Oracle Database 11*g*: Advanced Queuing

An Oracle White Paper
November 2010

INTRODUCTION

Businesses evolve rapidly; Information Technology (IT) systems need to adapt to changing business requirements while managing costs. Businesses should design the IT infrastructure to allow for automation of business process workflows and easy integration of all information systems. The real-time flow of information across these systems is critical to the success of the overall business. Globalization, M & A and Supply Chain innovations such as new 3rd party suppliers or fulfillment partners introduce additional requirements to integrate disparate and geographically distributed systems. Costs and time constraints due to an extremely competitive marketplace make investments in new platforms a non-starter. 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 11*g* provides 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 11*g* for enterprise messaging needs without the need for expensive, 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 buffered messaging, and message ordering priorities that offer flexibility and powerful messaging functionality to applications. With AQ in the Oracle database, Oracle has significantly lowered the barrier for including enterprise-messaging functionality in any applications.

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 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, 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 the queues asynchronously and 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 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 and reliability of the system.

ORACLE ADVANCED QUEUING

Oracle Advanced Queuing (AQ) is a database-integrated messaging infrastructure in Oracle Database 11*g*. 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 & recovery, security and manageability are available to AQ. Oracle technologies such as Data Guard, Real Application Clusters (RAC), Automatic Storage Management (ASM) can be combined with AQ to deliver a highly available and scalable messaging system. Using standard, off-the-shelf server and storage, customers can build AQ-based messaging systems that can scale linearly without sacrificing performance or reliability.

AQ Components

The four main components of AQ are:

1. Message - A message consists of message content, or payload, which can be specified using typed or raw data and message attributes or control information.
2. 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. AQ supports enqueue, dequeue, and propagation operations where the queue type is an abstract datatype (ADT). AQ also supports the ANYDATA queue type, which allows applications to enqueue different message types in a single queue. A queue is persisted in the database using one or more database tables where messages in a queue correspond to rows in the underlying table.
3. Message Interfaces – AQ can integrate seamlessly with the existing applications through support for popular standards. AQ messages can be created, queried, propagated and consumed using popular programming interfaces (API) such as PL/SQL, C/C++, Java and Visual Basic (Oracle Objects for OLE). AQ provides support for the Java Message Service (JMS) API that allows Java applications to utilize the message queuing functionality using the oracle.jms Java package.
4. Message Handling – Messages can be routed according to data in the message payload or attributes. AQ also supports rules based message routing where complex rules can be created by combining payload-based and attribute-based rules. Additionally, message transformations can be applied to messages to re-format data and delivered automatically to target applications or subscribers. Oracle Database 11*g* can also exchange AQ messages with IBM WebSphere MQ 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.

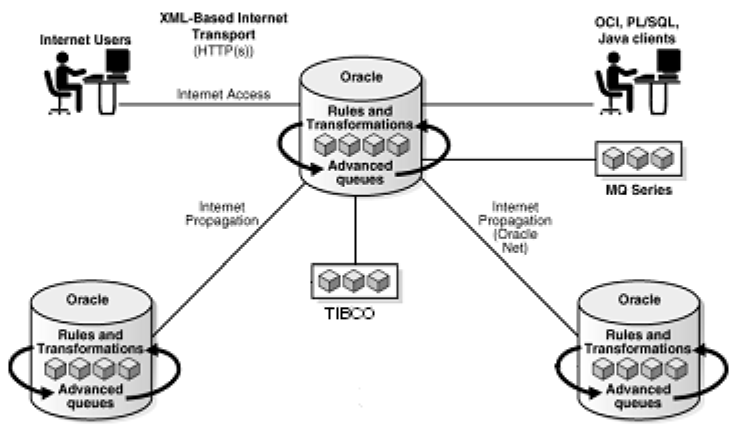

XML-Based Internet Transport (HTTP(s))
Internet Users
OCI, PL/SQL, Java clients
Oracle
Internet Access
Rules and Transformations
Advanced queues
MQ Series
Internet Propagation
Internet Propagation (Oracle Net)
TIBCO
Oracle
Rules and Transformations
Advanced queues
Oracle
Rules and Transformations
Advanced queues

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.

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

Transaction Support

AQ provides the 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. Applications can manipulate the underlying data and its exchange through 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. Messages are stored along with the other data using the same storage devices and do not require any special setup 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.

Quality of Service

Messages need to be persisted for various reasons – regulatory compliance, business process auditing, 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 mechanism. 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 messages be exchanged and retained until order process is complete. Applications exchange data in the form of messages and need to retain these messages for guaranteeing business process workflows.

In other cases, messages need to have the lowest latency, as measured by the time delay in enqueing and dequeing 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 new location of the subscribers. For these types of applications, AQ provides in-memory or buffered messaging for the lowest latency, high performance message management infrastructure.

Oracle Database 11*g* AQ offers comprehensive capabilities for both persistent and transient 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 the enqueue time or commit time to get a LIFO or FIFO 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. Buffered Messaging

Certain applications require higher performance and are willing to tradeoff the reliability and the transactional support offered by the AQ Persistent Messaging. In such cases, AQ's Buffered Messaging is an ideal solution as all the queue operations such as enqueue and dequeue can be much faster compared to persistent queues. Message retention is only available to persistent messages and not to buffered messages. Queues setup for buffered 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 buffered 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 setup under different queuing models such as point-to-point and publish-subscriber to enable business applications can 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. Messages in a publish/subscribe queue can be dequeued by multiple consumers. This type of queue messaging can be used for broadcast or multicast dissemination. Applications can setup 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 to separate 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 setup 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 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 automatically applied to messages during enqueuing, dequeuing or subscribing to queues. 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 buffered messages. Transformations are exported with a schema or a full database export. If an AQ table is exported the transformation 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.

AQ Deployments

AQ is a popular infrastructure for building enterprise messaging functionality across many industries. 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 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.

BEST PRACTICES

AQ implements queues using user tables, 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.

- ❖ 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.
- ❖ When persistent messages are enqueued, they are stored in database tables. 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 multicolumn queue 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 up to 15M 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. In 10.2, with automatic space segment management (ASSM), an online shrink operation may be used for the same purpose. A well balanced index reduces queue monitor CPU consumption, and ensures optimal enqueue-dequeue performance.
- ❖ Oracle RAC can be used to provide high availability and scalability to AQ. 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. Setting instance affinities allows for the partitioning the queue tables for queue-monitor

scheduling and propagation. Oracle recommends setting instance affinities for the queue tables. If an instance affinity is not set, the queue tables are partitioned arbitrarily amongst the available instances that causes pinging between the application accessing the queue tables and the queue-monitor process monitoring the queue. For more information, please refer to the RAC Best Practices for AQ document at http://www.oracle.com/technetwork/database/features/availability/maa-wp-10g-aqrac-bestpractices-jan2-131805.pdf

- ❖ 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, if needed.
- ❖ Dequeue with a wait time should only be used 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.
- ❖ Other performance best practices include batching multiple dequeue operations on multi-consumer queues into a single transaction, using NEXT as the 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 VIEWS

New views for persistent messaging statistics, notification statistics and subscription management improve monitoring of system performance and troubleshooting in 11g. 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.

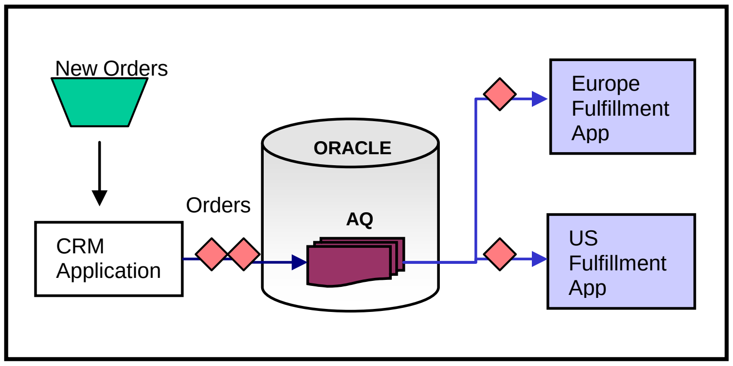
CONCLUSION

Oracle Advanced Queuing, built into the Oracle Database, offers a robust platform to standardize and integrate the various technologies and applications inside the datacenter. Businesses can leverage the 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 setup 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
CREATE USER aq_admin IDENTIFIED BY aq_admin
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. Setup 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
CREATE USER aq_user IDENTIFIED BY aq_user 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
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;
```

Advanced Queuing
June 2008
Author: RAVI RAJAMANI
Contributing Author: NEERJA BHATT

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com