
Oracle9i: Implement Advanced Queuing

Student Guide

D12611GC10
Edition 1.0
November 2001
D34137

ORACLE®

Author

Shankar Raman

**Technical Contributors
and Reviewers**

Brajesh Goyal
Stefan Lindblad
Mark McMellon
Paul Hood
Janet Stern
Metin Yilmaz
Michael Counsel
Samir Mehta
Peter Sharman
Debu Chatterjee
Shailendra Mishra

Publisher

May Lonn Chan-Villareal

Copyright © Oracle Corporation, 2001. All rights reserved.

This documentation contains proprietary information of Oracle Corporation. It is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited. If this documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend

Use, duplication or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

This material or any portion of it may not be copied in any form or by any means without the express prior written permission of Oracle Corporation. Any other copying is a violation of copyright law and may result in civil and/or criminal penalties.

If this documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights," as defined in FAR 52.227-14, Rights in Data-General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them in writing to Education Products, Oracle Corporation, 500 Oracle Parkway, Box SB-6, Redwood Shores, CA 94065. Oracle Corporation does not warrant that this document is error-free.

SQL, PL/SQL, Oracle DBMS, Oracle Universal Data Server, Oracle Application Server, LiveHTML are trademarks or registered trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

Contents

Preface

1 Introduction to Advanced Queuing

Objectives	1-2
Limitations of Traditional Applications	1-3
Asynchronous Application Interaction	1-4
Asynchronous Interaction between Applications	1-5
What Is Advanced Queuing?	1-6
Distinguishing Features of AQ	1-7
Distinguishing Features of AQ	1-8
Features of AQ	1-9
Architecture Overview	1-10
AQ Basic Elements	1-11
Rules Engine	1-15
Propagation	1-16
Transformation Infrastructure	1-17
Messaging Gateway	1-18
Management with Oracle Enterprise Manager	1-19
Summary	1-20

2 Initiating the Use of Advanced Queuing

Objectives	2-2
AQ Interfaces	2-3
PL/SQL Interfaces	2-4
C (OCI) Interface	2-5
Visual Basic Interface	2-6
Using Oracle JMS	2-7
Java Native Interface	2-9
Managing Java.AQ API	2-10
Using IDAP	2-11
Initiating Use of AQ	2-13
Initialization Parameters AQ_TM_PROCESSES	2-14
Initialization Parameters JOB_QUEUE_PROCESSES	2-15
Summary	2-16

3 Creating Queue Tables and Queues

Objectives	3-2
Building a Queue Model	3-3
Set Up Users: AQ Administrator	3-4
Create AQ Users	3-6
DBMS_AQADM.Grant_Queue_Privilege	3-7
Queue Tables	3-8
Create Queue Tables: Using the PL/SQL Interface	3-9
Creating a Queue Table Using OEM	3-11
Queues	3-17

- Create Queues 3-18
- Grant Privileges to Users 3-19
- Revoke Privileges 3-20
- Create Queues Using OEM 3-21
- Storage Queue Tables 3-26
- Starting a Queue 3-27
- Stopping a Queue 3-28
- Altering and Dropping Queues 3-29
- Dictionary Catalog Views 3-30
- Summary 3-31

4 Performing Queue Operations

- Objectives 4-2
- Queuing Operations 4-3
- Queue Operations Privileges 4-4
- Enqueue Operation 4-5
- Message Properties 4-7
- Enqueue Operation: Example 4-9
- Enqueue Features: Subscriptions and Recipient Lists 4-10
- Adding Subscribers Using OEM 4-12
- Priority and Ordering of Messages 4-16
- Time Specification: Delay and Expiration 4-17
- Message Grouping 4-18
- Dequeue Operation 4-19
- Dequeue Example 4-20
- Modes of Dequeue 4-21
- Dequeue Operation: Navigation 4-22
- Dequeue Operation: Search Criteria 4-23
- Dequeue Operation: Message Grouping 4-24
- Dequeue: Multiple Recipients 4-25
- Dequeue Operation: Retry with Delay Interval 4-26
- Dequeue Operation: Rule-Based Subscription 4-27
- Exception Handling for Multiple Recipient Queue 4-28
- Exception Queue 4-29
- Notification 4-31
- Event Monitor 4-32
- Listen Capability 4-33
- Listen on Queues 4-34
- Summary 4-36

5 Managing Propagation

Objectives 5-2

Propagation: Overview 5-3

Set Up Propagation 5-4

DBMS_AQADM. SCHEDULE_PROPAGATION 5-6

Sample Propagation Schedules 5-7

Monitoring Propagation 5-8

Exception Handling 5-9

Propagation over HTTP 5-10

Unschedule Propagation 5-11

Enabling and Disabling Propagation Schedule 5-12

Summary 5-13

6 Managing Transformation

Objectives 6-2

Message Transformation Concepts 6-3

Message Transformation with AQ 6-5

Defining Message Transformations 6-6

XML Transformations 6-7

DBMS_TRANSFORM Package 6-8

Create Transformation 6-10

Modify Transformation 6-11

Specifying Transformations 6-12

Managing Transformations 6-15

Summary 6-17

7 AQ Notifications

Objectives 7-2

AQ Notification 7-3

AQ Notification Database Events 7-4

Registering for Notifications 7-5

Register for Notification PL/SQL Procedure 7-6

Receive Notifications 7-7

OID Support for AQ 7-9

Register for AQ Notifications in OID: Setup 7-10

Register for Notification in OID Using OCI 7-12

Registering for Notification at an E-mail Address: Example 7-14

Summary 7-15

8	Configuring AQ with Internet Access
	Objectives 8-2
	Overview 8-3
	Internet Integration: Requirements 8-4
	Internet Client Communication 8-5
	Accessing AQ as an Internet User: Example 8-6
	What Is IDAP 8-7
	IDAP Message Structure 8-9
	IDAP Message 8-10
	IDAP Message Options: Producer 8-12
	IDAP Message Options: Consumer 8-13
	IDAP Message Options: Register 8-15
	IDAP Messages: Response 8-16
	Setting Up Internet Access for AQ 8-18
	IDAP: An Example 8-19
	Security Using HTTP 8-21
	Authorization Functions 8-23
	AQ Communication Using e-mail 8-26
	Setting Up AQ to Use E-mail 8-27
	AQ Privileges for Internet Users 8-28
	Managing Internet Users 8-29
	Internet Propagation 8-30
	Enabling HTTP Propagation 8-31
	Using XML Payloads with AQ 8-33
	Using XML Payloads with AQ Subscriptions 8-34
	Using XML Payloads with AQ 8-36
	Summary 8-37
9	AQ Tuning and Troubleshooting Tips
	Objectives 9-2
	Configuring the Tablespace 9-3
	Concurrent Transaction Activity 9-4
	Rollback Segments and Redo Logs 9-5
	Configure Background Processes 9-6
	Optimization: Enqueue/Dequeue 9-7
	Optimization Propagation 9-8
	Summary 9-9

10 Advanced Queuing and JMS (Optional)

Objectives	10-2
Java Message Service (JMS) Overview	10-3
Administered Objects in JMS Connection Factory	10-4
Connection	10-5
Connection Setup	10-6
Sessions	10-8
Administered Objects in JMS Destination	10-10
Access Control in JMS	10-11
Point-to-Point Model	10-12
Publish-Subscribe Model	10-14
JMS Message Producer Feature	10-16
API Comparison: Establishing Connection	10-18
API Comparison: Queue Table and Queue Management	10-19
API Comparison: Subscriptions and Propagation	10-20
JMS Message Structure	10-21
Mapping JMS Messages to AQ Payloads	10-22
API Comparison: Creating a Message	10-24
API Comparison: Enqueuing a Message	10-25
API Comparison: Dequeuing a Message	10-27
API Comparison: Notification	10-29
Summary	10-30

A Practices and Solutions

Preface

Profile

Before You Begin This Course

Before you begin this course, you should have the following qualifications:

- Working experience with Oracle RDBMS, PL/SQL

Prerequisites

- *Oracle9i Database Administration Parts I and II*
- *Oracle9i Advanced PL/SQL*

How This Course Is Organized

Oracle9i Implement Advanced Queuing is an instructor-led course featuring lecture and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills introduced.

Related Publications

Oracle Publications

Title

Oracle9i Applications Developer's Guide – Advanced Queuing

Oracle9i Database Reference

Oracle9i Database Administrators Guide

Oracle9i PL/SQL User Guide and Reference

Oracle Net Services Administrator's Guide

Additional Publications

- System release bulletins
- Installation and user's guides
- *read.me* files
- International Oracle User's Group (IOUG) articles
- *Oracle Magazine*

Typographic Conventions

Typographic Conventions in Text

Convention	Element	Example
Bold italic	Glossary term (if there is a glossary)	The <i>algorithm</i> inserts the new key.
Caps and lowercase	Buttons, check boxes, triggers, windows	Click the Executable button. Select the Can't Delete Card check box. Assign a When-Validate-Item trigger to the ORD block. Open the Master Schedule window.
Courier new, case sensitive (default is lowercase)	Code output, directory names, filenames, passwords, pathnames, URLs, user input, usernames	Code output: <code>debug.set ('I' , 300) ;</code> Directory: <code>bin (DOS), \$FMHOME (UNIX)</code> Filename: Locate the <code>init.ora</code> file. Password: User <code>tiger</code> as your password. Pathname: Open <code>c:\my_docs\projects</code> URL: Go to <code>http://www.oracle.com</code> User input: Enter <code>300</code> Username: Log on as <code>scott</code>
Initial cap	Graphics labels (unless the term is a proper noun)	Customer address (<i>but</i> Oracle Payables)
Italic	Emphasized words and phrases, titles of books and courses, variables	Do <i>not</i> save changes to the database. For further information, see <i>Oracle7 Server SQL Language Reference Manual</i> . Enter <code>user_id@us.oracle.com</code> , where <i>user_id</i> is the name of the user.
Quotation marks	Interface elements with long names that have only initial caps; lesson and chapter titles in cross-references	Select "Include a reusable module component" and click Finish. This subject is covered in Unit II, Lesson 3, "Working with Objects."
Uppercase	SQL column names, commands, functions, schemas, table names	Use the SELECT command to view information stored in the LAST_NAME column of the EMP table.

Convention	Element	Example
Arrow	Menu paths	Select File—> Save.
Brackets	Key names	Press [Enter].
Commas	Key sequences	Press and release keys one at a time: [Alternate], [F], [D]
Plus signs	Key combinations	Press and hold these keys simultaneously: [Ctrl]+[Alt]+[Del]

Typographic Conventions in Code

Convention	Element	Example
Caps and lowercase	Oracle Forms triggers	When-Validate-Item
Lowercase	Column names, table names	SELECT last_name FROM s_emp;
	Passwords	DROP USER scott IDENTIFIED BY tiger;
	PL/SQL objects	OG_ACTIVATE_LAYER (OG_GET_LAYER ('prod_pie_layer'))
Lowercase italic	Syntax variables	CREATE ROLE <i>role</i>
Uppercase	SQL commands and functions	SELECT userid FROM emp;

Typographic Conventions in Navigation Paths

This course uses simplified navigation paths, such as the following example, to direct you through Oracle Applications.

(N) Invoice—>Entry—>Invoice Batches Summary (M) Query—>Find (B) Approve

This simplified path translates to the following:

1. (N) From the Navigator window, select Invoice—>Entry—>Invoice Batches Summary.
2. (M) From the menu, select Query—>Find.
3. (B) Click the Approve button.

N = Navigator, **M** = Menu, **B** = Button

1

Introduction to Advanced Queuing

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- **Describe the need for asynchronous interaction between messages**
- **Describe the concepts and elements of Advanced Queuing such as:**
 - **Queues: enqueue and dequeue operations**
 - **Delay and expiration of messages**
 - **Multiconsumer queues, Publish / Subscribe**
 - **Rules and rule-based subscriptions**
 - **Propagation and transformation**

ORACLE

Limitations of Traditional Applications

Traditional applications have client-server architecture.

- **One application connects to another and maintains the connection.**
- **The requesting application blocks the serving application until a reply is received.**
- **It is an inefficient way of using resources.**
- **If a network or node is down the applications do not work.**

ORACLE

1-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Limitations of Traditional Applications

Traditionally applications have been built for single machine environments and an operating communication link between client and the server is presumed to be available at all times.

- Whenever an application needs to get data from the server it establishes a connection over the communication link.
- The connection remains until the user or application explicitly initiates a disconnect.

Such an implementation can be expensive in terms of network and memory resources.

Further if a server or a network is down it can have a dramatic affect on a business. To address these issues some configurations are offered to provide failover solutions, without addressing the applications themselves.

These solutions often require duplication and redundancy, but are often prohibitively expensive.

Asynchronous Application Interaction

- **The Internet is a driving factor for the new systems**
- **A mobile workforce is growing**
- **Acquisitions and new business demands result in the need to integrate disparate systems**
- **Location and transformation requirements are from a global perspective**
- **Complex distributed systems require integrated but not tightly coupled systems**

ORACLE

1-4

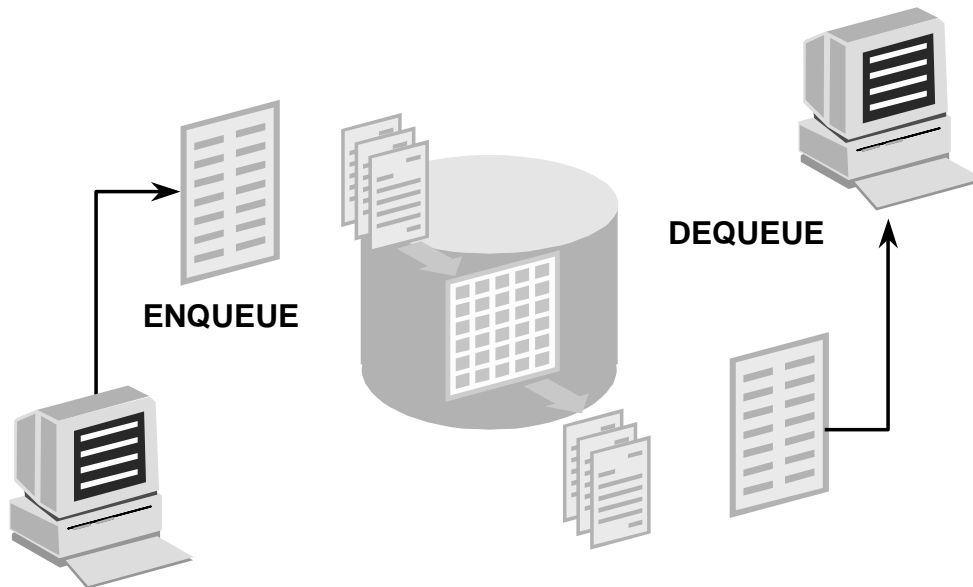
Copyright © Oracle Corporation, 2001. All rights reserved.

New Application Needs

There are, however, a number of factors that demand an alternative to these connected types of client-server environments. The Internet has tremendously changed the way we do business. Increasingly, mission critical systems are built for the Internet. Alliances, business mergers, and acquisitions require that diverse systems and applications integrate. This is where message-oriented application interaction can help.

A message-oriented approach provides benefits to a lot of implementations but it is important to choose the right solution for a particular problem.

Asynchronous Interaction between Applications



Asynchronous Interaction and Messages

Applications that interact asynchronously can do so regardless of the hardware or operating system. In this model, one application sends a request or message, and another in the role of consumer retrieves requests from the queue and acts on them. To protect against failure, requests are stored persistently and processed. For example, the applications of a retailer might use the message queues as follows:

- The inventory application can send a message to the ordering application when the inventory level for a product goes below a certain level.
- The ordering application can then verify the supplier or contract information and place the order with the suppliers.
- The ordering application also can send messages to accounts payable, inventory, and budgeting applications for their necessary actions.

As you can see, none of these applications need to be stopped or wait for interaction from others; using messaging for these tasks allows the different applications to interact with each other efficiently, without tying up network or other resources.

What Is Advanced Queuing?

Oracle Advanced Queuing:

- Is database integrated message queuing
- Enables asynchronous integration of heterogeneous and distributed applications
- Provides database functionality such as reliability, security, scalability, integrity, and disaster recovery
- Provides message management functionality necessary for e-business integration

ORACLE

1-6

Copyright © Oracle Corporation, 2001. All rights reserved.

What Is Advanced Queuing

Oracle Advanced Queuing (AQ) provides message queuing as an integrated part of the Oracle RDBMS, thereby creating a message-enabled database.

It supports both a local and distributed queue model. AQ is implemented as part of the database and queues can be accessed through normal SQL.

Oracle Advanced Queuing enables distributed, heterogeneous, and autonomous applications to communicate asynchronously using message queuing. This functionality is usually provided by a separate category of products called message-oriented middleware (MOM).

Oracle Advanced Queuing supports "publish and subscribe" communication in addition to the point-to-point communication model provided by most message-oriented middleware products.

Message Management

Using message queuing, applications communicate by exchanging messages. The communication is asynchronous. An application creates a message over the message queuing system. It is the responsibility of the message queuing product to guarantee the delivery of messages to their destinations. Message queuing systems offer this functionality by implementing a store-and-forward mechanism.

Distinguishing Features of AQ

- **Retention**
 - Retains messages after consumption
 - Maintains history of messages
 - History can be used for tracking, data warehousing, and data mining
- **Querying/SQL access**
 - Stores messages in database tables as rows
 - Use SQL to access message properties, payload, and history
- **Ordering and priority**
 - First in, first out (FIFO)
 - Message priority-based ordering

ORACLE

1-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Distinguishing Features of AQ

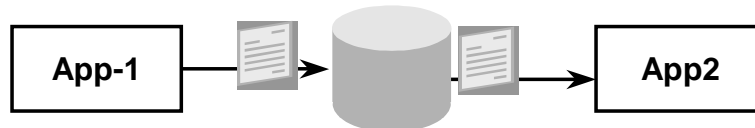
- **Retention and Audit:** All messages are retained even after they are dequeued. This enables auditing and for future reference.
- **Querying:** Messages can be queried using SQL. SQL features such as indexing, and so on, can be used for accessing messages efficiently.
- **Security:** Access right of users to queues can be specified per queue.
- **Ordering and priority:** You can set the priority and position of an enqueued message in the queue.

When enqueued, messages can be placed in a queue relative to another message in the queue, which can be used for ordering messages outside of FIFO and Priority.
- **Time specification:** Delay and expiration intervals can be specified for an enqueued message.
- **Nonpersistent queues:** Queues can be made nonpersistent to be retained in memory to serve for notifications only.
- **Explanation for exception handling:** A message is said to be processed “normally” if it is consumed within the specified time interval and within the specified number of attempts. Messages not consumed normally are placed in a separate queue called the exception queue.

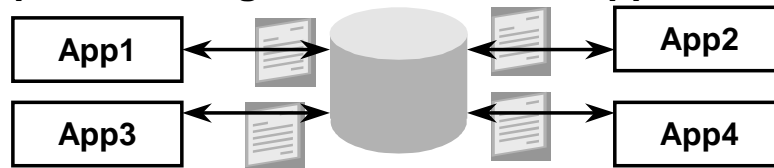
Distinguishing Features of AQ

Addressing modes: AQ supports point-to-point and publish / subscribe modes of interaction.

- **Point-to-point: Enables individual applications to interact with each other.**



- **Publish / Subscribe: Enables an AQ queue to act as a point of integration for several applications.**



ORACLE

Distinguishing Features of AQ (continued)

AQ supports point-to-point and Publish / Subscribe modes of addressing modes.

Point-to-point mode of addressing would be suitable where a few traditional applications are to be integrated. You can define their interaction individually with minimal effort. However when the number of applications that interact increases, and changes, it leads to the following problems.

- Each new application must be integrated with each of the other existing applications.
- There is no central place to monitor the integrated infrastructure.

The Publish / Subscribe model provides the following benefits:

- **Multiple recipients:** A message in a queue can be retrieved by multiple recipients, without making multiple copies of the message.
- **Subscription list:** A queue administrator can specify the list of subscribers who can retrieve messages from a queue.
- **Dynamic management of subscription:** New applications or subscribers can be added and old ones removed dynamically.
- **You can route messages based on the time of day that the message is processed even though the time of processing is not held in the message but can be queried from system variables.**

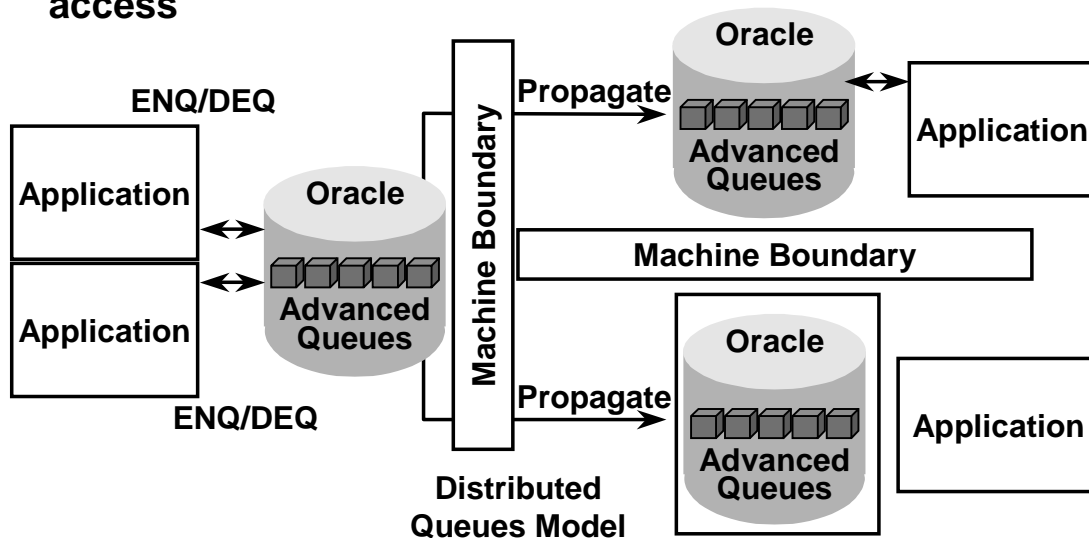
Features of AQ

- **Multiple ways of consuming messages**
 - Blocking dequeue
 - Nonblocking dequeue
 - Asynchronous notifications: e-mail, PL/SQL, OCI, and Java
- **Extensive type support for message structure**
 - Scalar types such as number, date, varchar, raw
 - Oracle object types
 - Large objects such as CLOB and BLOB
- **Dynamic message structure with XML**
- **Transformation plug-ins**
- **JMS support**

ORACLE

Architecture Overview

Database-integrated message queuing with SQL access



ORACLE

1-10

Copyright © Oracle Corporation, 2001. All rights reserved.

AQ - Architecture

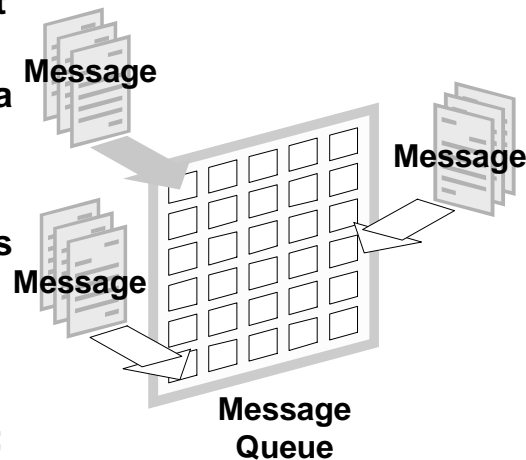
AQ is the industry's first database-integrated message queuing system. It supports both a local and distributed queue model. AQ is implemented as part of the database and queues can be accessed through normal SQL. The point should be emphasized that AQ has two interfaces: a queuing interface (enqueue/dequeue) and SQL access.

Applications that can connect to a single database can communicate using a shared queue model. This does not involve propagation of messages across a network.

AQ queues are implemented using normal tables. Each queue table can have many queues in them. Messages in queues can be unstructured (raw) or structured (any Oracle object type). Each queue table is associated with a specific message type. A queue can store only messages of a particular type.

AQ Basic Elements

- **Message**
 - A message is the smallest unit of information.
 - Its content can be raw or a user-defined type
- **Queue**
 - A repository for messages
 - Normal queues
 - Exception queues
 - Persistent queues are stored in database tables; nonpersistent queues are held only in memory.



ORACLE

1-11

Copyright © Oracle Corporation, 2001. All rights reserved.

AQ Basic Elements

Messages

A message is the smallest unit of information inserted into and retrieved from a queue. It can reside in only one queue, and is created by the enqueue call and consumed by the dequeue call. A message consists of the following:

- Control information (metadata): The control information represents message properties used by AQ to manage messages
- Payload (data): The payload data is the information stored in the queue and is transparent to Oracle AQ.

AQ Basic Elements (continued)

Queues

A queue is a repository for messages. Queues can be created, altered, started, stopped, and dropped by using the Oracle AQ administrative interfaces. There are two types of queues:

- User queues, also known as normal queues: The user queue is for normal message processing.
- Exception queues: Messages are transferred to an exception queue if they cannot be processed for some reason.

User queues can be persistent (the default) or nonpersistent queues. Persistent queues store messages in database tables whereas nonpersistent queues store messages in memory.

Exception Queues

An `exception_queue` is a repository for all expired or unserviceable messages. Applications cannot directly enqueue into exception queues. Also, a multiconsumer exception queue cannot have subscribers associated with it. However, an application that intends to handle these expired or unserviceable messages must dequeue from the exception queue.

AQ Basic Elements

- **Queue table**
 - Queues are stored in queue tables.
 - A queue table contains one or more queues, but they must contain messages with the same payload definition.
 - Each queue table contains a default exception queue.
- **Agent**
 - An agent is an end user or an application that uses the queue. There are two types of agents:
 - Producers who place messages in a queue (enqueue),
 - Consumers who retrieve messages (dequeue)
 - Agents insert messages into a queue and retrieve messages from the queue.
 - An agent is identified by its name, address, and protocol.

ORACLE

1-13

Copyright © Oracle Corporation, 2001. All rights reserved.

AQ Basic Elements (continued)

Queue Tables

Queues are stored in queue tables. Each queue table is a database table and contains one or more queues. Each queue table contains a default exception queue. All queues within a queue table must contain messages with the same payload definition.

Agents

An agent is an end user or an application that uses the queue. There are two types of agents:

- Producers who place messages in a queue (enqueue)
- Consumers who retrieve messages (dequeue)

Any number of producers and consumers can access the queue at a given time. Agents insert messages into a queue and retrieve messages from the queue by using the Oracle AQ operational interfaces.

An agent is identified by its name, address, and protocol.

- The name of the agent can be the name of the application or a name assigned by the application. A queue itself can be an agent enqueueing or dequeuing from another queue.
- The address field is a character field of up to 1024 bytes which is interpreted in the context of the protocol.

AQ Basic Elements

- **Recipients**

The recipient of a message can be a user:

- On the same database
- On another database
- On another queue
- Recipients dequeue messages depending on the conditions specified in message properties in the enqueue operation.

- **Subscription**

To enable multiple consumers for a message, you can also specify a subscription list.

ORACLE

1-14

Copyright © Oracle Corporation, 2001. All rights reserved.

AQ Basic Elements (continued)

Recipient

The recipient of a message can be a user on the same database, on another database, or on another queue.

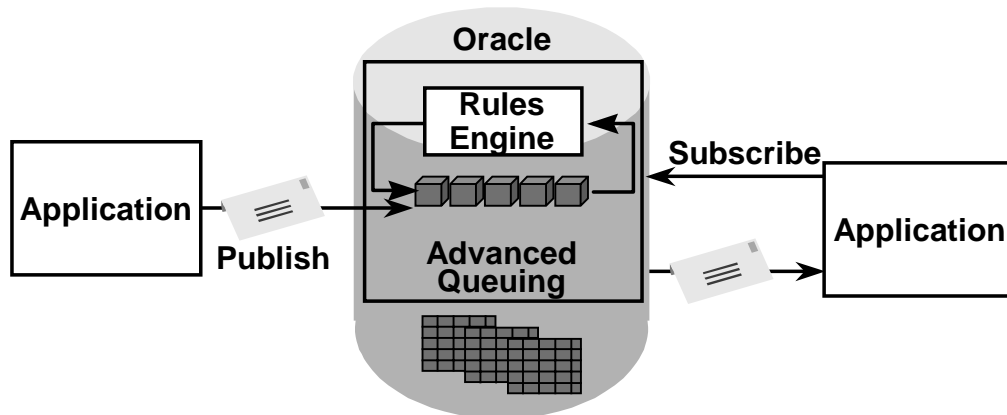
The recipient can dequeue messages based on the message properties specified at the time of enqueueing.

Subscription Lists

Multiple consumers can consume a single message:

- The enqueuer can explicitly specify the consumers who can retrieve the message as recipients of the message. A recipient is an agent identified by a name, address, and protocol.
- A queue administrator can specify a default list of recipients who can retrieve messages from a queue. The recipients specified in the default list are known as subscribers. If a message is enqueued without specifying the recipients, the message is sent to all the subscribers.
- A recipient can be a subscriber to more than one queue. Further, specific messages in a queue can be directed toward specific recipients who may or may not be subscribers to the queue, thereby overriding the subscriber list.

Rules Engine



ORACLE

1-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Rules

A rule is used to define one or more subscribers' interest in subscribing to messages. The messages that meet the rule criterion are delivered to the interested subscribers. A rule is specified as a boolean expression using syntax similar to the WHERE clause of a SQL query. The boolean expression can include conditions on the following:

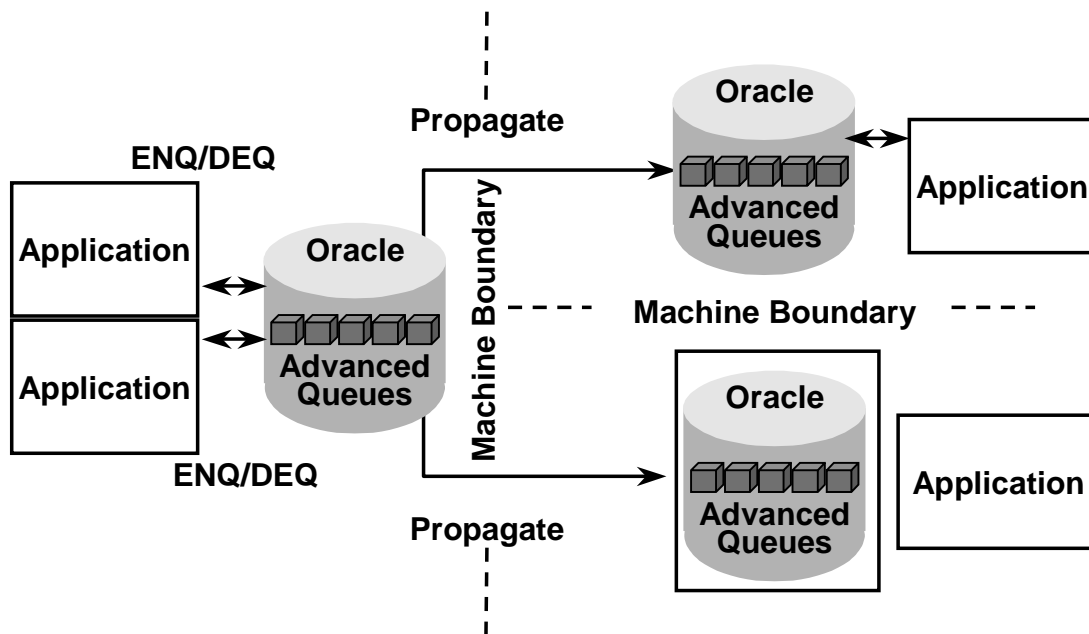
- Message properties (currently priority and correlation identifier)
- User data properties (object type payloads only)
- Functions (as specified in the WHERE clause of a SQL query)

The rules engine applies SQL conditions to a message in a queue. Given a set of conditions, called rules, the rules engine serves to efficiently determine which of them evaluates to true on a given piece of data, represented as a row in a table. This evaluation is performed at the time that the message is enqueued.

Rule-Based Subscribers

A rule-based subscriber is a subscriber with associated rules in the default recipient list. If an associated rule evaluates to TRUE for a message, the message is sent to the rule-based subscriber even if the message has no specified recipients.

Propagation



ORACLE

1-16

Copyright © Oracle Corporation, 2001. All rights reserved.

AQ Propagation

AQ Propagation enables applications that reside on different databases (or even different systems) to communicate with each other.

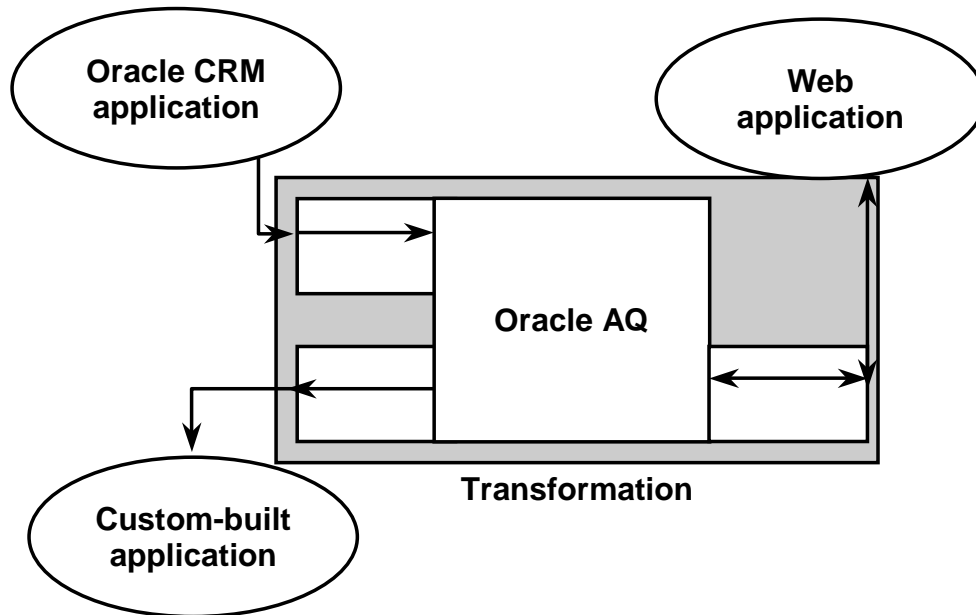
Messages can be propagated from one Oracle AQ to another, irrespective of whether these are local or remote.

The source and recipient of propagation are queues, and not applications.

Propagation is performed by snapshot (`job_queue_processes`) background processes. Propagation to remote queues is done using database links and OracleNet.

Using Oracle9i, you can also enable propagation over the Internet and across firewalls using the HTTP, HTTPS, or SMTP protocol.

Transformation Infrastructure



ORACLE

1-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Transformation

AQ provides a transformation infrastructure, that enables applications that use different data models to communicate with each other. A transformation defines a mapping from one Oracle data type to another. The transformation is represented by a function that takes the source data type as input and returns an object of the target data type.

Transformation functions can be specified during enqueue, dequeue, notification, or propagation.

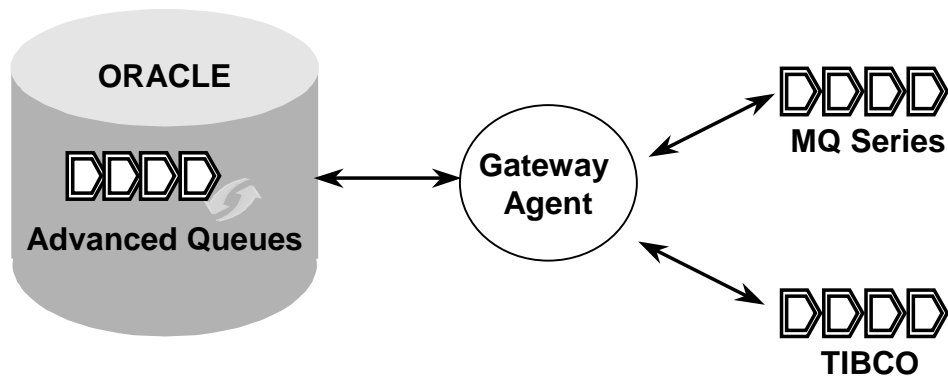
A transformation can be specified during enqueue to transform the message to the correct type before inserting it into the queue.

It can be specified during dequeue to receive the message in the desired format.

If specified with a remote subscriber, the message will be transformed before propagating it to the destination queue.

Messaging Gateway

**Gateway to third-party messaging systems:
MQ Series, TIBCO**



ORACLE

Messaging Gateway

Messaging gateway enables communication of AQ with other messaging systems such as MQ Series, and TIBCO.

Management with Oracle Enterprise Manager

- **Oracle Enterprise Manager provides two main sections for managing AQ:**
 - **AQ administration**
 - **AQ objects – They are managed under their respective queue tables**
- **The Diagnostics and Tuning Pack supports alerts and monitoring for AQ queues.**

ORACLE

1-19

Copyright © Oracle Corporation, 2001. All rights reserved.

AQ Management with Oracle Enterprise Manager

Oracle Enterprise Manager can be used to manage AQ. AQ administrative functionality has been provided through Oracle DBA studio. DBA studio can be used to create queue tables, create queues, browse through AQ messages, archive or purge AQ messages, add AQ subscribers, and manage propagation. DBA studio also shows topology for propagation of messages between queues at the database level and the queue level.

Oracle Enterprise Manager Packs support alerts and monitoring for AQ queues. Alerts can be sent when the number of messages for a particular subscriber exceeds a threshold. Alerts can be sent when there is an error in propagation. In addition, queues can be monitored for the number of messages in ready state or the number of messages per subscriber, and so on.

Summary

In this lesson, you should have learned about:

- **The limitations of traditional applications**
- **The need for asynchronous interaction between applications**
- **The different features of AQ that extend beyond just asynchronous interaction**

ORACLE

2

Initiating the Use of Advanced Queuing

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- **Set up the initialization parameters for advanced queues**
- **Set up the database with advanced queues**
- **Verify that the proper setup is complete**

ORACLE

AQ Interfaces

- **Administrative interface**
 - Administrative interfaces are used to manage AQ resources such as queue tables, queues, and so on.
 - PL/SQL and JMS support administrative functions.
- **Operational interface**
 - Operational interfaces are for used to perform enqueue, dequeue, and similar message management operations.
 - Operational interfaces are provided for PL/SQL, JMS, C (OCI), Visual Basic (OO4O), and Internet Data Access Protocol (IDAP)
 - IDAP defines an API for AQ operations that use XML message formats.

ORACLE

2-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle AQ Interface

The following programmatic environments are used to access the Advanced Queuing functions of Oracle:

- Native AQ Interface
 - PL/SQL (DBMS_AQADM and DBMS_AQ packages) supports administrative and operational functions.
 - C (OCI) supports operational functions
 - Visual Basic (OO4O) supports operational functions
- JMS Interface to AQ
 - Java (javax.jms and oracle.jms packages using JDBC) supports the standard JMS administrative and operational functions and Oracle JMS Extensions
- IDAP
 - IDAP provides an API for AQ operations that use XML messages. The Oracle AQ XML servlet provides operational functions using an XML message format in HTTP or SMTP.

PL/SQL Interfaces

- **Administrative interface is provided through the DBMS_AQADM package.**
 - Contains over 30 procedures to manage AQ configuration and administration
 - Is used to manage queue tables, queues, subscriptions, and propagation.
- **Operational interface is provided through the DBMS_AQ package.**
 - Provides a programmatic interface to Oracle's AQ
 - Contains the following procedures:
DEQUEUE, LISTEN, REGISTER, UNREGISTER, POST,
BIND_AGENT, UNBIND_AGENT

ORACLE

2-4

Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL interfaces

PL/SQL packages, DBMS_AQADM and DBMS_AQ, support access to Oracle Advanced Queuing administrative and operational functions using the native AQ interface. These functions include the following:

- Create: queue, queue table, nonpersistent queue, multiconsumer queue or topic, raw type message, message with structured data
- Get: queue table, queue, multi-consumer queue/topic
- Alter: queue table, queue/topic
- Drop: queue/topic
- Start or stop: queue/topic
- Grant and revoke privileges including database users including AQ internet agents
- Add, remove, alter subscriber, AQ internet agents
- Enable, disable, and alter propagation schedule
- Enqueue or publish messages to queue/topic
- Subscribe for messages in multiconsumer queue
- Receive and browse messages from queue/topic
- Register to receive messages asynchronously
- Listen for messages on multiple queues/topics
- Post messages to anonymous subscriptions
- Bind or unbind agents in a LDAP server
- Add or remove aliases to AQ objects in a LDAP server

C (OCI) Interface

- **Oracle Call Interface (OCI) provides an operational interface to AQ functions.**
- **An OCI client can perform the following actions:**
 - **Enqueue messages**
 - **Dequeue messages**
 - **Listen for messages on sets of queues**
 - **Register to receive message notifications**

ORACLE

2-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle Call Interface for C

Oracle Call Interface (OCI) provides an interface to Oracle Advanced Queuing functions using the native AQ interface.

An OCI client can perform the following actions:

- Enqueue messages
- Dequeue messages
- Listen for messages on sets of queues
- Register to receive message notifications
- In addition, OCI clients can receive asynchronous notifications for new messages in a queue using `OCISubscriptionRegister`

For queues with a user-defined payload type, OTT can be used to generate OCI mapping for the Oracle type. The OCI client is responsible for freeing the memory of the AQ descriptors and the message payload.

Visual Basic Interface

- **Visual Basic (OO4O) supports access to operational functions using the native AQ interface.**
- **These functions include the following:**
 - **Create: connection, RAW message, message with structured data**
 - **Enqueue or publish messages to queue/topic**
 - **Browse messages in a queue**
 - **Receive messages from queue/topic**
 - **Register to receive messages asynchronously**

ORACLE

2-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Visual Basic – Using OO4O interface

Visual Basic (OO4O) supports access to Oracle Advanced Queuing operational functions using the native AQ interface.

These functions include the following:

- Create: connection, RAW message, message with structured data
- Enqueue messages to single consumer queue (point-to-point model)
- Publish messages to multiconsumer queue/topic (publish-subscribe model)
- Browse messages in a queue
- Receive messages from queue/topic
- Register to received messages asynchronously.

Using Oracle JMS

- **Java Messaging Service (JMS) is a Java standard.**
- **Oracle JMS provides a Java API for AQ based on JMS.**
- **Oracle JMS extends the standard JMS features. Oracle JMS extensions include:**
 - **Message propagation between destinations**
 - **Supports transacted sessions that enable you to perform JMS as well as SQL operations in one atomic transaction**
 - **Message retention after messages have been dequeued**
 - **Message delay: messages can be made visible after a certain delay**
 - **Exception handling: messages are moved to exception queues if they cannot be processed successfully**

ORACLE

2-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Using Oracle Java Messaging Service to Access AQ

Oracle Java Messaging Service provides a Java API for AQ based on the JMS standard. Oracle JMS supports the standard JMS interfaces and has extensions to support the AQ administrative operations and other AQ features that are not a part of the standard.

Standard JMS features include:

- Point-to-point model of communication, using queues
- Publish-subscribe model of communication, using topics
- Five types of messages: `ObjectMessage`, `StreamMessage`, `TextMessage`, `BytesMessage`, `MapMessage`
- Synchronous and asynchronous delivery of messages
- Message selection based on message header fields or properties

Oracle JMS extensions include the following:

- **Administrative API:** to create queue tables, queues and topics
- **Point-to-multipoint communication:** using recipient lists for topics
- **Message propagation between destinations:** allows the application to define remote subscribers
- **Supports transacted sessions** that enable you to perform JMS as well as SQL operations in one atomic transaction
- **Message retention** after messages have been dequeued
- **Message delay:** messages can be made visible after a certain delay
- **Exception handling:** messages are moved to exception queues if they cannot be processed successfully

Using Oracle Java Messaging Service to Access AQ (continued)

Oracle JMS extensions

In addition to the standard JMS message types, Oracle supports `AdtMessages`. These are stored in the database as Oracle objects and hence the payload of the message can be queried after it is enqueued. Subscriptions can be defined on the contents of these messages as opposed to just the message properties.

Topic browsing allows durable subscribers to browse through the messages in a publish-subscribe (topic) destination, and optionally allows these subscribers to purge the browsed messages (so that they are no longer retained by AQ for that subscriber).

Accessing Standard and Oracle JMS

Oracle JMS uses JDBC to connect to the database, hence its applications can run as follows:

- Outside the database using the OCI or thin JDBC driver
- Inside Oracle JServer using the Oracle server driver

The standard JMS interfaces are in the `javax.jms` package.

The Oracle JMS interfaces are in the `oracle.jms` package.

Using OCI8 or Thin JDBC Driver: To use JMS with clients running outside the database, you must include the appropriate JDBC driver, JNDI jar files and the following AQ jar files in your CLASSPATH:

For JDK 1.1 include the following:

- `$ORACLE_HOME/rdbms/jlib/jmscommon.jar`,
`$ORACLE_HOME/rdbms/jlib/aqapi11.jar`,
`$ORACLE_HOME/jlib/jndi.jar`, and
`$ORACLE_HOME/jdbc/lib/classes111.jar`

For JDK 1.2 include the following in the `$ORACLE_HOME/rdbms/jlib/` folder

- `$ORACLE_HOME/rdbms/jlib/jmscommon.jar`,
`$ORACLE_HOME/rdbms/jlib/aqapi.jar`, `$ORACLE_HOME/jlib/jndi.jar`,
and `$ORACLE_HOME/jdbc/lib/classes12.jar`

Using Oracle Server Driver in JServer:

- If your application is running inside the JServer, you should be able to access the Oracle JMS classes that have been automatically loaded when the JServer was installed. If these classes are not available, you may have to load `jmscommon.jar` followed by `aqapi.jar` using the `loadjava` utility.

Privileges

Users must have the `EXECUTE` privilege on the `DBMS_AQIN` and `DBMS_AQJMS` packages in order to use the Oracle JMS interfaces. Users can also acquire these rights through the `AQ_USER_ROLE` or the `AQ_ADMINISTRATOR_ROLE`.

Users must also have the appropriate system and queue or topic privileges to send or receive messages.

Java Native Interface

- **Java AQ API supports both the administrative and operational features of AQ.**
- **Use JDBC to open a connection to the database and then to AQ.**
- **Common interfaces are prefixed with "AQ".**
- **The Java AQ classes are located in folder \$ORACLE_HOME/rdbms/jlib/.**
- **For JDK 1.2 include aqapi.jar in the CLASSPATH**
- **For JDK 1.1 include aqapi11.jar in the CLASSPATH**
- **Oracle recommends using the JMS interface to perform AQ operations.**

ORACLE

2-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Java Native Interface

Java Native AQ API supports both the administrative and operational features of AQ. In developing Java programs for messaging applications, you should use JDBC to open a connection to the database and then the Java AQ API for message queuing. This means that you will no longer need to use PL/SQL interfaces.

Common interfaces are prefixed with "AQ". In this course you learn about the common interfaces and their corresponding Oracle9i implementations, which are in turn prefixed with "AQOracle".

Accessing Java AQ Classes

The Java AQ classes are located in \$ORACLE_HOME/rdbms/jlib/aqapi.jar. These classes can be used with any Oracle9i JDBC driver.

- Using OCI or Thin JDBC Driver:
 - For JDK 1.2 you must include \$ORACLE_HOME/rdbms/jlib/aqapi.jar in the CLASSPATH
 - For JDK 1.1 you must include \$ORACLE_HOME/rdbms/jlib/aqapi11.jar in the CLASSPATH.
- Using Oracle server driver in JServer:
 - If the application is using the Oracle server driver and accessing the Java AQ API from Java-stored procedures, the Java files are generally automatically preloaded in a Java-enabled database. If the Java files are not loaded, you must first load the jmscommon.jar and aqapi.jar files into the database using the loadjava utility.

Managing Java.AQ API

- **AQDriverManager** is used to create an **AQSession**.
- **AQSession** is used to perform messaging tasks.
- When **createSession** method is invoked, it calls the appropriate driver.
- Users must have the **EXECUTE** privilege on the **DBMS_AQIN** package to use **AQ Java** interface.
- **AQ_USER_ROLE** and **AQ_ADMINISTER_ROLE** contain the **EXECUTE** privilege on **DBMS_AQIN**.

ORACLE

2-10

Copyright © Oracle Corporation, 2001. All rights reserved.

Managing the Java AQ API

The various implementations of the Java AQ API are managed through an **AQDriverManager**. The driver manager is used to create an **AQSession** that can be used to perform messaging tasks.

The AQ driver is registered using the `Class.forName(oracle.AQ.AQOracleDriver)` command.

When the `AQDriverManager.createAQSession()` method is invoked, it calls the appropriate **AQDriver** (amongst the registered drivers) depending on the parameter passed to the `createAQSession()` call.

The Oracle9i **AQDriver** expects a valid **JDBC** connection to be passed in as a parameter to create an **AQSession**. Users must have the **execute** privilege on the **DBMS_AQIN** package to use the **AQ Java** interfaces. Users can also acquire these rights through the **AQ_USER_ROLE** or the **AQ_ADMINISTRATOR_ROLE**. Users also must have the appropriate system and queue privileges for 8.1-style queue tables.

Using IDAP

- You can use IDAP to access AQ by using open protocols like HTTP, HTTPS, and SMTP.
- The following actions can be performed:
 - Send messages to single-consumer queues
 - Publish messages to multiconsumer queues/topics
 - Receive messages from queues
 - Register to receive message notifications
- The servlet can be deployed on any Web server or ServletRunner that implements Javasoft's Servlet 2.0 or Servlet 2.2 interfaces.

ORACLE

2-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Using IDAP to Access AQ

You can use the AQ XML servlet to access AQ by using open protocols such as HTTP and SMTP and using an XML message format called Internet Data Access Presentation (IDAP).

Using the AQ servlet, a client can perform the following actions:

- Send messages to single-consumer queues
- Publish messages to multiconsumer queues/topics
- Receive messages from queues
- Register to receive message notifications

The servlet can be deployed on any Web server or ServletRunner that implements Javasoft's Servlet 2.0 or Servlet 2.2 interfaces.

To deploy the AQ Servlet with a Web server that implements Javasoft's interfaces, you must define a class that extends:

- The `oracle.AQ.xml.AQxmlServlet20` class for Javasoft's Servlet2.0 interfaces
- The `oracle.AQ.xml.AQxmlServlet` class for Javasoft's Servlet2.2 interfaces

These classes in turn extend the `javax.servlet.http.HttpServlet` class.

Using IDAP to Access AQ (continued)

Location of JDK Library

The servlet can be compiled using JDK 1.1.x or JDK 1.2.x libraries.

For JDK 1.1.x the CLASSPATH must contain the following:

- \$ORACLE_HOME/jdbc/lib/classes111.jar
\$ORACLE_HOME/jlib/jta.jar
\$ORACLE_HOME/jdbc/lib/nls_charset11.jar
\$ORACLE_HOME/jlib/jndi.jar
\$ORACLE_HOME/lib/lclasses11.zip
\$ORACLE_HOME/lib/xmlparserv2.jar
\$ORACLE_HOME/lib/xschem.jar \$ORACLE_HOME/rdbms/jlib/aqapi11.jar
\$ORACLE_HOME/rdbms/jlib/jmscommon.jar
\$ORACLE_HOME/rdbms/jlib/aqxml.jar
\$ORACLE_HOME/rdbms/jlib/xsul11.jar
\$ORACLE_HOME/lib/servlet.jar.

For JDK 1.2.x the CLASSPATH must contain:

- \$ORACLE_HOME/jdbc/lib/classes12.jar
\$ORACLE_HOME/jlib/jta.jar
\$ORACLE_HOME/jdbc/lib/nls_charset12.jar
\$ORACLE_HOME/jlib/jndi.jar \$ORACLE_HOME/lib/lclasses12.zip
\$ORACLE_HOME/lib/xmlparserv2.jar
\$ORACLE_HOME/lib/xschem.jar
\$ORACLE_HOME/rdbms/jlib/aqapi.jar
\$ORACLE_HOME/rdbms/jlib/jmscommon.jar
\$ORACLE_HOME/rdbms/jlib/aqxml.jar
\$ORACLE_HOME/rdbms/jlib/xsul2.jar
\$ORACLE_HOME/lib/servlet.jar

Because the servlet uses JDBC OCI drivers to connect to the Oracle9i database server, the 9i Oracle Client libraries must be installed on the machine that hosts the servlet. The LD_LIBRARY_PATH must contain \$ORACLE_HOME/lib.

Initiating Use of AQ

- **Oracle9i Advanced Queuing executables are already installed with Oracle9i.**
- **To use the AQ infrastructure in a database, you might need to:**
 - **Set up initialization parameters**
 - **Decide on the programmatic and administrative environments**
 - **Set up users and agents**

ORACLE

Initialization Parameters

AQ_TM_PROCESSES

- Use the **AQ_TM_PROCESSES** parameter to enable the queue monitoring background process (QMN).
- QMN processes perform the cleanup of messages, making messages visible after a specified delay time, and moving expired messages to the exception queue.
- **AQ_TM_PROCESSES** can have value from 0 to 10. Default is 0.
- If **AQ_TM_PROCESSES** is set to 1, one queue monitor process is enabled.
- If this parameter is set to 0, the queue monitor process is not created.

ORACLE

2-14

Copyright © Oracle Corporation, 2001. All rights reserved.

AQ_TM_PROCESSES

This parameter specifies the number of background processes that are enabled for queue monitoring. These processes perform AQ operations such as cleaning up messages, enabling delay, and moving expired messages to exception queue.

When you use the delay and expiration properties in AQ messages, or you want to perform time monitoring on queue messages, you should set the **AQ_TM_PROCESSES** parameter appropriately.

Initialization Parameters

JOB_QUEUE_PROCESSES

- Propagation of messages is handled by job queue (SNPn background) processes.
- The number of job queue processes is controlled by the `JOB_QUEUE_PROCESSES` parameter. The default value of this parameter is 0.
- Set `JOB_QUEUE_PROCESSES` to at least 1 to enable propagation.
- The maximum value depends on the Oracle database version.
 - In Oracle8i, the maximum is 36
 - In Oracle9i, the maximum is 1000

ORACLE

2-15

Copyright © Oracle Corporation, 2001. All rights reserved.

JOB_QUEUE_PROCESSES

This initialization parameter enables the background processes that handle propagation.

Summary

In this lesson, you should have learned how to :

- **Setup the initialization parameters for advanced queues.**
- **Prepare the tablespaces for queue tables and queues.**

ORACLE[®]

3

Creating Queue Tables and Queues

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- **Create AQ users and roles**
- **Create queue tables**
- **Create persistent and nonpersistent queues**
- **Start and stop queues**
- **Manage subscriptions to queues**
- **Monitor queue usage with views**

ORACLE

Building a Queue Model

- **Build a queue model before you start to create queue administrators, queue tables, and queues.**
- **The model should answer the following questions:**
 - Which users will be the queue administrators?
 - Who (all) will own the queues?
 - Who will enqueue and who will dequeue?
 - Which queues will be involved in propagation.
- **The model helps in:**
 - Setting up the necessary users and providing the appropriate privileges
 - Setting up necessary database links

ORACLE

3-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Building a Queue model

Much like the data model is used before building the applications, you should consider building a queue model before starting to create queue administrators, queue tables, and queues.

Also consider to implement the queue tables in locally managed tablespaces for

- Easier management
- Performance implications

Set Up Users: AQ Administrator

- **Users of AQ are one of two types: AQ administrators, and AQ users**
- **Log in as SYSTEM user, create the AQ administrator, and grant the following roles and privileges to the AQ administrative user:**
 - **AQ_ADMINISTRATOR_ROLE**
 - **EXECUTE privilege on DBMS_AQADM and DBMS_AQ packages**
 - **CREATE ANY TABLE**
 - **ENQUEUE ANY QUEUE and DEQUEUE ANY QUEUE**
 - **CREATE DATABASE LINK**
- **View the creation of AQ Administrator using OEM**

ORACLE

3-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating AQ Administrator

To set up a user aqadm as an AQ administrator, do the following:

```
CONNECT system/manager
CREATE USER aqadm IDENTIFIED BY aqadm
DEFAULT TABLESPACE USERS TEMPORARY TABLESPACE TEMP
QUOTA UNLIMITED ON USERS;
GRANT CONNECT, RESOURCE TO aqadm;
GRANT AQ_ADMINISTRATOR_ROLE TO aqadm;
```

You should set the default tablespace for AQ administrator to be the one that holds the queue tables. It is advisable to keep this a locally managed tablespace to improve manageability and performance. Also ensure that a temporary tablespace is defined for the AQADM user and is of the type temporary.

Grant execute privilege on the AQ packages to allow aqadm user to execute the procedures in the AQ packages from within any other procedure of aqadm.

```
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT EXECUTE ON DBMS_AQ TO aqadm;
GRANT CREATE ANY TABLE, CREATE ANY QUEUE TO aqadm;
GRANT CREATE DATABASE LINK TO aqadm;
```

Creating AQ Administrator (continued)

The CREATE DATABASE LINK privilege is necessary for the AQ administrator to set up propagation across databases.

Also, if you consider using Oracle Enterprise Manager to perform AQ administration, then grant the AQ administrator with the SELECT ANY CATALOG privilege.

Create AQ Users

- **Connect as SYSTEM user and create the AQ users.**
- **Grant the AQ user EXECUTE privilege on DBMS_AQ in addition to the normal database user privileges such as CREATE SESSION, and so on.**
- **Depending on usage, you can grant privileges to**
 - **An individual queue, or**
 - **All the queues at the system level**

Creating AQ Users AQUSER1 and AQUSER2

To create AQ users who create and access queues within their own schemas, follow the steps outlined in the slide but do not grant the AQ_ADMINISTRATOR_ROLE.

```
CONNECT system/manager
CREATE USER aqadm IDENTIFIED BY aquser1;
GRANT CONNECT, RESOURCE TO aquser1;
GRANT EXECUTE ON DBMS_AQADM TO aquser1;
GRANT EXECUTE ON DBMS_AQ TO aquser1;
```

If you want to create an AQ user who does not create queues but uses a queue in another schema, first follow the steps outlined in the previous section. In addition, you must grant object level privileges. However, note that this applies only to queues defined using 8.1 compatible queue tables.

For aquser2 to access the queue, aquser1_q1 in aquser1 schema, aquser1 must execute the following statements:

```
CONNECT aquser1/aquser1
EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
    'ENQUEUE', 'aquser1_q1', 'aquser2', FALSE);
```


DBMS_AQADM.Grant_Queue_Privilege

- Grants queue-level access

```
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (  
    privilege          IN    VARCHAR2,  
    queue_name         IN    VARCHAR2,  
    grantee            IN    VARCHAR2,  
    grant_option        IN    BOOLEAN := FALSE);
```

- The privilege can be ENQUEUE, DEQUEUE, ALL (includes both enqueue and dequeue)
- grant_option allows you to grant the privilege to someone else
- API available in PL/SQL and Java

ORACLE

Queue Tables

- Queue tables are the storage bins for messages.
- You should specify the following when you create a queue table:
 - The type of payload the messages contain: RAW or Object Type.
 - Whether the queue is meant for single or multiple consumers
 - The preferred order of retrieval of messages from the queue
- You should use `STORAGE_CLAUSE` to specify the tablespace if you do not want the queue table to be in the default tablespace.

ORACLE

3-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Queue Tables

Queues are stored in queue tables. Each queue table is a database table and contains one or more queues. Each queue table contains a default exception queue. Queue tables are locations that hold the interface between applications which may be on different schemas and different databases as well. You should give due consideration to the location of queue tables.

RAW payload should only be used where a common understanding of the format of the message is available in all communicating applications. Queue payload is opaque in these queues. Sort order is the order in which messages will be dequeued.

After you specify the tablespace in `STORAGE_CLAUSE`, all IOTs and indexes created for that queue table will go to the specified tablespace.

If you intend that the queue in this table is intended for multiple consumers, then you should specify that when creating the queue table.

Create Queue Tables: Using the PL/SQL Interface

- **Create queue table for queues containing messages of object type**

```
– CREATE type aq.Message_typ as object
  (Subject VARCHAR2(30), Text VARCHAR2(80));
– EXECUTE dbms_aqadm.create_queue_table
  (Queue_table => 'aq.ObjMsgs_qtab',
   Queue_payload_type => 'aq.Message_typ'
   storage_clause 'tablespace aq_tbs' );
```

- **Create queue table for queues containing messages of RAW type**

```
– EXECUTE dbms_aqadm.create_queue_table
  (Queue_table => 'aq.RawMsgs_qtab',
   Queue_payload_type => 'RAW');
```

ORACLE

3-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Queue Tables

Queue names and queue table names are converted to uppercase. Mixed case (upper and lowercase together) is not supported.

The following clauses are important:

storage_clause

- Useful to specify tablespace and other storage parameters such as inittrans, maxtrans, freelists, etc.

queue_payload_type

- RAW, Oracle object type, or AQ\$_JMS_*

sort list

- enqueue_time, priority or priority, enqueue_time
- The sort keys for dequeue ordering, if any, need to be defined at table creation time.

Compatible (option deprecated in Oracle9i)

- The default value of the COMPATIBLE clause in CREATE_QUEUE_TABLE procedure depends on the database COMPATIBLE initialization parameter of the database.
- If the database has COMPATIBLE set to 8.1 or higher, the default value is '8.1'
- If the database is in 8.0 compatible mode, the default value is '8.0'.

Creating Queue Tables (continued)

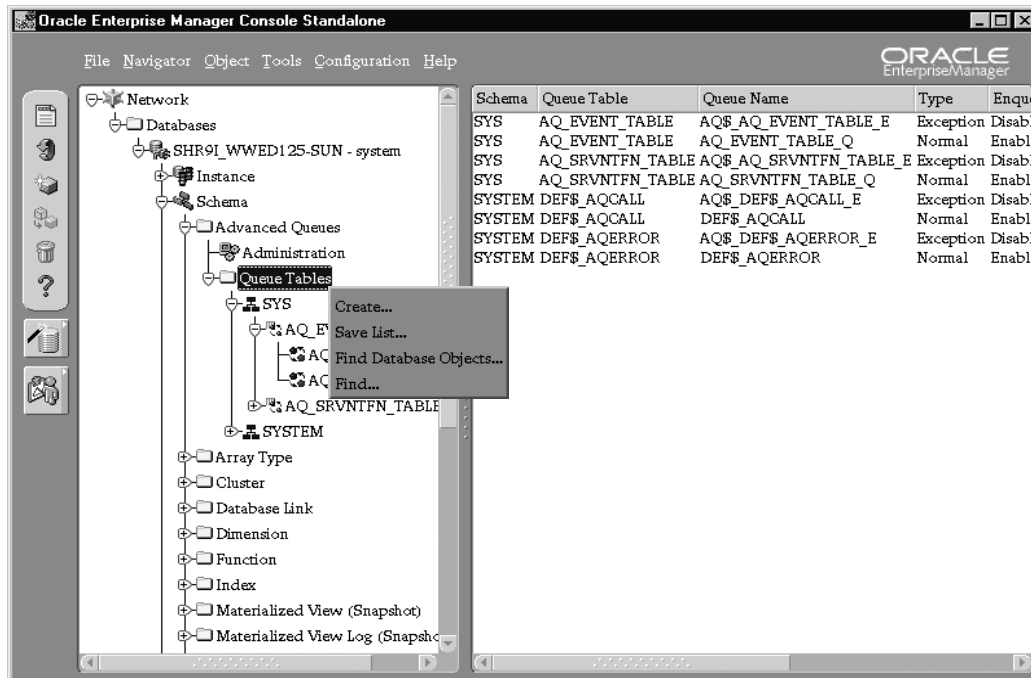
The following objects are created when you create a queue table:

- The default exception queue associated with the queue table called `aq$_<queue_table_name>_e`.
- A read-only view which is used by AQ applications for querying queue data called `aq$_<queue_table_name>`.
- An index for the queue monitor operations called `aq$_<queue_table_name>_t`.
- An index or an index organized table (IOT) in the case of multiple consumer queues for dequeue operations called `aq$_<queue_table_name>_i`.

For 8.1-compatible multiconsumer queue tables, the following additional objects are created:

- A table called `aq$_<queue_table_name>_s`. This table stores information about the subscribers.
- A table called `aq$_<queue_table_name>_r`. This table stores information about rules on subscriptions.
- An index organized table (IOT) called `aq$_<queue_table_name>_h`. This table stores the dequeue history data.
- CLOB, BLOB, or BFILE objects are valid in an AQ message. You can propagate these object types using AQ propagation. To enqueue an object type that has an LOB, you must first set the `LOB_attribute` to `EMPTY_BLOB()` and perform the enqueue. You can then select the LOB locator that was generated from the queue table's view and use the standard LOB operations.
- You can specify and modify the `primary_instance` and `secondary_instance` only in 8.1 compatible mode (used in Real Application Clusters).
- You cannot specify a secondary instance unless there is a primary instance.
- When a queue, queue table, or subscriber is created, modified, or dropped, and if `GLOBAL_TOPIC_ENABLED` clause is set to `TRUE`, a corresponding LDAP entry is also created.

Creating a Queue Table Using OEM



3-11

Copyright © Oracle Corporation, 2001. All rights reserved.

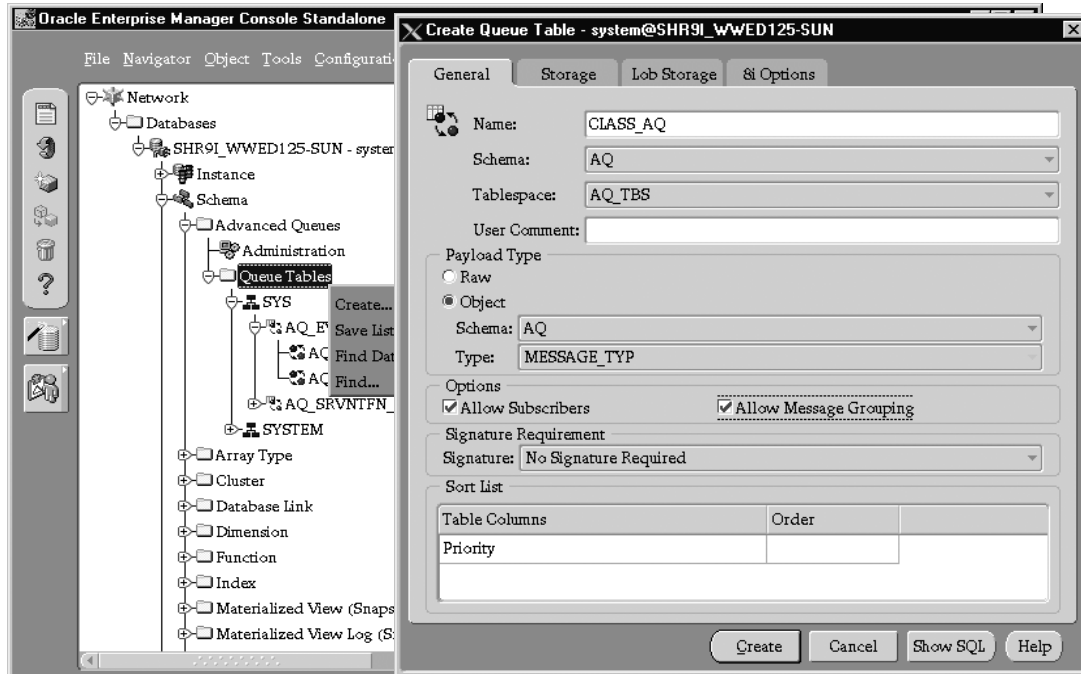
ORACLE

Creating a Queue Table Using Oracle Enterprise Manager (OEM)

Use the following steps to create a queue table.

1. Right click the Queue Table container in the Navigation panel to get the context sensitive menu.
2. Click Create ... on the context sensitive menu.
3. Queue Table Property window with four tabs appears.
4. Enter the relevant details and click the Create button. If you intend the queue table to hold queues for multiple consumers, select the Allow Subscribers checkbox under Options.
5. The confirmation appears.

Creating a Queue Table Using OEM



3-12

Copyright © Oracle Corporation, 2001. All rights reserved.

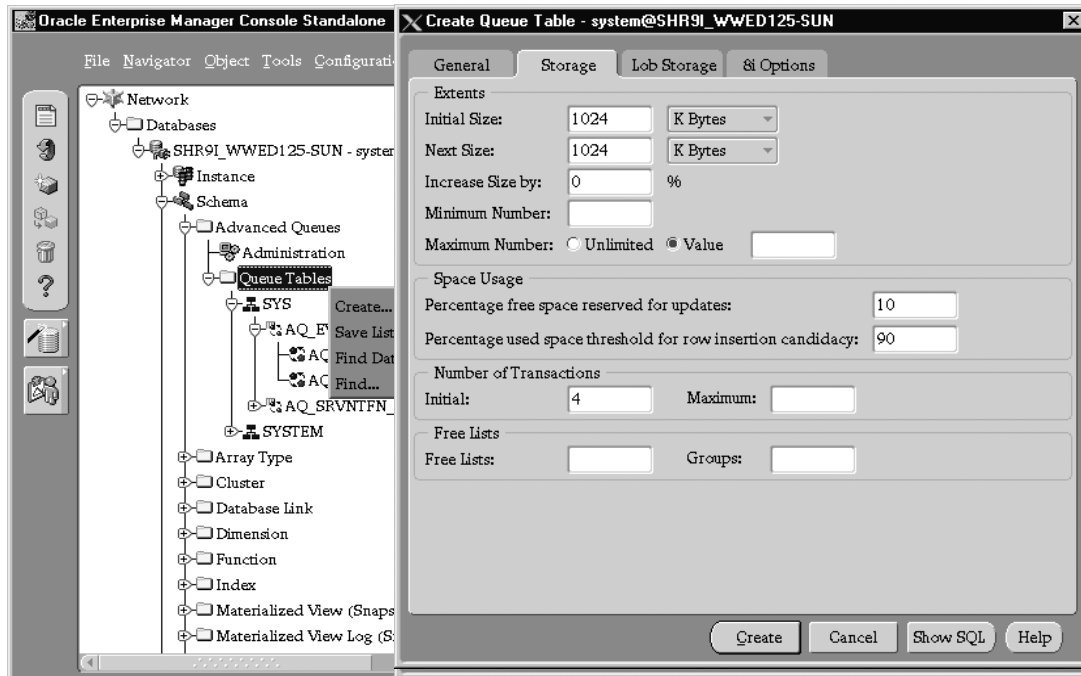
ORACLE

Creating Queue Table Using Oracle Enterprise Manager

Use the following steps to create a queue table.

1. Right Click the Queue Table container in the Navigation Panel to get the context sensitive menu.
2. Click Create ... on the context sensitive menu.
3. Queue Table Property window with four tabs appears.
4. Enter the relevant details and finally click create button. If you intend that the queue table would hold queues for multiple consumers, check the “Allow Subscribers” box under Options.
5. The confirmation appears.

Creating a Queue Table Using OEM



3-13

Copyright © Oracle Corporation, 2001. All rights reserved.

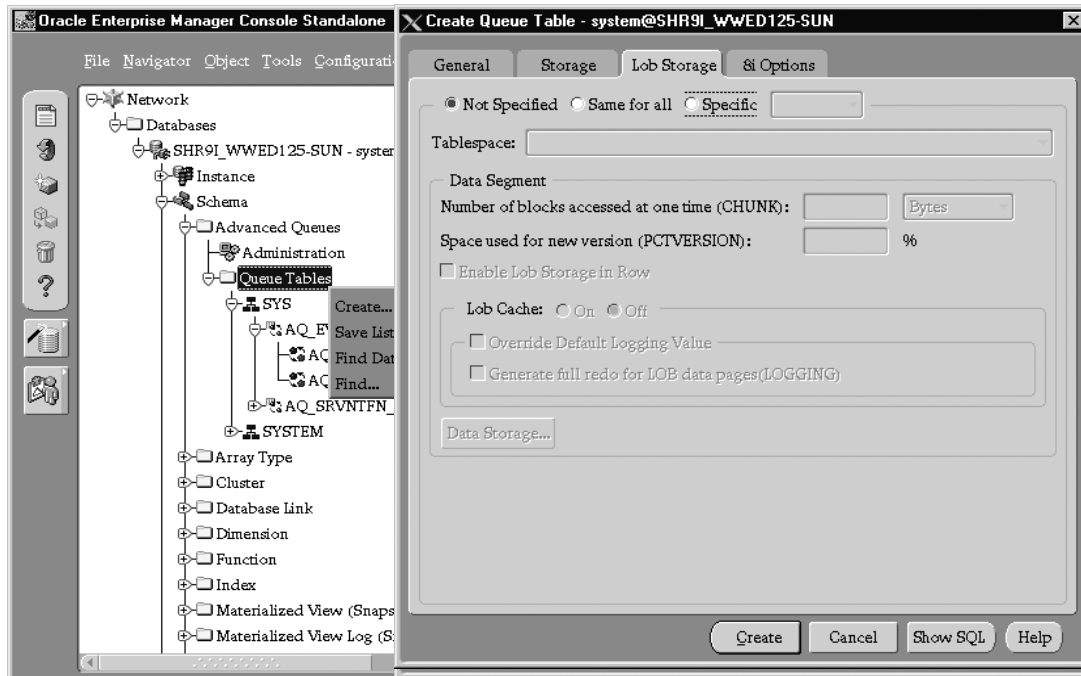
ORACLE

Creating Queue Table Using Oracle Enterprise Manager

Use the following steps to create a queue table.

1. Right Click the Queue Table container in the Navigation Panel to get the context sensitive menu.
2. Click Create ... on the context sensitive menu.
3. Queue Table Property window with four tabs appears.
4. Enter the relevant details and finally click create button. If you intend that the queue table would hold queues for multiple consumers, check the “Allow Subscribers” box under Options.
5. The confirmation appears.

Creating a Queue Table Using OEM



3-14

Copyright © Oracle Corporation, 2001. All rights reserved.

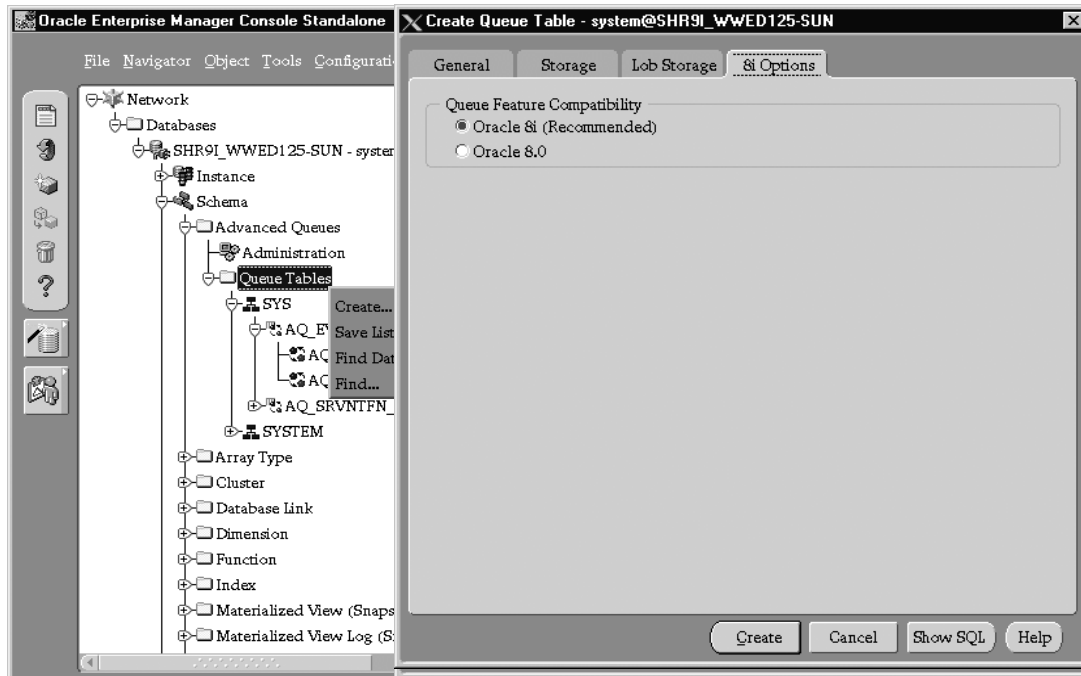
ORACLE

Creating Queue Table Using Oracle Enterprise Manager

Use the following steps to create a queue table.

1. Right Click the Queue Table container in the Navigation Panel to get the context sensitive menu.
2. Click Create ... on the context sensitive menu.
3. Queue Table Property window with four tabs appears.
4. Enter the relevant details and finally click create button. If you intend that the queue table would hold queues for multiple consumers, check the “Allow Subscribers” box under Options.
5. The confirmation appears.

Creating a Queue Table Using OEM



3-15

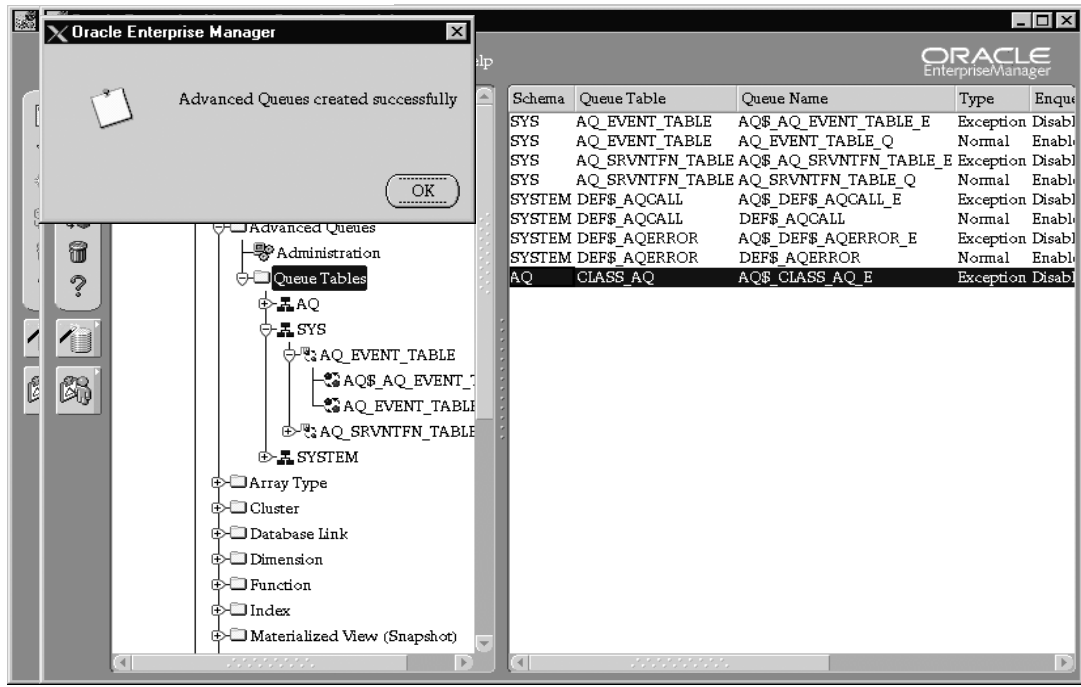
Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Queue Table Using Oracle Enterprise Manager

Use the following steps to create a queue table.

1. Right Click the Queue Table container in the Navigation Panel to get the context sensitive menu.
2. Click Create ... on the context sensitive menu.
3. Queue Table Property window with four tabs appears.
4. Enter the relevant details and finally click create button. If you intend that the queue table would hold queues for multiple consumers, check the “Allow Subscribers” box under Options.
5. The confirmation appears.

Creating a Queue Table Using OEM



3-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Queue Table Using Oracle Enterprise Manager

Use the following steps to create a queue table.

1. Right Click the Queue Table container in the Navigation Panel to get the context sensitive menu.
2. Click Create ... on the context sensitive menu.
3. Queue Table Property window with four tabs appears.
4. Enter the relevant details and finally click create button. If you intend that the queue table would hold queues for multiple consumers, check the "Allow Subscribers" box under Options.
5. The confirmation appears.

Queues

- A queue is a repository for messages.
- The user queue is for normal message processing.
- Messages are transferred to an exception queue if they cannot be retrieved and processed for some reason.
- Queues can be created, altered, started, stopped, and dropped by using the Oracle AQ administrative interfaces.
- User queues can be persistent (the default) or nonpersistent queues.
 - Persistent queues store messages in database tables.
 - Nonpersistent queues store messages in memory.

ORACLE

3-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Message Queues

A message queue represents a collection of messages. At the time of creation the a name is assigned to the queue, so that the application or user can specify which particular queue a message should be stored in or retrieved from.

You can use the Oracle-supplied package DBMS_AQADM for administrative tasks to allow queues to be created, altered, started, stopped, and dropped.

There is no limit to the number of queues defined in the database.

Create Queues

- To create queue you must be granted **EXECUTE** on the **DBMS_AQADM** package.
- Specify the table (along with the schema) in which you want the queue to be stored.
- You should explicitly enable the queue with a **START** command to allow enqueue and dequeue operations.
- After you have created the queue, you should grant suitable privileges to the database users for them to perform queue operations.
- Use **USER_QUEUES** or **DBA_QUEUES** views to monitor the queues that have been created.

ORACLE

3-18

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Queues

When creating the queues, you should specify the queue table (accordingly the schema) in which you want them to be stored. Queue names and queue table names are converted to upper case. All queue names must be unique within the schema. After a queue is created with **CREATE_QUEUE**, it can be enabled by calling **START_QUEUE**. By default, the queue is created with both enqueue and dequeue disabled.

- To view retained messages, you can either dequeue by message ID or use SQL.
- When a queue, queue table, or subscriber is created and if **GLOBAL_TOPIC_ENABLED = TRUE**, a corresponding LDAP entry is also created.

Create a Queue for Messages of Object Type – Using PL/SQL Interface

```
/* Create a message type: */
CREATE type aq.Message_typ as object (
  Subject VARCHAR2(30), Text VARCHAR2(80));
/* Create a object type queue table and queue: */
EXECUTE dbms_aqadm.create_queue_table (
  Queue_table => 'aq.ObjMsgs_qtab',
  Queue_payload_type => 'aq.Message_typ');
EXECUTE dbms_aqadm.create_queue (
  Queue_name => 'msg_queue', Queue_table => 'aq.ObjMsgs_qtab');
```

Grant Privileges to Users

- When created, only the owner of a queue table can perform enqueue or dequeue operations.
- There are two ways to enable other users to perform enqueue or dequeue operations:
 - Grant system privileges using the `DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE` procedure
 - Grant object privilege using the `DBMS_AQADM.GRANT_QUEUE_PRIVILEGE` procedure
- Use `GRANT_SYSTEM_PRIVILEGE` to grant privileges on all queues.
- System privileges cannot be granted using the Java (JDBC) interface.

ORACLE

3-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Grant Privileges to Users

When queues are created, only the queue table owner can use this procedure to grant privileges on the queues. You should grant `ENQUEUE` or `DEQUEUE` privileges on a queue to other users and roles to enable them to perform enqueue or dequeue operations on your queue.

Example: Using the `DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE`

```
CONNECT aqadm/aqadm;  
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(  
  privilege => 'ENQUEUE_ANY',  
  grantee => 'Jones',  
  admin_option => FALSE);
```

Example Using Java (JDBC)

```
/* Grant enqueue and dequeue privileges on queue to user 'Jones' */  
public static void example(AQSession aq_sess) throws AQException {  
  AQQueue queue;  
  /* Get the queue object */  
  queue = aq_sess.getQueue("AQ", "multiconsumermsg81_queue");  
  /* Enable enqueue and dequeue */  
  queue.grantQueuePrivilege("ALL", "Jones", true); }  
}
```

Revoke Privileges

- **To revoke privileges, use either of these procedures:**
 - DBMS_ADADM.REVOKE_SYSTEM_PRIVILEGE
 - DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE
- **Use REVOKE_SYSTEM_PRIVILEGE for revoking privileges from several queues.**

ORACLE

3-20

Copyright © Oracle Corporation, 2001. All rights reserved.

Revoke Privileges

- Example using system privilege

```
connect aqadm/aqadm
EXECUTE DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE(
  privilege=>'DEQUEUE_ANY',
  grantee=>'Jones');
```

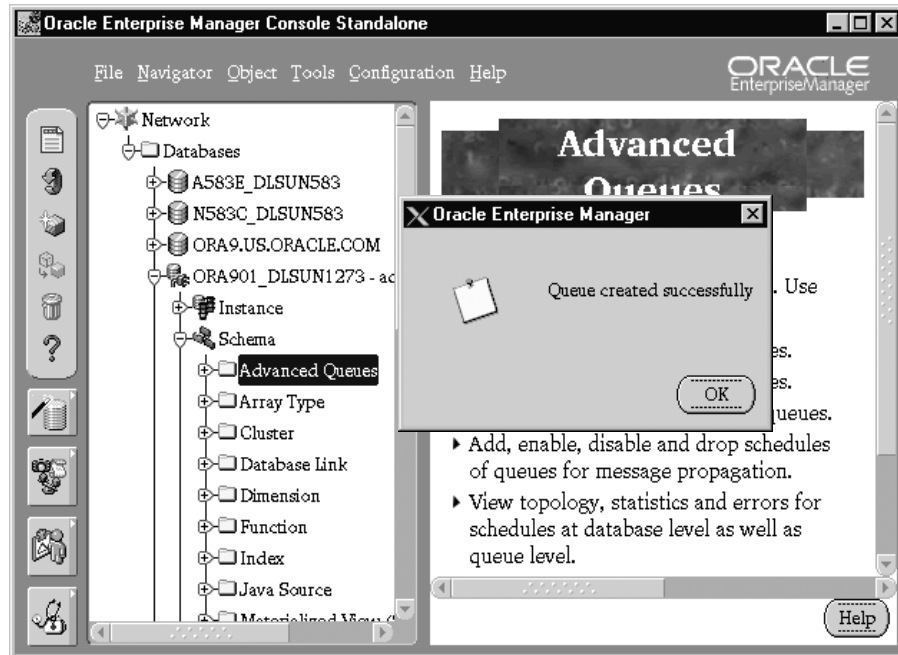
- Example using object privilege

```
EXECUTE DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE(
  privilege => 'DEQUEUE',
  queue_name => 'aq.aqmsgs_queue',
  grantee => 'Jones');
```

- Example using Java interface

```
public static void example(AQSession aq_sess) throws AQException {
  AQQueue queue;
  /* Get the queue object */
  queue = aq_sess.getQueue("SCOTT", "ScottMsgs_queue");
  /* Enable enqueue and dequeue */
  queue.revokeQueuePrivilege("DEQUEUE", "Jones"); }
```

Create Queues Using OEM

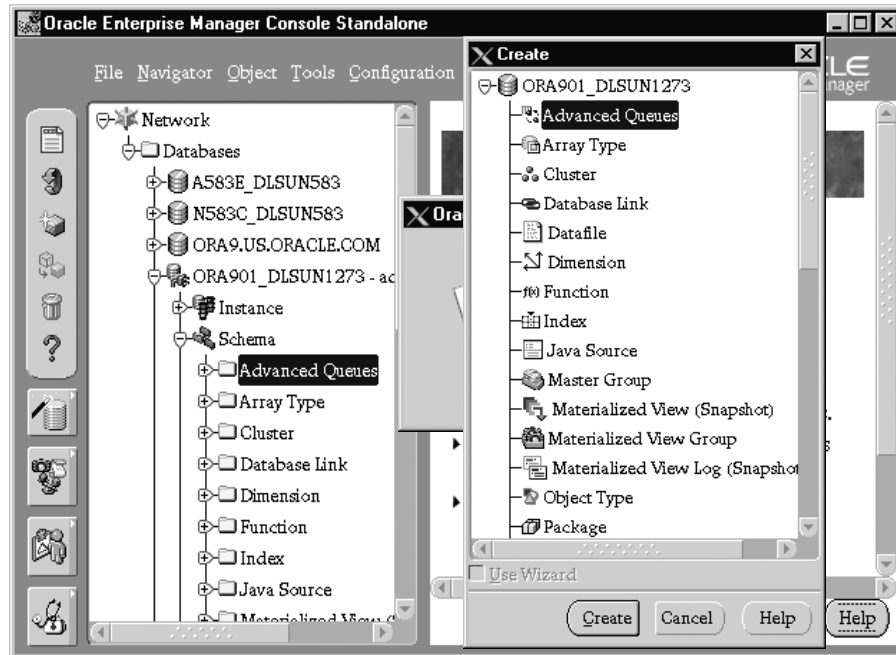


Create Queues Using OEM

To create queues using OEM, log in to the Console as AQ administrator, and perform the following steps:

1. Select Create from Object menu.
2. Select Queue from the object menu tree and click the Create button.
3. The Queue property window appears.
4. Enter the name and the correct queue table.
5. Click the Create button.

Create Queues Using OEM

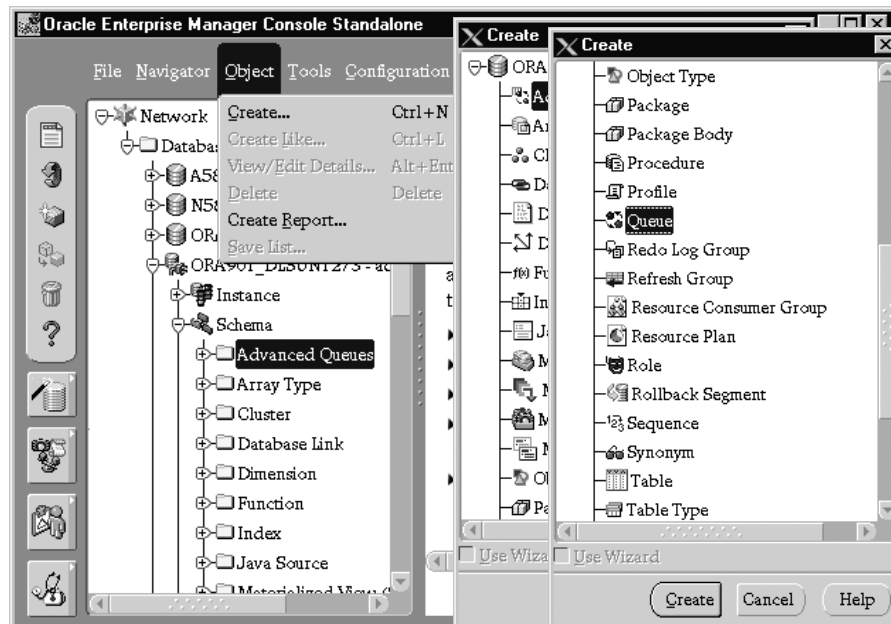


Create queues using OEM

To create queues using OEM, login to the Console as AQ Administrator, and perform the following steps:

1. Select Create from Object menu.
2. Select Queue from the object menu tree and click Create button.
3. The Queue property window appears.
4. Enter the name and the correct queue table.
5. Click Create button.

Create Queues Using OEM

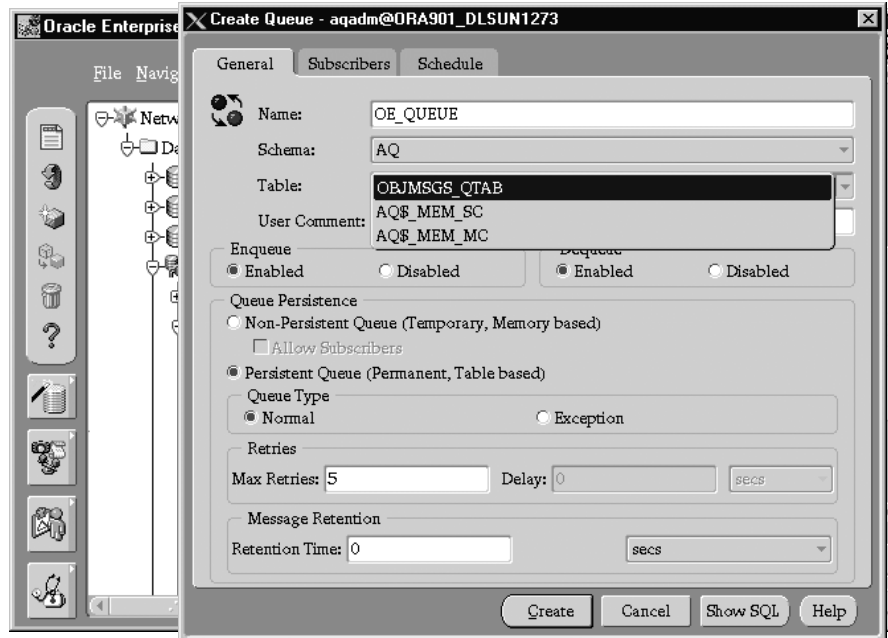


Create Queues Using OEM

To create queues using OEM, login to the Console as AQ Administrator, and perform the following steps:

1. Select Create from Object menu.
2. Select Queue from the object menu tree and click Create button.
3. The Queue property window appears.
4. Enter the name and the correct queue table.
5. Click Create button.

Create Queues Using OEM

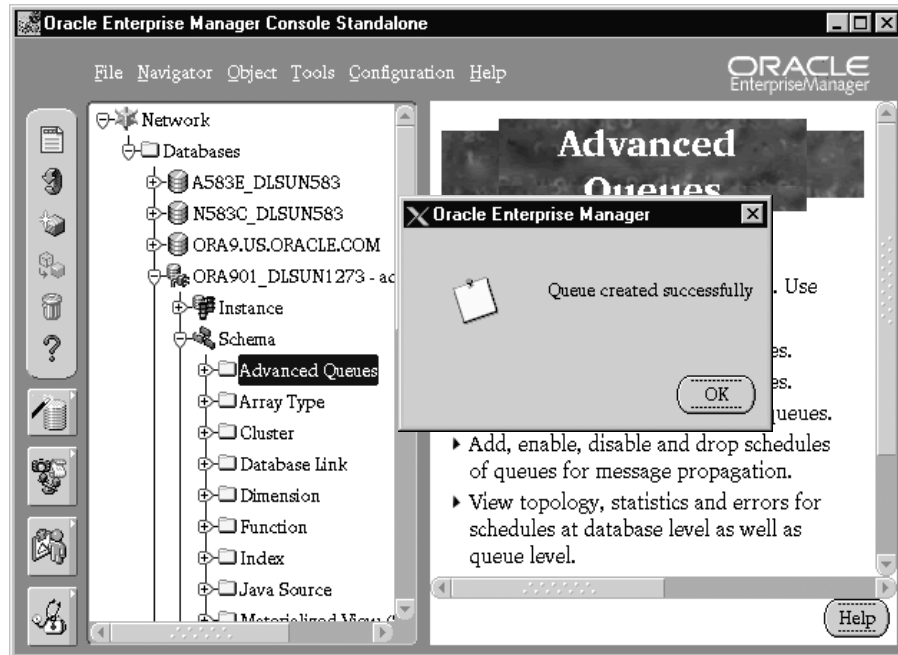


Create Queues Using OEM

To create queues using OEM, login to the Console as AQ Administrator, and perform the following steps:

1. Select Create from Object menu.
2. Select Queue from the object menu tree and click Create button.
3. The Queue property window appears.
4. Enter the name and the correct queue table.
5. Click Create button.

Create Queues Using OEM



Create Queues Using OEM

To create queues using OEM, login to the Console as AQ Administrator, and perform the following steps:

1. Select Create from Object menu.
2. Select Queue from the object menu tree and click Create button.
3. The Queue property window appears.
4. Enter the name and the correct queue table.
5. Click Create button.

Storage Queue Tables

- **Payload and message properties**
- **One row per message**
- **Message ordering is specified at the queue table level.**
- **Storage clause is inherited by indexes, IOTs, and tables.**

ORACLE

3-26

Copyright © Oracle Corporation, 2001. All rights reserved.

Storage Queue Tables

Message properties stored along with the message:

- Tracking (message ID, correlation ID, priority, state)
- Timing (delay, expiration)
- Enqueue (time, uid, txn id)
- Dequeue (time, uid, txn id)
- Retry count
- Exception queue
- Sender info (name, address, protocol)

Messages are inserted by enqueue, and deleted by dequeue, or by the queue monitor.

Messages may be retained after being processed. They may be moved to an exception queue in some cases.

The dequeue IOT (aq\$_QT_I) keys are based on the message ordering. Currently, there is one dequeue IOT per queue table.

Storage clause must be carefully specified since it is inherited by other related database objects. It can also affect performance of AQ operations.

Starting a Queue

- **Enables the specified queue for enqueueing or dequeuing**

```
DBMS_AQADM.START_QUEUE (
    queue_name      IN      VARCHAR2,
    enqueue         IN      BOOLEAN default TRUE,
    dequeue         IN      BOOLEAN default TRUE)
```

```
execute dbms_aqadm.start_queue (
    queue_name      => 'bzcardorders_q');
```

- **Only dequeue operations are allowed on an exception queue.**

```
execute dbms_aqadm.start_queue (
    queue_name      => 'aq$_bzcardorders_qt_e',
    enqueue         => FALSE, dequeue      => TRUE);
```

ORACLE

Starting a Queue

When queues are created, they may not be enabled for AQ operations such as enqueue or dequeue. To enable enqueue or dequeue operation, the queue should be started using the START_QUEUE procedure.

The exception queues should also be started, but only for dequeue operation as no explicit enqueue operation is possible on exception queue.

Stopping a Queue

- **Disables enqueueing or dequeuing on the specified queue**

```
DBMS_AQADM.STOP_QUEUE (
    queue_name      IN      VARCHAR2,
    enqueue          IN      BOOLEAN default TRUE,
    dequeue          IN      BOOLEAN default TRUE,
    wait             IN      BOOLEAN default TRUE);
```

- **Always stop_queue before doing any administrative operations**

```
execute
    dbms_aqadm.stop_queue(queue_name=>'bzcardorders_q');
```

ORACLE

3-28

Copyright © Oracle Corporation, 2001. All rights reserved.

Stopping a Queue

To stop a queue use DBMS_AQADM.STOP_QUEUE procedure. It takes the following input parameters:

queue_name : Specify the name of the queue to be disabled.

enqueue: Use this parameter to specify if ENQUEUE should be disabled.

TRUE: Disable ENQUEUE. This is the default.

FALSE: Do not alter the current setting.

dequeue: Specifies whether DEQUEUE should be disabled on this queue.

TRUE: Disable DEQUEUE. This is the default.

FALSE: Do not alter the current setting.

wait: Use this parameter so that the queue is stopped only after the completion of outstanding transactions.

TRUE: Set the queue in wait mode. No new enqueue or dequeue operations are allowed on the queue.

FALSE: Return immediately either with a success or an error.

Altering and Dropping Queues

- **Alter:**
 - Use the `dbms_aqadm.alter_queue` PL/SQL procedure or `queue.alterQueue` Java to alter properties of a queue.
 - Only `max_retries`, `comment`, `retry_delay`, and `retention_time` can be altered.
- **Drop:**
 - The queue should be stopped before being dropped.
 - All queue data is deleted as part of the drop operation.

ORACLE

3-29

Copyright © Oracle Corporation, 2001. All rights reserved.

Alter Queue Properties

You can use the `dbms_aqadm.alter_queue` PL/SQL procedure or `queue.alterQueue` Java procedure to alter the properties of an existing queue. You can alter `max_retries`, `comment`, `retry_delay`, and `retention_time` properties.

Example:

```
EXECUTE dbms_aqadm.alter_queue (  
    queue_name => 'aq.Anothermsg_queue', retention_time => 86400);
```

Drop Queue

You should first disable the queue for both enqueue and dequeue operations before dropping a queue. All the queue data is deleted as part of the drop operation.

Example:

```
• /* Stop Queue */  
EXECUTE dbms_aqadm.stop_queue ( Queue_name => 'aq.Msg_queue' );  
/* Drop queue: */  
EXECUTE dbms_aqadm.drop_queue ( Queue_name => 'aq.Msg_queue' );
```

Dictionary Catalog Views

```
SQL> select name, max_retries  
2      , retry_delay, retention  
3      from user_queues;
```

- The **USER_QUEUES** view provides the details of each queue.

```
SQL> select queue_table, type,  
2      object_type, sort_order  
3      from dba_queue_tables;
```

- **DBA_QUEUE_TABLES** provides information on all the queue tables in the database.
- You can also use **AQ\$<name of queue table>** to monitor queue information.

ORACLE

Monitor Queue Information

Use the catalog views to monitor the queues and queue tables.

Summary

In this lesson, you should have learned how to do the following:

- **Create AQ administrators**
- **Create AQ users**
- **Create queue tables**
- **Create queues**
- **Manage privileges on queues**

ORACLE

4

Performing Queue Operations

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- **Perform an enqueue operation**
- **Use time specifications in enqueue operations**
- **Use subscriptions and recipient lists in enqueue operations**
- **Perform a dequeue operation**
- **Check exception queues**

ORACLE

Queuing Operations

- The queue operations are:
 - Enqueue
 - Dequeue
 - Exception Handling
 - Listen
 - Notification
- To perform queue operations, use the `DBMS_AQ` package.

ORACLE

Queue Operations Privileges

- To perform any queue operation, a user must have the **EXECUTE** privilege on the **DBMS_AQ** procedure.
- The **EXECUTE** privilege on **DBMS_AQ** should be granted explicitly to the user if the user is making AQ calls from PL/SQL procedures.
- In addition AQ privileges are required. AQ privileges are at two levels.
 - System level (for all queues): Granted using the **DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE** procedure.
 - Queue level (for specific queue): Granted using the **DBMS_AQADM.GRANT_QUEUE_PRIVILEGE** procedure.

ORACLE

4-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Privileges

In addition to the database level **EXECUTE** privilege on the **DBMS_AQ** package, a user should be granted some AQ privileges to perform enqueue or dequeue operations.

Use **DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE** procedure to grant system level privilege to the user. The system level privileges can be

- **ENQUEUE_ANY**, **DEQUEUE_ANY**, or **MANAGE_ANY**

The **admin_option** allows the user (grantee) to grant this privilege to someone else. The system-level privilege can be granted in PL/SQL.

Use the **DBMS_AQADM.GRANT_QUEUE_PRIVILEGE** procedure to grant queue-level privileges to the user. The queue-level privileges are

- **ENQUEUE**, **DEQUEUE**, **ALL** (meaning both enqueue and dequeue).

The **grant_option** allows the user (grantee) to grant this privilege to others. The queue level privilege can be granted in PL/SQL and Java interfaces.

Messages of JMS data types that are larger than 2K in size are stored in LOB columns. If the object type contains a LOB attribute, the message is stored as a LOB. In such cases, the user also needs select, update, and insert privileges on the queue table.

Enqueue Operation

- Enqueue operations are performed using PL/SQL, Java, or Visual Basic operational interfaces.
- Enqueue options define the way a message is enqueued.
- Message properties define the way a message are handled when being dequeued

```
DBMS_AQ.ENQUEUE (
  QUEUE_NAME          IN   varchar2,
  ENQUEUE_OPTIONS     IN   enqueue_options_t,
  MESSAGE_PROPERTIES  IN   message_properties_t,
  PAYLOAD             IN   <object_type|RAW>,
  MSGID              OUT  raw)
```

ORACLE

4-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Enqueue Operation

Enqueue operation is performed using a call to DBMS_AQ.ENQUEUE procedure.

```
DBMS_AQ.ENQUEUE (
  queue_name IN VARCHAR2,
  enqueue_options IN enqueue_options_t,
  message_properties IN message_properties_t,
  payload IN "<type_name>",
  msgid OUT RAW);
```

Enqueue options:

There are mainly 4 enqueue options. They are visibility, squence_deviation, relative_msgid, and transformation.

Visibility:

- It is used to specify when the message should be visible to the other queue users. The values are IMMEDIATE, to specify immediately visible on completion of enqueue operation, and ON_COMMIT, to specify that the message should become visible when the transaction is committed.

Enqueue Operation (continued)

Enqueue options (continued)

Visibility

- Do not use the immediate option when you want to use LOB locators because LOB locators are valid only for the duration of the transaction. Because the immediate option automatically commits the transaction, your locator will not be valid.
- The visibility option must be immediate for nonpersistent queues. Only local recipients are supported for nonpersistent queues.

Transformation

- The transformation parameter specifies the name of the procedure that performs message transformation from an input form to an output form.
- It is used mainly when the two interacting applications have distinctly separate forms of messages to handle. This is usually the case in traditional applications and applications across different organizations.
- If a transformation is specified, it will be applied to the message before it is enqueued. The transformation must map the message into an object whose type is the ADT type of the queue.

Message Properties

- **Message properties describe messages.**
- **They are set when a message is enqueued, and their values are generally used at dequeue time.**
- **Enqueue specific message properties are:**

PRIORITY	BINARY_INTEGER	default 1,
DELAY	BINARY_INTEGER	default NO_DELAY,
EXPIRATION	BINARY_INTEGER	default NEVER,
CORRELATION	VARCHAR2(128)	default NULL,
RECIPIENT_LIST	AQ\$RECIPIENT_LIST_T,	
EXCEPTION_QUEUE	VARCHAR2(51)	default NULL,
ATTEMPTS	BINARY_INTEGER,	
ENQUEUE_TIME	DATE,	
ORIGINAL_MSGID	RAW(16)	default NULL
SENDER_ID	AQ\$AGENT	DEFAULT NULL,

ORACLE

4-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Message Properties

Message properties describe the message being enqueued and are set at the time the message is enqueued. When the message is dequeued the message properties are generally used to perform the dequeue operation appropriately.

- **Priority**
 - Specifies the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.
- **Delay**
 - Specifies the number of seconds after which a message is available for dequeuing. Dequeuing by msgid overrides the delay specification. A message enqueued with delay set will be in the WAITING state until the delay expires and then is set to the READY state. DELAY processing requires the queue monitor to be started. Note that delay is set at when the message is enqueued.
 - NO_DELAY the message is available for immediate dequeuing.
 - number: the number of seconds to delay the message.

Message Properties (continued)

- Expiration
 - Specifies in seconds, the duration the message is available for dequeuing. This parameter is an offset from the delay.
 - Expiration processing requires the queue monitor to be running.
 - NEVER: message does not expire.
 - number: number of seconds message remains in READY state. If the message is not dequeued before it expires, then it is moved to the exception queue in the EXPIRED state.
- Correlation
 - Specifies the identification for a message at enqueueing.
- Attempts
 - Specifies the number of attempts that have been made to dequeue this message. This parameter cannot be set at enqueue time. The maximum attempts allowed is specified when the queue is created.
- Recipient_list
 - This parameter is only valid for queues which allow multiple consumers. The default recipients are the queue subscribers. This parameter is not returned to a consumer at dequeue time.
- Exception_queue
 - Specifies the name of the queue to 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 or the message has expired. All messages in the exception queue are in the EXPIRED state.
 - The default is the exception queue associated with the queue table.
 - If the exception queue specified in the parameter does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert file.
- enqueue_time
 - Returns the time at which the message was enqueued. This value is determined by the system and cannot be set by the user. This parameter can not be set at enqueue time.
- state
 - Returns the state of the message at the time of the dequeue. This parameter can not be set at enqueue time.
 - 0: The message is ready to be processed.
 - 1: The message delay has not yet been reached.
 - 2: The message has been processed and is retained.
 - 3: The message has been moved to the exception queue.
- sender_id
 - Specifies the application-specified sender identification. Default is NULL.
- original_msgid
 - This parameter is used by Oracle AQ for propagating messages. Default is NULL

Enqueue Operation: Example

```
/* Enqueue to msg_queue: */
DECLARE enqueue_options
dbms_aq.enqueue_options_t;
message_properties
dbms_aq.message_properties_t;
message_handle RAW(16);
message aq.message_typ;
BEGIN message := message_typ('NORMAL
MESSAGE', 'enqueued to msg_queue first. ');
dbms_aq.enqueue(
    queue_name => 'msg_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => message,
    msgid => message_handle);
COMMIT;
```

ORACLE

Example: Enqueue an Object Type Messages Using PL/SQL

To enqueue a single message without any other parameters specify the queue name and the payload. It is assumed that the message type `aq.Message_typ` has already been created before the queue table is created.

```
/* Enqueue to msg_queue: */
DECLARE enqueue_options dbms_aq.enqueue_options_t;
message_properties dbms_aq.message_properties_t;
message_handle RAW(16);
message aq.message_typ;
BEGIN message := message_typ('NORMAL MESSAGE', 'enqueued to msg_queue
first. ');
dbms_aq.enqueue(
    queue_name => 'msg_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => message,
    msgid => message_handle);
COMMIT;
```

Enqueue Features: Subscriptions and Recipient Lists

- **To enable consumption of a message by more than one consumer, you must:**
 - Create multiconsumer queues; and
 - Enqueue the messages into multiconsumer queues
- **A list of consumers for a message is identified using:**
 - **Subscription:**
Add a subscription to a queue using the `DBMS_AQADM.ADD_SUBSCRIBER` procedure
 - **Recipient list:**
Specified in `recipient_list` message property when the message is enqueued

ORACLE

4-10

Copyright © Oracle Corporation, 2001. All rights reserved.

Enqueue Feature: Subscriptions and Recipient List

AQ allows a single message to be processed/consumed by more than one consumer. To use this feature, you must create multiconsumer queues and enqueue the messages into these multiconsumer queues. AQ allows two methods of identifying the list of consumers for a message: subscriptions and recipient lists.

Subscriptions

- You can add a subscription to a queue by using the `AQ$_AGENT` parameter of the `DBMS_AQADM.ADD_SUBSCRIBER` procedure. You can add up to a maximum of 1024 subscribers for a multiconsumer queue.
- All consumers that are added as subscribers to a multiconsumer queue must have unique values for the `AQ$_AGENT` parameter.
- You cannot add subscriptions to single-consumer queues or exception queues.
- You can remove a subscription by using the `DBMS_AQADM.REMOVE_SUBSCRIBER` procedure. AQ will automatically remove from the `AQ$<QueueTable>` view the rows corresponding to the consumer identified by the `AQ$_AGENT` parameter.

Enqueue Features (continued)

Recipient Lists

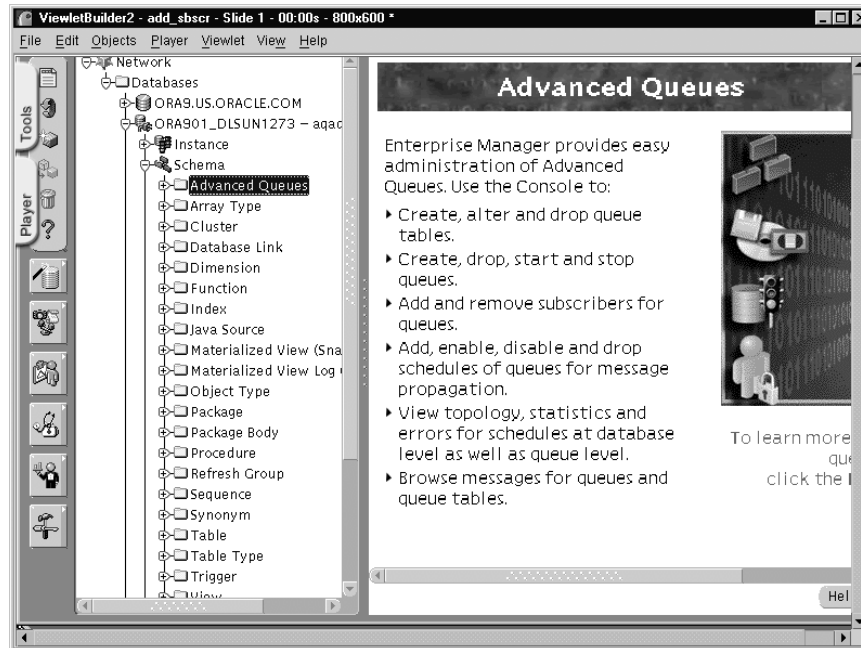
In some situations it may be desirable to enqueue a message that is targeted to a specific set of consumers rather than the default list of subscribers. You accomplish this by specifying a recipient list at the time of enqueueing the message.

You do not need to specify subscriptions for a multiconsumer queue if messages are enqueued with recipient list specified in message properties. If a recipient list is specified during enqueue, it overrides the subscription list. The consumers specified in the recipient list may or may not be subscribers for the queue. It is an error if the queue does not have any subscribers and the enqueue does not specify a recipient list

Messages that have a specified recipient list will not be available for dequeue by other subscribers of the queue.

- In PL/SQL you specify the recipient list by adding elements to the `recipient_list` field of the `message_properties` record.
- In OCI the recipient list is specified by using the `OCISetAttr` procedure to specify an array of `OCI_DTYPE_AQAGENT` descriptors as the recipient list (`OCI_ATTR_RECIPIENT_LIST` attribute) of an `OCI_DTYPE_AQMSG_PROPERTIES` message properties descriptor.

Adding Subscribers Using OEM

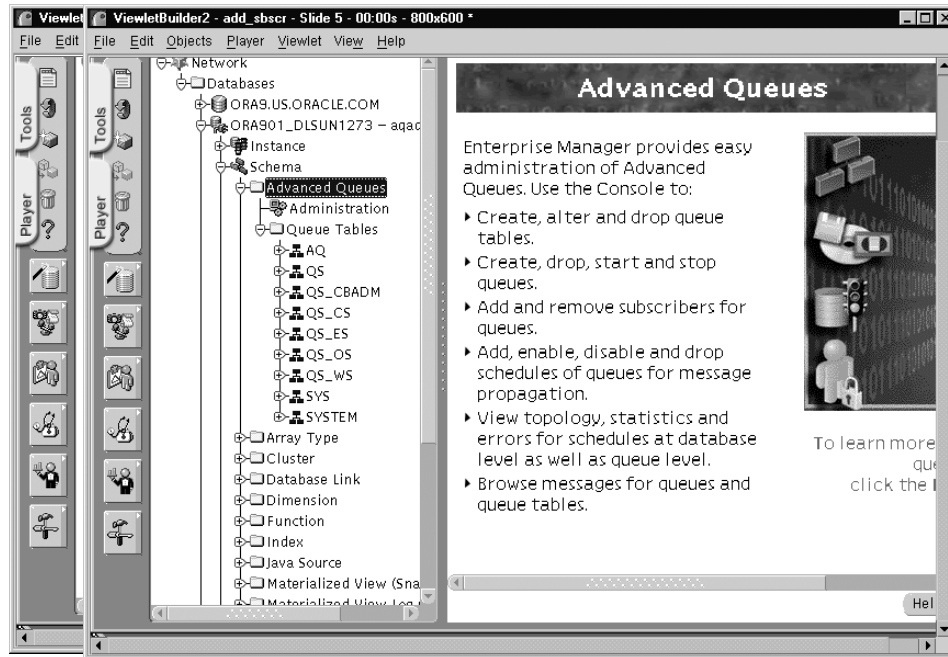


Adding Subscribers

The following steps show how to add subscribers:

1. **Connect to the database as the AQ administrator using OEM console.**
2. **Expand the Schema container.**
3. **Expand the Advanced Queues container.**
4. Expand the Queue Table container.
5. Expand the Schema container that holds the queue table you want.
6. Expand the queue table and click the correct queue. The queue property sheet appears on the right panel. Ensure that the queue table allows subscription.
7. From the queue property sheet select the Subscriber tab.
8. Enter the name of the subscriber, and any rule that you want to specify.
9. Click the Apply button.

Adding Subscribers Using OEM

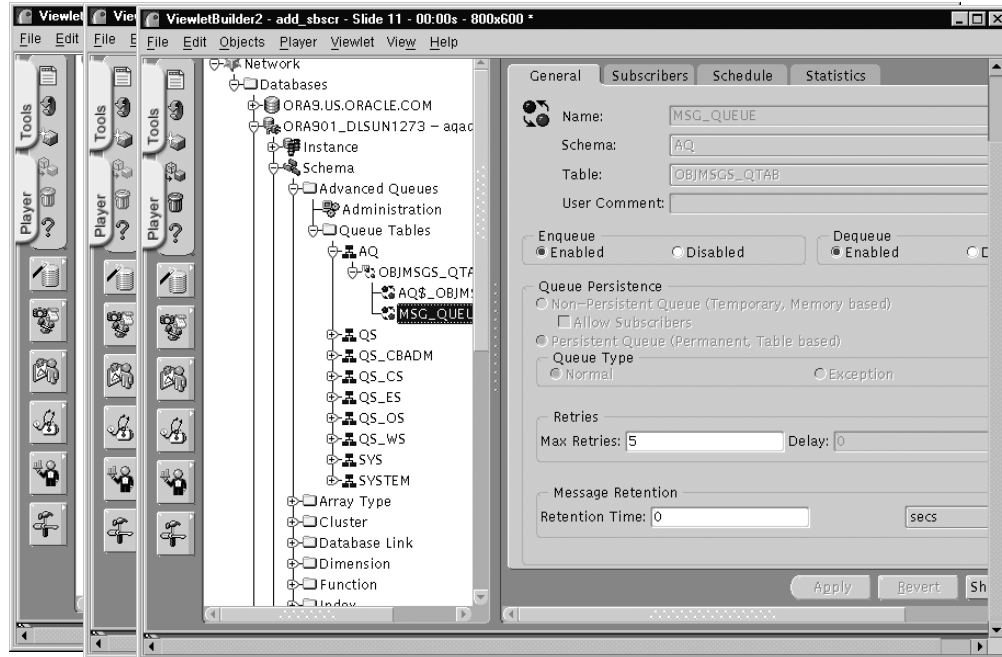


Adding Subscribers

The following path shows how to add subscribers:

1. Connect to the database as the AQ administrator using OEM console.
2. Expand the schema container
3. **Expand the Advanced Queue container.**
4. **Expand the Queue Table Container**
5. Expand the schema container that holds the queue table you want.
6. Expand the queue table and click the correct queue. The queue property sheet appears on the right panel. Ensure that the queue table allows subscription.
7. From the queue property sheet select the Subscriber Tab.
8. Enter the name of the subscriber, and any rule that you want to specify.
9. Click Apply button.

Adding Subscribers Using OEM



4-14

Copyright © Oracle Corporation, 2001. All rights reserved.

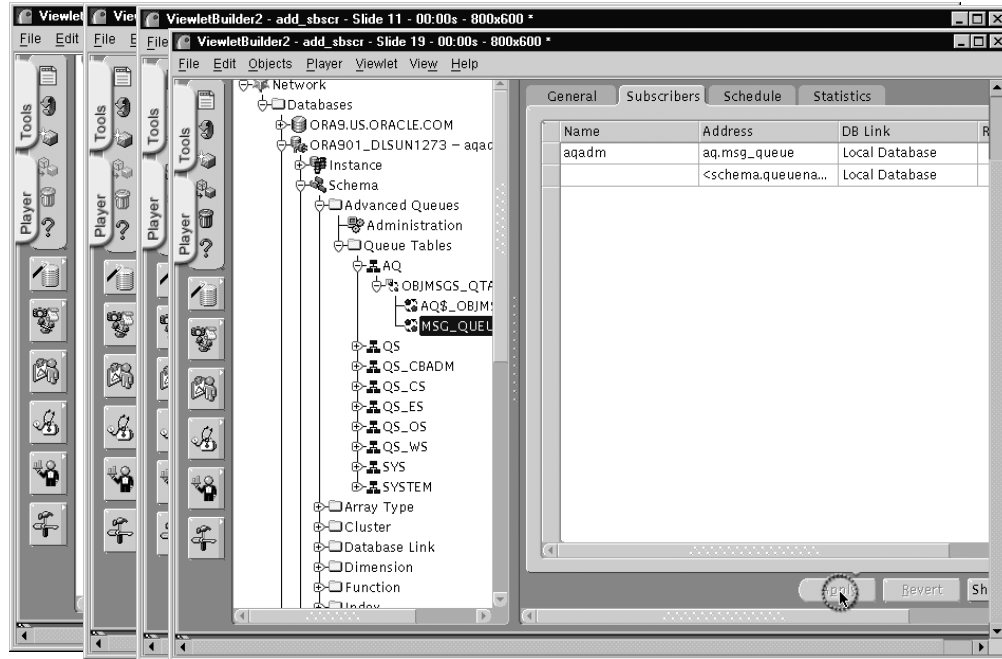
ORACLE

Adding Subscribers

The following path shows how to add subscribers:

1. Connect to the database as the AQ administrator using OEM console.
2. Expand the schema container
3. Expand the Advanced Queue container.
4. **Expand the Queue Table Container**
5. **Expand the schema container that holds the queue table you want.**
6. **Expand the queue table and click the correct queue. The queue property sheet appears on the right panel. Ensure that the queue table allows subscription.**
7. From the queue property sheet select the Subscriber Tab.
8. Enter the name of the subscriber, and any rule that you want to specify.
9. Click Apply button.

Adding Subscribers Using OEM



4-15

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE

Adding Subscribers

The following path shows how to add subscribers:

1. Connect to the database as the AQ administrator using OEM console.
2. Expand the schema container
3. Expand the Advanced Queue container.
4. Expand the Queue Table Container
5. Expand the schema container that holds the queue table you want.
6. Expand the queue table and click the correct queue. The queue property sheet appears on the right panel. Ensure that the queue table allows subscription.
7. **From the queue property sheet select the Subscriber Tab.**
8. **Enter the name of the subscriber, and any rule that you want to specify.**
9. **Click Apply button.**

Priority and Ordering of Messages

- **Message ordering dictates the order that messages are dequeued from a queue.**
- **The default order of dequeue operation from a queue is specified when a queue table is created.**
- **To dequeue in an order of priority, specify “priority,enq_time” in the `sort_list` parameter of the `CREATE_QUEUE_TABLE` procedure.**

ORACLE

4-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Priority and Ordering of Messages

The order that messages are dequeued from a queue is specified by the `sort_list` parameter when queue table is created.

Priority ordering of messages is achieved by specifying “priority, enqueue time” as the sort order for the message. If priority ordering is chosen, each message will be assigned a priority at enqueue time by the enqueuer. At dequeue time, the messages will be dequeued in the order of the priorities assigned. If two messages have the same priority, the order in which they are dequeued is determined by the enqueue time. A first in, first out (FIFO) priority queue can also be created by specifying the enqueue time, priority as the sort order of the messages.

Time Specification: Delay and Expiration

- **DELAY** indicates that the message is available for dequeue after the specified interval of time from enqueue operation.
- **EXPIRE** indicates that the message expires after the specified interval of time from enqueue operation.
- When **DELAY** time is set, the enqueued message is set in **WAIT** state for the time specified before it becomes available for dequeue.
- For **DELAY** and **EXPIRE** specifications to be effective, a background time manager should be operational.

ORACLE

4-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Time Specification: Delay and Expire

The delay interval determines when an enqueued message is marked as available to the dequeuers after the message is enqueued. The producer can also specify the time when a message expires. If the message is not consumed within that time, the message is moved to an exception queue.

When a message is enqueued with a delay time set, the message is marked in a **WAIT** state. Messages in **WAIT** state are masked from the default dequeue calls. They can be dequeued by explicitly using the message ID.

A background time-manager daemon wakes up periodically, scans an internal index for all **WAIT** state messages, and marks messages as **READY** if their delay time has passed. The time-manager will then post to all foreground processes that are waiting on queues for messages that have just been made available.

Messages can be enqueued with an expiration that specifies the interval of time the message is available for dequeuing. Note that expiration processing requires that the queue monitor be running.

Message Grouping

- **Messages belonging to a queue can be grouped to form a set that can be consumed only by one user at a time.**
- **All messages belonging to a group must be created in the same transaction.**
- **All messages created in one transaction belong to the same group.**
- **Message grouping is possible in queues created in a queue table that is enabled for transactional message grouping.**

ORACLE

4-18

Copyright © Oracle Corporation, 2001. All rights reserved.

Message Grouping

Messages belonging to a queue can be grouped to form a set that can be consumed only by one user at a time. This requires that the queue be created in a queue table that is enabled for transactional message grouping. All messages belonging to a group have to be created in the same transaction and all messages created in one transaction belong to the same group. This feature allows you to segment complex messages into simple messages.

For example, messages directed to a queue containing invoices can be constructed as a group of messages starting with the header message, followed by messages representing details, followed by the trailer message. Message grouping is also useful if the message payload contains complex large objects such as images and video that can be segmented into smaller objects.

The general message properties (priority, delay, expiration) for the messages in a group are determined solely by the message properties specified for the first message (head) of the group, irrespective of which properties are specified for subsequent messages in the group.

The message grouping property is preserved across propagation. However, it is important to note that the destination queue where messages have to be propagated must also be enabled for transactional grouping.

Dequeue Operation

- The operation of retrieving messages from a queue is known as dequeuing.
- `DBMS_AQ.DEQUEUE` procedure is used for dequeue, with one of the following options:
 - Correlation identifier, Message identifier, or Dequeue condition.
- A correlation identifier is a user-defined message property.
- A message identifier is a system-assigned value (of RAW data type).
- A dequeue condition is an expression that is similar in syntax to the `WHERE` clause of a SQL query.

ORACLE

4-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Dequeue Operation

A message can be dequeued by specifying one of the dequeue identifiers:

- Correlation identifier:
 - A correlation identifier is a user-defined message property. Multiple messages with the same correlation identifier can be present in a queue. If there are multiple messages with the same correlation identifier, the enqueue ordering of messages may not be preserved on dequeue calls.
 - A dequeue call with a correlation identifier will directly remove a message of specific interest, and the correlation identifier usually contains the most recognizable and useful attribute of a payload.
 - The correlation identifier cannot be changed between successive dequeue calls without specifying the first message navigation option.
- Message identifier:
 - A message identifier is a system-assigned value (of RAW data type). Only one message with a given message identifier can be present.
- Dequeue condition:
 - A dequeue condition is an expression that is similar in syntax to the `WHERE` clause of a SQL query. Dequeue conditions are expressed in terms of the attributes that represent message properties or message content. The messages in the queue are evaluated against the conditions and a message that satisfies the given condition is returned.

Dequeue Example

```
/* Dequeue from msg_queue: */
DECLARE
  dequeue_options
  dbms_aq.dequeue_options_t;
  message_prop
  dbms_aq.message_properties_t;
  message_handle RAW(16);
  message aq.message_typ;
BEGIN DBMS_AQ.DEQUEUE(
  queue_name => 'msg_queue',
  dequeue_options => dequeue_options,
  message_prop => message_properties,
  payload => message,
  msgid => message_handle);
  DBMS_OUTPUT.PUT_LINE('MSG:.' ||
    message.subj || ' . ' ||
message.text );
  COMMIT;
END;
```

ORACLE

An example of dequeue operation

```
/* Dequeue from msg_queue: */
DECLARE
  dequeue_options
  dbms_aq.dequeue_options_t;
  message_prop dbms_aq.message_properties_t;
  message_handle RAW(16);
  message aq.message_typ;
BEGIN DBMS_AQ.DEQUEUE(
  queue_name => 'msg_queue',
  dequeue_options => dequeue_options,
  message_prop => message_properties,
  payload => message,
  msgid => message_handle);
  DBMS_OUTPUT.PUT_LINE('MSG:.' ||
    message.subj || ' . ' || message.text );
  COMMIT;
END;
```

Modes of Dequeue

- **Dequeue modes are:**
 - **Browse, Lock, Remove, and Remove with no data**
- **If a message is browsed, it remains available for future dequeue.**
- **A locked message is skipped by a concurrent dequeue operation.**
- **If a message is consumed using remove or remove with no data mode, it is not available for future dequeue.**
- **Remove with no data mode does not fetch the payload of the message.**

ORACLE

4-21

Copyright © Oracle Corporation, 2001. All rights reserved.

Modes of Dequeueing

A dequeue request can either view a message or delete a message.

- To view a message, you can use the browse mode or locked mode.
- To consume a message, you can use remove mode or remove with no data mode.

If a message is browsed, it remains available for further processing. Similarly if a message is locked, it remains available for further processing after the lock is released by performing a transaction commit or rollback. To prevent a viewed message from being dequeued by a concurrent user, you should view the message in the locked mode.

After a message is consumed, using either of the remove modes, it is no longer available for dequeue requests.

When a message is dequeued using REMOVE_NODATA mode, the payload of the message is not retrieved. This mode can be useful when the user has already examined the message payload, possibly by means of a previous BROWSE dequeue. In this way, you can avoid the overhead of payload retrieval that can be substantial for large payloads

A message is retained in the queue table after it has been consumed only if a retention time is specified for a queue. Messages cannot be retained in exception queues

Dequeue Operation: Navigation

- Use the **NAVIGATION** parameter of the **DBMS_AQ.DEQUEUE** procedure to determine the order in which you want the messages to be dequeued.
- The default order is **NEXT MESSAGE**.
- To access a high priority message, the dequeuing application must specify **FIRST MESSAGE** in the **NAVIGATION** parameter.

ORACLE

4-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Navigating Through a Queue

Use **NAVIGATION** parameter of the **DBMS_AQ.DEQUEUE** operation to determine the sequence in which you want the messages to be dequeued.

The default **NAVIGATION** parameter during dequeue is **NEXT MESSAGE**. This means that subsequent dequeue operation will retrieve the messages from the queue based on the snapshot obtained in the first dequeue. In particular, a message that is enqueued after the dequeue command will be processed only after processing all the messages already enqueued before in the queue. This is usually sufficient when all the messages have already been enqueued into the queue, or when the queue does not have a priority-based ordering.

NEXT MESSAGE with some delay is the optimal way of processing processing AQ messages.

When the first message in the queue needs to be processed by every dequeue command applications must explicitly use the **FIRST MESSAGE** navigation option. This usually becomes necessary when a higher priority message arrives in the queue while messages already-enqueued are being processed, and the 'sort_list' parameter of the queue table is message is set to **PRIORITY,ENQ_TIME**.

Dequeue Operation: Search Criteria

- The search criteria is determined by the `consumer name`, `msgid`, `correlation`, and `condition` parameters in the dequeue options.
- Only messages in the **READY** state are dequeued unless the `msgid` is specified.
- The dequeue order is determined by the ordering criteria specified when creating the queue table.
- The dequeue order can be overridden by the `msgid` and `correlation ID` in dequeue options.
- The database-consistent read mechanism is applicable for dequeue operations.

ORACLE

4-23

Copyright © Oracle Corporation, 2001. All rights reserved.

Search Criteria

The search criteria for messages to be dequeued is determined by the consumer name, `msgid` and correlation parameters in the dequeue options. `Msgid` uniquely identifies the message to be dequeued. Correlation identifiers are application-defined identifiers that are not interpreted by AQ.

- Only messages in the **READY** state are dequeued unless a `msgid` is specified.
- The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the `msgid` and correlation id in dequeue options.
- The database consistent read mechanism is applicable for queue operations. A browse transaction will not see a message that is enqueued after the beginning of the browsing transaction.

Dequeue Operation: Message Grouping

- **Messages enqueued in the same transaction into a queue will form a group.**
- **A dequeue operation that seeks to dequeue a message from a group will lock the entire group.**
- **The application can use the `NEXT TRANSACTION` to start dequeuing messages from the next available group.**
- **In the event that no groups are available, the dequeue will time-out after the specified `WAIT` period.**

ORACLE

4-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Dequeue by Message Grouping

Messages enqueued in the same transaction into a queue that has been enabled for message grouping will form a group. If only one message is enqueued in the transaction, this will effectively form a group of one message. There is no upper limit to the number of messages that can be grouped in a single transaction.

In queues that have not been enabled for message grouping, a dequeue in `LOCKED` or `REMOVE` mode locks only a single message. By contrast, a dequeue operation that seeks to dequeue a message that is part of a group will lock the entire group. This is useful when all the messages in a group need to be processed as an atomic unit.

When all the messages in a group have been dequeued, the dequeue returns an error indicating that all messages in the group have been processed. The application can then use the `NEXT TRANSACTION` to start dequeuing messages from the next available group. In the event that no groups are available, the dequeue will time-out after the specified `WAIT` period.

Dequeue: Multiple Recipients

- A message can be retrieved by multiple recipients.
- A message is considered **PROCESSED** only when all intended consumers have successfully dequeued the message.
- In PL/SQL the consumer name is supplied using the consumer_name field of the dequeue_options_t record.
- In OCI the consumer name is supplied using the OCISetAttr procedure to specify a text string as the OCI_ATTR_CONSUMER_NAME of an OCI_DTYPE_AQDEQ_OPTIONS descriptor.
- In OO4O, the consumer name is supplied by setting the consumer property of the OraAQ object.

ORACLE

4-25

Copyright © Oracle Corporation, 2001. All rights reserved.

Multiple Recipients

A consumer can dequeue a message from a multiconsumer normal queue by supplying the name that was used in the AQ\$_AGENT type of the DBMS_AQADM.ADD_SUBSCRIBER procedure or the recipient list of the message properties.

- In PL/SQL the consumer name is supplied using the `consumer_name` field of the `dequeue_options_t` record.
- In OCI the consumer name is supplied using the `OCISetAttr` procedure to specify a text string as the `OCI_ATTR_CONSUMER_NAME` of an `OCI_DTYPE_AQDEQ_OPTIONS` descriptor.
- In OO4O, the consumer name is supplied by setting the consumer property of the OraAQ object.

Multiple processes or operating system threads can use the same `consumer_name` to dequeue concurrently from a queue. Unless the message ID of a specific message is specified during dequeue, the consumers can dequeue messages that are in the **READY** state.

Dequeue Operation: Retry with Delay Interval

- **AQ records the number of failed attempts in message history.**
- **Message history is also moved to exception queue when a message is moved to the exception queue.**

ORACLE

4-26

Copyright © Oracle Corporation, 2001. All rights reserved.

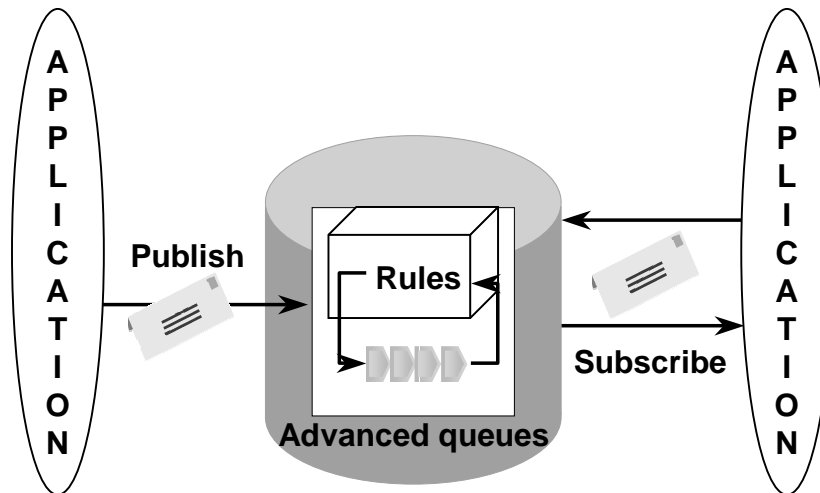
Dequeue Operation: Retry with Delay

If the transaction dequeuing the message from a queue fails, it is regarded as an unsuccessful attempt to consume the message. AQ records the number of failed attempts to consume the message in the message history. Applications can query the `retry_count` column of the queue table view to find out the number of unsuccessful attempts on a message. In addition, AQ allows the application to specify, at the queue level, the maximum number of retries for messages in the queue. If the number of failed attempts to remove a message exceeds this number, the message is moved to the exception queue and is no longer available to applications.

Retry Delay

A bad condition can cause the transaction receiving a message to abort. AQ allows users to hide the bad message for a pre-specified interval. A `retry_delay` can be specified along with maximum retries. This means that a message that has had a failed attempt will be visible in the queue for dequeue after the `retry_delay` interval. Until then it will be in the `WAITING` state. In the AQ background process, the time manager enforces the retry delay property. The default value for maximum retries is 5. The default value for retry delay is 0.

Dequeue Operation: Rule-Based Subscription



ORACLE

4-27

Copyright © Oracle Corporation, 2001. All rights reserved.

Rule Based Subscription

Messages can be routed to various recipients based on message properties or message content. Users define a rule-based subscription for a given queue to specify interest in receiving messages that meet particular conditions.

Rules are Boolean expressions that evaluate to TRUE or FALSE. Similar in syntax to the WHERE clause of a SQL query, rules are expressed in terms of the attributes that represent message properties or message content. These subscriber rules are evaluated against incoming messages and those rules that match are used to determine message recipients. This feature thus supports the notions of content-based subscriptions and content-based routing of messages.

Rules are applied to messages that are enqueued without explicit recipient list. Also, rule can be specified when adding the subscriber.

```
execute dbms_aqadm.add_subscriber ( -  
    queue_name => 'BZCARDORDERS_Q', -  
    subscriber => sys.aq$_agent('RUSH_ORDER',null,null), -  
    rule => 'tab.user_data.ord_typ = ''RUSH''');
```

Exception Handling for Multiple Recipient Queue

- A message is considered processed only if all the intended subscribers have consumed it successfully.
- A message is considered expired if one or more consumers did not dequeue the message before the expiration time.
- Expired messages are retained in exception queue.
- Message history is also moved to exception queue when a message is moved to exception queue.
- Expired messages can be dequeued only once.
- You can look at message history to determine who successfully received the message and who did not.

ORACLE

4-28

Copyright © Oracle Corporation, 2001. All rights reserved.

Multiple Recipients

A message is considered:

- Processed when all intended consumers have successfully dequeued the message.
- Expired if one or more consumers did not dequeue the message before the expiration time. When a message has expired, it is moved to an exception queue, which is also a multiple consumer type.

Expired messages from multiconsumer queues cannot be dequeued by the intended recipients of the message. However, they can be dequeued in the REMOVE mode once by specifying a NULL consumer name in the dequeue options.

All exception queues are single-consumer queues.

The queue monitor removes expired messages from multiconsumer queues. This allows dequeuers to complete the dequeue operation by not locking the message in the queue table. Because the queue monitor removes messages that have been processed by all consumers from multiconsumer queues at regular intervals, users may see a delay when the messages have been completely processed and when they are physically removed from the queue.

Exception Queue

- **Exception queue is a message property that can be specified in the enqueue operation.**
- **If an exception queue is not specified, the default exception queue is used.**
- **Messages are moved to exception queue:**
 - **If the message is not being dequeued within the specified expiration interval**
 - **If each of the unsuccessful dequeuers has exceeded the maximum number of retries**
 - **If the dequeuer is terminated or rolls back the transaction. The attempt to dequeue is, however, counted as a retry.**

ORACLE

4-29

Copyright © Oracle Corporation, 2001. All rights reserved.

Exception Queue

- The exception queue is a message property that can be specified during enqueue time. In PL/SQL users can use the `exception_queue` attribute of the `DBMS_AQ.MESSAGE_PROPERTIES_T` record to specify the exception queue. In OCI users can use the `OCISetAttr` procedure to set the `OCI_ATTR_EXCEPTION_QUEUE` attribute of the `OCIAQMsgProperties` descriptor.
- If an exception queue is not specified, the default exception queue is used. For example, if the normal queue is created in `QTAB` queue table, the default exception queue will be called `AQ$_QTAB_E`. The default exception queue is automatically created when the queue table is created.
- Messages are moved to the exception queues by AQ under the following conditions:
 - The message is not being dequeued within the specified expiration interval. For messages intended for more than one recipient, the message will be moved to the exception queue if one or more of the intended recipients was not able to dequeue the message within the specified expiration interval. The default expiration interval is `DBMS_AQ.NEVER`, meaning the messages will not expire.
 - The application that dequeues the message chooses to roll back the transaction or is terminated. In this case, the message is returned to the queue and will be available for any applications that are waiting to dequeue from the same queue.

Exception Queue (continued)

- The message is moved to the exception queue when retry counts for all recipients of the message have exceeded the specified retry limit. The default retry limit is 5 for single-consumer queues and 8.1-compatible multiconsumer queues.
- The statement executed by the client contains a dequeue that succeeded but the statement itself was undone later due to an exception. To understand this case, consider a PL/SQL procedure that contains a call to DBMS_AQ.DEQUEUE. If the dequeue procedure succeeds but the PL/SQL procedure raises an exception, AQ will attempt to increment the RETRY_COUNT of the message returned by the dequeue procedure.
- Messages cannot be dequeued by the intended recipients once the messages have been moved to an exception queue. These messages should instead be dequeued in the REMOVE or BROWSE mode by specifying a NULL consumer name in the dequeue options. The messages can also be dequeued by their message IDs.
- Users can associate a RETRY_DELAY with a queue. The default value for this parameter is 0, meaning that the message will be available for dequeue immediately after the RETRY_COUNT is incremented. Otherwise the message will be unavailable for RETRY_DELAY seconds. After RETRY_DELAY seconds, the queue monitor marks the message as READY.
- For a multiconsumer queue, RETRY_DELAY can be specified per subscriber.

Notification

- **This feature allows users or clients to receive notification of a message of interest.**
- **The client can use it to monitor multiple subscriptions.**
- **The client does not have to be connected to the database to receive notifications regarding its subscriptions.**
- **Clients should register for notification.**
- **The notification details are stored in `reg$`.**
- **The `DBMS_AQ.REGISTER` procedure is used to register for notification.**

ORACLE

4-31

Copyright © Oracle Corporation, 2001. All rights reserved.

Notification

The client can specify a callback function that is invoked for every new message that is enqueued. The callback can be a C function in an OCI client or a user-defined PL/SQL procedure. The user can also specify an email address to which notifications will be mailed.

For nonpersistent queues, the message is delivered to the client as part of the notification.

For persistent queues, only the message properties are delivered as part of the notification.

Consequently, in the case of persistent queues, the client has to make an explicit dequeue to access the contents of the message.

Clients can also specify the presentation for notifications, either RAW or XML.

Event Monitor

- **Starts up one background process per instance (EMN) during first notification**
- **Is event driven; woken up when notifications need to be delivered**
- **Tracks registrations from various clients and locations of the clients**
- **Finds clients subscribed to an event, and sends notification messages to them**
- **Keeps a pool of connections to the clients**
- **Starts off monitor process in the case of an abnormal termination of another event**

ORACLE

4-32

Copyright © Oracle Corporation, 2001. All rights reserved.

Event Monitor

- Registrations from clients are tracked in dictionary tables (reg\$ and loc\$).
- Based on the event, the interested clients and their whereabouts are looked up from reg\$ and loc\$.
- Currently, some notifications will be lost when there are more than 1024 concurrent notifications being delivered by the event monitor. This is fixed in later releases.
- When the event monitor dies, some notifications can get lost.

Listen Capability

- An application can use LISTEN to wait for messages for multiple subscriptions without having to repeatedly poll the queue.
- The LISTEN call is well suited for such applications where there are long periods with no traffic in any of the queues.
- LISTEN returns with the first agent for whom there is a message.
- To prevent one agent from “starving” other agents for messages, the application can change the order of the agents in the agent list.

ORACLE

4-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Listening for Messages

AQ can monitor multiple queues for messages with a single `LISTEN` call. An application can use `LISTEN` to wait for messages for multiple subscriptions. It can also be used by gateway applications to monitor multiple queues. If the `LISTEN` call returns successfully, a `dequeue` must be used to retrieve the message. Without the `LISTEN` call, an application which sought to `dequeue` from a set of queues would have to continuously poll the queues to determine if there were a message. Alternatively, you could design your application to have a separate `dequeue` process for each queue. However, if there are long periods with no traffic in any of the queues, these approaches will create unacceptable overhead. The `LISTEN` call is well suited for such applications.

When there are messages for multiple agents in the agent list, `LISTEN` returns with the first agent for whom there is a message. Design your application suitably to prevent one agent from “starving” other agents for messages, the application can change the order of the agents in the agent list.

Listen on Queues

- You can use PL/SQL, Java (JDBC), or OCI interfaces to listen on queues.
- In PL/SQL uses the `DBMS_AQ.LISTEN` procedure.
- You must specify queue name and agent name to listen on multiconsumer queues.
- Only local queues can be listened on.
- You cannot listen on nonpersistent queues.

```
DBMS_AQ.LISTEN(  
    Agent_list => My_agent_list,  
    Wait => 0,  
    Agent => agent_w_msg);
```

ORACLE

4-34

Copyright © Oracle Corporation, 2001. All rights reserved.

Listen on Queues

You can use the `listen` call to monitor receipt of messages on one or more queues on behalf of a list of agents. The call takes a list of agents as an argument. You specify the queue to be monitored in the address field of each agent listed. You also must specify the name of the agent when monitoring multiconsumer queues. For single-consumer queues, an agent name must not be specified. Only local queues are supported as addresses. Protocol is reserved for future use.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, only the first agent listed is returned. If there are no messages found when the wait time expires, an error is raised.

A successful return from the `listen` call is only an indication that there is a message for one of the listed agents in one the specified queues. The interested agent should dequeue the relevant message.

You cannot `LISTEN` on nonpersistent queues.

Listen on Queues (continued)

Example: Listen to Single-Consumer Queue (Timeout of Zero)

```
DECLARE Agent_w_msg aq$_agent;
My_agent_list dbms_aq.agent_list_t;
BEGIN
/* Here SCQn are single cons. queues in SCOTT's schema */
Qlist(1):= aq$_agent(NULL, 'scott.SCQ1', NULL);
Qlist(2):= aq$_agent(NULL, 'SCQ2', NULL);
Qlist(3):= aq$_agent(NULL, 'SCQ3', NULL);
/* Listen with a time-out of zero: */
DBMS_AQ.LISTEN(
  Agent_list => My_agent_list,
  Wait => 0,
  Agent => agent_w_msg);
DBMS_OUTPUT.PUT_LINE('Message in Queue :- ' || agent_w_msg.address);
DBMS_OUTPUT.PUT_LINE('');
END;
```

Listen to Mixture of Multiconsumer Queues (Timeout 100 Seconds)

```
DECLARE Agent_w_msg aq$_agent;
My_agent_list dbms_aq.agent_list_t;
BEGIN
/* NOTE: MCQ1, MCQ2, MCQ3 are multiconsumer queues in SCOTT's schema */
Qlist(1):= aq$_agent('agent1', 'MCQ1', NULL);
Qlist(2):= aq$_agent(NULL, 'scott.SQ1', NULL);
Qlist(3):= aq$_agent('agent3', 'scott.MCQ3', NULL);
/* Listen with a time-out of 100 seconds */
DBMS_AQ.LISTEN(
  Agent_list => My_agent_list,
  Wait => 100,
  Agent => agent_w_msg);
DBMS_OUTPUT.PUT_LINE('MSG in Q: ' || agent_w_msg.address || 'for ' ||
  agent_w_msg.name);
DBMS_OUTPUT.PUT_LINE('');
END; /
```

Summary

In this lesson, you should have learned how to:

- **Perform enqueue operations**
- **Use time specifications in enqueue operations**
- **Use subscriptions and recipient lists in enqueue operations**
- **Perform a dequeue operation**
- **Monitor exception queues**

ORACLE

5

Managing Propagation

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

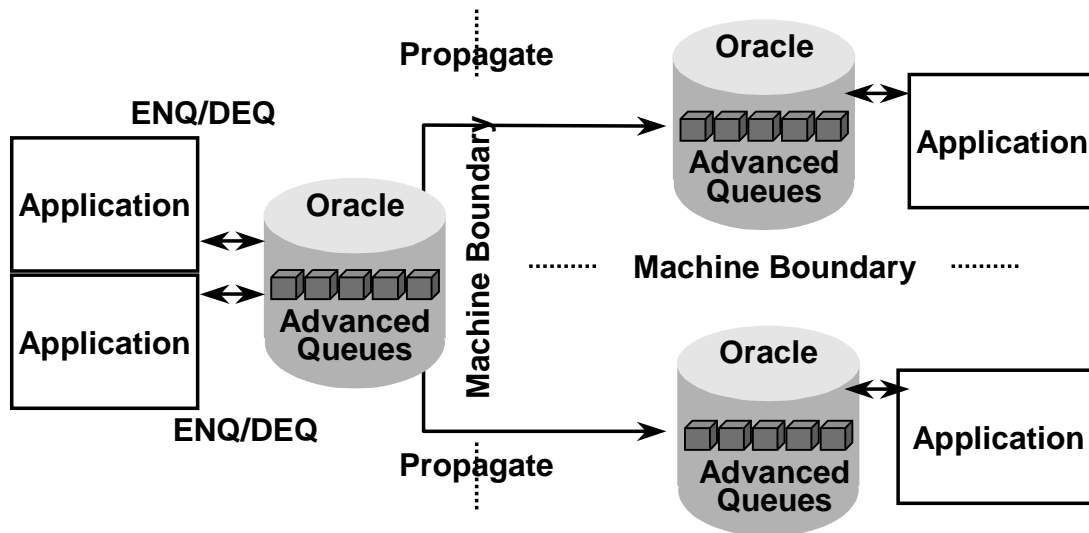
Objectives

After completing this lesson, you should be able to do the following:

- **Set up AQ message propagation**
- **Schedule AQ message propagation**
- **Verify completion of propagation**
- **Diagnose problems with AQ propagation**

ORACLE

Propagation: Overview



ORACLE

5-3

Copyright © Oracle Corporation, 2001. All rights reserved.

AQ Propagation: Overview

AQ Propagation enables applications that reside on different databases (or even different systems) to communicate with each other. Messages can be propagated from one Oracle AQ to another, irrespective of whether these are local or remote. Propagation is performed by snapshot (`job_queue_processes`) background processes. Propagation to remote queues is done using database links and OracleNet.

The propagation feature is used as follows:

- Two or more SNP background processes are initiated.
- One or more subscribers is defined for the queue from which messages are to be propagated.
- A schedule is defined for each destination where messages are to be propagated from the queue.
- Messages are enqueued, and enqueued messages are then propagated and made available for dequeuing at the destination queues.

If you want to deploy propagation across databases, you must ensure that the database link specified for the schedule is valid and has proper privileges for enqueueing into the destination queue.

Set Up Propagation

- **Set up the JOB_QUEUE_PROCESSES parameter.**
 - Use the ALTER DATABASE command.

```
ALTER DATABASE SET JOB_QUEUE_PROCESSES = 6;
```

- This starts up six SNP background processes
- **Create or add subscribers to the queue.**

```
execute dbms_aqadm.add_subscriber ( -  
    queue_name => 'BZCARDORDERS_Q', -  
    subscriber =>  
        sys.aq$_agent(null,'bzcardorders_prpq', null));
```

ORACLE

Setup Message Propagation

To schedule propagation, you should set up two or more additional job_queue background processes. Also, if you want to deploy remote propagation, you must ensure that the database link specified for the schedule is valid and has proper privileges for enqueueing into the destination queue.

Considering that the number of job queue processes is limited (36 is the maximum) and some of these may be used to handle jobs that are unrelated to propagation, propagation has built-in support for multitasking and load balancing.

The propagation algorithms are designed such that multiple schedules can be handled by a job queue process. The propagation load on a job_queue processe can be skewed based on the arrival rate of messages in the different source queues. If one process is overburdened with several active schedules whereas another is less loaded with many passive schedules, propagation automatically redistributes the schedules so that they are loaded uniformly.

Set Up Propagation

- **Set up a propagation schedule for the subscriber**

```
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(  
    Queue_name => 'AQ.MSG_QUEUE' );
```

- **Enqueue the message**
- **The job queue process will propagate the message.**

ORACLE

5-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Propagation Scheduling

A schedule indicates the time frame during which messages can be propagated from the source queue. If a queue has messages to be propagated to several queues, a schedule must be defined for each of the destination queues. The schedule therefore must be tailored for the specific source and destination. When a schedule is created, a job is automatically submitted to the job_queue facility to handle propagation.

If you intend to propagate messages across to another database, you should set up a database link, before setting up propagation schedule.

Example:

Create the database link:

- `CREATE DATABASE LINK link2q
 CONNECT TO remotq IDENTIFIED BY db4away USING linkonnet;`

Set up subscriber:

- `EXECUTE dbms_aqadm.add_subscriber (queue_name => 'BZCARDORDERS_Q', -
 subscriber => sys.aq$agent(null,'bz_prpq@link2q', null));`

Then set up propagation:

- `EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(-
 queue_name => 'OE.OE_bookedorders_que', DESTINATION => 'link2q');`

DBMS_AQADM. SCHEDULE_PROPAGATION

The **DBMS_AQADM.SCHEDULE_PROPAGATION** procedure uses the following parameters:

```
DBMS_AQADM.SCHEDULE_PROPAGATION (  
  QUEUE_NAME      IN varchar2,  
  DESTINATION     IN varchar2 default NULL,  
  START_TIME      IN date      default SYSDATE,  
  DURATION        IN number    default NULL,  
  NEXT_TIME       IN varchar2 default NULL,  
  LATENCY         IN number    default 60);
```

ORACLE

5-6

Copyright © Oracle Corporation, 2001. All rights reserved.

DBMS-AQADM.SCHEDULE_PROPAGATION

The **DBMS_AQADM.SCHEDULE_PROPAGATION** procedure uses these parameters:

- **QUEUE_NAME**: Specifies the originating queue from which message should be propagated out.
- **DESTINATION**: Specifies the name of the destination—the queue or agent to which the message should be sent.
- **START_TIME**: Specifies when the message propagation should be started.
- **DURATION**: Specifies the time frame during which propagation must take place. It is also referred to as propagation window. If the duration is unspecified, the time frame is an infinite single window.
- **NEXT_TIME** : Defines the interval between successive windows. If a window must be repeated periodically, a finite duration is specified along with a **NEXT_TIME** parameter.
- **LATENCY**: Within a propagation window, latency specifies the time interval within which a queue must be rechecked for messages. It is relevant only when a queue does not have any messages to be propagated.

The propagation schedules defined for a queue can be changed or dropped at any time during the life of the queue. In addition, there are calls for temporarily disabling a schedule (instead of dropping the schedule) and enabling a disabled schedule.

Sample Propagation Schedules

- Propagates messages every night for 2 hours

```
execute dbms_aqadm.schedule_propagation( -  
    queue_name => 'bzcardorders_q', -  
    destination => 'loopback', -  
    start_time => TO_DATE(sysdate, 'DD-MM-RR'), -  
    duration => 7200, -  
    next_time => TO_DATE(sysdate+1-(7200/86400),  
        'DD_MM_RR'));
```

- Propagates messages continuously with a 5-second

```
execute dbms_aqadm.schedule_propagation( -  
    queue_name => 'bzcardorders_q', -  
    destination => 'loopback', -  
    latency => 5);
```

ORACLE

Monitoring Propagation

- The **USER_QUEUE_SCHEDULES** dictionary view contains information about propagation.
- To get average information on propagation:

```
select avg_time, avg_number, avg_size
       from user_queue_schedules;
```

- To get maximum duration of propagation windows:

```
select max_number, max_bytes
       from user_queue_schedules;
```

- To get current status information of schedule

```
select process_name, session_id, schedule_disabled
       from user_queue_schedules;
```

ORACLE

Monitoring Propagation

Detailed information about schedules such as number of messages sent, their size, and time can be obtained from the **USER_QUEUE_SCHEDULES** view. Information such as the name of the background process handling that schedule can also be obtained from the view.

The detailed statistics that can be obtained include:

- The total number of messages propagated in a schedule
- Total number of bytes propagated in a schedule
- Maximum number of messages propagated in a window
- Maximum number of bytes propagated in a window
- Average number of messages propagated in a window
- Average size of propagated messages
- Average time to propagated a message

Exception Handling

- **Propagation reports exceptions such as**
 - Invalid database link
 - Unavailability of remote queue
- **The retry attempts use an exponential back off scheme**
- **A maximum of 16 retries are attempted after which the schedule is disabled.**
- **The errors are reported in the alert log of the initiating instance.**

ORACLE

5-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Exception Handling

Propagation has built-in support for handling failures and reporting errors. For example, if the specified database link is invalid, the remote database is unavailable, or if the remote queue is not enabled for enqueueing, then the appropriate error message is reported.

Propagation uses an exponential back off scheme for retrying propagation from a schedule that encountered a failure.

If a schedule continuously encounters failures, the first retry happens after 30 seconds, the second after 60 seconds, the third after 120 seconds and so forth. If the retry time is beyond the expiration time of the current window, the next retry is attempted at the start time of the next window. A maximum of 16 retry attempts is made, after which the schedule is automatically disabled. When a schedule is disabled automatically due to failures, the relevant information is written into the alert log.

Propagation over HTTP

- **Propagation over HTTP (or HTTPS) has a few differences over the OracleNet based propagation.**
- **When you create database link, for HTTP propagation,**
 - You should specify the protocol as HTTP (or HTTPS).
 - The user name/password combination should be that of internet user.
- **XML is used in communication**
- **AQPropServlet is deployed at the destination.**

ORACLE

5-10

Copyright © Oracle Corporation, 2001. All rights reserved.

Propagation Using HTTP

In Oracle9i, you can set up AQ propagation over HTTP and HTTPS (HTTP over SSL). HTTP propagation uses the Internet access infrastructure and requires deployment of an AQ servlet that connects to the destination database. The database link must be created with the connect string indicating the Web server address and port to which the HTTP protocol is assigned. The source database must be created for running Java and XML.

Unschedule Propagation

- Use the `DBMS_AQADM.UNSCHEDULE_PROPAGATION` procedure to unschedule a propagation.

```
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(  
    queue_name => 'aq.q1def');
```

- Two parameters are used if another database is involved.
 - **Queue_Name:** Specifies the originating queue
 - **Destination:** Specifies the scheduled destination

```
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(  
    Queue_name => 'aq.q1def',  
    Destination => 'another_db.world');
```

ORACLE

5-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Unscheduling Propagation

To unschedule a previously scheduled propagation of messages from a queue, use the `DBMS.AQADM.UNSCHEDULE_PROPAGATION` procedure. If the destination of the scheduled propagation was a database different from that of the source queue, then you can specify the `DESTINATION` parameter in the procedure. The value of the `DESTINATION` parameter would be the database link of the destination database.

Unscheduling Propagation from Queue to Other Queues in the Same Database

- `EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(queue_name => 'aq.q1def');`

Unscheduling Propagation from a Queue to other Queues in Another Database

- `EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
 Queue_name => 'aq.q1def', Destination => 'another_db.world');`

Unschedule propagation to other queues in another database using Java

- ```
public static void example(AQSession aq_sess) throws AQException {
 AQQueue queue; AQAgent agent1; AQAgent agent2;
 /* Get the queue object */
 queue = aq_sess.getQueue("AQ", "q1def");
 queue.unschedulePropagation("another_db.world"); }

```

## Enabling and Disabling Propagation Schedule

- When a propagation schedule is created, it is enabled already.
- `DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE` procedure can be used to disable a propagation schedule.

```
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
 queue_name => 'aq.q1def');
```

- `DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE` procedure can be used to enable a propagation schedule.

```
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
 queue_name => 'aq.q1def');
```

ORACLE

5-12

Copyright © Oracle Corporation, 2001. All rights reserved.

### Enabling and Disabling Propagation Schedule

Propagation schedules are enabled when they are set up. Under some circumstances, you may find it necessary to disable a propagation schedule.

To disable a propagation schedule use the `DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE` procedure. This procedure takes the source queue name as the input. When a queue from another database is involved in this propagation schedule, you should specify the database link of the destination database in the `DESTINATION` parameter.

You can perform this task using the PL/SQL or Java interface.

#### Disable Propagation Schedule From a Queue in Another Database

```
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
 Queue_name => 'aq.q1def', Destination => 'another_db.world');
```

#### Enable Propagation schedule

To re-enable, you should use the `ENABLE_PROPAGATION_SCHEDULE` procedure. You should specify the database link if the destination queue is another database.

```
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
 Queue_name => 'aq.q1def', Destination => 'another_db.world');
```

## Summary

**In this lesson, you should have learned to do the following:**

- **Set up a propagation schedule**
- **Enable propagation**
- **Enable and disable propagation schedule**
- **Distinguish steps for enabling propagation over HTTP**

ORACLE



# 6

## Managing Transformation

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

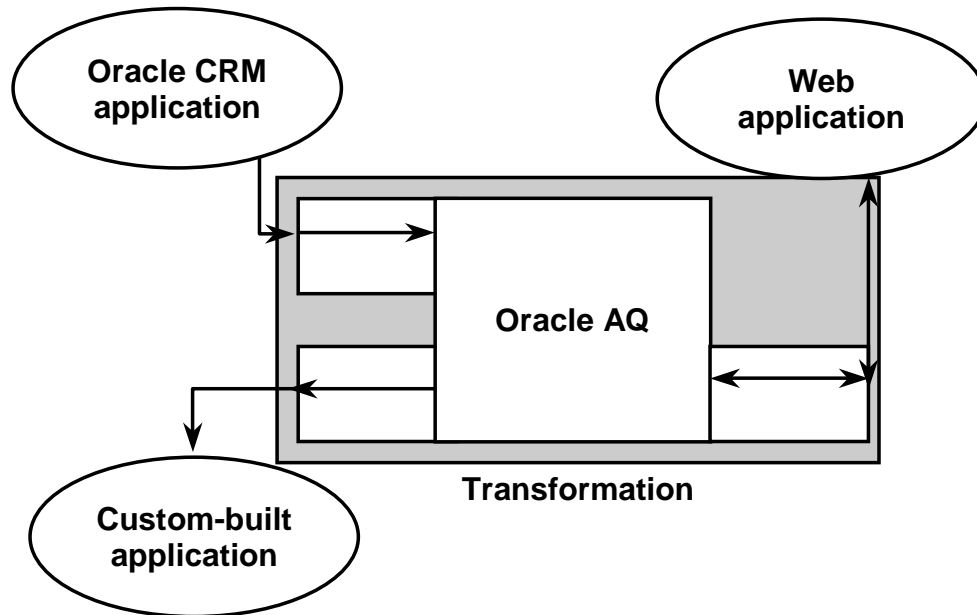
# Objectives

**After completing this lesson you should be able to do the following:**

- **Describe message transformation**
- **Create Transformation**
- **Modify Transformation**
- **Use XML and XSLT features**

ORACLE

## Message Transformation Concepts



### Message Transformation Concepts

Different applications such as Oracle CRM applications, custom-built applications, and Web applications have their own message format. For these applications to exchange messages, effectively, messages from one application should be transformed into a format recognized by the receiving application. The Oracle AQ transformation feature provides the following:

- Allows users to define the mappings from one Oracle object type to another using the DBMS\_TRANSFORM package.
- Uses the above defined mappings for AQ operations such as enqueue, dequeue, and propagation.

For example, when the Oracle CRM, custom-built, or Web application enqueues the message into an AQ queue automatically the message is transformed to the message format of the queue. Similarly at dequeue, the message is automatically transformed from the queue format to the format of the application.

## Message Transformation Concepts

- **Mappings are defined as expressions from source type to destination type**
- **Expressions can be either SQL, PL/SQL, or Java stored functions**
- **Expressions can be defined for the entire destination type or for each attribute of the destination type**

ORACLE

6-4

Copyright © Oracle Corporation, 2001. All rights reserved.

### Message Transformation Concepts

Message transformation consists of a two-step process:

1. Define the mapping between the two message formats.
2. Define a procedure (or function) to convert the message.

For example, consider a transformation routine `MODIFY_TRANSFORMATION` to transform the phone numbers from an application that uses the phone numbers in U.S format into an international phone number format. The transformation routine will append the country code for the United States (+1) to the beginning of each U.S phone number. The transformation function can be defined in two ways:

1. A separate expression specified for each attribute of the target type. This representation allows the application fine grain control over the transformation.
2. A single PL/SQL function that returns an object of the target type or the constructor of the target type. The application may prefer this representation for very simple transformations or transformations that are not easily broken down into independent transformations for each attribute.

The PL/SQL functions used for mappings can be implemented as either PL/SQL, Java, or external C callout. XSLT transformation can be defined for XML type attributes of an object type. This provides an interface for third-party transformation vendors.



## Message Transformation with AQ

- The defined mappings can be used for AQ operations: enqueue, dequeue, or propagation
- Mapping can be specified in enqueue\_options at enqueue
- Mapping can be specified with add\_subscriber or in dequeue\_options to be used for propagation (for remote subscriber) and dequeue

ORACLE

6-5

Copyright © Oracle Corporation, 2001. All rights reserved.

### Message Transformation with AQ

Because transformations and messaging often go hand in hand, it makes sense to allow the AQ messaging system to automatically invoke transformations when needed. Users can specify that transformation mapping be used at various points—enqueue, dequeue, and propagation.

- Transformation mapping for enqueue must be specified in enqueue options.
- Transformation mapping for dequeue can be specified either in dequeue options or in add\_subscriber. Mapping specified at add\_subscriber will be called every time the specified subscriber dequeues a message.
- If you specify the transformation mapping at both dequeue options and add\_subscriber, the mapping specified in dequeue options will override the mapping specified at add\_subscriber.
- If the subscriber is a queue on a different database (for propagation), the mapping will be used before propagating to the subscriber.

## Defining Message Transformations

```
create or replace function FnOrdertoShip
(ord oe.ORDER_TYP)
return Ship.SHIP_TYP is
newship ship.ship_TYP;
BEGIN
newship := SHIP_TYP(ord, 'EASTERN');
if ord.region = 'WS'
 then newBill.agent := 'WESTERN';
 end if;
return newship;
END;
```

ORACLE

### Specifying Message Transformations

In this example, you use a transformation to convert the order type to shipping type.

## XML Transformations

- Oracle9i supports a new data type 'XMLType' to efficiently handle XML data.
- Transformation mappings can be defined from Oracle object to XML and from XML to Oracle object.
- XSLT transformations can be used to transform data from one XML format to another.
- SYS\_XMLGEN, XML Parser and XML Processor functionality can be used to define XML transformations.

ORACLE

6-7

Copyright © Oracle Corporation, 2001. All rights reserved.

### XML Transformations

Example:

```
Create type Ord_xml_typ as object (orderno number, details XMLType);
```

```
create function Ord2XML (in_ord OE.Order_typ)
 return OE.Ord_xml_typ as
 xm SYS.XMLType;
 new_order OE.Ord_xml_typ;
begin
 select sys_xmlgen(in_ord) into xm from dual;
 new_order:= OE.Ord_xml_typ(in_ord.orderno,xm);
 return new_order;
```

```
End Ord2XML;/
```

```
execute DBMS_TRANSFORM.CREATE_TRANSFORMATION(
 schema => 'OE', name => 'Obj2XML',
 from_schema => 'OE', from_name => 'OE.Order_typ',
 to_schema => 'OE' to_type=>'OE.Order_xml_typ',
 transformation => 'Ord2XML(source.user_data)');
```

## DBMS\_TRANSFORM Package

The DBMS\_TRANSFORM package provides:

- An API to create, alter, and drop transformations
- An interface to invoke the transformation run-time engine.

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION
DBMS_TRANSFORM.MODIFY_TRANSFORMATION
DBMS_TRANSFORM.DROP_TRANSFORMATION
```

ORACLE

## DBMS\_TRANSFORM Package

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION (
 schema IN VARCHAR2,
 name IN VARCHAR2,
 from_schema IN VARCHAR2,
 from_type IN VARCHAR2,
 to_schema IN VARCHAR2,
 to_type IN VARCHAR2,
 transformation IN VARCHAR2 default NULL)
```

```
DBMS_TRANSFORM.DROP_TRANSFORMATION (
 schema IN VARCHAR2,
 name IN VARCHAR2)
```

ORACLE

6-9

Copyright © Oracle Corporation, 2001. All rights reserved.

### DBMS\_TRANSFORM Package

DBMS\_TRANSFORM.CREATE\_TRANSFORMATION

- Creates a transformation of the specified name that transforms an object of the source type to an object of the destination type.

PROCEDURE CREATE\_TRANSFORMATION(

```
 schema IN VARCHAR2,
 name IN VARCHAR2,
 from_schema IN VARCHAR2,
 from_type IN VARCHAR2,
 to_schema IN VARCHAR2,
 to_type IN VARCHAR2,
 transformation IN VARCHAR2 default null)
```

DBMS\_TRANSFORM.DROP\_TRANSFORMATION

- Drops a given transformation.
  - DBMS\_TRANSFORM.DROP\_TRANSFORMATION (
  - schema IN VARCHAR2,
  - name IN VARCHAR2 )

## Create Transformation

- The **DBMS\_TRANSFORM.CREATE\_TRANSFORMATION** procedure is used to create a transformation.

```
EXECUTE dbms_transform.create_transformation(-
 schema => 'aq', name => 'test_transf', -
 from_schema => 'aq', from_type => 'type1', -
 to_schema => 'aq', to_type => 'type2', -
 transformation =>
 'aq.trans_func(source.user_data)');
```

- To use this feature, you must have the **EXECUTE** privilege on:
  - The **dbms\_transform** package
  - The user-defined types, and have execute privileges
  - Any PL/SQL function used in transformation function

ORACLE

6-10

Copyright © Oracle Corporation, 2001. All rights reserved.

### Create Transformation

The **CREATE\_TRANSFORMATION** procedure creates a transformation that maps an object of the source type to an object of the target type. The transformation expression can be a SQL expression or a PLSQL function. It must return an object of the target type. The description of this procedure is:

| Parameter      | Description                                 |
|----------------|---------------------------------------------|
| schema         | Specifies the schema of the transformation. |
| name           | Specifies the name of the transformation.   |
| from_schema    | Specifies the schema of the source type     |
| from_name      | Specifies the source type.                  |
| to_schema      | Specifies the target type schema            |
| to_type        | Specifies the target type.                  |
| transformation | Specifies the transformation expression,    |

To create transformation, you must be granted execute privileges on

- **dbms\_transform** package,
- the user-defined types that are the source and destination types of the transformation,
- any PL/SQL function being used in the transformation function.

# Modify Transformation

The `MODIFY_TRANSFORMATION` procedure is used to modify an existing transformation.

```
EXECUTE dbms_transform.modify_transformation(-
 schema => 'aq', name => 'test_transf', -
 attribute_number 0, -
 transformation =>
 'aq.trans_func(source.user_data)');
```

ORACLE

6-11

Copyright © Oracle Corporation, 2001. All rights reserved.

## Modify Transformation

To change the transformation function and to specify transformations for an attribute of the target type use the `DBMS_TRANSFORM.MODIFY_TRANSFORMATION` procedure.

If the attribute number 0 is specified, then the transformation expression singularly defines the transformation from the source to target types. You should be granted execute privileges on `dbms_transform`. You must also have execute privileges on the user-defined types that are the source and destination types of the transformation, and have execute privileges on any PL/SQL function being used in the transformation function.

| Parameter                 | Description                                                                                                  |
|---------------------------|--------------------------------------------------------------------------------------------------------------|
| schema                    | Specifies the schema of the transformation                                                                   |
| name                      | Specifies the name of the transformation.                                                                    |
| attribute_number          | Must be zero for the entire target type.<br>(1 for the first attribute, 2 for second ...)                    |
| transformation_expression | A SQL expression or a PLSQL function<br>returning the type of the specified attribute of the<br>target type. |

## Specifying Transformations

- **DBMS\_AQ.enqueue\_options\_t** and **DBMS\_AQ.dequeue\_options\_t** object types include an attribute called transformation.
- The **DBMS\_AQADM.ADD\_SUBSCRIBER** procedure allows a transformation to be specified for a subscription.

```
execute DBMS_AQADM.ADD_SUBSCRIBER(
 queue_name => 'QS_WS.QS_WS_backorders_q',
 subscriber => subscriber,
 transformation => 'QS_WS_trans_backorders');
```

ORACLE

6-12

Copyright © Oracle Corporation, 2001. All rights reserved.

### Specifying Transformations

The example shown above creates a queue subscriber for the QS\_WS\_backorders\_q queue, specifying that the QS\_WS\_trans\_backorders transformation is to be used by default when retrieving messages from a queue.

Following are the modifications which have been made to DBMS\_AQ and DBMS\_AQADM.

TYPE enqueue\_options\_t IS RECORD(  
 • visibility            BINARY\_INTEGER NOT NULL DEFAULT ON\_COMMIT  
 • relative\_msgid        RAW(16) DEFAULT NULL,  
 • sequence\_deviation    BINARY\_INTEGER DEFAULT NULL,  
 • transformation        VARCHAR2(60) DEFAULT NULL);

TYPE dequeue\_options\_t IS RECORD(  
 • consumer\_name        VARCHAR2(30) DEFAULT NULL,  
 • dequeue\_mode         BINARY\_INTEGER NOT NULL DEFAULT REMOVE,  
 • navigation           BINARY\_INTEGER NOT NULL DEFAULT NEXT\_MESSAGE,  
 • visibility           BINARY\_INTEGER NOT NULL DEFAULT ON\_COMMIT,  
 • wait                 BINARY\_INTEGER NOT NULL DEFAULT FOREVER,  
 • msgid                RAW(16) DEFAULT NULL,  
 • correlation          VARCHAR2(128) DEFAULT NULL,  
 • transformation        VARCHAR2(60) DEFAULT NULL);



## Specifying Transformations

The transformation field in `enqueue_options_t` can be used to specify transformation at enqueue.

```
create procedure FwdOrd2Ship(booked_order
OE.order_typ)
 enq_opt dbms_aq.enqueue_options_t;
begin
 enq_opt.transformation := 'OE.OE2WS';
 dbms.aq.enqueue(
 queue_name => 'WS.ws_orders_topic',
 enqueue_options => enq_opt,
 payload => booked_order);
end;
/
```

ORACLE

## Specifying Transformations

The transformation field in `dequeue_options_t` can be set to specify transformation at dequeue time.

```
create procedure FwdOrd2Ship(booked_order
OE.order_typ)
 deq_opt dbms_aq.dequeue_options_t;
 booked_order WS.order_typ_sh;
begin
 deq_opt.transformation := 'OE.OE2WS';
 dbms_aq.dequeue(
 queue_name => 'WS.ws_orders_topic',
 dequeue_options => enq_opt,
 payload => booked_order);
end;
/
```

ORACLE

# Managing Transformations

Use `DBA_TRANSFORMATIONS` and `USER_TRANSFORMATIONS` to monitor.

|                                  |                              |
|----------------------------------|------------------------------|
| <b>transformation_id</b>         | <b>NUMBER</b>                |
| <b>owner</b>                     | <b>VARCHAR2(30) NOT NULL</b> |
| <b>name</b>                      | <b>VARCHAR2(30) NOT NULL</b> |
| <b>namespace</b>                 | <b>INTEGER</b>               |
| <b>from_type_schema</b>          | <b>VARCHAR2(30)</b>          |
| <b>from_type_name</b>            | <b>VARCHAR2(30)</b>          |
| <b>to_type_schema</b>            | <b>VARCHAR2(30)</b>          |
| <b>to_type_name</b>              | <b>VARCHAR2(30)</b>          |
| <b>type</b>                      | <b>TYPE</b>                  |
| <b>attribute_name</b>            | <b>VARCHAR2(30)</b>          |
| <b>transformation_expression</b> | <b>VARCHAR2(4000)</b>        |
| <b>comment</b>                   | <b>VARCHAR2(4000)</b>        |

ORACLE

6-15

Copyright © Oracle Corporation, 2001. All rights reserved.

## Data Dictionary Views

Two new data dictionary views have been added to assist in managing transformations:

`DBA_TRANSFORMATIONS`

`USER_TRANSFORMATIONS`

The difference between the two views is that `USER_TRANSFORMATIONS` does not contain the owner column, and only list transformations owned by the current schema user.

`DBA_TRANSFORMATIONS`

- `Transformation_id`: The transformation identifier
- `Name`: The name of the transformation
- `Owner`: The owner of the transformation
- `Namespace`: This is set to 1 for Oracle transformations. Transformations from third party are in different namespaces.
- `From_type_schema`: The schema of the source type.
- `From_type_name`: The name of the source type.
- `To_type_schema`: The schema of the target type.
- `To_type_name`: The name of the target type.
- `Type`: The type of transformation: PL/SQL, SQL, or XSL.
- `Attribute_name`: The destination attribute name
- `Transformation_expression`: The transformation expression.

## Managing Transformations

- **Transformations can be exported and imported:**
  - All transformations are exported during a full database export
  - Transformations owned by a user are exported during a schema level export
- **If an AQ queue table is exported, the transformations corresponding to the queue table will also be exported.**

ORACLE

6-16

Copyright © Oracle Corporation, 2001. All rights reserved.

### Managing Transformations

Transformations will be exported on a full database export. Transformations owned by a user will also be exported on a schema level export. Transformations used by AQ will also be exported when the corresponding queue table is exported.

In both queue table export and schema level export, the source and target ADTs are also exported and created on import, if they don't already exist.

## Summary

**In this lesson, you should have learned how to:**

- **List the message transformation types that are available**
- **Create a transformation mapping**
- **Transform messages using the `DBMS_TRANSFORM` package.**

ORACLE





# AQ Notifications

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

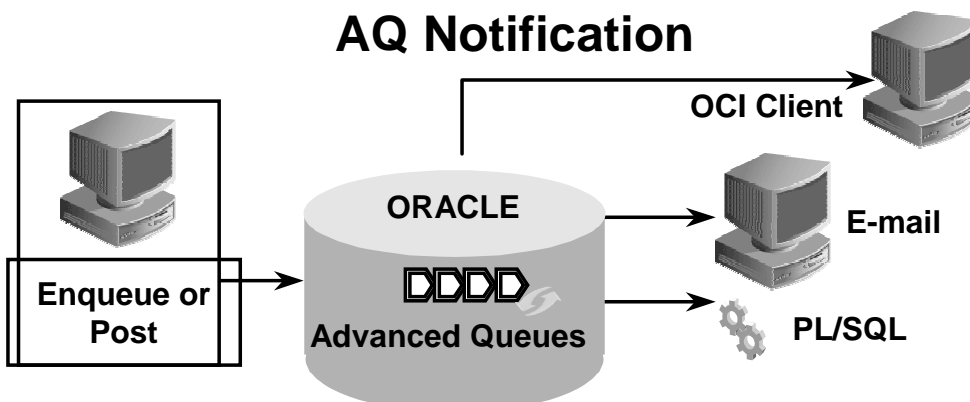
# Objectives

**At the end of this lesson, you should be able to:**

- **Describe Notification feature of AQ**
- **Setup AQ Notification**
- **Describe AQ Notification over Internet.**
- **Describe AQ notification at an Email address.**

ORACLE





The AQ Notification feature enables enterprises to provide

- Automatic execution of business logic
- E-mail notification to a set of users
- Notification of database events such as
  - Startup or shutdown of a database
  - Creation or dropping of a schema or table
  - Database access by a specific user

### AQ Notification: Overview

AQ notification allows a client (application or agents) to receive notification of a message of interest. The client can use it to monitor multiple subscriptions. The client does not have to access a queue or be connected to the database to receive notifications regarding its subscriptions.

AQ supports notifications that can be received in a PL/SQL procedure, an OCI callback function, a JMS onMessage callback, an email notification, and an HTTP post. You can register for notifications using any of the APIs such as PL/SQL, OCI, Java (JMS), or IDAP (XML-based API for AQ). There are two steps to use notifications:

- Register for notifications: Users can register for notifications through OCI, PL/SQL, or IDAP (for event registration over the Internet) APIs.
- Receive notifications: Notifications can be an OCI Callback function, a PL/SQL procedure, an e-mail, or an HTTP post. The presentation for notification can be either RAW or XML. Only XML presentation is supported for Oracle object type payload, and e-mail and HTTP post notifications. Message ID and message properties are sent for notifications. Message payload is also sent for nonpersistent queues.

## AQ Notification Database Events

- **AQ users can register for notification of database events such as**
  - Startup or shutdown of a database
  - Creating or dropping a schema or table
  - Database access by a specific user
- **To enable database event notification, you should set up corresponding triggers in the database.**
- **Triggers should contain the command to enqueue or post to the relevant queue.**
- **A client can register for "global events" that apply to multiple databases. Such registrations can be stored in the Oracle Internet Directory.**

ORACLE

7-4

Copyright © Oracle Corporation, 2001. All rights reserved.

### AQ Notification

AQ users can register for notification of database events such as startup, shutdown, creation or dropping of schema, table, and so on. By registering such events in AQ, users can be notified of the events through AQ.

To enable database event notification, corresponding triggers for the database event should be written. In the body of the trigger, a message should be enqueued to the appropriate queue. On the message enqueue, the subscribers to the queue are notified of the event automatically.

If notifications for events from multiple databases are involved, then the trigger mechanism has to be set up in each database that participates in a notification.

The postoperation is used as mechanism for notification to nonpersistent queues. In postoperation the message is delivered to the client as a part of the notification.

Where persistent queues are involved, only the message properties are delivered as part of the notification. The client must make an explicit dequeue to access the contents of the message.

Clients can also specify the presentation for notifications, either RAW or XML messages.

## Registering for Notifications

**AQ Notification is defined by the AQ\$\_REG\_INFO object type. It contains the following attributes:**

- **NAME:** Specifies the schema qualified name of the subscription
- **NAMESPACE:** For receiving subscriptions from AQ queues, the namespace must be `DBMS_AQ.NAMESPACE_AQ`; for other applications it should be `DBMS_AQ.NAMESPACE_ANONYMOUS`.
- **CALLBACK:** Specifies the function to be invoked on receipt of notification
- **CONTEXT:** Specifies the contextual parameters that are needed by the callback function

ORACLE

7-5

Copyright © Oracle Corporation, 2001. All rights reserved.

### Registering for Notifications

AQ Notifications are defined by the object type `SYS.AQ$_REG_INFO`. This object type has four attributes.

|           |                |
|-----------|----------------|
| NAME      | VARCHAR2(128)  |
| NAMESPACE | NUMBER         |
| CALLBACK  | VARCHAR2(4000) |
| CONTEXT   | RAW(2000)      |

- **NAME:** Defines the name of subscription to notification. It is of the form `<schema>.<queue>` for single consumer queue, and `<schema>.<queue>.<consumer>` for multiconsumer queue. For postoperation, it is just the name.
- **NAMESPACE:** Defines the namespace for the subscription. It must be
  - `DBMS_AQ.NAMESPACE_AQ` for notifications from AQ queues
  - `DBMS_AQ.NAMESPACE_ANONYMOUS` for non AQ queues used with `DBMS_AQ.POST`, or other applications using `OCISubscriptionPost()`.
- **CALLBACK:** Defines the function to be performed on receipt of message notification
  - For e-mail and HTTP post it is of the form `mailto://xyz@company.com`
  - For PL/SQL with raw payload it is `plsql://<schema>.<procedure>?PR=0`
  - For XML payload, it is `plsql://<schema>.<procedure>?PR=1`
- **CONTEXT:** Specifies the parameters to be passed to `CALLBACK` function

## Register for Notification PL/SQL Procedure

- The **DBMS\_AQ.REGISTER** procedure is used to
  - Register an e-mail address for message notification
  - Invoke a user-defined PL/SQL procedure on message enqueue
- The **DBMS\_AQ.UNREGISTER** procedure unregisters a subscription and turns off notifications to the subscriber.
- The **DBMS\_AQ.POST** procedure posts to a list of subscriptions to get notifications.
- Several subscriptions can be registered or unregistered at one time with the use of the **reglist** option.

ORACLE

7-6

Copyright © Oracle Corporation, 2001. All rights reserved.

### The DBMS\_AQ Package Procedures Manage AQ Registrations

Use the REGISTER procedure to register an e-mail address or call a user-defined PL/SQL procedure for message notification. Interest in several subscriptions can be registered at one time. Following is an example of registering two PL/SQL procedures:

```
DECLARE
 reginfo sys.aq$_reg_info;
 reginfo1 sys.aq$_reg_info;
 reginfo1list sys.aq$_reg_info_list;
BEGIN
 reginfo := sys.aq$_reg_info('U1.EVENTS:ADMIN', DBMS_AQ.NAMESPACE_AQ,
 'plsql://plsqlregprocl', HEXTORAW('FF'));
 reginfo1 := sys.aq$_reg_info('U1.EVENTS:ADMIN', DBMS_AQ.NAMESPACE_AQ,
 'plsql://plsqlregprocl?PR=1', HEXTORAW('FF'));
 reginfo1list := sys.aq$_reg_info_list(reginfo);
 reginfo1list.EXTEND;
 reginfo1list(2) := reginfo1;
 sys.dbms_aq.register(reginfo1list, 2);
commit;
END;
```

## Receive Notifications

- **Notifications can be received in**
  - PL/SQL procedure
  - OCI callback function
  - An e-mail
  - HTTP post
- **Notifications from nonpersistent queues contain message payload along with it.**
- **Queue-based notifications contain only the message ID and message properties.**
- **Presentation for Oracle object payload is XML.**
- **Presentation for RAW payloads is RAW or XML.**

ORACLE

7-7

Copyright © Oracle Corporation, 2001. All rights reserved.

### Receive Notifications

Following is an example of a PL/SQL procedure registered for notification:

```
<PL/SQL Notification Function> (
 context IN RAW, -- context provided during registration
 reginfo IN sys.aq$_reg_info,
 descr IN sys.aq$_descriptor,
 payload IN RAW | VARCHAR,
 payloadl IN NUMBER -- length of the payload
)
```

Consumer name, message ID, and message properties are returned in AQ\$\_DESCRIPTOR. Payload for from nonpersistent queues is received along with the notifications. It would be of VARCHAR for Oracle object type queues, or if the presentation is specified as XML; and RAW for RAW type payloads.

## PL/SQL Procedure

Consumer name, message ID, and message properties are returned in AQ\$\_DESCRIPTOR.

An example of the PL/SQL notification procedure:

```
create or replace procedure plsqrnotif (
 context raw,
 reginfo sys.aq$_reg_info,
 descr sys.aq$_descriptor, payload raw, payloadl number)
AS
 dequeue_options DBMS_AQ.dequeue_options_t;
 message_prop DBMS_AQ.message_properties_t;
 message_hdl RAW(16);
 message BZ_ADM.bzcardorder_typ;
BEGIN
 -- get the consumer name and message id from the descriptor
 dequeue_options.msgid := descr.msg_id;
 dequeue_options.consumer_name := descr.consumer_name;
 -- dequeue the message
 DBMS_AQ.DEQUEUE(queue_name => descr.queue_name,
 dequeue_options => dequeue_options,
 message_properties => message_prop,
 payload => message,
 msgid => message_hdl);
 commit;
END
```

## OID Support for AQ

- **LDAP client can register for events that apply to multiple databases. Such a registration is referred to as Global Registration.**
- **OID can be used as the repository for event registration.**
- **JMS queue/topics, Connection factories, and their subscriptions can be looked up in OID.**
- **OID can be used to look up AQ Aliases, Agents, and queues.**
- **OID can be used to modify registered events.**

ORACLE

7-9

Copyright © Oracle Corporation, 2001. All rights reserved.

### OID Support for AQ

In Oracle9i, the registration via LDAP is enabled. This will allow registration of an LDAP client so they can receive notifications when the specified queues are affected.

A client can register for events that apply to multiple databases in an enterprise. For example, the client can register for a DATABASE OPEN event through LDAP. When the database of interest comes back up, it queries the LDAP server and picks up the new registrations.

Also all the users who have registered for the concerned queue are notified that the database is being started up.

## Register for AQ Notifications in OID: Setup

- **Prerequisites:**

- Client should be an enterprise user.

- Database parameters:

- `COMPATIBLE = 9.0 (or higher)`

- `LDAP_REG_ENABLED = "TRUE"`

- **Setup procedure:**

- In each enterprise domain create `ENTERPRISE_AQ_USER_ROLE` in OID.

- Add `GLOBAL_AQ_ROLE` to the `ENTERPRISE_AQ_USER_ROLE` in the database.

- Add `ENTERPRISE_AQ_USER_ROLE` to privilege group `cn=OracleDBAUsers`.

- Grant `ENTERPRISE_AQ_USER_ROLE` to AQ users.

ORACLE

7-10

Copyright © Oracle Corporation, 2001. All rights reserved.

### Open Registration Steps

Prerequisites:

- Registering through LDAP (open registration) requires the client to be an enterprise user. (For more information on enterprise user management refer to the section titled “Managing Enterprise User Security” in the Oracle9i Advanced Security Guide.)

- The `COMPATIBLE` database parameter has to be 9.0 or higher

- The `LDAP_REG_ENABLED` parameter must be set to true. This can be changed in operating database instance using the command

- `ALTER SYSTEM SET ldap_reg_enabled = 'TRUE';`

- `REG_SYNC_INTERVAL` parameter must be set to the time interval (in seconds) to refresh registrations from LDAP. This can be changed in operating database instance using the command

- `ALTER SYSTEM SET reg_sync_interval = Number of Seconds`



## **Open Registration Steps (continued)**

### **Register AQ notifications in OID: Manual Setup**

The following steps must be performed before the open registration feature can be used:

1. In each enterprise domain, create the `ENTERPRISE_AQ_USER_ROLE` enterprise role using Oracle Enterprise Security Manager (OESM). This role is stored in the OID, and is applicable to all the databases in the domain.
2. For each database in the enterprise domain, add the `GLOBAL_AQ_USER_ROLE` global role to the `ENTERPRISE_AQ_USER_ROLE` enterprise role using OESM.
3. For each enterprise domain, add the `ENTERPRISE_AQ_USER_ROLE` enterprise role to privilege group `cn=OracleDBAQUUsers` under `cn=oraclecontext` in the administrative context.
4. For each enterprise user that is authorized to register for events in the database, grant the `ENTERPRISE_AQ_USER_ROLE` enterprise role to it using OESM.

## Register for Notification in OID Using OCI

- Execute `OCIInitialize()` with `OCI_EVENTS` or `OCI_USE_LDAP` mode.
- Execute `OCIAttrSet()` to set the environment handle attributes for accessing OID.
- Execute `OCIHandleAlloc()` to allocate a subscription handle.
- Execute `OCIDescriptorAlloc()` to allocate a server DN descriptor.
- Execute `OCIAttrSet()` to set the server DN descriptor.
- Execute `OCIAttrSet()` to set the subscription handle.
- Execute `OCISubscriptionRegister()` to register with the subscriptions.

ORACLE

7-12

Copyright © Oracle Corporation, 2001. All rights reserved.

### Register for Notification in OID Using OCI

Steps to register via LDAP using OCI:

- Execute `OCIInitialize()` with `OCI_EVENTS` or `OCI_USE_LDAP` mode to specify that the application is interested in registering for notifications via LDAP.
- Execute `OCIAttrSet()` to set the environment handle attributes for accessing LDAP:
  - `OCI_ATTR_LDAP_HOST`: the host name on which the LDAP server resides
  - `OCI_ATTR_LDAP_PORT`: the port on which the LDAP server is listening
  - `OCI_ATTR_BIND_DN`: the distinguished name to login to the LDAP server; usually the DN of the enterprise user
  - `OCI_ATTR_LDAP_CRED`: the credential used to authenticate the client
  - `OCI_ATTR_WALL_LOC`: for SSL authentication, the location of the client wallet
  - `OCI_ATTR_LDAP_AUTH`: the authentication method code, (for a complete list of authentication modes see Appendix A, "Handle and Descriptor Attributes" in the *Oracle9i Application Developer Guide – Advanced Queuing*)
  - `OCI_ATTR_LDAP_CTX`: the administrative context for oracle in the LDAP server
- Execute `OCIHandleAlloc()` with handle type `OCI_HTYPE_SUBSCRIPTION` to allocate a subscription handle.
- Execute `OCIDescriptorAlloc()` with descriptor type `OCI_DTYPE_SRVDN` to allocate a server DN descriptor.

## Register for Notification in OID Using OCI (continued)

- Execute OCIAttrSet() to set the server DN descriptor attributes for:
  - OCI\_ATTR\_SERVER\_DN: the distinguished name of the database in which the client is interested receiving notifications;
  - OCIAttrSet() can be called multiple times for this attribute so that more than one database server is included in the registration.
- Execute OCIAttrSet() to set the subscription handle attributes for:
  - OCI\_ATTR\_SUBSCR\_NAME: subscription name
  - OCI\_ATTR\_SUBSCR\_NAMESPACE: subscription namespace
  - OCI\_ATTR\_SUBSCR\_CALLBACK: notification callback
  - OCI\_ATTR\_SUBSCR\_CTX: callback context
  - OCI\_ATTR\_SUBSCR\_PAYLOAD: payload buffer for posting
  - OCI\_ATTR\_SUBSCR\_RECPT: recipient name
  - OCI\_ATTR\_SUBSCR\_RECPTPROTO: protocol to receive notification
  - OCI\_ATTR\_SUBSCR\_RECPTRES: presentation to receive notification with
  - OCI\_ATTR\_SUBSCR\_SERVER\_DNs: the descriptor handle we populated in step 5.
- Execute OCISubscriptionRegister() to register with the subscription(s). The registration takes effect when the database comes to LDAP to pick up new registrations, whose frequency is determined by the value of REG\_SYNC\_INTERVAL.

## Registering for Notification at an E-mail Address: Example

```
<?xml version="1.0"?>
<Envelope xmlns=
"http://www.oracle.com/schemas/IDAP/envelope">
<Body>
 • <AQXmlRegister xmlns =
 "http://www.oracle.com/schemas/AQ/access">
 – <register_options>
 – <destination>AQUSER.EMP_TOPIC</destination>
 – <consumer_name>APP1</consumer_name>
 – <notify_url>mailto:app1@hotmail.com</notify_url>
 – </register_options>
 – <AQXmlCommit/>
 • </AQXmlRegister>
</Body>
</Envelope>
```

ORACLE

7-14

Copyright © Oracle Corporation, 2001. All rights reserved.

### Register Request Example

Registering for notification at an e-mail address:

To notify an e-mail address of messages enqueued for consumer APP1 in queue AQUSER.EMP\_TOPIC, the register request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://www.oracle.com/schemas/IDAP/envelope">
<Body>
<AQXmlRegister xmlns = "http://www.oracle.com/schemas/AQ/access">
<register_options>
<destination>AQUSER.EMP_TOPIC</destination>
<consumer_name>APP1</consumer_name>
<notify_url>mailto:app1@hotmail.com</notify_url>
</register_options>
<AQXmlCommit/>
</AQXmlRegister>
</Body>
</Envelope>
```

## Summary

**In this lesson, you have learned how to:**

- **Setup AQ notification**
- **Use AQ notification**
- **Describe AQ notification over Internet.**
- **Describe AQ notification at an Email address.**

ORACLE



# 8

## Configuring AQ with Internet Access

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

## Objectives

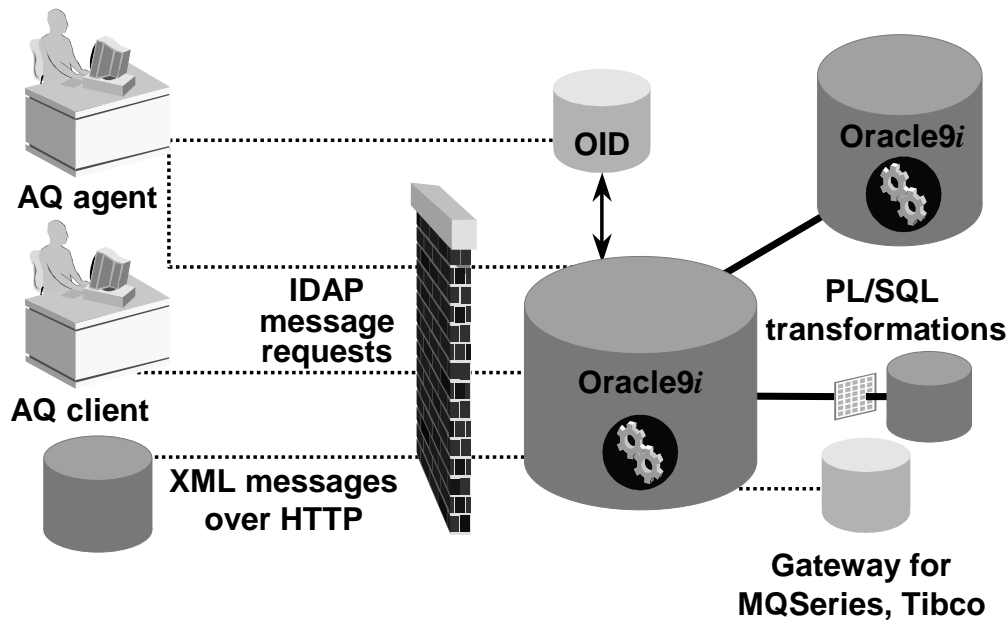
**After completing this lesson, you should be able to do the following:**

- **Describe how to use IDAP to transmit AQ messages over the Internet**
- **Describe how an Internet user is authorized to access Oracle AQ**
- **Set up AQ to use HTTP or HTTPS**
- **Set up AQ to use e-mail**
- **Describe the structure of XML payloads under IDAP**
- **Describe AQ message propagation between Oracle databases using HTTP**
- **Describe the use of Messaging gateways**

**ORACLE**



## Overview



### Overview of Internet Integration

Oracle9i provides a standard infrastructure that makes it easier for customers to integrate storefronts, exchanges and portals with other backend and external systems. This allows Oracle9i to support a vast number of messages in industry standard business messaging formats.

To enable messaging operations (such as enqueue and dequeue) to be performed across the Internet, a new XML-based Internet Document Access Presentation (IDAP) has been created by Oracle that allows message operations to be sent through HTTP across firewalls.

Advanced Queuing agents can now also be defined in Oracle Internet Directory, providing a centralized, easy to manage, secure infrastructure for global messaging. In addition, AQ information can be externalized in Oracle Internet Directory, providing a single place for the configuration and management of Advanced Queuing operations that span one or more systems, such as subscribing to queues across multiple systems.

Enterprise application integration may involve legacy applications and/or customer sites where non-Oracle products are widely used. It is not feasible to convert all customer applications to use Oracle AQ or RDBMS entirely. For this reason, Oracle9i provides a Messaging Gateway which allows customer applications to remain unchanged and which seamlessly integrates connectivity between AQ and other messaging systems.

## Internet Integration: Requirements

- **B2B applications on the Internet require that**
  - Messages work with firewalls and proxies
  - The protocol used not be proprietary
- **Oracle has evolved an XML-based protocol called Internet Data Access Presentation (IDAP).**
- **Security mechanisms for Internet usage have also been incorporated.**
- **Oracle AQ offers two models for communication over the Internet:**
  - Internet client communication
  - Message propagation over the Internet

ORACLE

8-4

Copyright © Oracle Corporation, 2001. All rights reserved.

### Internet Integration: Requirements

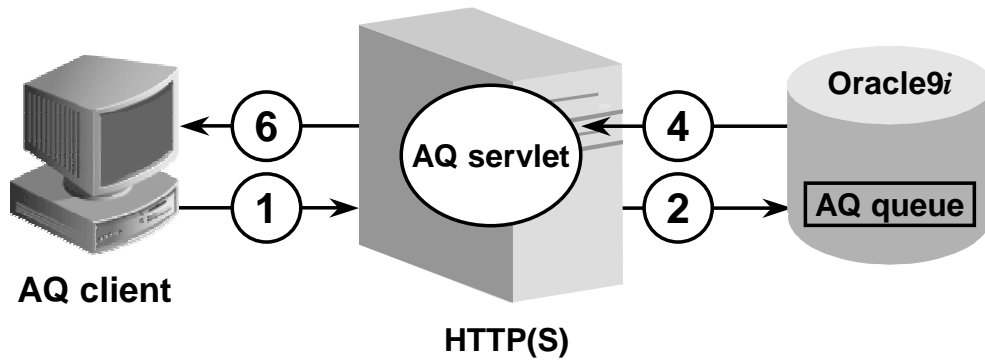
Currently client-to-server and server-to-server communication of Oracle database uses the Oracle Net protocol. Being proprietary, it is faced with difficulty for B2B applications for transporting messages between organizations. Another limitation is the inability to transmit SQL\*Net messages through a firewall.

A simple and open protocol that demands very little by way of prerequisites at the client end is needed. It should also work with firewalls and proxies. Considering this, Oracle has evolved an XML-based Internet Data Access Presentation (IDAP) , a standard protocol to make B2B communications over the Internet possible. In addition, authentication and authorization mechanisms have also been built in to Advanced Queuing to support Internet-based users.

The two models offered by AQ for communication over the Internet are:

- Internet client communication
- Message propagation

## Internet Client Communication



### Internet Client Communication

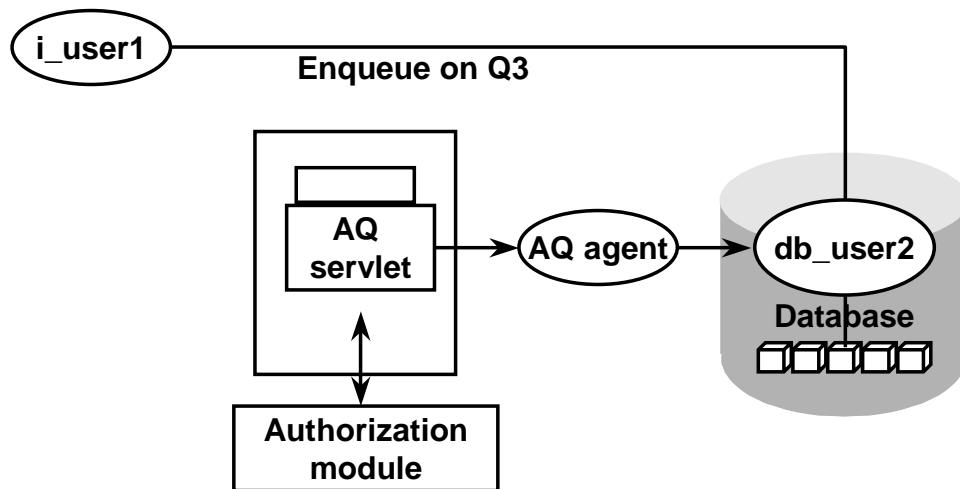
An Internet client application can communicate over HTTP and e-mail with an Oracle AQ using Internet Data Access Protocol (IDAP). IDAP is XML-based protocol and is the Simple Object Access Protocol (SOAP) specification for AQ operations.

The broad steps in Internet client communication are:

1. Internet client sends IDAP request to AQ servlet through HTTP using HTTP Post.
2. Internet client authenticates to the Web server.
3. AQ Server parses the XML messages and connects to the database server using a database user proxy.
4. Database performs authentication and authorization checks database user proxy.
5. Database sends response to AQ servlet.
6. AQ servlet sends the response to the Internet client.

At the first AQ operation, a cookie is dropped with the database session information. Subsequent operations use the session info if the session remains active.

## Accessing AQ as an Internet User: Example



### Accessing AQ as an Internet User: Example

Consider an Internet user `i_user1`. Inside the authorization module, we have associated `i_user1` with database users `db_user1` and `db_user2`, using the authorization module functions. User `db_user1` has privileges to enqueue and dequeue on `q1` and `q2`. User `db_user2` has enqueue and dequeue privileges on `q2` and `q3`.

If `i_user1` issues an enqueue request on `q3`, the AQ servlet obtains the appropriate database username from the Authorization module function. In this case, the function would return the username `db_user2` because it is the only username in the `i_user1` privilege set that is allowed to enqueue on `q3`. The AQ servlet would then proxy as this database username when issuing the enqueue command.

If a request arrived for enqueue on `q2`, the Authorization module could return either `db_user1` or `db_user2` because both have privileges on `q2`.

If a request is issued on a queue that neither `db_user1` or `db_user2` has privileges on, the Authorization module signals an authorization error.

# What Is IDAP

- **IDAP is the SOAP specification for AQ operations.**
- **The IDAP document encapsulates transactions, security, transformation, and character set IDs for the requests.**
- **Third-party clients can enqueue and dequeue messages, if they have HTTP support and the ability to decode XML messages.**

ORACLE

8-7

Copyright © Oracle Corporation, 2001. All rights reserved.

## Internet Protocol Support

An Internet Data Access Presentation (IDAP) document encapsulates AQ operation request and response in XML. To perform AQ operations—enqueue, dequeue, send notifications, register for notifications, propagation, and so on—over the Internet standard transport (HTTP(S) and e-mail), IDAP is used. In addition, IDAP encapsulates transactions, security, transformation, and character set IDs for the requests.

The IDAP uses the content-type of `text/xml` to specify the body of the request containing an XML-encoded method request. XML provides the presentation for IDAP request and response messages as follows:

- All protocol tags are scoped to the IDAP namespace.
- The sender includes namespaces in IDAP elements and attributes.
- The receiver processes IDAP messages that have correct namespaces; for requests with incorrect namespaces, the receiver returns an invalid request error.
- The receiver processes IDAP messages without namespaces as though they had the correct namespaces, if the context is valid.
- The IDAP namespace has the value `http://ns.oracle.com/AQ/schemas/envelope`
- An XML document forming the request of an IDAP invocation does not require the use of an XML DTD or a schema.

## **Internet Protocol Support (continued)**

The XML message that is propagated will contain the actual payload in XML format along with meta information present in the header which is also represented in XML format.

All of these types of queues can still be accessed by way of the Internet. If the queue holds messages in RAW or Oracle Object (ADT) format, the XML payloads are first transformed to the appropriate internal format and then stored. During dequeue, if the messages are obtained from the queue in RAW or ADT format, they are converted to XML before sending them to the client.

# IDAP Message Structure

**IDAP structures a message request or response as follows:**

- **The IDAP envelope is the root element that defines the namespace of an XML message**
- **The IDAP body**
  - **Contains the AQ method request and its parameters**
  - **Must contain an AQ XML document to perform AQ operations**

ORACLE

8-9

Copyright © Oracle Corporation, 2001. All rights reserved.

## IDAP Message Structure

IDAP messages are structured into IDAP Envelope, IDAP header, and IDAP body.

The IDAP envelope:

- Is the root element of the XML message, and defines the namespace of the message. In Oracle9i the namespace is  
`<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">`  
In Oracle9iAS, you can also use  
`<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope">`
- Also contains additional attributes, provided they are namespace-qualified. Additional namespace-qualified subelements can follow the body.

The IDAP body:

- Defines the method request and its parameters. AQ methods are defined in the namespace `http://ns.oracle.com/AQ/schemas/access`. The body of an IDAP message is an AQ XML document which represents:
  - Client requests for enqueue, dequeue, and registration for notification
  - Server responses to client requests
  - Notifications from server to clients
- IDAP client requests also have tags for commit and rollback of a transaction.

## IDAP Message

- A message set consists of one or more messages.
- Each message consists of a message\_header and message\_payload
- An AQ message is provided as an argument to the send/publish call.
- A message consists of the following:
  - Message number
  - Message header
  - Message payload

ORACLE

8-10

Copyright © Oracle Corporation, 2001. All rights reserved.

### AQ Message

MESSAGE\_Set: A Message\_Set consists of one or more messages.

MESSAGE: A Message consists of the following:

- Message Number: The sequence number of the message in the Message\_Set. This information is useful for the client when logging in.
- Message Header: The Message Header comprises the following:
  - MESSAGE\_ID: The identifier of the message
  - CORRELATION: The correlation identifier of the message
  - EXPIRATION: The duration in seconds that a message is available for dequeuing
  - DELAY: Duration in seconds after which a message is available for processing
  - PRIORITY: The priority of the message. A smaller number indicates higher priority.
  - SENDER\_ID: The application-specified identifier, agent\_name, address, protocol agent\_alias—if specified, resolves to a name, address, protocol using LDAP
  - RECIPIENT\_LIST: Overrides the default subscriber list; lookup\_type defines if the recipients are specified or looked up in LDAP
  - MESSAGE\_STATE: The state of the message is filled in automatically during dequeue
  - EXCEPTION\_QUEUE: In case of exceptions the name of the queue to which the message is moved if it cannot be processed successfully
  - MESSAGE\_PAYLOAD The payload of the message is of type ANY and will be interpreted at run time.



## IDAP Methods

- **There are two types of AQ methods in IDAP:**
  - **AQ access methods:**  
**SEND, PUBLISH, RECEIVE, REGISTER, COMMIT, ROLLBACK**
  - **AQ status response:**  
**Response from the AQ**
- **Attributes can be specified for each method**
- **Three classes of options:**
  - **Producer (for SEND/PUBLISH operations)**
  - **Consumer (for RECEIVE operations)**
  - **Register (for notification registration)**

ORACLE

8-11

Copyright © Oracle Corporation, 2001. All rights reserved.

### AQ Message Methods

AQ methods can be classified into two areas, namely:

AQ access methods:

- **SEND:** Enqueue a message from a client to a single consumer queue
- **PUBLISH:** Enqueue a message from a client to a multi consumer queue
- **RECEIVE:** Dequeue a message from a client to a single consumer queue
- **REGISTER:** Register for notifications on a queue or topic
- **COMMIT:** Commits the transaction
- **ROLLBACK:** Rolls back the transaction

Status response:

- Status Response following an Publish request
- Status Response following a Receive request
- Status Response following an error

## IDAP Message Options: Producer

- **Indicate options associated with send/publish method calls.**
- **Destination: Queue**
- **Visibility: When the message is visible**
- **Transformation: The transformation function (PL/SQL) to be invoked before enqueue**

ORACLE

8-12

Copyright © Oracle Corporation, 2001. All rights reserved.

### AQL Message Options: Producer

Producer options indicate options associated with enqueue operations. The following parameters are provided through the Producer options:

- **Destination:** The queue to which messages are to be sent (enqueued). It includes a `lookup_type` attribute that defines whether the queue is specified directly or must be looked up through LDAP.
- **Visibility:** Specifies the transactional behavior of the Send (enqueue) request. Visibility can be either `ON_COMMIT` or `IMMEDIATE`.
- **Transformation:** The PL/SQL transformation to be invoked before the message is enqueued.

## IDAP Message Options: Consumer

- Indicates options associated with receive method calls
- Destination: Source Queue
- Consumer\_name
- Selector: Message\_ID, Correlation or <expression>
- Visibility
- Dequeue\_mode
- Navigation
- Transformation

ORACLE

8-13

Copyright © Oracle Corporation, 2001. All rights reserved.

### AQL Message Options: Consumer

Consumer options indicate options associated with dequeue operations. The following parameters are provided through the Consumer options:

- Destination: The queue from which messages are to be received (dequeued). It includes a `lookup_type` attribute defines whether the queue must be specified directly or looked up through LDAP.
- Consumer\_Name: The name of the consumer
- Selector: Indicates the selection criterion to be used for dequeuing a message. This can be one of the following:
  - Message\_ID: Specifies the message identifier of the message to be dequeued
  - Correlation: Specifies the correlation identifier of the message to be dequeued

### **AQL Message Options: Consumer (continued)**

– <expression>: A condition is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query).

Message properties include priority, corrid and other columns in the queue table

- **Visibility:** Specifies whether the message is dequeued as part of the current transaction. Visibility can be either ON\_COMMIT or IMMEDIATE.
- **Dequeue\_mode:** Specifies the mode in which the dequeue is performed (The locking behavior associated with the dequeue). The possible values of dequeue\_mode are BROWSE, LOCKED, REMOVE.
- **Navigation:** Specifies the position of the method that will be retrieved. The possible values are NEXT\_MESSAGE, NEXT\_TRANSACTION, and FIRST\_MESSAGE.
- **Transformation:** The PL/SQL transformation to be invoked after the message is dequeued

## IDAP Message Options: Register

- Indicates options associated with events registration
- Destination
- Notify\_URL

ORACLE

8-15

Copyright © Oracle Corporation, 2001. All rights reserved.

### AQL Message Options: Register

Register options indicate options associated with event notification registration. The following parameters are provided through the Register Options:

- Destination: The queue to which the consumer is registering for notification. It includes a `lookup_type` attribute defines whether the queue must be specified directly or looked up through LDAP.
- Notify\_URL: Where notification is sent when a message is enqueued. The form can be `http://<url>` or `mailto://<e-mail address>` or `plsql://<pl/sql procedure>`.

## IDAP Messages: Response

- **AQ uses Response Messages to return the result of an AQ method.**
- **Each Response consists of a Status Response and possibly also one of the following:**
  - **Send\_result**
  - **Publish\_result**
  - **Receive\_result**

ORACLE

8-16

Copyright © Oracle Corporation, 2001. All rights reserved.

### AQ Response Messages

The response to an enqueue request to a single-consumer queue uses the AQXmlSendResponse method. The components of the response may include:

#### Status Response

- **status\_code**—indicates success (0) or failure (-1)
- **error\_code**—Oracle code for the error
- **error\_message**—description of the error

#### Send\_result (for an Enqueue to a Single-Consumer Queue)

- **destination**—where the message was sent
- **message\_id**—identifier for every message sent

#### Publish\_result (for an Enqueue to a Multiconsumer Queue or Topic)

- **destination**—where the message was sent
- **message\_id**—identifier for every message sent

#### Receive\_result (for a Dequeue from a Queue or Topic)

- **destination**—where the message was sent
- **message\_set**—the set of messages dequeued

## **AQ Response Messages (continued)**

The response to a Register request uses the AQXmlRegisterResponse method, which consists of status\_response.

The response to a Commit request uses the AQXmlCommitResponse method, which consists of status\_response.

The response to a Rollback request uses the AQXmlRollbackResponse method, which consists of status\_response.

In case of an error in any of the above requests, a FAULT is generated. The FAULT element consists of:

- faultcode: Error code for fault
- faultstring: Indicates a client error or a server error. A client error means that the request is not valid. Server error indicates that the AQ servlet has not been set up correctly detail, which consists of status\_response

## Setting Up Internet Access for AQ

**Summarized steps required to set up Internet Access for AQ queues:**

- **Set up AQ servlet:** Create a servlet that extends the class `oracle.AQ.xml.AQxmlServlet`.
- **Set up user authentication:** Configure the Web server to authenticate all the users that send POST requests to the AQ servlet.
- **Set up user authorization:** Register AQ agent used to perform AQ operations using the `DBMS_AQADM.CREATE_AQ_AGENT` procedure.
- **Map the AQ agent to the database users using the `DBMS_AQADM.ENABLE_DB_ACCESS` procedure.**

ORACLE

8-18

Copyright © Oracle Corporation, 2001. All rights reserved.

### Set Internet Access for AQ

Following are the summarized steps required to set up Internet Access for AQ queues:

Set up AQ Servlet:

- Create a servlet that extends `oracle.AQ.xml.AQxmlServlet` class. Implement `init()` method in the servlet to specify database connection parameters

Set up User Authentication:

- Configure the Web server to authenticate all the users that send POST requests to the AQ servlet. Only authenticated users are allowed to access the AQ servlet.

Set up User Authorization:

- Register AQ agent name which will be used to perform AQ operations using `DBMS_AQADM.CREATE_AQ_AGENT`. Map the AQ agent to the database users using `DBMS_AQADM.ENABLE_DB_ACCESS`.

Now clients can write IDAP requests and send by way of HTTP POST to the AQ servlet.



## IDAP: An Example

- This example shows an Order Entry message in its IDAP format.
- As you can see the main component is the body comprises
  - Producer options
  - Message set
  - Message payload

ORACLE

8-19

Copyright © Oracle Corporation, 2001. All rights reserved.

### AQ Communication Using HTTP(S)

An enqueue request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns="http://www.oracle.com/schemas/IDAP/envelope">
 <Body>
 <AQXmlSend xmlns = "http://www.oracle.com/schemas/AQ/access">
 <producer_options>
 <destination>OE.OE_NEW_ORDERS_QUE</destination>
 </producer_options>
 <message_set>
 <message_count>1</message_count>
 <message>
 <message_number>1</message_number>
 <message_header>
 <correlation>XML_ADT_SINGLE_ENQ</correlation>
 <sender_id>
 <agent_name>john</agent_name>
 </sender_id>
 </message_header>
 <message_payload>
```

## AQ Communication Using HTTP(S) (continued)

```
<ORDER_TYP>
<ORDERNO>100</ORDERNO>
<STATUS>NEW</STATUS>
<ORDERTYPE>NORMAL</ORDERTYPE>
<ORDERREGION>EAST</ORDERREGION>
<CUSTOMER>
<CUSTNO>1001233</CUSTNO>
<CUSTID>JOHN</CUSTID>
<NAME>AMERICAN EXPRESS</NAME>
<STREET>EXPRESS STREET</STREET>
<CITY>REDWOOD CITY</CITY>
<STATE>CA</STATE>
<ZIP>94065</ZIP>
<COUNTRY>USA</COUNTRY>
</CUSTOMER>
<PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
<ITEMS>
<ITEMS_ITEM>
<QUANTITY>10</QUANTITY>
<ITEM>
<TITLE>Perl</TITLE>
<AUTHORS>Randal</AUTHORS>
<ISBN>ISBN20200</ISBN>
<PRICE>19</PRICE>
</ITEM>
<SUBTOTAL>190</SUBTOTAL>
</ITEMS_ITEM>
</ITEMS>
<CCNUMBER>NUMBER01</CCNUMBER>
<ORDER_DATE>2000-08-23 0:0:0</ORDER_DATE>
</ORDER_TYP>
</message_payload>
</message>
</message_set>
</AQXmlSend>
</Body>
</Envelope>
```

# Security Using HTTP

**Security is considered under two aspects:**

- **Authorization**
  - Provided by the Web server
  - Oracle9iAS uses SSL
- **Authentication is performed in one of two ways:**
  - Username and password-based authentication
  - Digital certificate-based authentication

ORACLE

8-21

Copyright © Oracle Corporation, 2001. All rights reserved.

## Security Using HTTP

Authorization for HTTP propagation is provided by the Web server.

- For Oracle9iAS, security will be provided by the SSL (Secure Socket Layer) protocol. It handles the details of security management using public key cryptography to encrypt all client/server communication.
- SSL includes support for server certificates only. It provides authentication of the server, confidentiality and integrity. Server authentication requires that the server obtains a digital certificate from a certificate authority (CA) and install it on its site. The digital format used in SSL is X.509
- Here's how server authentication works:
  - A user connects to the secure site using HTTPS
  - The server sends its X.509 certificate (which has the server's public key) to the client
  - The client application checks to see whether a trusted certificate authority signed the key. If one didn't, the client asks the user if the key can be trusted and proceeds as directed.
  - The client generates an asymmetric key for the session, which is encrypted with the server's public key and sent back to the server. This new key is used to encrypt all subsequent transactions.

## **Security Using HTTP (continued)**

Client authentication is done using SSL also, thus after the client has authenticated the server, the server establishes the client's identity. You can accomplish this in one of the following ways:

- Username and password-based authentication: The user is required to enter a username and password which is encrypted and then sent to the server over SSL. The server checks the users credentials against its userid and password database and then grants access to the user. Note that because all of the data is going over an SSL connection, it cannot be stolen during transmission. Also, no malicious server can successfully pretend to be the original Web server in an attempt to steal the user's password.
- Digital certificate-based authentication: In the SSL certificate-based client authorization model, the client must obtain its own digital certificate from an external CA and install it on the client machine (or possibly a smartcard). The browser then sends the client's X.509 certificate to the server. The server checks this certificate from a library of existing certificates or from a certificate authority (CA) to verify the client's identity and then grants appropriate access to the user.

# Authorization Functions

**To create and manage Internet users, use the following:**

- **DBMS\_AQADM.CREATE\_AQ\_AGENT**
- **DBMS\_AQADM.ALTER\_AQ\_AGENT**
- **DBMS\_AQADM.DROP\_AQ\_AGENT**

```
DBMS_AQADM.CREATE_AQ_AGENT(
 agent_name IN VARCHAR2,
 certificate_location IN VARCHAR2 DEFAULT NULL,
 enable_http IN BOOLEAN DEFAULT FALSE,
 enable_smtp IN BOOLEAN DEFAULT FALSE,
 enable_anyp IN BOOLEAN DEFAULT FALSE)
```

ORACLE

8-23

Copyright © Oracle Corporation, 2001. All rights reserved.

## **DBMS\_AQADM.CREATE\_AQ\_AGENT**

The parameter description is as follows:

**agent\_name:** Specifies the username of the Internet user. This function enforces the restriction that two Internet users cannot share the same username, and will throw an error if this username has already been created with a previous call to **CREATE\_AQ\_USER**.

**Certificate\_location:** Will be used later for authentication purposes. Presently ignored.

**Enable\_http:** Specifies whether the agent should be enabled for HTTP access.

This procedure will throw an error if none of the protocol access methods are enabled.

**Enable\_smtp:** Specifies whether the agent should be enabled for SMTP access.

This procedure will throw an error if none of the protocol access methods are enabled.

**Enable\_anyp:** Specifies whether the agent should be enabled for all access types.

If this parameter is set to **TRUE**, all other enable inputs are ignored.  
This procedure will throw an error if none of the protocol access methods are enabled.

## **DBMS\_AQADM.CREATE\_AQ\_AGENT (continued)**

Example:

```
DBMS_AQADM.CREATE_AQ_AGENT(
agent_name => 'KURT', enable_http => TRUE, enable_smtp => TRUE,
certificate_location =>
'ldap://ldaphst:389 cn=kurt,cn=OracleDBAgents,cn=db1,cn=acme,cn=com');
```

# Authorization Functions

To manage database privileges for Internet users, use the following:

- **DBMS\_AQADM.ENABLE\_DB\_ACCESS**
- **DBMS\_AQADM.DISABLE\_DB\_ACCESS**

```
DBMS_AQADM.ENABLE_DB_ACCESS(
 agent_name IN VARCHAR2,
 db_username IN VARCHAR2)
```

```
DBMS_AQADM.DISABLE_DB_ACCESS(
 agent_name IN VARCHAR2,
 db_username IN VARCHAR2)
```

ORACLE

8-25

Copyright © Oracle Corporation, 2001. All rights reserved.

## DBMS\_AQADM.ENABLE\_DB\_ACCESS and DISABLE\_DB\_ACCESS

### Parameter Description

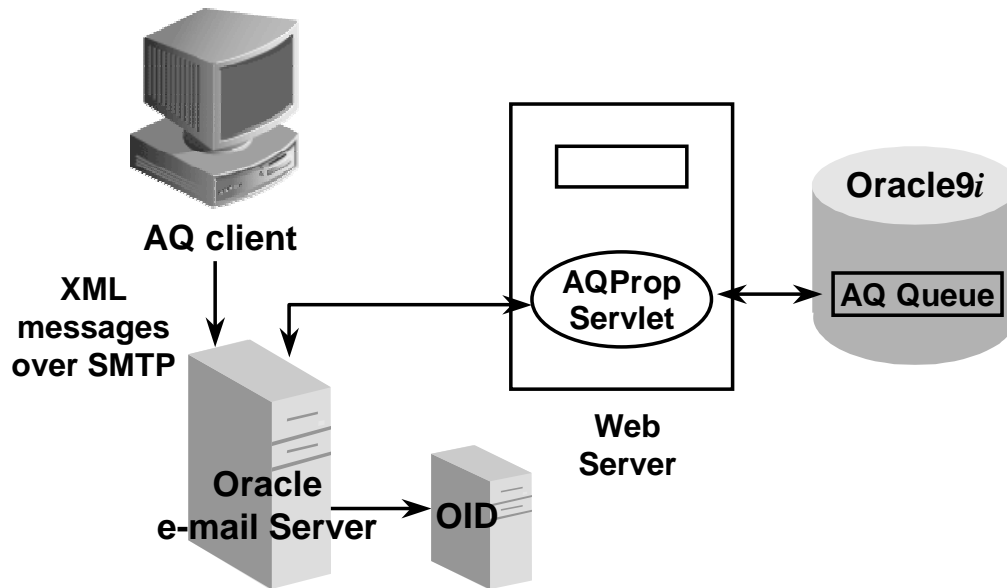
**agent\_name** Specifies the username of the Internet user to be granted or to lose the privileges of a database username. This procedure will throw an error if this username has not already been created with a previous call to **CREATE\_AQ\_AGENT**.

**db\_username** This is the database username that the Internet user will add to or remove from his or her privilege set. This procedure will throw an error if this username does not exist.

### Examples :

```
DBMS_AQADM.ENABLE_DB_ACCESS(agent_name => 'KURT', db_username=>
'AQ_USER_ENQ');
DBMS_AQADM.ENABLE_DB_ACCESS(agent_name => 'KURT', db_username=>
'AQ_USER_DEQ');
```

## AQ Communication Using E-mail



### To Use SMTP Protocol:

- The client constructs the request (send, receive, or register) in the specified XML format and signs it.
- The message is sent to the database's e-mail address <database-name>@<domain-name>
- The e-mail server receives the message, parses it, and sends an HTTP POST message containing the client request.
- The AQ Servlet is invoked to process the request.
- A response is then sent back to the client as an XML formatted e-mail.



## Setting Up AQ to Use E-mail

- These operations are supported by Oracle E-mail Server 5.5 and above.
- In addition steps as defined in the set for Internet Access for AQ perform the following steps:
  - Set up Web server: Configure Web server to receive requests from `ORACLE_SMTP_AGENT`.
  - Set up Oracle e-mail server:
    - Create an AQ schema on the e-mail server database.
    - Create an e-mail account for the destination database.
    - Set up an e-mail rule for the destination database.
- Now clients can write IDAP requests and send them by e-mail to AQPropServlet.

ORACLE

8-27

Copyright © Oracle Corporation, 2001. All rights reserved.

### Setting up AQ to Use E-mail

Currently, these operations are supported by Oracle E-mail Server 5.5 and above. Follow the steps as defined in the set for Internet Access for AQ. In addition, perform the following steps:

- Set up the Web server: Configure the Web server to receive requests from a user called `ORACLE_SMTP_AGENT`. This user is used to access the AQPropServlet. In addition also specify `sete-mailServerAddr` or `sete-mailServerHost` in `init()` method of AQPropServlet.
- Set up Oracle E-mail Server:
  - Run `$ORACLE_HOME/admin/e-mailrule.sql` to create an AQ schema on the e-mail server database.
  - Create an e-mail account for the destination database (database in which the AQ operations are to be performed)
  - Set up an e-mail rule for the destination database, so that it can route the AQ requests to the AQ Servlet on the Web server.
- Now clients can write IDAP requests and send them by e-mail to AQ servlet.

## AQ Privileges for Internet Users

- The privileges for an Internet user can be represented as the union of privileges of several database users.
- The Internet user is registered for Internet access in the database by using `DBMS_AQADM.CREATE_AQ_AGENT`.
- Each AQ agent is then granted mapped to one or more database users using `DBMS_AQADM.ENABLE_DB_ACCESS`.
- AQ privileges are granted to the database users.

ORACLE

## Managing Internet Users

- The AQ\$INTERENT\_USERS data dictionary view can be used to manage Internet users.
- It displays the database usernames associated with an agent name and their protocols.

Agent_name	db_username	http_en	smtp_en
scott	system	YES	NO
scott	buyer	YES	NO
Kurt	Aq_user_enq	YES	YES
Kurt	Aq_user_deq	YES	YES

ORACLE

8-29

Copyright © Oracle Corporation, 2001. All rights reserved.

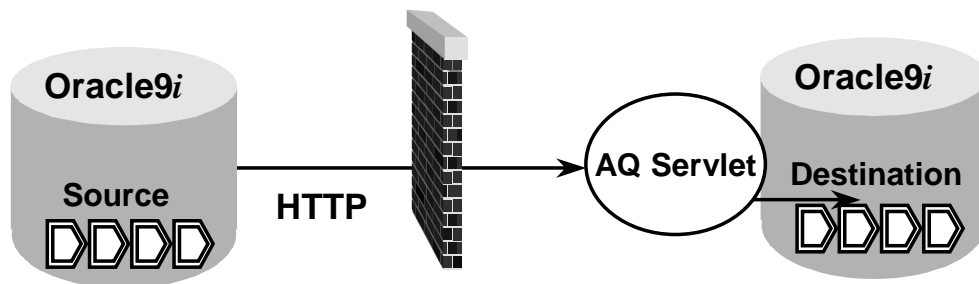
### Managing Internet Users

```
SQL> describe aq$Internet_users;
```

Name	Null?	Type
AGENT_NAME		VARCHAR2(30)
DB_USERNAME		VARCHAR2(30)
HTTP_ENABLED		VARCHAR2(4)
SMTP_ENABLED		VARCHAR2(4)

## Internet Propagation

- **AQ messages between queues on different databases can be propagated over HTTP.**
- **Such propagation can occur across firewalls.**



ORACLE

8-30

Copyright © Oracle Corporation, 2001. All rights reserved.

### Propagation of Messages Using HTTP

In release 9i, AQ propagation can be enabled over HTTP and HTTPS as well as OracleNet. The primary advantage of using an open protocol like HTTP is that, unlike Oracle Net, it is easy to configure for firewalls.

HTTP based propagation uses the infrastructure for Internet to access AQ.

The background process doing propagation pushes the messages to an AQ servlet which enqueues them into the destination database.

## Enabling HTTP Propagation

- **Steps required to configure HTTP propagation**
  - Enable the source database for Java and XML.
  - Create DBLINK on source Database with protocol HTTP, Host, and port of the Web server.
  - Username and password will correspond to those for the Internet user.
  - Create and deploy the AQPropServlet.
  - Ensure that AQ HTTP agent is authorized to perform AQ operation.
- **Then invoke propagation at the source queue.**

ORACLE

8-31

Copyright © Oracle Corporation, 2001. All rights reserved.

### Enabling HTTP Propagation

The additional steps to configure for propagation using HTTP are:

- The source database should be created for running java and XML.
- The dblink at the source database should specify the protocol as http. The host and the port of the Web server running the AQ servlet should also be specified. The username, password of the dblink will be used for authentication with the Web server.

For Example if the Web server is on Webdest.acme.com, at port 8081, then the connect string of the database would look like:

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=http) (HOST=
Webdest.acme.com)(PORT=8081)))
```

The database link for would be created (where John is a user with password welcome).

```
CREATE PUBLIC DATABASE LINK destq CONNECT TO john IDENTIFIED BY
welcome USING '(DESCRIPTION=(ADDRESS=(PROTOCOL=http) (HOST=
Webdest.acme.com)(PORT=8081)))'
```

When using the SSL, Oracle wallet should be created for the user. The wallet is used for secure communication of messages over HTTP with encryption; however, it is not used for authentication of user in AQ.

## Enabling HTTP Propagation (continued)

- An AQ servlet which connects to the destination database should be deployed. Create a class AQPropServlet that extends AQxmlServlet that is provided with AQ. This servlet should be deployed on the Web server in the path aqserv/servlet and connect to the destination database. Make sure that the AQ HTTP agent (John) is authorized to perform AQ operation. This is done by the following two steps at the destination database:
  - Register the AQ agent.

```
dbms_aqadm.create_aq_agent(agent_name => 'John',
enable_http => true);
```
  - Map the AQ agent to a database user.

```
dbms_aqadm.enable_db_access(agent_name => 'John',
db_username=> 'CBADM') '
```
- Start propagation at the source queue site by calling the `dbms_aqadm.schedule_propagation` procedure.
  - `DBMS_AQADM.SCHEDULE_PROPAGATION( 'src_queue', 'destq');`

## Using XML Payloads with AQ

- The queue table is created specifying the payload type as an object type.
- The object type can contain an attribute of type XMLType.
- XMLType stores XML documents internally as NCLOBs.
- XMLType columns can be queried using the XML operators `ExistsNode()` and `SchemaMatch()`.

ORACLE

8-33

Copyright © Oracle Corporation, 2001. All rights reserved.

### Using XML Payloads with AQ

To use XML payloads, you create a queue table with the payload type set to an Oracle object type. This can be any object type, or it can contain an attribute of type XMLType. The XMLType stores XML documents as CLOBs. If an object type is used which does not contain an XML attribute, the XML nodes are translated into object type attributes.

## Using XML Payloads with AQ Subscriptions

The steps to create and use XML Payloads in AQ are:

- Create an object type with the XMLTYPE attribute.
- Create a queue table that uses the object type.
- Create a queue in the queue table.
- Create procedures to enqueue and dequeue messages.
- Add subscribers to the queue.

ORACLE

8-34

Copyright © Oracle Corporation, 2001. All rights reserved.

### Using XML Payloads with AQ: An Example

Create an object with XML Type attribute.

```
CREATE TYPE order_xml_typ as OBJECT (orderno NUMBER, details XMLTYPE);
```

Create the queue table.

```
BEGIN
dbms_aqadm.create_queue_table(
 queue_table => 'OS_orders_pr_mqtab',
 comment => 'Overseas Shipping MultiConsumer Orders queue table',
 multiple_consumers => TRUE,
 queue_payload_type => 'OS.order_xml_typ',
 compatible => '8.1');
END;
```

Create Queue in the Queue Table.

```
BEGIN
dbms_aqadm.create_queue (
 queue_name => 'OS_bookedorders_que',
 queue_table => 'OS_orders_pr_mqtab');
END;
```



## Using XML Payloads with AQ – An Example (continued)

Create Procedures for enqueue and dequeue operations.

Add a rule-based subscriber (for Overseas Shipping) to the Booked orders queues with Transformation Overseas Shipping handles all non-US orders: \*/

```
DECLARE
 subscriber aq$_agent;
BEGIN
 subscriber := aq$_agent('Overseas_Shipping','OS.OS_bookedorders_que',null);
 dbms_aqadm.add_subscriber(
 queue_name => 'OE.OE_bookedorders_que',
 subscriber => subscriber,
 rule => 'tab.user_data.orderregion = "INTERNATIONAL"'
 transformation => 'OS.OE2XML');
END;
```

## Using XML Payloads with AQ

- **AQ is integrated with a server-based data transformation engine.**
- **The payload for IDAP conformant messages is mapped automatically to an object type payload if the XML values and object type attributes match exactly.**
- **With DBMS\_TRANSFORM you can convert AQ messages between object types in order to:**
  - **Enqueue or dequeue messages from a queue in the format of your choice**
  - **Specify the desired message format when subscribing to queues**

ORACLE

8-36

Copyright © Oracle Corporation, 2001. All rights reserved.

### Using XML Payloads with AQ (continued)

Transformation is a key requirement for application integration and business to business e-commerce. Existing applications that need to be integrated might have data in different formats. Thus any application integration solution requires a way to transform data.

Since Oracle's aim is to provide a complete solution for application integration, Oracle9i introduces a transformation tool. The transformation engine is integrated with Advanced Queues. The engine supports those transforms expressible in SQL and PL/SQL and transform data represented as relational scalar data or as Abstract Data Types (ADTs).

Because XML is the format of choice of many B2B applications, the transformation engine also supports XSLT transformations of XML documents. XML documents that conform to the IDAP model must use values and attributes that match the Oracle object payload type exactly for the transformation to take place.

## Summary

**In this lesson, you should have learned about the following:**

- **How to use IDAP to transmit AQ messages over the Internet**
- **How an Internet user is authorized to access Oracle AQ**
- **AQ message propagation between Oracle databases using HTTP and e-mail.**

ORACLE



# AQ Tuning and Troubleshooting Tips

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

## Objectives

**After completing this lesson, you should be able to do the following:**

- **Setup the tablespaces for high performance of AQ.**
- **Setup rollback segments for high performance of AQ.**
- **Set up parameters for optimal propagation.**

ORACLE

## Configuring the Tablespaces

- **When configuring tablespaces that contain queue tables, enable the following:**
  - **EXTENT MANAGEMENT LOCAL and EXTENT ALLOCATION UNIFORM (for most OLTP messaging)**
- **Tablespaces should be large enough to accommodate:**
  - **Data**
  - **IOTs**
  - **History**
- **Use retention prudently**
- **For high IO operation, set up the tablespace over striped disk volumes.**

ORACLE

9-3

Copyright © Oracle Corporation, 2001. All rights reserved.

### Performance Considerations: Tablespace

Very often, tablespaces involved in frequent online transactions need extent allocation and de-allocation activity. The extent allocation and de-allocation can result in performance degradation at two levels.

- The activity uses some CPU cycles, depriving the application of some CPU time.
- The extent allocation becomes serialized if the tablespace is Dictionary Managed.

This degradation can be avoided if Extent management is LOCAL and extents are pre-allocated.

Based on retention, and the enqueue/dequeue rate, you can get an estimate for the sizes of the queue table, history IOT, and so on. Some planning based on such estimates will help.

A very high value for retention increases the sizes of the queue table and the history IOT.

Some of the optimizations for high update tables are also applicable for the AQ tables and IOTs.

## Concurrent Transaction Activity

**Where a high level of concurrent activity is involved, create the tables with appropriate PCTFREE and INITRANS options.**

- **PCTFREE should be high where**
  - A high level of update activity exists, and
  - The rows increase in size with updates.
- **INITRANS should correspond to average number of concurrent transactions**

ORACLE

9-4

Copyright © Oracle Corporation, 2001. All rights reserved.

### Concurrent Transaction Activity

#### PCTFREE:

- The PCTFREE parameter is used to set the percentage of a data block to be reserved for possible updates to rows that already are contained in that block. For example, if you had specified a PCTFREE 20 when you created a table, 20% of each data block in this table's data segment would be kept free and available for possible updates.

#### INITRANS:

- If a table is usually accessed by many users at the same time then you should consider preallocating transaction entry space by using a high INITRANS. This eliminates the overhead of having to allocate transaction entry space, as required when the object is in use. Also, allow a higher MAXTRANS so that no user has to wait to access necessary data blocks.



# Rollback Segments and Redo Logs

- **Rollback segments**
  - If on Oracle9i, use the Automatic Undo Segment to manage undo (rollback) operations.
  - Use multiple rollback segments, with many equally sized extents.
  - Create multiple tablespaces for rollback segments.
- **Redo logs**
  - Locate the redo log files on separate disks.
  - Spread redo log files across multiple disks.
  - Have multiple redo log groups.

ORACLE

9-5

Copyright © Oracle Corporation, 2001. All rights reserved.

## Rollback Segments

In Oracle9i, the Automatic Undo Management feature has been introduced to provide easier management of undo (rollback) operations. Use Automatic Undo Management over the rollback segments.

If using rollback segments, use numerous (smaller sized) rollback segments rather than a few large rollback segments.

If possible, create rollback segments in different tablespaces of their own. Using multiple rollback segments that are spread across different disk controller distributes rollback segment contention across many segments and improves system performance.

## Redo Log Files

Create multiple redo log groups with files on different disks. Separate them from the data files and rollback segments.

## Configure Background Processes

- If operations are database write operation intensive, enable additional DB writer processes.
- If the AQ operations involve time-based operations (such as Delay, Expiry, and so on), enable a sufficient number of AQ\_TM\_PROCESSES.
  - Start up the queue monitor to ensure block cleanout of the dequeue IOT
  - Multiple queue monitor processes share the load; make sure there are enough.
- For propagation, enable a sufficient number of JOB\_QUEUE\_PROCESSES (roughly 2 per propagation).

ORACLE

## Optimization: Enqueue/Dequeue

- **Segment the queues into smaller queues to enable parallel operation.**
- **Where possible, spread the load dynamically among numerous enqueueers and dequeuers.**
- **It is better to have more subscribers, than multiple processes dequeuing with the same subscriber.**
- **Use NO\_DATA where applicable (if dequeuing is in BROWSE mode).**
- **Where dequeue operations are performed in a particular order, consider creating suitable indexes.**

ORACLE

9-7

Copyright © Oracle Corporation, 2001. All rights reserved.

### Optimization: Enqueue/Dequeue Operations

It is beneficial to segment large queues into smaller ones thereby enabling a higher level of parallel operation. For example, if an order queue is segmented on the basis of geographic region, suppliers from different regions can access order messages in parallel.

Where possible, distribute the enqueue and dequeue operations among different producers and consumers so that they can perform the task in parallel.

In operations such as BROWSE, especially where the payload is large sized, use NO\_DATA option.

When the dequeue operations are performed based on Correlation ID, consider creating an index on correlation ID.

## Optimization Propagation

- **Set `JOB_QUEUE_PROCESSES` to two times the number of scheduled propagations.**
- **Maximum number of `JOB_QUEUE_PROCESSES` is 1000 for Oracle9i**
- **Tune latency or `next_time` to control the frequency of propagation**
- **Remember to disable propagation when necessary**
- **Monitor for propagation errors frequently in**
  - `dba_queue_schedules`
  - Alert log

ORACLE

9-8

Copyright © Oracle Corporation, 2001. All rights reserved.

### Optimizing Propagation

For each propagation, you should consider enabling at least two job queue processes. Having a lower number for latency or for `next_time`, will increase the frequency of propagation. This extra overhead needs to be considered, and matched against the requirements.

There are cases when you might be able to anticipate propagation problems in advance: for example, scheduled down time, network outages, and so on. In such cases, unscheduling or disabling propagation will reduce unnecessary generation and processing of errors.

Propagation errors can be monitored through `dba_queue_schedules`, and the alert log.

## Summary

**In this lesson, you should have learned how to do the following:**

- **Configure tablespaces for performance**
- **Configure rollback segments and redo logs**
- **Configure background processes**
- **Optimize enqueue and dequeue operations**
- **Optimize propagation**

ORACLE



# 10

## **Advanced Queuing and JMS (Optional)**

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

# Objectives

**After completing this lesson, you should be able to do the following:**

- **Describe the JMS features provided in AQ such as**
  - Connection Factory
  - Topics, Subscription, and so on
- **Compare the PL/SQL and JMS APIs for**
  - Administration and operation

ORACLE



# Java Message Service (JMS) Overview

- **JMS defines standard Java-based API to access message systems.**
- **A JMS application deals with the following:**
  - **JMS providers, JMS clients, messages, and administered objects.**
- **The basic components of JMS are:**
  - **Connection factories**
  - **Connections and sessions**
  - **Destinations: topics and queues**
  - **Message producers and consumers**
  - **Messages**

ORACLE

10-3

Copyright © Oracle Corporation, 2001. All rights reserved.

## Java Message Service (JMS) Overview

Java Message Service, provides standard APIs that Java developers can use to access the common features of message systems. JMS supports the publish/subscribe and point-to-point models, and allows the creation of message types consisting of arbitrary Java objects.

A JMS application deals with the following:

- A JMS provider is a messaging system that implements the JMS interfaces and provides administrative and control features. Oracle9i JMS is the JMS provider for Oracle9i AQ.
- JMS clients are the programs or components written in the Java programming language that produce and consume messages.
- Messages are the objects that communicate information between JMS clients.
- Administered objects are preconfigured JMS objects created by an administrator for the use of clients. There are two kinds of administered objects: destinations and connection factories.

## Administered Objects in JMS Connection Factory

- **Connection Factory is used to create connection.**
- **It encapsulates connection configuration.**

```
public static void get_Factory() throws JMSEException
{
 QueueConnectionFactory qc_fact = null;
 /* get queue connection factory for database
 "aqdb", host "sun-123", port 5521, driver "thin"*/
 qc_fact =
 AQjmsFactory.getQueueConnectionFactory(
 "sun-123", "aqdb", 5521, "thin");
}
```

ORACLE

10-4

Copyright © Oracle Corporation, 2001. All rights reserved.

### Administered Objects in JMS Connection Factory

**ConnectionFactory:** A ConnectionFactory is the object a client uses to create a connection with a provider (in this case AQ). A ConnectionFactory encapsulates a set of connection configuration parameters that has been defined by an administrator. ConnectionFactory objects support concurrent use.

There are two types of ConnectionFactory objects: QueueConnectionFactory, and TopicConnectionFactory

ConnectionFactory objects can be obtained in one of the following ways:

- Static methods in AQjmsFactory
- Java Naming and Directory Interface (JNDI) lookup from a LDAP directory server.

The AQjmsFactory class can be used to obtain a handle to Queue or Topic ConnectionFactory objects.

- Use AQjmsFactory.getQueueConnectionFactory() method to obtain a QueueConnectionFactory. The queue connection factory can be created using hostname, port number, SID driver or by using JDBC URL and properties.
- Use AQjmsFactory.getTopicConnectionFactory() method To obtain a TopicConnectionFactory. The topic connection factory can be created using hostname, port number, SID driver or by using JDBC URL and properties.

## Connection

- **Connection object of JMS provides the following services:**
  - It represents an open TCP/IP socket (or a set of open sockets) between a client and a provider's service daemon.
  - It provides a structure for authenticating clients at the time of its creation.
  - It creates sessions.
- **Connections can be obtained in one of the three ways:**
  - Through an already established JDBC connection
  - Using an OCI connection pool
  - Create connection

ORACLE

10-5

Copyright © Oracle Corporation, 2001. All rights reserved.

### Connection

A JMS connection is a client's active connection to its JMS provider. A connection provides several critical services:

- It encapsulates either an open connection or a pool of connections with a JMS provider.
- It represents an open TCP/IP socket (or a set of open sockets) between a client and a provider's service daemon.
- It provides a structure for authenticating clients at the time of its creation.
- It creates sessions.
- It provides connection metadata.
- It supports an optional Exception Listener

A JMS connection to the database can be created by invoking `createQueueConnection()` or `createTopicConnection()` and passing the parameters `username` and `password` on the `QueueConnectionFactory` and `TopicConnectionFactory` object respectively.

## Connection Setup

- **A JMS client creates a connection, sessions, and a number of Message producers and consumers.**
- **Several connection objects are available to manage connection.**
- **Create a connection in stopped mode until the setup is complete.**
- **Start() method will enable messages to be received.**

ORACLE

10-6

Copyright © Oracle Corporation, 2001. All rights reserved.

### Connection Setup

A JMS client typically creates a connection, session, and a number of Message producers and Message consumers. Multiple connections are possible in the following cases:

- If the JDBC OCI driver is used to create the JMS Connection
- If the user provides an OracleOCIConnectionPool instance during JMS Connection creation.

When a Connection is created it is in stopped mode and no messages can be delivered to it. It is typical to leave the Connection in stopped mode until setup is complete. When setup is complete, the Connection's start() method is called and messages are allowed to arrive at the Connection's consumers. This setup convention minimizes the conflicts that may result from asynchronous message delivery while the client is still in the process of being set up.

It is possible to start a Connection and then to perform setup subsequently. Clients should be coded to handle asynchronous message delivery while connections are being set up. A Message producer can send messages while a Connection is stopped.

## Connection Setup (continued)

Some of the methods that are supported on the Connection object are:

- `start()`: Start, or restart, a Connection's delivery of incoming messages.
- `stop()`: Used to temporarily stop a Connection's delivery of incoming messages. When stopped, delivery to all the Connection's message consumers is inhibited. Also, synchronous receive's block and messages are not delivered to message listener
- `close()`: Close the JMS session and release all associated resources
- `createQueueSession(true, 0)`: Create a queue session. Currently only transacted sessions are allowed.
- `createTopicSession (true, 0)`: Create a topic session. Currently only transacted sessions are allowed.
- `setExceptionListener (ExceptionListener)`: Set an exception listener for the connection. This allows a client to be asynchronously notified of a problem. Some connections only consume messages so they have no other way to learn the connection has failed.
- `getExceptionListener()`: Get the ExceptionListener for this connection.

# Sessions

- **A Session encapsulates the context for messages.**
- **Session object provides a way to handle destinations, Message producers, and consumers.**
- **It also serializes execution of message production and consumption.**

ORACLE

10-8

Copyright © Oracle Corporation, 2001. All rights reserved.

## Sessions

A JMS Session is a single threaded context for producing and consuming messages. A Session serves several purposes such as:

- Constitutes a factory for its message producers and consumers
- Provides a way to get a handle to a destination objects (queues/topics)
- Supplies provider-optimized message factories
- Supports a single series of transactions that combines work spanning this session's producers and consumers, organizing these into atomic units
- Defines a serial order for the messages it consumes and the messages it produces
- Serializes execution of message listeners registered with it

Multiple sessions are possible per connection if connection pooling is used. Since a provider may allocate some resources on behalf of a session outside the JVM, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough. The same is true for the MessageProducers and MessageConsumers created by a session.

## Sessions (continued)

Methods on the Session object include:

- `commit()`: Commits all messages performed in this transaction and releases locks currently held
- `rollback()`: Rolls back any messages done in the transaction and release locks currently held
- `close()`: Closes the session
- `getDBConnection()`: Gets a handle to the underlying JDBC connection. This handle can be used to perform other SQL DML operations as part of the same session. The method is Oracle JMS specific.

The following are some of the extensions to JMS made by Oracle. The Session object must be cast to `AQjmsSession` to use any of the extensions. QueueTables, Queues, and Topics can be created from the Session object.

- `createQueueTable()`: Creates a queue table
- `getQueueTable()`: Gets a handle to an existing queue table
- `createQueue()`: Creates a Queue
- `getQueue()`: Gets a handle to an existing queue
- `createTopic()`: Creates a topic
- `getTopic()`: Gets a handle to an existing topic

## Administered Objects in JMS Destination

- **A destination object describes where**
  - Messages are sent to
  - Messages are received from
- **There are two types of destination objects: queue and topic**
- **Destination objects can be accessed in one of the following ways:**
  - Using domain-specific methods in the JMS Session
  - Java Naming and Directory Interface (JNDI) Lookup from a LDAP directory server

ORACLE

10-10

Copyright © Oracle Corporation, 2001. All rights reserved.

### Destination

A Destination is an object a client uses to specify the destination where it sends messages, and the source from which it receives messages.

There are two types of destination objects: queue and topic. In AQ, these would map to a <schema>.<queue> at a specific database. Queue maps to single consumer queue in AQ and topic maps to multiconsumer queue in AQ.

Destination objects can be obtained in one of the following ways:

- Using domain specific methods in the JMS Session
- Java Naming and Directory Interface (JNDI) Lookup from a LDAP directory server

Destination objects are created from a Session object using domain specific session methods.

- `AQjmsSession.getQueue(queue_owner, queue_name)`: Used to get a handle to a JMS queue
- `AQjmsSession.getTopic(topic_owner, topic_name)`: Used to get a handle to a JMS topic



# Access Control in JMS

**The fundamental access controls in JMS are defined at two levels:**

- **System level:**
  - Access to any queue
  - Suitable for administrators
- **Destination level:**
  - Access to specific queues
  - Suitable for operational users

ORACLE

10-11

Copyright © Oracle Corporation, 2001. All rights reserved.

## **System-Level Access Control in JMS**

System-level access control allows an application designer or DBA to create users as queue administrators. A queue or topic administrator can invoke any JMS interface on any queue in the database. This simplifies the administrative work and helps in administering all the scripts for queues in a database with in one schema.

## **Destination-Level Access Control**

Queue- or topic-level access control for enqueue and dequeue operations allows the application designer to protect queues and topics created in one schema from applications running in other schemas. You need to grant only minimal access privileges to the applications that run outside the queue's or topic's schema. The supported access privileges on a queue or topic are `ENQUEUE`, `DEQUEUE` and `ALL`.

## Point-to-Point Model

- It is a single consumer queue model.
- You can create a single consumer queue using the `createQueue` method.
- A queue processing request can get the queue details using the `getQueue` method.
- A client uses `QueueSender` to send (enqueue) messages to a queue.
- At the destination, `QueueReceiver` is used to receive (dequeue) message.
- A `QueueBrowser` is used to browse a message without removing it from the queue.

ORACLE

10-12

Copyright © Oracle Corporation, 2001. All rights reserved.

### Point-to-Point Model

Queues involved in point-to-point model are described as single-consumer queues because a message can be consumed by only a single consumer.

An administrator creates single-consumer queues by means of the `createQueue` method of `AQjmsSession`. A client obtains a handle to a queue using the `getQueue` method of `AQjmsSession`.

You should start the queue using `start` call in `AQjmsDestination` before an enqueue or dequeue operation. The message is removed based on the retention time of the queue. Retained messages can be queried using SQL on the queue table or dequeued using a `QueueBrowser`.

A client uses a `QueueSender` to send messages to a Queue. A `QueueSender` is created by passing a Queue to a session's `createSender` method. A client can specify a default delivery mode, priority, and time-to-live for all messages sent by the `QueueSender`. A client also has the option of creating a `QueueSender` without providing a queue name. In that case a queue name must be specified on every send operation along with options on a per message basis.

A client uses a `QueueReceiver` to receive messages from a queue. A `QueueReceiver` is created using the session's `createQueueReceiver` method. A `QueueReceiver` can be created with a message selector to restrict messages delivered to the consumer.

## Examples of QueueSender and QueueReceiver

An Example of QueueSender that is used to send (enqueue) message:

```
public void enqueue_new_orders(QueueSession jms_session, BolOrder
new_order)
{QueueSender sender;
Queue queue;
ObjectMessage obj_message;
try {
 /* get a handle to the new_orders queue */
 queue=((AQJmsSession) jms_session).getQueue("OE",
 "OE_neworders_que");
 sender = jms_session.createSender(queue);
 obj_message = jms_session.createObjectMessage();
 obj_message.setJMSCorrelationID("RUSH");
 obj_message.setObject(new_order); sender.send(obj_message);
 jms_session.commit();
}
catch (JMSEException ex) {
 System.out.println("Exception: " + ex);
}
}
```

An Example of QueueReceiver that is used to receive (dequeue) message:

```
public void get_new_orders(QueueSession jms_session) {
 QueueReceiver receiver;
 Queue queue;
 ObjectMessage obj_message;
 BolOrder new_order;
 BolCustomer customer;
 String state;
 String cust_name;
 try {
 /* get a handle to the new_orders queue */
 queue=((AQJmsSession) jms_session).getQueue("OE", "OE_neworders_que");
 receiver = jms_session.createReceiver(queue);
 for(;;) {
 /* wait for a message to show up in the queue */
 obj_message = (ObjectMessage)receiver.receive(10);
 new_order = (BolOrder)obj_message.getObject();
 customer = new_order.getCustomer();
 state = customer.getState();
 obj_message.clearBody();
 obj_message.setStringProperty("Region", "WESTERN");
 cust_name = new_order.getCustomer().getName();
 obj_message.setStringProperty("Customer", cust_name);
 book_new_order(obj_message);
 jms_session.commit();
 } }
 catch (JMSEException ex)
 { System.out.println("Exception: " + ex);
 } }
}
```

## Publish-Subscribe Model

- **The Publish-Subscribe model is similar to the multiconsumer queues of AQ.**
- **In the Publish-Subscribe model, messages are published to and received from topics.**
- **A topic is created using the CreateTopic method in an AQjmsSession.**
- **A client may obtain a handle to a previously created topic using the getTopic method in AQjmsSession**

ORACLE

10-14

Copyright © Oracle Corporation, 2001. All rights reserved.

### Publish-Subscribe Model

Publish-subscribe model enables applications to interact based on messages and message content as publishers and subscribers.

In distributing messages, publisher applications do not have to explicitly handle or manage message recipients. This allows for the dynamic addition of new subscriber applications to receive messages without changing any publisher application logic.

Subscriber applications receive messages based on message content without regard to which publisher applications are sending messages. This allows the dynamic addition of subscriber applications without changing any subscriber application logic. Subscriber applications specify interest by defining a rule-based subscription on message properties and/or the message content of a topic. The system automatically routes messages by computing recipients for published messages using the rule-based subscriptions.

In the Publish-Subscribe model, messages are published to and received from topics. A topic is created using the CreateTopic method in an AQjmsSession. A client may obtain a handle to a previously created Topic using the getTopic method in AQjmsSession.

## Publish-Subscribe Model

- **Topic publishers publish (enqueue) messages to the topic.**
- **A recipient list can be explicitly stated with the published message.**
- **If a recipient list is not specified, all the subscribers to the topic will receive the message.**
- **Topic receivers will receive (dequeue) the message.**

ORACLE

10-15

Copyright © Oracle Corporation, 2001. All rights reserved.

### Using Publish-Subscribe Model

You use the publish-subscribe model of communication in JMS by taking the following steps:

- Enable enqueue/dequeue on the topic using the start call in `AQjmsDestination`.
- Set up one or more topics to hold messages. These topics should represent an area or subject of interest. For example, a topic can be used to represent billed orders.
- Create a set of subscribers. Each subscriber may specify a selector that represents a specification (selects) for the messages that the subscriber wishes to receive. A null selector indicates that the subscriber wishes to receive all messages published on the topic.
- Subscribers may be local or remote. Local subscribers are durable subscribers defined on the same topic on which the message is published. Remote subscribers are other topics, or recipients on other topics that are defined as subscribers to a particular queue. In order to use remote subscribers, you must set up propagation between the two local and remote topic.
- Create `TopicPublishers` using the session's `createPublisher` method. Messages are published using the `publish` call. Messages may be published to all subscribers to the topic or to a specified subset of recipients on the topic.
- Subscribers may receive messages on the topic by using the `receive` method.
- Subscribers may also receive messages asynchronously by using `Message Listeners`. The concepts of `Remote Subscribers` and `Propagation` are Oracle extensions to JMS.

## **JMS Message Producer Feature**

- **Message Ordering:** Dictates the order in which messages are received. Two ordering attributes, Priority or Enqueue Time, are supported.
- **Time Specification (Delay):** Messages can be sent or published to a queue or topic with Delay.
- **Time Specification (Expiration):** Producers of messages can specify the period of time the message is available for a Message Consumer.
- **Message Grouping:** Messages can be grouped to form a set that can only be consumed by one consumer at a time

ORACLE

# JMS Message Consumer Features

## Receiving Messages

- Messages can be received synchronously using the receive call or asynchronously with a Message Listener. There are three modes of receive:
  - Block until a message arrives for a consumer
  - Block for a maximum of the specified time
  - Nonblocking
- Message Navigation: Message Navigation mode provided by AQ are all available for JMS
- Message Listener: Enables a JMS client to receive messages asynchronously

ORACLE

## API Comparison: Establishing Connection

PL/SQL	JMS
By Default database uses the OracleNet connection with PL/SQL	<code>get[queue topic]ConnectionFactory</code> <code>get[queue topic]Connection</code> <code>create[queue topic]Session</code>

- Can specify established JDBC connection in `get*Connection`
- Can get JDBC connection from JMS connection

ORACLE

10-18

Copyright © Oracle Corporation, 2001. All rights reserved.

### API Comparison

To establish connection, in PL/SQL, you would normally use the OracleNet connection setup.

When using the JMS, you establish connection using the JDBC. Then create `ConnectionFactory`, and establish connection / session.



## API Comparison: Queue Table and Queue Management

PL/SQL	JMS
[create   alter   drop]_queue_table	AQjmsSession.createQueueTable AQQueueTable.[alter   drop] AQjmsSession.createQueue
[create   alter   drop   start   stop]_queue	AQjmsDestination.[alter   drop   start   stop] AQjmsSession.createTopic

ORACLE

## API Comparison: Subscriptions and Propagation

PL/SQL	JMS
[add   alter   remove] _subscriber	TopicSession.createDurableSubscriber Or AQjmsSession.unsubscribe
[un]schedule_propagation [enable   alter   disable]_propagation_schedule	AQjmsDestination.[un]schedulePropagation AQjmsDestination.[enable   alter   disable]PropagationSchedule

ORACLE

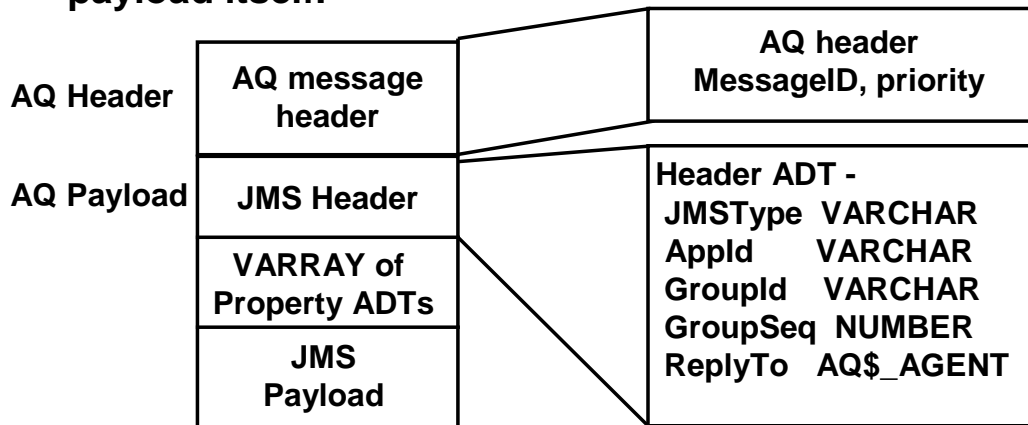
## JMS Message Structure

<b>Header</b> MessageID: 123456781234 Priority: 2 Destination: Orders_Topic Expiration: 1000	<b>Message Header</b> Has standard fields - messageID, priority, destination, correlationID etc.
<b>Properties</b> Status: New OrderId: 2000 Region: West	<b>Properties</b> User defined - name/value pairs. Name (string), Value (any java primitive type)
<b>Payload</b> Depends on message type	<b>Payload</b> TextMessage: body contains a string BytesMessage: contains uninterpreted bytes StreamMessage: stream of java primitive values MapMessage: name/value pairs ObjectMessage: contains a java serializable object AdtMessage: java object that maps to a SQL ADT

ORACLE

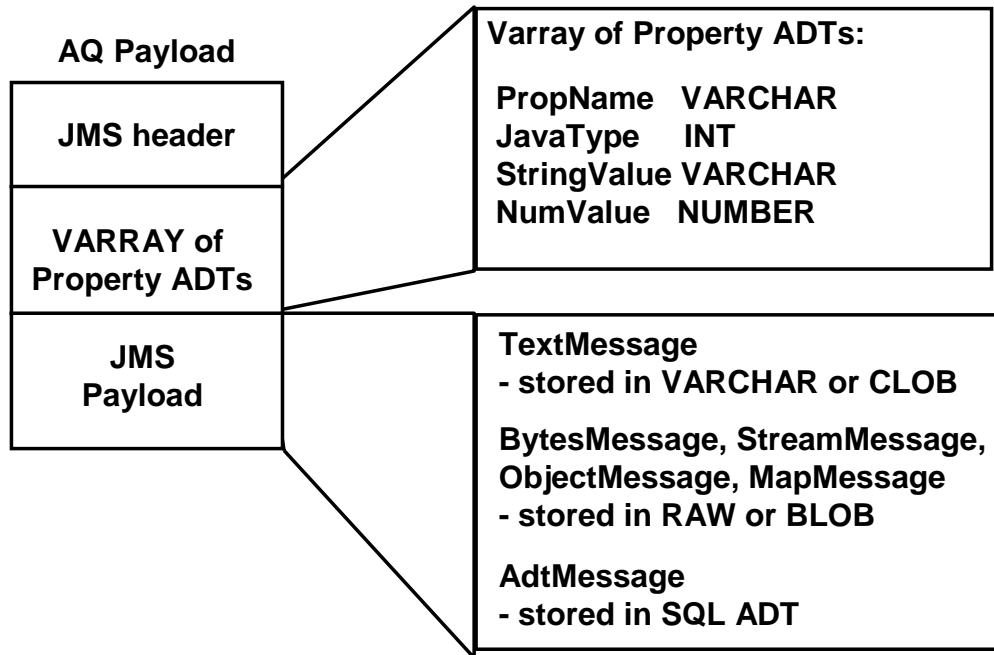
## Mapping JMS Messages to AQ Payloads

- AQ messages consist of message header fields (priority, messageID, correlationID) and payload (RAW or ADT)
- JMS defines has additional header fields, user-defined properties and . These are stored in the AQ payload itself.



ORACLE

## Mapping JMS Messages to AQ Payloads



ORACLE

## API Comparison: Creating a Message

Operation	PL/SQL	JMS
Create a raw message	RAW	Not supported
Create a message with JMS message types	AQ\$_JMS_* message type	Session.create[ Text   Object   Map   Bytes   Stream]Message
Create a message with user-defined message type	user-defined type	AQjmsSession.createAdtMessage

ORACLE

## API Comparison: Enqueuing a Message

Operation	PL/SQL	JMS
Enqueue	enqueue	QueueSender.send TopicPublisher.publish
Message properties - recipients list	specify recipients list	AQjmsTopicPublisher.publish - specify array of AQjmsAgent
Message properties - priority, expiration, correlation	priority, expiration, correlation	MessageProducer.set[TimeToLive Priority JMSCorrelationID] or during send or publish

ORACLE

## API Comparison: Enqueuing a Message

Operation	PL/SQL -	JMS
Message properties - Delay	delay	JMS_OracleDelay
Exception queue	exception_ queue	JMS_OracleExcpQ
JMS User-defined message properties	Specify in payload	Message.set[ Int   String   Boolean   etc]. Property

ORACLE



## API Comparison: Dequeueing a Message

Operation	PL/SQL	JMS
Browse message	BROWSE	QueueSession.createBrowser TopicSession.createBrowser
Locked mode	LOCKED	[Queue Topic]Session.createBrowser - set locked to TRUE
Receive messages	REMOVE	QueueSession.createQueueReceiver TopicSession. createDurableSubscriber AQjmsSession.createTopicReceiver

ORACLE

## API Comparison: Dequeueing a Message

Operation	PL/SQL	JMS
Navigation	dequeue option	AQjmsQueueReceiver.setNavigationMode AQjmsTopicSubscriber.setNavigationMode AQjmsTopicReceiver.setNavigationMode
Specify wait	dequeue option	AQjms[Queue Topic]Receiver.receive[   NoWait]

ORACLE

## API Comparison: Notification

Operation	PL/SQL	JMS
Asynchronous notification	DBMS_AQ.register	[QueueReceiver   TopicSubscriber   TopicReceiver].setMessageListener
Listen on multiple queues	DBMS_AQ.listen	[Queue   Topic]Session.setMessageListener

ORACLE

## Summary

**In this lesson you should have learned how to:**

- **Describe JMS features and AQ support for them**
- **Compare PL/SQL and JMS interfaces**

ORACLE

# **Practices and Solutions**



**ORACLE®**

Copyright © Oracle Corporation, 2001. All rights reserved.



## Oracle9i: Implement Advanced Queuing Practices and Solutions

### Lesson 2 (Initial Setup)

Lab 2 -1. Check that the SAMPLE tablespace has at least 10MB free space.

```
SQL> connect system/manager
Connected.
SQL> select tablespace_name, sum(bytes/(1024*1024))
 2 from dba_free_space
 3 where tablespace_name = 'SAMPLE'
 4 group by tablespace_name;
```

TABLESPACE_NAME	BYTES/(1024*1024)
SAMPLE	17.9375

Lab 2-2 Create users BZ\_ADM (password bz\_adm) and BZ (password bz)  
Their schema objects should be in SAMPLE tablespace,  
They should use TEMP tablespace for any disk sort.

```
SQL> connect system/manager
Connected.
SQL> CREATE USER bz_adm IDENTIFIED BY bz_adm
 2 DEFAULT TABLESPACE sample
 3 TEMPORARY TABLESPACE temp
 4 QUOTA UNLIMITED ON sample;
```

User created.

```
SQL> CREATE USER bz IDENTIFIED BY bz
 2 DEFAULT TABLESPACE SAMPLE
 3 TEMPORARY TABLESPACE TEMP
 4 QUOTA UNLIMITED ON sample;
```

User created.

Lab 2-2. Check that the users have been created correctly  
(Use DBA\_USERS)

```
SQL> SELECT username, default_tablespace, temporary_tablespace
 2 FROM dba_users
 3 WHERE username in ('BZ', 'BZ_ADM');
```

USERNAME	DEFAULT_TABLESPACE	TEMPORARY_TABLESPACE
BZ_ADM	SAMPLE	TEMP
BZ	SAMPLE	TEMP

Lab 2-4 Check that the parameters JOB\_QUEUE\_PROCESSES and AQ\_TM\_PROCESSES are set to 4, and 4 respectively. (If not set then set them as indicated).

```
SQL> SELECT name, value FROM v$parameter
 2 WHERE name in ('job_queue_processes', 'aq_tm_processes');
```

NAME	VALUE
job_queue_processes	6
aq_tm_processes	2

```
SQL> alter system set job_queue_processes=4;
```

System altered.

```
SQL> alter system set aq_tm_processes=4;
```

System altered.

```
SQL> SELECT name, value FROM v$parameter
 2 WHERE name in ('job_queue_processes', 'aq_tm_processes');
```

NAME	VALUE
job_queue_processes	4
aq_tm_processes	4



## Lesson 3 (Create queue tables and queues)

Lab 3-1. Grant privileges to the users as outlined here:

BZ\_ADM will be the Administrator of Queue operations. BZ will be AQ user  
Both of them should be able to enqueue and dequeue from queues owned by others.

```
SQL> GRANT ALTER SESSION TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE CLUSTER TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE DATABASE LINK TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE SEQUENCE TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE SESSION TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE SYNONYM TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE TABLE TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE VIEW TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE CLUSTER TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE INDEXTYPE TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE OPERATOR TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE PROCEDURE TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE SEQUENCE TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE TABLE TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE TRIGGER TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT CREATE TYPE TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT aq_administrator_role TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT EXECUTE ON dbms_aq TO bz_adm;
```

Grant succeeded.

```
SQL> GRANT EXECUTE ON dbms_aqadm TO bz_adm;
Grant succeeded.
```

```
SQL> GRANT select any dictionary to bz_adm;
Grant succeeded.
```

```
SQL>
SQL> GRANT connect, resource to bz;
Grant succeeded.
```

```
SQL> GRANT EXECUTE ON dbms_aq to bz;
Grant succeeded.
```

```
SQL> GRANT EXECUTE ON dbms_aqadm to bz;
Grant succeeded.
```

Lab 3-2. Create the following object type in BZ\_ADM's schema.

```
bzcardorder_typ (
 employee_id NUMBER(6),
 first_name VARCHAR2(20),
 last_name VARCHAR2(25),
 ordtyp VARCHAR(10))

SQL> connect bz_adm/bz_adm
Connected.
SQL> CREATE OR REPLACE TYPE bzcardorder_typ AS OBJECT (
2 employee_id NUMBER(6),
3 first_name VARCHAR2(20),
4 last_name VARCHAR2(25),
5 ordtyp VARCHAR(10));
6 /
```

Type created.

Lab 3-3. Check that the object type has been created. Grant access on object type to BZ user

```
SQL> connect bz_adm/bz_adm
Connected.
SQL> SELECT substr(object_name,1,25), object_type
2 FROM dba_objects
3 WHERE owner = 'BZ_ADM';
```

SUBSTR(OBJECT_NAME,1,25)	OBJECT_TYPE
BZCARDORDER_TYP	TYPE

```
SQL> grant execute on bzcardorder_typ to bz;
```

Grant succeeded.

Lab 3-4. Create Queue tables as follows as BZ\_ADM user:

```

queue_table => 'bzcardorders_qt',
comment => 'Business Card Orders queue table',
multiple_consumers => true,
queue_payload_type => 'bzcardorder_typ'

connect bz_adm/bz_adm
Connected.

execute dbms_aqadm.create_queue_table(-
 queue_table => 'bzcardorders_qt', -
 comment => 'Business Card Orders queue table', -
 multiple_consumers => true, -
 queue_payload_type => 'bzcardorder_typ');

PL/SQL procedure successfully completed.

```

Lab 3-5. Check the queue tables have been created in SAMPLE tablespace.

```

SQL> connect bz_adm/bz_adm
Connected.
SQL> select table_name, tablespace_name from dba_tables
 2 where table_name = 'BZCARDORDERS_QT';

```

TABLE_NAME	TABLESPACE_NAME
BZCARDORDERS_QT	SAMPLE

Lab 3-6. Create the following queue: queue\_name => 'bzcardorders\_q', queue\_table => 'bzcardorders\_qt'

```

SQL> connect bz_adm/bz_adm
Connected.
SQL> BEGIN
 2 dbms_aqadm.create_queue (
 3 queue_name => 'bzcardorders_q',
 4 queue_table => 'bzcardorders_qt');
 5 END;
 6 /
PL/SQL procedure successfully completed.

```

Lab 3-7. Check that the queues have been created.

```

SQL> select name, queue_table, queue_type
 2 from user_queues;

```

NAME	QUEUE_TABLE	QUEUE_TYPE
AQ\$_BZCARDORDERS_QT_E	BZCARDORDERS_QT	EXCEPTION_QUEUE
BZCARDORDERS_Q	BZCARDORDERS_QT	NORMAL_QUEUE

Lab 3-8. Grant enqueue and dequeue privileges on bzcardorders\_q queue to BZ user.

```

SQL> begin
 2 dbms_aqadm.grant_queue_privilege(privilege => 'ALL',
 3 queue_name => 'bzcardorders_q',
 4 grantee => 'bz',
 5 grant_option => TRUE);
 6 end;
PL/SQL procedure successfully completed.

```

- Lab 3-9. Add the following subscribers to bzc\_cardorders\_q
- a) subscriber => sys.aq\$\_agent('SHIPPING',null,null)
  - b) subscriber => sys.aq\$\_agent('BILLING',null,null)

```
SQL> connect bz_adm/bz_adm
Connected.
SQL> execute dbms_aqadm.add_subscriber (-
2 queue_name => 'BZCARDORDERS_Q', -
3 subscriber => sys.aq$_agent('SHIPPING',null,null));

PL/SQL procedure successfully completed.
```

```
SQL> connect bz_adm/bz_adm
Connected.
SQL> execute dbms_aqadm.add_subscriber (-
2 queue_name => 'BZCARDORDERS_Q', -
2 subscriber => sys.aq$_agent('BILLING',null,null));

PL/SQL procedure successfully completed.
```

- Lab 3-10. Add a rule-based subscriber 'RUSH\_ORDER' to BZCARDORDERS\_Q  
(Use the following syntax)

```
BEGIN
 dbms_aqadm.add_subscriber (
 queue_name => 'BZCARDORDERS_Q',
 subscriber => sys.aq$_agent('RUSH_ORDER',null,null),
 rule => 'tab.user_data.ordtyp = ''RUSH''');
END;
```

- Lab 3-11. Start the queue for AQ operations.

```
connect bz_adm/bz_adm

EXECUTE dbms_aqadm.start_queue (queue_name => 'bzc_cardorders_q');
```

- Lab 3-12. Check the subscriptions have been added to the queue;

```
SELECT * FROM aq$bzc_cardorders_qt_s;
QUEUE NAME

ADDRESS

PROTOCOL TRANSFORMATION

BZCARDORDERS_Q RUSH_ORDER
0
BZCARDORDERS_Q SHIPPING
0
```

## Lesson 4 (Queue Operations)

Lab 4-1 Perform an enqueue to the BZCARDORDERS\_Q with following values  
payload => bzcardorder\_typ(101, 'First', 'User', 'NORMAL')

```
SQL> connect bz/bz
Connected.
SQL> declare
2 enqopt dbms_aq.enqueue_options_t;
3 mprop dbms_aq.message_properties_t;
4 enq_msgid RAW(16);
5 begin
6 dbms_aq.enqueue(
7 queue_name => 'bz_adm.bzcardorders_q',
8 enqueue_options => enqopt,
9 message_properties => mprop,
10 payload => bz_adm.bzcardorder_typ(
11 101, 'First', 'User', 'NORMAL'),
12 msgid => enq_msgid);
13 commit;
14 end;
2 /
```

PL/SQL procedure successfully completed.

Lab 4-2. Perform enqueue with subscription list.

('Billing', null, null); and ('Shipping', null, null);

```
SQL> Connect bz /bz
Connected.
SQL> declare
2 enqopt dbms_aq.enqueue_options_t;
3 mprop dbms_aq.message_properties_t;
4 enq_msgid RAW(16);
5 rcpt_list dbms_aq.aq$recipient_list_t;
6 begin
7 rcpt_list(0) := sys.aq$agent('Shipping', null, null);
8 rcpt_list(1) := sys.aq$agent('Billing', null, null);
9 mprop.recipient_list := rcpt_list;
10 dbms_aq.enqueue(
11 queue_name => 'bzcardorders_q',
12 enqueue_options => enqopt,
13 message_properties => mprop,
14 payload =>
15 bzcardorder_typ(102, 'Second', 'User', 'NORMAL'),
16 msgid => enq_msgid);
17 commit;
18 end;
18 /
```

PL/SQL procedure successfully completed.

Lab 4-3. Using the AQ\$ view, check to see the messages have been queued.

```
SQL> select ENQ_TIME, CONSUMER_NAME, USER_DATA
2 from AQ$BZCARDORDERS_QT;
```

```
ENQ_TIME CONSUMER_NAME

USER_DATA(EMPLOYEE_ID, FIRST_NAME, LAST_NAME, ORDTYP)

14-SEP-01 BILLING
BZCARDORDER_TYP(101, 'First', 'User', 'NORMAL')

14-SEP-01 SHIPPING
BZCARDORDER_TYP(101, 'First', 'User', 'NORMAL')

14-SEP-01 BILLING
BZCARDORDER_TYP(102, 'Second', 'User', 'NORMAL')

3 rows selected.
```

**Note: This view returns one row per subscription.**

Lab 4-4. Connect as BZ user and dequeue the first message from the queue as the SHIPPING consumer. And then check the AQ\$ view.

```
SQL> declare
2 deqopt dbms_aq.dequeue_options_t;
3 mprop dbms_aq.message_properties_t;
4 msgid RAW(16);
5 payload bz_adm.bzcardorder_typ;
6 begin
7 deqopt.consumer_name := 'SHIPPING';
8 deqopt.navigation := DBMS_AQ.FIRST_MESSAGE;
9 deqopt.wait := 0;
10 dbms_aq.dequeue(
11 queue_name => 'bz_adm.bzcardorders_q',
12 dequeue_options => deqopt,
13 message_properties => mprop,
14 payload => payload,
15 msgid => msgid);
16 commit;
17 end;
18 /
```

PL/SQL procedure successfully completed.

```
SQL> select enq_time, deq_time, user_data from AQ$BZCARDORDERS_QT;
```

```
ENQ_TIME DEQ_TIME, CONSUMER_NAME
```

```

```

```
USER_DATA(EMPLOYEE_ID, FIRST_NAME, LAST_NAME, ORDTYP)
```

```

```

```
14-SEP-01 BILLING
BZCARDORDER_TYP(101, 'First', 'User', 'NORMAL')
```

```
14-SEP-01 14-SEP-01 SHIPPING
BZCARDORDER_TYP(101, 'First', 'User', 'NORMAL')
```

```
14-SEP-01 BILLING
BZCARDORDER_TYP(102, 'Second', 'User', 'NORMAL')
```

```
3 rows selected.
```

**Note: Edit the dequeue procedure and dequeue as consumer BILLING, and then you would see that the first record enqueued would no longer be visible in this view.**

Lab 4-5. Perform an Enqueue with Delay (Use LABS/Lab0406.sql. Edit as indicated in the file).

```
SQL> declare
2 enqopt dbms_aq.enqueue_options_t;
3 mprop dbms_aq.message_properties_t;
4 enq_msgid RAW(16);
5 rcpt_list dbms_aq.aq$_recipient_list_t;
6
7 begin
8 rcpt_list(0) := sys.aq$_agent('Billing', null, null);
9 rcpt_list(1) := sys.aq$_agent('Shipping', null, null);
10 mprop.delay := 180; -- 3 minutes delay
11 mprop.recipient_list := rcpt_list;
12 dbms_aq.enqueue(
13 queue_name => 'bz_aqadm.bzcardorders_q',
14 enqueue_options => enqopt,
15 message_properties => mprop,
16 payload=> bz_aqadm.bzcardorder_typ(103, 'Third', 'User', 'NORMAL'),
17 msgid => enq_msgid);
18 commit;
19 end;
20 /
```

PL/SQL procedure successfully completed.

Lab 4-6. Perform an Enqueue with Expiration of 2 minutes (Use LABS/Lab0406.sql. Edit as indicated in the file)

```
SQL> declare
2 enqopt dbms_aq.enqueue_options_t;
3 mprop dbms_aq.message_properties_t;
4 enq_msgid RAW(16);
5 rcpt_list dbms_aq.aq$_recipient_list_t;
6 begin
7 rcpt_list(0) := sys.aq$_agent('Billing', null, null);
8 rcpt_list(1) := sys.aq$_agent('Shipping', null, null);
9 mprop.expiration := 120; -- expires after 2 minutes
10 mprop.recipient_list := rcpt_list;
11 dbms_aq.enqueue(
12 queue_name => 'bzcardorders_q',
13 enqueue_options => enqopt,
14 message_properties => mprop,
15 payload => bzcardorder_typ(104, 'Fourth', 'User', 'RUSH'),
16 msgid => enq_msgid);
17 commit;
18* end;
SQL> /
```

PL/SQL procedure successfully completed.

```
SQL> select enq_time, expiration,msg_state,user_data from
AQ$bzcardorders_qt;
```

ENQ_TIME	EXPIRATION	MSG_STATE	USER_DATA(EMPLOYEE_ID, FIRST_NAME, LAST_NAME, ORDTYP)
14-SEP-01	120	<b>EXPIRED</b>	BZCARDORDER_TYP(104, 'Fourth', 'User', 'RUSH')
14-SEP-01	120	<b>EXPIRED</b>	BZCARDORDER_TYP(104, 'Fourth', 'User', 'RUSH')

Lab 4-7. Perform a Dequeue with Blocking

```
SQL> declare
2 deqopt dbms_aq.dequeue_options_t;
3 mprop dbms_aq.message_properties_t;
4 msgid RAW(16);
5 payload bz_adm.bzcardorder_typ;
6 begin
7 deqopt.consumer_name := 'SHIPPING';
8 deqopt.navigation := DBMS_AQ.FIRST_MESSAGE;
9 deqopt.wait := 30; -- wait for 30 seconds
10 dbms_aq.dequeue(
11 queue_name => 'bz_adm.bzcardorders_q',
12 dequeue_options => deqopt,
13 message_properties => mprop,
14 payload => payload,
15 msgid => msgid);
16
17 commit;
18 end;
```



Lab 4-8. Dequeue Messages enqueued within the past 30 seconds. (Edit lab0408.sql as indicated in the file).

```
SQL> declare
2 deqopt dbms_aq.dequeue_options_t;
3 mprop dbms_aq.message_properties_t;
4 msgid RAW(16);
5 payload bz_adm.bzcardorder_typ;
6 begin
7 deqopt.consumer_name := 'SHIPPING';
8 deqopt.navigation := DBMS_AQ.FIRST_MESSAGE;
9 deqopt.deq_condition := 'sysdate - enq_time > 30/86400';
10 dbms_aq.dequeue(
11 queue_name => 'bz_adm.bzcardorders_q',
12 dequeue_options => deqopt,
13 message_properties => mprop,
14 payload => payload,
15 msgid => msgid);
16
17 commit;
18 end;
19 /
PL/SQL procedure successfully completed.
```

## Lesson 5 (Propagation)

Lab 5-1. Create another queue table (name bzcardorders\_prpqt), and a queue (name bzcardorders\_prpq) in the queue table. And start it.

```
SQL> connect bz_adm/bz_adm
2 execute dbms_aqadm.create_queue_table(-
3 queue_table => 'bzcardorders_prpqt', -
4 comment => 'Business Card Orders queue table for Propagation', -
5 multiple_consumers => true, -
6 queue_payload_type => 'bzcardorder_typ');
7 /
```

PL/SQL procedure successfully completed.

```
SQL BEGIN
2 dbms_aqadm.create_queue (
3 queue_name => 'bzcardorders_prpq',
4 queue_table => 'bzcardorders_prpqt');
5 END;
6 /
```

PL/SQL procedure successfully completed.

```
SQL> BEGIN
2 dbms_aqadm.start_queue (
3 queue_name => 'bzcardorders_prpq');
4 END;
5 /
```

PL/SQL procedure successfully completed.

Lab 5-2. Add another subscriber who uses the new queue table use the following parameters

```
queue_name => 'BZCARDORDERS_PRPQ',
subscriber => sys.aq$_agent('RUSH_ORDER','bzcardorders_prpq', null),
rule => 'tab.user_data.ordtyp = "RUSH"'
```

```
SQL> begin
2 dbms_aqadm.add_subscriber (
3 queue_name => 'BZCARDORDERS_Q',
4 subscriber=> sys.aq$_agent('RUSH_RMT',
5 'bz_adm.bzcardorders_prpq', null),
6 rule => 'tab.user_data.ordtyp = ''RUSH'' ');
7 end;
8 /
```

Lab 5-3. Add a propagation schedule for the queue

```
SQL> execute dbms_aqadm.schedule_propagation(-
2 queue_name => 'bzcardorders_q');
```

Lab 5-4. Check the schedule has been enabled using USER\_QUEUE\_SCHEDULES.

```
SQL> select QNAME, DESTINATION
2 from user_queue_schedules;
```

QNAME	DESTINATION
BZCARDORDERS_Q	AQ\$_LOCAL

Lab 5-5 Grant enqueue and dequeue privileges on bzcardorders\_prpq to BZ user.

```
SQL> begin
 2 dbms_aqadm.grant_queue_privilege(privilege => 'ALL',
 3 queue_name => 'bzcardorders_prpq',
 4 grantee => 'bz',
 5 grant_option => TRUE);
 6 end;
```

Lab 5-6. Enqueue a message for RUSH ORDER

```
declare
 enqopt dbms_aq.enqueue_options_t;
 mprop dbms_aq.message_properties_t;
 enq_msgid RAW(16);
begin
 dbms_aq.enqueue(
 queue_name => 'bz_adm.bzcardorders_q',
 enqueue_options => enqopt,
 message_properties => mprop,
 payload => bz_adm.bzcardorder_typ(116, 'Person in', 'Rush', 'RUSH'),
 msgid => enq_msgid);
 commit;
end;
/
```

Lab 5-7. Dequeue from the new queue.

```
declare
 deqopt dbms_aq.dequeue_options_t;
 mprop dbms_aq.message_properties_t;
 msgid RAW(16);
 payload bz_adm.bzcardorder_typ;
begin
 deqopt.consumer_name := 'RUSH_RMT';
 deqopt.navigation := DBMS_AQ.FIRST_MESSAGE;
 deqopt.wait := 0;
 dbms_aq.dequeue(
 queue_name => 'bz_adm.bzcardorders_prpq',
 dequeue_options => deqopt,
 message_properties => mprop,
 payload => payload,
 msgid => msgid);

 commit;
end;
/
```

Lab 5-8. Disable the propagation schedule.

```
execute dbms_aqadm.unschedule_propagation(-
 queue_name => 'bzcardorders_q');
```

## Lesson 6 (Transformation)

Lab 6-1 Create a new type BZBILL\_TYP that adds one more attribute PRICE to BZCARDORDER\_TYP. Grant execute privilege on the type to BZ user.

```
create or replace type BZBILL_TYP AS object(
 ord bzcorder_ttyp,
 price number);
/

GRANT EXECUTE ON BZBILL_TYP TO BZ;
```

Lab 6-2. Create BZBILLS\_QT queue table using the BZBILL\_TYP. It should allow multiple consumers.

```
execute dbms_aqadm.create_queue_table(-
 queue_table => 'bzbills_qt', -
 comment => 'Business Card Billing queue table', -
 multiple_consumers => true, -
 queue_payload_type => 'bzbill_ttyp');
```

Lab 6-3. Create and start queue BZBILLS\_Q in the BZBILLS\_QT queue table.

```
BEGIN
 dbms_aqadm.create_queue (
 queue_name => 'bzbills_q',
 queue_table => 'bzbills_qt');
END;
/

execute dbms_aqadm.start_queue(queue_name=>'bzbills_q');
```

Lab 6-4. Grant enqueue and dequeue privileges on BZBILLS\_Q queue to BZ.

```
grant execute on bzbill_ttyp to bz;

begin
 dbms_aqadm.grant_queue_privilege(privilege => 'ALL',
 queue_name => 'bzbills_q',
 grantee => 'bz',
 grant_option => TRUE);
end;
/
```

Lab 6-5. Create a mapping function that adds a price to order submitted. And grant execute on the function to BZ. The pricing is \$10 for NORMAL, \$20 for RUSH, and \$30 for URGENT.

```
create or replace function FnOrdertoBill
(ord BZ_ADM.BZCARDORDER_TYP)
return BZBILL_TYP is
newBill BZBILL_TYP;
BEGIN
newBill := BZBILL_TYP(ord, 10);
if ord.ordtyp = 'RUSH'
then newBill.price :=20;
end if;
if ord.ordtyp = 'URGENT'
then newBill.price := 30;
end if;
return newBill;
END;
/

grant execute on FnOrdertoBill to bz;
```

Lab 6-6. Create a transformation that uses the above function, and transforms order messages to bills.

```
execute dbms_transform.create_transformation (-
schema => 'BZ_ADM', name => 'OrdertoBill', -
from_schema => 'BZ_ADM', to_schema => 'BZ_ADM', -
from_type => 'BZ_ADM.BZCARDORDER_TYP', to_type => 'BZ_ADM.BZBILL_TYP', -
transformation => 'BZ_ADM.FnOrdertoBill(SOURCE.USER_DATA)');

show errors;
```

Lab 6-7. Add Subscription to BZCARDORDERS\_Q with following information.

Source queue is 'BZCARDORDERS\_Q'

The subscriber name is 'CUST\_BILL'

DESTINATION is BZBILLS\_Q

Transformation used is ORDERTOBILL.

```
connect bz_adm/bz_adm
begin
dbms_aqadm.add_subscriber (
queue_name => 'BZCARDORDERS_Q',
subscriber => sys.aq$_agent('CUST_BILL','bz_adm.bzbills_q', null),
transformation=> 'BZ_ADM.OrdertoBill');
end;
```

Lab 6-8. Schedule Propagation on BZCARDORDERS\_Q.

```
execute dbms_aqadm.schedule_propagation(queue_name => 'bzcardorders_q');
```

Lab 6-9. Enqueue an order using lab0609.sql in LABS folder.

Lab 6-10. Verify the order has been queued using the AQ\$BZCARDORDERS\_QT view.

```
SQL> select consumer_name, user_data from aq$bzcardorders_qt;

CONSUMER_NAME

USER_DATA(EMPLOYEE_ID, FIRST_NAME, LAST_NAME, ORDTYP)

RUSH_ORDER
BZCARDORDER_TYP(116, 'Person in', 'Rush', 'RUSH')

CUST_BILL
BZCARDORDER_TYP(116, 'Person in', 'Rush', 'RUSH')
```

Lab 6-11. Verify that the order has been propagated to BZBILLS\_Q using AQ\$BZBILLS\_QT view.

```
SQL> select user_data from aq$zbzills_qt;

USER_DATA(ORD(EMPLOYEE_ID, FIRST_NAME, LAST_NAME, ORDTYP), PRICE)

BZBILL_TYP(BZCARDORDER_TYP(116, 'Person in', 'Rush', 'RUSH'), 10)
```

Lab 6-12. Use LABS/lab0612.sql to dequeue the bill from BZBILLS\_Q.