies a transformation that will be applied before enqueuing the message. The return type of the transformation function must match the type of the queue.

Note:

Transformations must be created in PL/SQL using `DBMS_TRANSFORM.CREATE_TRANSFORMATION(...)`.

- Visibility: Specifies the transactional behavior of the enqueue request. The default value for this option is `ENQUEUE_ON_COMMIT`. It indicates that the enqueue operation is part of the current transaction. `ENQUEUE_IMMEDIATE` indicates that the enqueue operation is an autonomous transaction, which commits at the end of the operation. For buffered messaging, you must use `ENQUEUE_IMMEDIATE`.

The following code snippet illustrates how to set the enqueue options and enqueue the message:

```
...
AQEnqueueOptions opt = new AQEnqueueOptions();opt.setRetrieveMessageId(true);
conn.enqueue(queueName, opt, mesg);
...
```

30.7 Dequeuing Messages

Enqueued messages can be dequeued using the `dequeue` method of the `OracleConnection` interface. Before you dequeue a message you must set the dequeue options. The `AQDequeueOptions` class enables you to specify the following dequeue options:

- Condition: Specifies a conditional expression based on the message properties, the message data properties, and PL/SQL functions. A dequeue condition is specified as a `Boolean` expression using syntax similar to the `WHERE` clause of a SQL query.
- Consumer name: If specified, only the messages matching the consumer name are accessed.

> **Note:**
>
> If the queue is a single-consumer queue, do *not* set this option.

- Correlation: Specifies a correlation criterion (or search criterion) for the dequeue operation.
- Delivery Filter: Specifies the type of message to be dequeued. You dequeue buffered messages only (`DEQUEUE_BUFFERED`) or persistent messages only (`DEQUEUE_PERSISTENT`), which is the default, or both (`DEQUEUE_PERSISTENT_OR_BUFFERED`).
- Dequeue Message ID: Specifies the message identifier of the message to be dequeued. This can be used to dequeue a unique message whose ID is known.
- Dequeue mode: Specifies the locking behavior associated with the dequeue operation. It can take one of the following values:
  - `DequeueMode.BROWSE`: Message is dequeued without acquiring any lock.
  - `DequeueMode.LOCKED`: Message is dequeued with a write lock that lasts for the duration of the transaction.
  - `DequeueMode.REMOVE`: (default) Message is dequeued and deleted. The message can be retained in the queue based on the retention properties.
  - `DequeueMode.REMOVE_NO_DATA`: Message is marked as updated or deleted.
- Maximum Buffer Length: Specifies the maximum number of bytes that will be allocated when dequeuing a message from a `RAW` queue. The default maximum is `DEFAULT_MAX_PAYLOAD_LENGTH` but it can be changed to any other nonzero value. If the buffer is not large enough to contain the entire message, then the exceeding bytes will be silently ignored.

- Navigation: Specifies the position of the message that will be retrieved. It can take one of the following values:
  - `NavigationOption.FIRST_MESSAGE`: The first available message matching the search criteria is dequeued.
  - `NavigationOption.NEXT_MESSAGE`: (default) The next available message matching the search criteria is dequeued. If the previous message belongs to a message group, then the next available message matching the search criteria in the message group is dequeued.
  - `NavigationOption.NEXT_TRANSACTION`: Messages in the current transaction group are skipped, and the first message of the next transaction group is dequeued. This setting can be used *only* if message grouping is enabled for the queue.
- Retrieve Message ID: Specifies whether or not the message identifier of the dequeued message needs to be retrieved. By default, it is not retrieved.
- Transformation: Specifies a transformation that will be applied after dequeuing the message. The source type of the transformation must match the type of the queue.

> **Note:**
>
> Transformations must be created in PL/SQL using `DBMS_TRANSFORM.CREATE_TRANSFORMATION(...)`.

- Visibility: Specifies whether or not the message is dequeued as part of the current transaction. It can take one of the following values:
  - `VisibilityOption.ON_COMMIT`: (default) The dequeue operation is part of the current transaction.
  - `VisibilityOption.IMMEDIATE`: The dequeue operation is an autonomous transaction that commits at the end of the operation.

> **Note:**
>
> The Visibility option is ignored in the `DequeueMode.BROWSE` dequeue mode. If the delivery filter is `DEQUEUE_BUFFERED` or `DEQUEUE_PERSISTENT_OR_BUFFERED`, then this option *must* be set to `VisibilityOption.IMMEDIATE`.

- Wait: Specifies the wait time for the dequeue operation, if none of the messages matches the search criteria. The default value is `DEQUEUE_WAIT_FOREVER` indicating that the operation waits forever. If set to `DEQUEUE_NO_WAIT`, then the operation does not wait. If a number is specified, then the dequeue operation waits for the specified number of seconds.

> **Note:**
>
> If you use `DEQUEUE_WAIT_FOREVER`, then the dequeue operation will not return until a message that matches the search criterion is available in the queue. However, you can interrupt the dequeue operation by calling the `cancel` method on the `OracleConnection` object.

The following code snippet illustrates how to set the dequeue options and dequeue the message:

```
...
AQDequeueOptions deqopt = new AQDequeueOptions();
deqopt.setRetrieveMessageId(true);
deqopt.setConsumerName(consumerName);
AQMessage msg = conn.dequeue(queueName,deqopt,queueType);
```

30.8 Examples: Enqueuing and Dequeuing

This section provides a few examples that illustrate how to enqueue and dequeue messages.

Example 30-2 illustrates how to enqueue a message, and Example 30-3 illustrates how to dequeue a message.

Example 30-2   Enqueuing a Single Message

This example illustrates how to obtain access to a queue, create a message, and enqueue it.

```
AQMessageProperties msgprop = AQFactory.createAQMessageProperties();
msgprop.setPriority(1);
msgprop.setExceptionQueue("EXCEPTION_QUEUE");
msgprop.setExpiration(0);
AQAgent agent = AQFactory.createAQAgent();
agent.setName("AGENTNAME");
agent.setAddress("AGENTADDRESS");
msgprop.setSender(agent);
AQMessage mesg = AQFactory.createAQMessage(msgprop);
mesg.setPayload(buffer); // where buffer is a byte array (for a RAW queue)
AQEnqueueOptions options = new AQEnqueueOptions();
conn.enqueue("HR.MY_QUEUE", options, mesg);
```

Example 30-3   Dequeuing a Single Message

This example illustrates how to obtain access to a queue, set the dequeue options, and dequeue the message.

```
AQDequeueOptions options = new AQDequeueOptions();
options.setDeliveryFilter(AQDequeueOptions.DeliveryFilter.BUFFERED);
AQMessage mesg = conn.dequeue("HR.MY_QUEUE", options, "RAW");
```