ORACLE DATABASE 12c

Oracle Database 12c: Advanced Queuing

ORACLE WHITE PAPER | JANUARY 2015

ORACLE

Table of Contents

Introduction

IT managers need a standards-based enterprise-messaging infrastructure that can integrate the different systems and technologies on a scalable, reliable and powerful platform for the real-time flow of information. Oracle Database 12c provides such an enterprise-messaging infrastructure with Oracle Advanced Queuing (AQ), which is a key component in automating business process workflows for distributed applications. Using AQ, businesses can take advantage of the Oracle Database 12c for enterprise-messaging needs without high-end, message-oriented middleware products. Organizations not only can manage all the data inside the Oracle database, but also manage the flow and exchange of data using messages to different systems in one highly reliable, available and scalable Oracle Database. AQ implements the message queuing functionality natively inside the database and leverages its easy manageability, high availability, high performance and security. AQ supports point-to-point and publish/subscribe queues, persistent and nonpersistent messaging, and message ordering priorities that offer flexibility and powerful messaging functionality to applications. Interfaces to AQ include PL/SQL, JMS 1.1, JDBC, ODP.NET and OCI.

This paper focuses on the typical requirements of an enterprise-messaging infrastructure and discusses how Advanced Queuing technologies available in the Oracle Database can help automate business workflows in a distributed environment. The paper highlights some of the advanced messaging, routing and propagation features of AQ and how businesses can leverage the database-integrated messaging functionality in the Oracle Database. This will allow businesses to maximize the return on their investments on infrastructure and build robust, highly scalable, distributed applications with better quality of service to users.

Message Queuing

Message queuing infrastructure enables information sharing and integration amongst different, and in many cases distributed, applications. Producer applications send or enqueue messages into queues from which consumer applications receive or dequeue messages. Producers and consumers interact with each other asynchronously through the queue; this “decoupling” is the centerpiece of message queuing.

Messages often represent critical business events and impose certain characteristics on the underlying messaging infrastructure. The creation, consumption and propagation of the messages must be handled with the highest levels of integrity. Messages must be protected against failures in any component in the enterprise stack and be recoverable in all cases. Message content and attributes must be easily retrievable through standard interfaces.

Finally, the infrastructure should be scalable without compromising the performance, availability and reliability of the system.

Oracle Advanced Queuing

Oracle Advanced Queuing (AQ) is a database-integrated messaging infrastructure in Oracle Database 12c. AQ leverages the functionality of the Oracle database to store messages in persistent queues. All operational benefits of the Oracle database such as high availability, scalability and reliability are applicable to the messages and queues in AQ. Standard database features such as backup and recovery, security and manageability are available to AQ. Oracle technologies such as Data Guard, Real Application Clusters (RAC), and Automatic Storage Management (ASM) can be combined with AQ to deliver a highly available and scalable messaging system. Using standard, off-the-shelf servers and storage, customers can build AQ-based messaging systems that can scale linearly without sacrificing performance, availability or reliability.

AQ Components

The four main components of AQ are:

- Message - A message consists of message content, or payload, which can be specified using typed or raw data and message attributes or control information.
- Message Queue – Messages are stored in queues, and these queues act as “postal boxes” where different applications can look for “mail” in the form of messages. Thus, when one application wants to contact certain applications for certain tasks, it can leave messages in these queues, and the receiving applications will be able to find these messages for processing.
- Message Interfaces – AQ supports enqueue, dequeue, and propagation and notification operations that integrate seamlessly with existing applications by supporting popular standards and APIs. AQ messages can be created, queried, propagated and consumed using popular application programming interfaces (API) such as PL/SQL, C/C++, Java, Visual Basic (through Oracle Objects for OLE) and ODP.NET. AQ provides support for the Java Message Service 1.1 (JMS) API that allows Java applications to utilize the message queuing functionality.
- Message Handling – AQ supports rule-based routing of messages according to data in the message payload or attributes. Additionally, message transformations can be applied to messages to re-format data before the messages are automatically delivered to target applications or subscribers. Oracle Database 12c can also exchange AQ messages with IBM WebSphere MQ and TIBCO/Rendezvous through the Oracle Messaging Gateway.

The various components in AQ provide the functionality needed for enterprise application integration or distributed applications. In a typical integrated environment as shown below in Figure 1, messages are created, propagated and consumed between the Oracle Database server, applications and users.

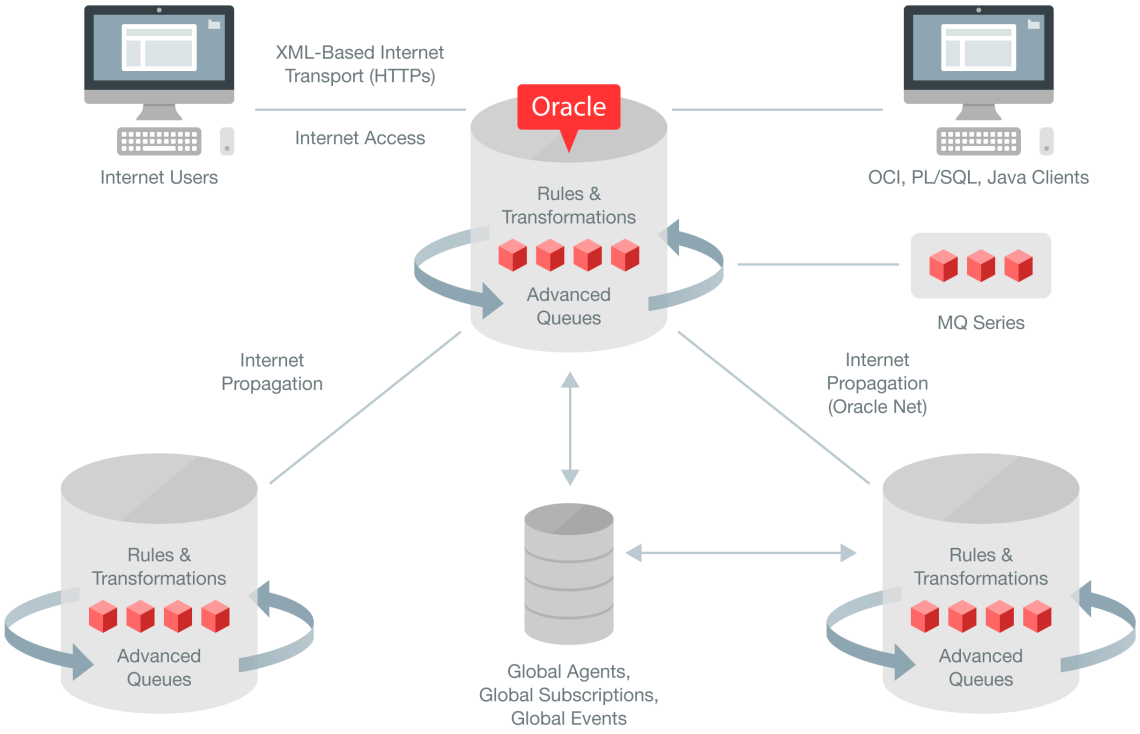


Figure 1. Integrated Application Environment using Oracle AQ

AQ Capabilities

AQ is an integrated messaging infrastructure inside the Oracle database and offers many key capabilities for developing message-based, distributed applications including:

- Transaction support
- Quality of Service (QoS)
- Queue Models
- Security
- Message Propagation
- Message Transformation
- Rules-based Message Routing

Transaction Support

AQ provides transactional semantics to messages using the same underlying infrastructure in the Oracle Database used for relational data. Queue operations such as enqueue and dequeue are atomic, and the Oracle database guarantees the consistency of the messages in persistent queues. Messages are stored along with the other data using the same storage devices and do not require any special set up or management. With AQ, messages can be retained for any length of time (infinite if required), and used for tracking/auditing purposes or for building message warehouses for data mining and analytics. Applications can manipulate the relational data and messages in a single transaction. When external or third party messaging systems are used, applications often need to use 2-phase commit algorithms to achieve transactional semantics, which could be expensive. In contrast, Oracle WebLogic uses

a one-phase commit optimization when Java Transaction API (JTA) transactions consist only of AQ JMS and JDBC operations to the same Oracle database.

Quality of Service

Messages need to be persisted for various reasons. Regulatory compliance, business process auditing, and analytics are a few examples where messages need to be retained for different periods of time. Applications can rely on AQ's message queuing infrastructure for guaranteed exactly-once delivery. Financial services hubs that integrate portfolio management systems with trade processing systems need to retain client order messages to satisfy legal requirements. Integrating with partners or 3rd party fulfillment contractors for order processing requires that messages be exchanged and retained until the order process is complete. Applications exchange data in the form of messages and need to retain these messages for processing business process workflows.

By being integrated with the Oracle database, AQ can provide a higher quality of service than most messaging systems. AQ queues can be backed up like other Oracle tables. Oracle Data Guard provides high availability to AQ messages. AQ scales to very large concurrent message volumes. Concurrent smaller message operations are not blocked by larger message operations. AQ can handle extremely large message backlogs without requiring producers to be throttled. AQ handles very large message payloads (e.g., gigabytes) and leverages the LOB streaming infrastructure. AQ messages are visible through SQL, and AQ tables can be indexed.

In addition to the database-backed quality of service that AQ delivers for persistent messages, AQ provides integrated support for nonpersistent or buffered messages. In some cases, messages need to have the lowest latency, as measured by the time between enqueuing and dequeuing the message. Such messages can be transient i.e. need not be retained. Subscribers to stock quotes need updates at regular intervals, however failures need not be re-tried as the most current stock update is sufficient for most consumers. Cell phone coupons based on location, for example, should be delivered to mobile subscribers as soon as possible. In case of errors, the application can decide to re-send the deals to those subscribers or send new deals based on the new locations of the subscribers. For these types of applications, AQ provides in-memory or buffered messaging for the low latency, high performance message management infrastructure.

Oracle Database 12c AQ offers comprehensive capabilities for both persistent and nonpersistent messages.

1. Persistent Messaging

   AQ provides the queues and the underlying queue tables to persist messages that must be guaranteed to be processed exactly once, even in the event of network, hardware or software failures. Applications can use AQ queues to process messages arriving simultaneously from external programs or from modules within applications. AQ supports different mechanisms to control the order in which messages are processed. Applications can specify a 'priority' for each message at enqueue time, which can be used to control the order in which messages are consumed. Alternately, messages can also be sorted according to enqueue time, commit time, or priority to get a FIFO or even LIFO order for consuming the messages. Commit time is the time at which the transaction was committed and this is especially useful when transactions are interdependent. The persistent quality of service is the default in AQ.

2. Nonpersistent Messaging

   Certain applications require higher performance and are willing to tradeoff the reliability and the transactional support offered by the AQ Persistent Messaging. Queues for nonpersistent messages store messages in memory and do not involve disk I/O. The memory for buffered queues is allocated from the SGA and can be controlled using the 'streams_pool_size' parameter. Alternately, Oracle can automatically allocate the

appropriate memory using SGA auto-tuning. All message ordering schemes available for persistent messages are available to buffered messages.

AQ supports persistent and nonpersistent messages through a common API and provides a messaging infrastructure that effectively separates the application logic and the message integration logic. AQ queues can be set up under different queuing models such as point-to-point and publish-subscriber to let business applications communicate with each other flexibly and reliably.

Queue Models

AQ supports two queue models, namely point-to-point and publish/subscribe queues. A point-to-point or single-consumer queue is aimed at a specific target. Producers and consumers decide on a common queue in which to exchange messages. A message in the point-to-point can be dequeued only once. A publish/subscribe or multi-consumer queue is aimed at multiple targets. A message in a publish/subscribe queue can be dequeued by each of multiple consumers. This type of queue messaging can be used for broadcast or multicast dissemination. Applications can set up rules for delivery to consumers, and these rules can be defined on message payload, attributes or both. Subscriber applications can receive messages that match the subscription rules automatically at dequeue time. Publishers need not be aware of the different consumers or rules and can continue to publish messages. AQ tracks the subscribers and can notify the subscriber applications using the Oracle Call Interface (OCI) or PL/SQL notification mechanism. This allows for a push mode of message delivery.

Security

AQ supports flexible security mechanisms that separate the queue administration and the queue operational tasks. System-level access control allows the application designer or DBA to control access for all queue operations and designate certain users as queue administrators. A queue administrator can perform both the administrative and operational tasks on any queue in the database. AQ also supports queue-level access control for enqueue and dequeue operations. Access to particular queues can be limited to only the applications running in the same schema.

Message Propagation

AQ can propagate messages from one queue to another queue in the same database or in a remote database. This allows applications to communicate asynchronously with each other in a distributed environment without being connected to the same database or to the same queue. The source queue is a multi-consumer queue while the target queue can be either a single-consumer or multi-consumer queue. Messages enqueued in the source queue are propagated automatically and are available for dequeuing at the destination queue or queues. Propagation can be set up to run either continuously as a background process or run only if there is a message to be propagated. With queue-to-queue propagation, a separate job is created to propagate messages for each source and destination queue pair. With queue to dblink propagation, propagation to all target queues at a dblink will share the same propagation job.

Message Transformation

Most business-to-business (B2B) applications need to manipulate data in different formats to integrate disparate applications and systems. AQ provides a complete data transformation engine to transform messages from one data type to another. AQ supports message transformations between different Oracle and user-defined data types. These transformations can be SQL expressions, PL/SQL functions or Java stored procedures. AQ also supports transformations of XML documents using XSLT.

Transformations change the format of a message, so that a message created by one application can be understood by another application. AQ message transformations can be specified during enqueue or when adding a subscriber

for dequeue or propagation. A single transformation must be specified when enqueuing or dequeuing a message, irrespective of the number of the recipients of the message. In the case of remote subscriptions, a single transformation must be specified for all messages sent to a particular queue at the destination. Message transformations can be applied to both persistent and nonpersistent messages. Transformations are exported with a schema or a full database export. If an AQ table is exported, the transformations corresponding to the queue table will also be exported.

Rules-based Message Routing

AQ can intelligently route messages to the right subscribers in a multi-consumer queue or propagate messages to the right queues based on rules specified by each application. The rules can be defined on message properties, message data content, or both. Similar in syntax to the WHERE clause of a SQL query, rules can be expressed in terms of attributes that represent message properties or message content. Starting in Oracle Database 11.2, the rules engine supports faster evaluation of many SQL-92 expressions such as BITAND , CEIL, FLOOR , LENGTH, POWER, CONCAT, LOWER, UPPER, LENGTH, INSTR, SYS_CONTEXT, and UID.

AQ Deployments

AQ is a popular infrastructure for building enterprise messaging functionality across many industries. AQ has been used for simple scalable workflows, messaging hubs, asynchronous processing, information integration, application integration, messaging warehouse, alerts, and message-based logical replication. The following provides examples of AQ deployments.

A leading online retailer integrated its CRM system that was hosted by a third party provider with its backend Order processing system using AQ's robust and reliable database-integrated messaging infrastructure. Customer and order data were synchronized in near real-time between the two systems in geographically distributed sites. Message persistence with AQ allowed the two systems to send and receive data changes through persistent queues. This asynchronous message passing de-coupled the two systems and allowed the online store to be available to customers to collect orders even if the order-processing site was down. With AQ, the company leveraged the reliability and scalability of the Oracle Database to handle peak traffic during holiday seasons and developed the integration in a matter of weeks using AQ's standards based interfaces.

A European financial services firm implemented AQ as the core platform to integrate the firm's global IT infrastructure. Enterprise messaging provided by AQ was used to connect the hubs in London, New York, Singapore, Hong Kong and Tokyo. The core applications in the hubs exchange financial transactions and other information through XML messages. Due to the sensitive nature of the information, the customer required 100% reliable messaging with zero message loss in the event of failure or malfunction of any software or hardware component. Messages had to be delivered in the same order of creation and also be available in a disaster recovery (DR) location for each hub. This customer used multi-consumer queues with persistent messaging in each hub. In addition, using Oracle Data Guard, messages were synchronously copied to the DR locations. Messages were propagated from the local hubs to remote hubs using AQ propagation and appropriate locale-specific transformations for messages were applied at the destination hubs. AQ and the Oracle Database provided the robust, scalable and reliable messaging infrastructure to satisfy the customers' extremely stringent requirements for guaranteed messaging at high throughputs.

Advanced Queuing is a popular feature of the Oracle Database. AQ is widely used as infrastructure by the Oracle Database itself and other Oracle products. AQ is used for Database Change Notifications, Database Alerts, the Database Scheduler including event based jobs and rule based job chains, Fast Application Notifications (FAN), and

Datapump. Other Oracle products using AQ include Enterprise Manager, Audit Vault, Data Vault, Oracle Weblogic Application Server, Oracle E-Business Suite, Oracle Fusion Applications, Oracle Eloqua and Oracle Retail.

Integration with Oracle WebLogic Server

Oracle WebLogic Server applications interoperate with AQ through the JMS API using either WebLogic Server resources (Web Apps, Enterprise Java Beans, Message-Driven Beans) or stand-alone clients. AQ JMS uses the WebLogic JMS Foreign Server framework. The required references to the database, JDBC driver, and data source are configured as part of this framework. The WebLogic Server installation includes all the necessary classes. WebLogic Server applications and stand-alone clients lookup AQ JMS connection factories and destinations using a standard the WebLogic JNDI context. WebLogic applications and clients load and invoke AQ JMS using standard Java EE APIs.

Best Practices

AQ implements queues using different types of user tables for JMS sharded queues and unsharded queues. JMS Sharded Queues automatically manage system-partitioned tables. Unsharded AQ queue tables use index organized tables (IOTs) and indexes in the Oracle database. AQ operations such as enqueue and dequeue generate corresponding database activity. Performance of the underlying database operations significantly impacts the overall performance of AQ. This section details Oracle's best practices, recommendations and tuning tips for optimal performance of the AQ messaging infrastructure.

- JMS Sharded Queues are the preferred JMS configuration for Oracle Advanced Queuing with
  - JMS queues that have enqueuers or dequeuers on multiple Oracle RAC instances
  - high throughput JMS queues
  - JMS queues that consume too many system resources when using unsharded queues.
  - JMS queues with a large number of subscribers.
- Oracle recommends using automatic segment-space management (ASSM) tablespaces for the AQ queue tables, especially for high concurrency applications. Otherwise, initrans, freelists and freelist groups must be tuned to achieve better AQ performance. Storage parameters can be specified during creation of the queue table using the *storage_clause* parameter.
- For unsharded queues, the performance characteristics of queue operations on persistent messages are similar to underlying database operations. The code path of an enqueue operation is comparable to SELECT and INSERT into a multi-column table with three index-organized tables. The code path of a dequeue operation is comparable to a SELECT operation on the multi-column table and a DELETE operation on the dequeue index-organized table. In many scenarios, for example when Oracle Real Application Clusters (Oracle RAC) is not used and there is adequate streams pool memory, the dequeue operation is optimized and is comparable to a SELECT operation on a multi-column table. To take advantage of the optimized dequeue operations, increase STREAMS_POOL_SIZE to allocate at least 20M per queue.
- The queue table indexes and IOTs are automatically coalesced by AQ background processes. However, they must continue to be monitored and coalesced if needed. From 10.2 onwards, with automatic space segment management (ASSM), an online shrink operation may be used for the same purpose. A well balanced index reduces CPU consumption by the queue-monitor process, and ensures optimal enqueue-dequeue performance.
- Oracle RAC can be used to provide high availability and scalability to AQ. For queues being accessed exclusively via the JMS drivers, JMS Sharded Queues provide optimized performance on Oracle RAC with improved manageability. For queues being accessed through the AQ PL/SQL APIs, the performance of AQ can be improved by allowing different queues to be managed by different RAC instances. Different instance affinities or preferences can be specified for the queue tables that allows for parallelization of queue operations

on different queues. Oracle recommends setting instance affinities for the queue tables. Setting instance affinities allows distribution of the background processing for queue-monitor scheduling and propagation. If an instance affinity is not set, queue table affinity is allocated arbitrarily amongst the available instances, which can cause pinging between the application accessing the queue tables and the queue-monitor process monitoring the queue under high loads.

- Ensure that statistics are being gathered so that the optimal query plans for retrieving messages are being chosen. By default, queue tables are locked out from automatic gathering of statistics. The recommended use is to gather statistics with a representative queue message load and lock them.
- Ensure that there are enough queue monitor processes running to perform the background tasks. The queue monitor must also be running for other crucial background activity. Multiple QMN processes share the load; make sure that there are enough of them. These are auto-tuned, but can be forced to a minimum number by specifying the *aq_tm_processes* Database parameter, if needed.
- Dequeue with a wait time should be used only with dedicated server processes. In a shared server environment, the shared server process is dedicated to the dequeue operation for the duration of the call, including the wait time. The presence of many such processes can cause severe performance and scalability problems and can result in deadlocking the shared server processes.
- If queues seem to be growing without bound, check for and drop any persistent subscribers that have been orphaned by their applications.
- Oracle Database Release 11 transparently introduced many performance optimizations for unsharded queues. For example, empty index blocks are now automatically freed incrementally by internally monitoring empty index blocks and the number of messages dequeued at queue and subscriber level. In previous releases, such periodic maintenance needed to be performed manually.
- Other performance best practices include batching multiple dequeue operations on multi-consumer queues into a single transaction, using the NEXT_MESSAGE navigation mode if not using message priorities, and using the REMOVE_NODATA dequeue mode if dequeuing in BROWSE mode followed by a REMOVE. Please see the AQ documentation for additional performance hints.

Performance Monitoring

In addition to AQ administrative functions, Enterprise Manager has support for monitoring AQ metrics and setting thresholds for alerts. These metrics cover both persistent and nonpersistent queues. They can be used to detect problems like orphaned subscribers that have not dequeued for a long time or to identify slow subscribers. The metrics can also monitor the health of a queue, such as throughput or how fast the length of a queue is growing or shrinking or space usage. Please see the **Oracle® Enterprise Manager Oracle Database Plug-in Metric Reference Manual** for more information.

Enhanced views for persistent messaging statistics, notification statistics and subscription management allow direct monitoring of system performance and troubleshooting in Oracle Database 12c. The Automatic Workload Repository (AWR) displays the most active queues for persistent messaging operations, allowing for easier diagnosability of AQ performance problems. Users can generate a report based on two AWR snapshots to compute enqueue rate, dequeue rate, and other statistics per queue or per subscriber. In addition, a performance monitoring PL/SQL package for AQ is available through Support Document 1163083.1.

Conclusion

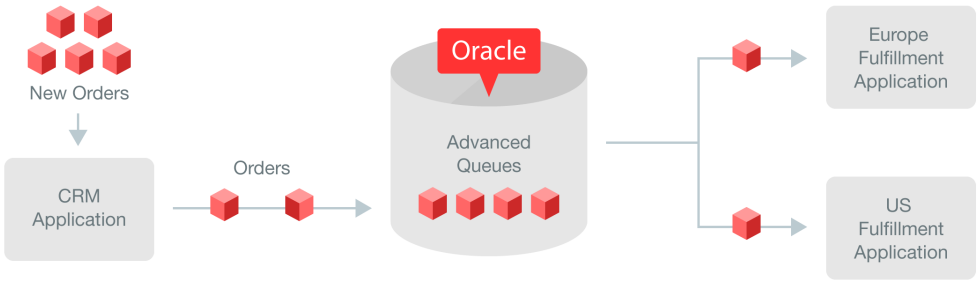
Oracle Advanced Queuing, built into the Oracle Database, offers a robust platform to standardize and integrate the various technologies and applications inside the data center. Businesses can leverage AQ's enterprise messaging infrastructure to build highly scalable and reliable distributed applications. Powerful AQ features such as differing

qualities of service, automatic message transformations, and propagations give businesses the tools needed to design a powerful and flexible messaging platform. Database integrated Advanced Queuing provides smooth, real-time flow of critical information, less management and more productivity for your ever growing, scalable, highly available business.

ORACLE ADVANCED QUEUING – SHORT TUTORIAL

The following is a short tutorial on how to configure and use Oracle Advanced Queuing (AQ). This section demonstrates the basic functionality and the simplicity of using AQ.

An electronic store needs to route the customer orders from its online store to the right warehouse in US or Europe for order fulfillment. The central CRM application collects the order along with the customer information and stores the order data in the Oracle Database. The order details are enqueued as ADT messages in AQ. The fulfillment applications for the different warehouses then dequeue the order messages (deleted from queue automatically) and process the customer orders. The CRM, Europe Fulfillment and US Fulfillment applications work asynchronously and should work even if other applications are down. For example, even if the Europe site is down, the online store should continue to process new customer orders through the CRM application and the US warehouse should continue to process the orders for the US region.



This tutorial explains the steps needed to set up and use the messaging infrastructure in AQ.

1. Configure AQ Administrator account

   The AQ Administrator user ('aq_admin') creates and owns the queuing infrastructure. The role AQ_ADMINISTATOR_ROLE that allows for the creation and administration of the queuing infrastructure needs to be granted to the 'aq_admin' user.

```
--create aq_admin administrator account
set verify off
ACCEPT password PROMPT 'Enter password for aq_admin:' HIDE
CREATE USER aq_admin IDENTIFIED BY &password
DEFAULT TABLESPACE users
TEMPORARY TABLESPACE temp;

ALTER USER aq_admin QUOTA UNLIMITED ON users;
--grant roles to aq_admin
GRANT aq_administrator_role TO aq_admin;
GRANT connect TO aq_admin;
```

```
GRANT create type TO aq_admin;
```

2. Set up Order message payload and Orders queues

The following steps must be executed as the aq_admin user.

- Create the content or the payload of the message.

```
CREATE TYPE orders_message_type AS OBJECT (
order_id       NUMBER(15),
Product_code   VARCHAR2(10),
Customer_id    VARCHAR2(10),
order_details  VARCHAR2(4000),
price          NUMBER(4,2),
region_code    VARCHAR2(100));
```

- Create Queue Table and Queue

After creating the payload, the queuing infrastructure can be created. Queues are implemented using a queue table that can hold multiple queues with the same payload type. The following creates a queue table ‘orders_qt’ and a queue ‘orders_msg_queue’.

```
DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table => 'aq_admin.orders_qt',
queue_payload_type =>
'aq_admin.orders_message_type');

DBMS_AQADM.CREATE_QUEUE (
queue_name => 'orders_msg_queue',
queue_table => 'aq_admin.orders_msg_qt',
queue_type => DBMS_AQADM.NORMAL_QUEUE,
max_retries => 0,
retry_delay => 0,
retention_time => 1209600,
dependency_tracking => FALSE,
comment => 'Test Object Type Queue',
auto_commit => FALSE);
```

- Start the queue

```
DBMS_AQADM.START_QUEUE('orders_msg_queue');
```

3. Configure AQ user account

The AQ user (‘aq_user’) accesses the queuing infrastructure created in the above step. The following creates the ‘aq_user’ account and grants the necessary privileges.

```
--create aq_user user account
set verify off
```

```
ACCEPT password PROMPT 'Enter password for aq_user:' HIDE
CREATE USER aq_admin IDENTIFIED BY &password
DEFAULT TABLESPACE users
TEMPORARY TABLESPACE temp;
--grant roles to aq_user
GRANT aq_user_role TO aq_user;
GRANT EXECUTE ON message_type TO aq_user;

DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
privilege => 'ALL',
queue_name => 'aq_admin.orders_msg_queue',
grantee => 'aq_user',
grant_option => FALSE);
```

4. Subscriptions to the Orders queue

The orders queue has two subscriptions, one for orders made from within the US, and another for orders made from Europe. The region_code in the orders_message_type distinguishes the two types of orders.

```
-- need administrator privileges to add subscriber
DBMS_AQADM.ADD_SUBSCRIBER(
Queue_name => 'aq_admin.orders_msg_queue',
Subscriber => 'US_ORDERS',
Rule => 'tab.user_data.region_code = ''USA''');

DBMS_AQADM.ADD_SUBSCRIBER(
Queue_name => 'aq_admin.orders_msg_queue',
Subscriber => 'EUROPE_ORDERS',
Rule => 'tab.user_data.region_code =
''EUROPE''', Transformation =>
'aq_admin.Dollar_to_Euro');
```

5. Create message transformations (optional)

Message transformations can be automatically applied to messages in AQ queues. The code below shows an example of translating currency from dollars to euros. The price field in the order message is specified in dollars. When the European warehouse dequeues the message, the price field is automatically changed to euros as shown in the simple example below.

```
CREATE FUNCTION
Fn_Dollars_to_Euro(src aq_admin.orders_msg_type
)Returns  aq_admin.orders_msg_type AS
   Target  aq_admin.orders_msg_type;
```

```
BEGIN
       Target :=
       aqadmin.orders_msg_type(src.order_id,
       src.product_code, src.customer_id,
       src.order_details, src.price*.5,
       src.region_code);
END;


DBMS_TRANSFORM.CREATE_TRANSFORMATION(
schema         => 'AQ_ADMIN',
name           => 'DOLLAR_TO_EURO',
from_schema    => 'AQ_ADMIN',
from_type      => 'ORDERS_MSG_TYPE',
to_schema      => 'AQ_ADMIN',
to_type        => 'ORDERS_MSG_TYPE',
transformation =>
'AQ_ADMIN.Fn_Dollars_to_Euro(source.user_data)');
```

6. Queue Operations – Enqueue and Dequeue Messages

The following steps must be executed as the aq_user user. The CRM application enqueues the order messages into the Orders queue that is then dequeued by the Fulfillment applications.

- Enqueue Message - Enqueue a new order into the orders_queue using the DBMS_AQ.ENQUEUE procedure. The order price is specified in dollars.

```
DECLARE
  enqueue_options dbms_aq.enqueue_options_t;
  message_properties
  dbms_aq.message_properties_t;
  message_handle RAW(16);
  message aq_admin.orders_message_type;
  message_id NUMBER;

  BEGIN
  message := AQ_ADMIN.MESSAGE_TYPE (1, 325, 49,
  'Details: Digital Camera. Brand: ABC. Model: XYX' , 23.2, 'EUROPE' );

  -- default for enqueue options VISIBILITY is ON_COMMIT.

  -- message has no delay and no expiration

  message_properties.CORRELATION :=  message.order_id;
```

```
DBMS_AQ.ENQUEUE (
queue_name => 'aq_admin.orders_msg_queue',
enqueue_options => enqueue_options,
message_properties => message_properties,
payload => message,
msgid => message_handle);

COMMIT;
END;
```

- Dequeue Message – This example shows how the European warehouse dequeues messages corresponding to orders from Europe. The DBMS_AQ.DEQUEUE procedure is used to read or dequeue the messages from the queue. The price is automatically transformed to euros before dequeue.

```
DECLARE

dequeue_options dbms_aq.dequeue_options_t;
 message_properties
 dbms_aq.message_properties_t;
 message_handle RAW(16);
 message aq_admin.orders_message_type;

 BEGIN

 -- defaults for dequeue_options
 -- Dequeue for the Europe_Orders subscriber
 -- Transformation Dollar_to_Euro is automatically applied
 dequeue_options.consumer_name :=
 'EUROPE_ORDERS';
 -- set immediate visibility
 dequeue_options.VISIBILITY :=
 DBMS_AQ.IMMEDIATE;

 DBMS_AQ.DEQUEUE (
 queue_name => 'aq_admin.orders_msg_queue',
 dequeue_options => dequeue_options,
 message_properties => message_properties,
 payload => message,
 msgid => message_handle);

 dbms_output.put_line('+---------------+');
 dbms_output.put_line('| MESSAGE PAYLOAD |');
```

```
dbms_output.put_line('+---------------+');
dbms_output.put_line('- Order ID := ' ||
message.order_id);

dbms_output.put_line('- Customer ID:= ' |
message.customer_id);
dbms_output.put_line('- Product Code:= ' ||
message.product_code);

dbms_output.put_line('- Order Details := ' ||
message.order_details);
dbms_output.put_line('- Price in Euros := ' ||
message.price);

COMMIT;

END;
```

7. Please drop any users (e.g. aq_admin, aq_user) that you created as part of this tutorial.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

ORACLE

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

White Paper Title
January 2015
Author: [OPTIONAL]
Contributing Authors: [OPTIONAL]

Oracle is committed to developing practices and products that help protect the environment