Application Development with Oracle Advanced Queuing

Jeffrey M. Jacobs

Senior Data Architect, PayPal

Qualifications(?)

- Two centuries experience with Oracle
- Extensive consulting and training experience
- Now FT Senior Site Data Architect with PayPal

Disclaimer

- All content is responsibility of author who is neither infallible nor omniscient
- This presentation is *not* about PayPal's use of messaging technology
- Paper and revised slides will be available at www.jeffreyjacobs.com after conference (permanently)

Survey Says

- Developers
- DBA
- Architect
- Manager
- Generally familiar with messaging concepts

Agenda

- Features and Capabilities
- Fundamental Concepts
- Creating Queue Tables
- RAC considerations
- Creating Queues
- Queuing techniques
- AQ Management
- Managing Error Queues
- Managing Propagation
- Dequeuing Performance Tips

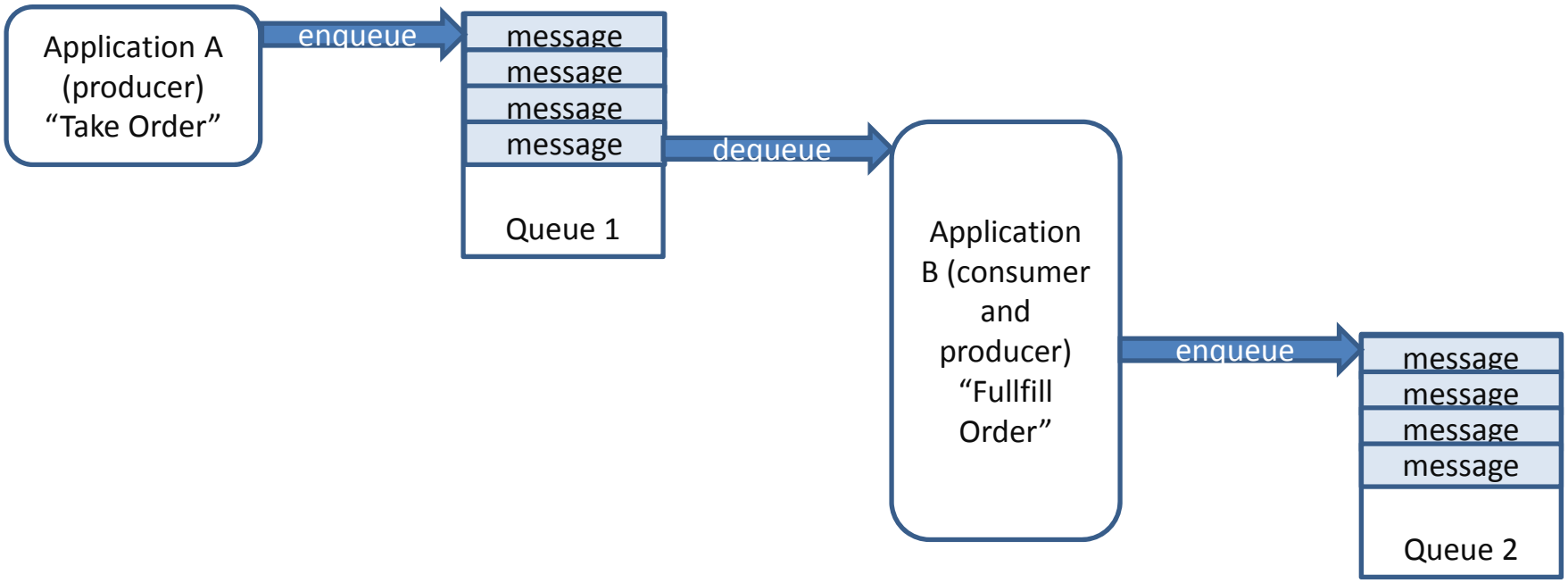
What is “Messaging”

- Messaging is the ability to send a message containing data from one application/process to another application/process
- Generally asynchronous
  - Oracle AQ does not support synchronous messaging
- Uses include:
  - Distributed applications
  - Batch processing
  - Deferred processing
  - Replication (Oracle Streams, “custom” replication)
  - Many more (extensively used in EBS)

Terminology

- *Message*
  - Data sent from one process to one or more other processes (includes both *payload* and appropriate delivery information)
- *Producer*
  - A process that enqueues a message
    - Any process with appropriate privileges can enqueue
- *Consumer(s)*
  - The recipient(s) of a message
  - When all Consumers have dequeued/consumed a message, the message is removed from the queue

Basic Messaging (FIFO)



Terminology

- *Payload*
  - The data part of a message
- *Subscriber* – named applications that have been declared as consumers for a queue (only for multi-consumer queues)
- *Browsing*
  - AQ provides the ability to examine messages in the queue without consuming them
- *Transaction*
  - An Oracle transaction

Oracle AQ Features

- Provides all common messaging capabilities
  - Point to point
  - Publish and Subscribe (aka multi-point)
    - Multicast– msg sent to receivers known by producer
    - Broadcast – Producer does not know recipients, consumers dynamically *subscribe* to queue
- Error handling
- Timeouts
- History
- Options for dequeuing (not just FIFO)

More Features

- Message grouping
- Propagation
  - Other Oracle databases via dblink
  - Pushing messages to external queues
    - JMS, middleware and gateways
- Persistent messages and meta-data
  - Guaranteed operations
- Lightweight, non-persistent, non-guaranteed “buffered” messages
- Very high performance

More Features

- API's for all operations, both PL/SQL and Java
  - DBMS_AQADM, DBMS_AQ
- AQ tables are accessible via SQL for monitoring
  - Query only; DML will damage operations
- Multiple payload types
  - Abstract Data Types
    - Definition must exist in all databases if propagating
  - XML
  - Raw, CLOB, BLOB, BFILE
  - anydata

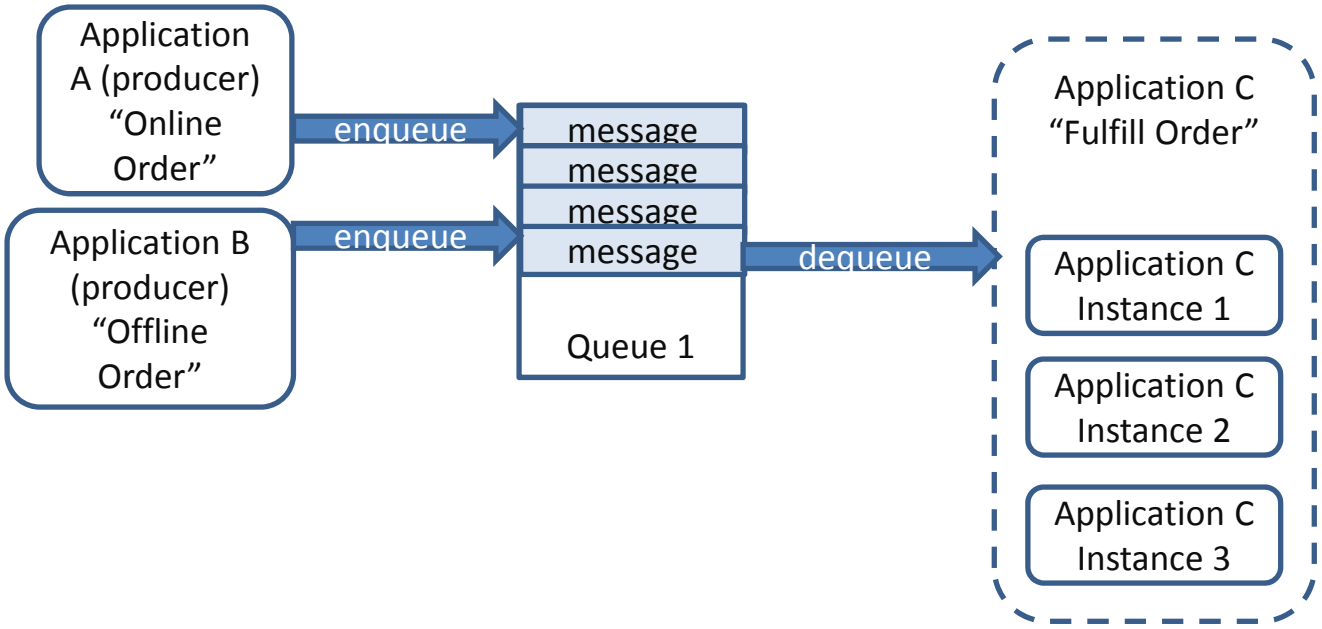
More Features

- Content-base routing
- Wait/listen for available message on multiple queues
- Notifications via email

Queue Types

- "Single consumer" queues
  - Messages are dequeued by only one consumer
  - Message is removed when dequeued
  - Multiple consumers may access queue, but a given message is only read by one consumer
    - E.g. multiple jobs accessing queue
  - Fastest
  - Multiple producers may enqueue messages
  - Simpler underlying structure

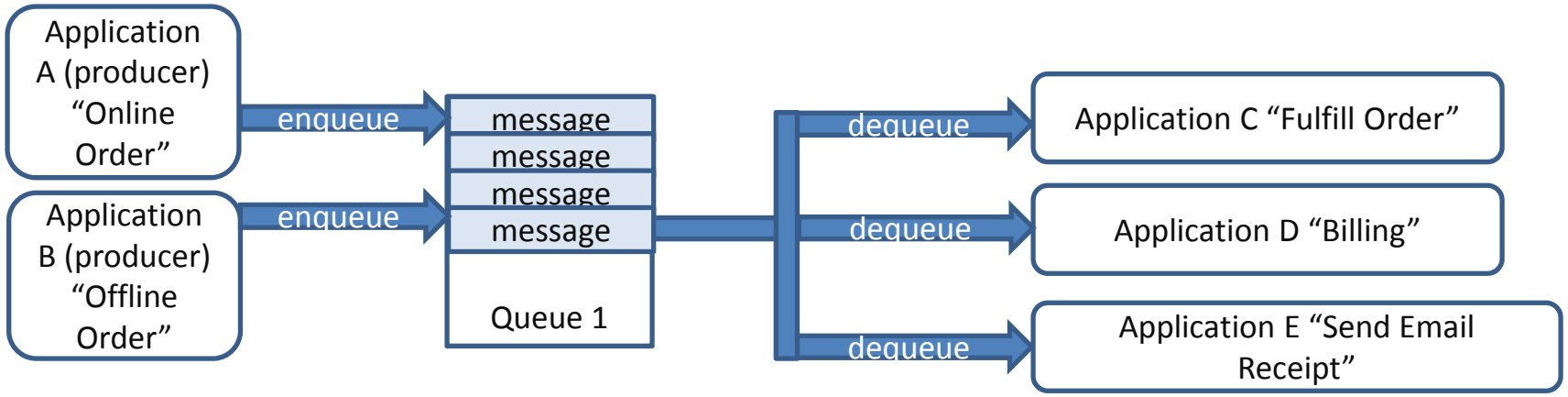
Single Consumer Queue, 1 "Instance" Consumes 1 Message



Queue Types

- Multi-consumer queues
  - Messages are read/dequeued by multiple consumers
  - ***Messages remain in queue until read by all consumers (or it expires)***

Multi-Consumer Queue, 3 Consumers



Multi-consumer Queues

- Two types
  - Multicast
    - Message recipients are designated by name
    - Usually propagated to queues in other databases
    - Those queues may be single or multi-consumer
  - Broadcast
    - Consumers (“subscribers”) are dynamic
  - More tables, more overhead

Buffered Messaging

- Light weight, non-persistent
  - Not guaranteed
- Single message only, no grouping

Message States

- `Ready` – message is available to be dequeued
- `Waiting` – availability for dequeuing is delayed
- `Expired` – message has timed out or exceeded retry count and been moved to an exception queue
  - The reason can be determined by examining the retry count in the exception queue
- `Processed` – message has been consumed by all consumers

Advanced Queuing (AQ) Tables

- Data structure for queues
- AQ Table is an “abstraction”, not a true Oracle table
- Two basic types of AQ table
  - Single consumer
    - All queues are single consumer
    - Creates a single Oracle table
  - Multiple consumer
    - Queues may have multiple consumers
    - Creates multiple Oracle heap and IOT tables
    - Managed by AQ monitor/service

AQ Table Structures

- Multi-consumer AQ table creates multiple (7) Oracle tables
  - Main table with data has same name as specified in CREATE_QUEUE_TABLE, e.g. ORDERS_QUEUETABLE
  - Other tables have name beginning with AQ$, e.g. AQ$_ORDERS_QUEUETABLE_H
- Single consumer AQ table creates single table with main table name

Creating AQ Tables

```
DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table IN VARCHAR2,
    queue_payload_type IN VARCHAR2,
    [storage_clause IN VARCHAR2 DEFAULT NULL,]
    sort_list IN VARCHAR2 DEFAULT NULL,
    multiple_consumers IN BOOLEAN DEFAULT FALSE,
    message_grouping IN BINARY_INTEGER DEFAULT NONE,
    comment IN VARCHAR2 DEFAULT NULL,
    primary_instance IN BINARY_INTEGER DEFAULT 0,
    secondary_instance IN BINARY_INTEGER DEFAULT 0,
    compatible IN VARCHAR2 DEFAULT NULL,
    secure IN BOOLEAN DEFAULT FALSE);
```

Parameters

- `queue_table` – AQ table name
- `queue_payload_type` – payload type
- `storage_clause` – any valid storage clause
  - Only tablespace needed
  - Oracle recommends ASSM
    - If not ASSM, may (rarely) need to modify `INITRANS` or `PCTFREE`
- `sort_list` – determines the order in which messages are dequeued
  - Applies to all queues
  - Cannot be changed after creation

Parameters

- `multiple_consumers` – 'TRUE' or 'FALSE'
- `message_grouping` – 'NONE' or 'TRANSACTIONAL'
  - 'TRANSACTIONAL' – all messages enqueued in one transaction may be treated as a group when dequeuing
- `comment` – description of queue table (stored in data dictionary)
- `primary_instance` – primary owner of the queue table service (RAC)
- `secondary_instance` – secondary owner of the queue table service (RAC)

Parameters

- `compatible` – lowest database version compatibility.
- `secure` – 'TRUE' for secure queues (beyond scope)

RAC Considerations

- Each AQ table effectively creates a service
- AQ table structures are typically hot tables with potential hot block instance
- `primary_instance` - specifies the preferred instance on which the service runs (aka the “owner”)
  - Effectively specifies node affinity for all queue operations associated with AQ table
- `secondary_instance` - specifies the preferred instance if primary instance is not available
- If neither is available, “random” choice for service

Creating Queues

```
DBMS_AQADM.CREATE_QUEUE (
    queue_name IN VARCHAR2,
    queue_table IN VARCHAR2,
    queue_type IN BINARY_INTEGER DEFAULT NORMAL_QUEUE,
    max_retries IN NUMBER DEFAULT NULL,
    retry_delay IN NUMBER DEFAULT 0,
    retention_time IN NUMBER DEFAULT 0,
    dependency_tracking IN BOOLEAN DEFAULT FALSE,
    comment IN VARCHAR2 DEFAULT NULL,
    auto_commit IN BOOLEAN DEFAULT TRUE);
```

Creating Queues

- `queue_name` – name of the queue
- `queue_table` – name of table holding queue
- `queue_type` – `NORMAL_QUEUE` or `EXCEPTION_QUEUE`
- `max_retries` – max number of dequeue retries (rollbacks) before moving to exception queue
- `retry_delay` – after failure, delay before msg can be dequeued again
- `retention_time` – time in which msg remains in the *queue table* after dequeuing
- `dependency_tracking` - not currently implemented
- `comment` – documentation (in data dictionary)
- `auto_commit` - deprecated

Adding Subscribers

```
DBMS_AQADM.ADD_SUBSCRIBER (
queue_name IN VARCHAR2,
subscriber IN sys.aq$_agent,
rule IN VARCHAR2 DEFAULT NULL,
transformation IN VARCHAR2 DEFAULT NULL
queue_to_queue IN BOOLEAN DEFAULT FALSE,
delivery_mode IN PLS_INTEGER DEFAULT
DBMS_AQADM.PERSISTENT);
```

Parameters

- `queue_name` – name of the queue
- `Subscriber(sys.aq$_agent)` – name, address and protocol of the subscriber
- `rule` – rule determines if message is to be processed by subscriber (beyond scope)
- `transformation` – specifies a transformation to be applied to message (beyond scope)
- `queue_to_queue` – used for propagation via dblink; subscriber may dequeue from local queue
- `delivery_mode` – delivery may be persistent or buffered

Enqueue Options and Features

- Enqueue single message
- Enqueue an array of messages (PL/SQL or OCI)
- Message Grouping
- Sender Identification
- Time Specification and Scheduling
- Correlation Identifier

Enqueuing Messages (note record types)

```
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);
```

DBMS_AQ. ENQUEUE_OPTIONS_T

```
TYPE SYS.ENQUEUE_OPTIONS_T IS RECORD (
   visibility BINARY_INTEGER DEFAULT ON_COMMIT,
   relative_msgid RAW(16) DEFAULT NULL,
   sequence_deviation BINARY_INTEGER DEFAULT NULL,
   transformation VARCHAR2(61) DEFAULT NULL,
   delivery_mode PLS_INTEGER NOT NULL DEFAULT
   PERSISTENT);
```

Enqueue Options

- `queue_name` – the name of the queue
- `enqueue_options_t` – PL/SQL type
  - `visibility`
    - 'ON_COMMIT' (default) – enqueue is part of transaction, added on COMMIT
    - 'IMMEDIATE' – added as part of autonomous transaction
  - `transformation`
    - Specifies a transformation function to be performed before enqueuing

`enqueue_options_t`

- `delivery_mode`
  - 'PERSISTENT' (default)
  - 'BUFFERED'
- The sequence deviation feature is deprecated as of 10.2
  - `relative_msg_id` – effectively deprecated
  - `sequence_deviation` – effectively deprecated

DBMS_AQ. message_properties_t

```
TYPE message_properties_t IS RECORD (
    priority BINARY_INTEGER NOT NULL DEFAULT 1,
    delay BINARY_INTEGER NOT NULL DEFAULT NO_DELAY,
    expiration BINARY_INTEGER NOT NULL DEFAULT NEVER,
    correlation VARCHAR2(128) DEFAULT NULL,
    attempts BINARY_INTEGER,
    recipient_list AQ$_RECIPIENT_LIST_T,
    exception_queue VARCHAR2(61) DEFAULT NULL,
    enqueue_time DATE,
    state BINARY_INTEGER,
    sender_id SYS.AQ$_AGENT DEFAULT NULL,
    original_msgid RAW(16) DEFAULT NULL,
    signature aq$_sig_prop DEFAULT NULL,
    transaction_group VARCHAR2(30) DEFAULT NULL,
    user_property SYS.ANYDATA DEFAULT NULL
    delivery_mode PLS_INTEGER NOT NULL DEFAULT
    DBMS_AQ.PERSISTENT);
```

Relevant ENQUEUE Attributes

- `priority` – priority of message. Smaller number = higher priority, may be negative
- `delay` – specifies number of seconds *before* msg is available for dequeue. Default is 0 (NO_DELAY)
- `expiration` – number of seconds msg is *available* for dequeuing (after delay)
  - Generally necessary for multi-consumer queues, as not all subscribers may be able to dequeue msg. Default is `NEVER`.
- `delivery_mode` - `DBMS_AQ.BUFFERED` or `DBMS_AQ.PERSISTENT`.

More ENQUEUE attributes

- `correlation` - correlation id for dequeuing by correlation id
  - *correlation* allows multiple messages to be *logically* grouped by an id and dequeued as a group
  - Unlike transaction group, need not be enqueued in a single transaction
  - Multiple producers may enqueue messages with same correlation id

Dequeuing Features

- Concurrent dequeues
- Multiple dequeue methods and options
- Array dequeue
- Message states
- Message navigation
- Wait for messages
- Retries with delays
- Transaction protection
- Exception queues

Dequeuing Messages (note record types)

```
DBMS_AQ.DEQUEUE(
    queue_name IN VARCHAR2,
    dequeue_options IN dequeue_options_t,
    message_properties OUT message_properties_t,
    payload OUT "type_name",
    msgid OUT RAW);
```

DEQUEUE_OPTION_T

```
TYPE DEQUEUE_OPTIONS_T IS RECORD (
    consumer_name VARCHAR2(30) DEFAULT NULL,
    dequeue_mode BINARY_INTEGER DEFAULT REMOVE,
    navigation BINARY_INTEGER DEFAULT NEXT_MESSAGE,
    visibility BINARY_INTEGER DEFAULT ON_COMMIT,
    wait BINARY_INTEGER DEFAULT FOREVER,
    msgid RAW(16) DEFAULT NULL,
    correlation VARCHAR2(128) DEFAULT NULL,
    deq_condition VARCHAR2(4000) DEFAULT NULL,
    signature aq$_sig_prop DEFAULT NULL,
    transformation VARCHAR2(61) DEFAULT NULL,
    delivery_mode PLS_INTEGER DEFAULT PERSISTENT);
```

Dequeue Modes

- REMOVE (with data) – standard dequeue. Message can remain in table for history based on retention period, but not eligible for future dequeuing
- REMOVE_NODATA – no data returned, but removed from queue
  - May be used for selective cleanup
- BROWSE – read, but does not actually dequeue
  - Remains available for future processing (unless dequeued by another process)
  - Non-repeatable, numerous “gotchas”

Dequeue Navigation

- Two navigation methods
  - `FIRST_MESSAGE`
  - `NEXT_MESSAGE`

Dequeue Methods

- Default - simple dequeue of 1st available msgs based on declared sort order
- Correlation ID – dequeue series of msgs based on `correlation`
  - Get correlation id from FIRST_MESSAGE
    - Creates “snapshot” (effectively a cursor)
    - Get additional mg's via NEXT_MESSAGE until exhausted
    - Only gets msg's enqueued at time of FIRST_MESSAGE
    - May use pattern matching
    - Typically needs index added (EBS DBAs take note!)

More Dequeue Methods

- Transaction group – similar to correlation, but uses transaction_group set by producer
- When dequeuing individual msgs in transaction_group
  - Dequeue 1st msg in group using `FIRST_MESSAGE`
  - Dequeue subsequent mgs using `NEXT_MESSAGE`
- Faster to use `DBMS_AQ.DEQUEUE_ARRAY`
  - Generally no need to dequeue messages individually

More DEQUEUE Navigation

- KISS – dequeuing single messages
  - No need for `FIRST_MESSAGE`
  - `NEXT_MESSAGE` is `faster`
    - Fewer `SELECT`s issued
    - Establishes snapshot/cursor for duration of transaction

Dequeue `visibility`

- Messages may be dequeued either
  - `ON_COMMIT` (transaction protection)
    - Message is removed from queue on COMMIT of transaction
  - `IMMEDIATE`
    - Messages is removed from queue as autonomous transaction
    - Use if application does not have retry capabilities

Transaction Protection (`visibility`)

- With transaction protection, dequeue operation is considered part of transaction, same as INSERT, UPDATE, DELETE
- Transaction failure leaves message in queue
  - ROLLBACK increments retry count
  - If retry count exceeded, msg is moved to exception queue
  - Killing session, abort shutdown, etc. do not increment retry count
- Only useful if application has retry capabilities

Retries with Delays

- After failure, a delay may be specified
  - Message placed in `WAITING` state for specified duration

Message Expiration

- If expiration is specified `in message_properties_t.expiration`
  - ***All*** consumers must dequeue msg before expiration
  - Otherwise, msg is moved to an exception queue
- If multi-consumer queue, expiration is generally good practice
  - Consumers may or may not be active

Waiting for Messages

- A consumer may wait for messages
  - DEQUEUE operation `wait` parameter to wait on specific queue when no message available to dequeue
    - `FOREVER` – waits forever, default
      - best for high frequency queues
    - `NO_WAIT` – don't wait
    - Number –wait time in seconds
    - Message is dequeued on wake up
  - LISTEN operation to wait on multiple queues
    - Returns name of queue with message

Exception Queues

- Each AQ table has at least one exception queue which contains messages that have expired or exceeded retry count from all of the other queues
- Messages in exception queues may be dequeued *once* by only one consumer for reprocessing

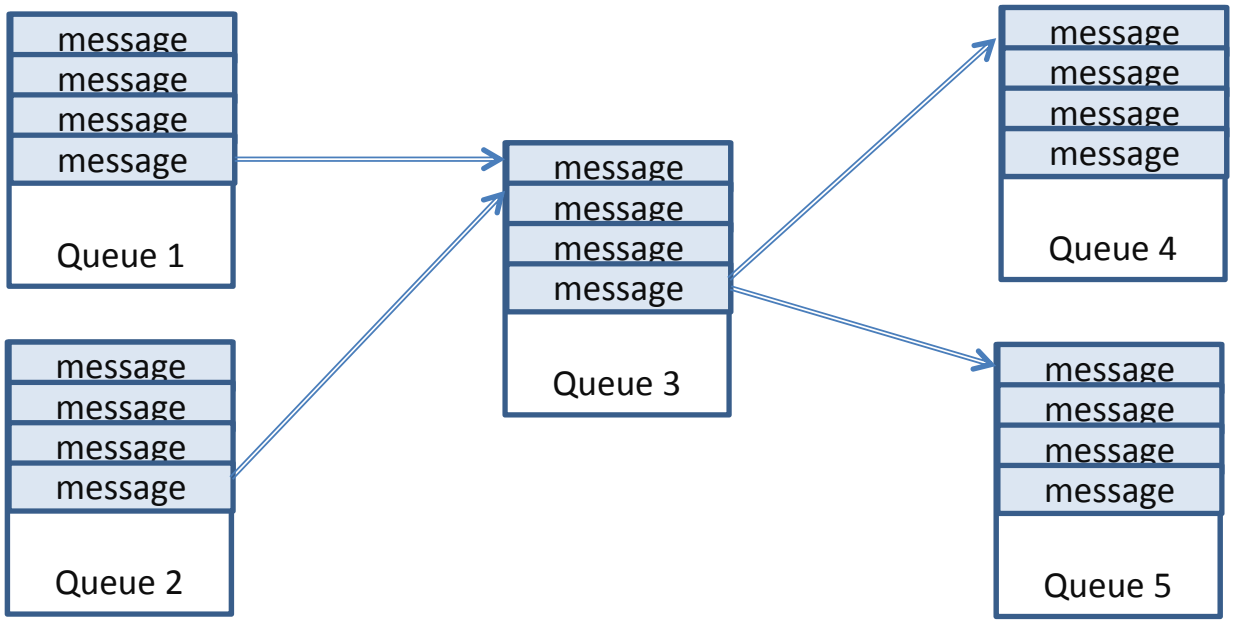
Propagation

- Messages may be pushed to other queues via *propagation*
- Specify destination queues, typically in other database (via dblink)
  - Specify propagation schedule
  - Occurs via scheduled jobs (managed by AQ)
- Message properties become subject to all specifications of destination queue

Propagation

- Messages may be “fanned” out to multiple queues
- Messages may be “combined” from multiple propagators into a single queue
- Destination queue must be of same payload type

Fan In, Fan Out Propagation



Propagation APIs

- ALTER_PROPAGATION_SCHEDULE
- DISABLE_PROPAGATION_SCHEDULE
- ENABLE_PROPAGATION_SCHEDULE
- SCHEDULE_PROPAGATION
- VERIFY_QUEUE_TYPES

Performance Tips for Dequeuing (EBS DBAs!)

- May need to add additional indexes on main queue table, e.g. `CORRID`
  - May need to generate statistics on added indexes to change plan
  - Due to volatility of queue, statistics usually need to be generated manually
    - Hand crafted
    - Generate in dev environment (enqueue without dequeue)
  - Import statistics
  - Lock statistics (avoid auto stats gathering)

Query to be Tuned

- Search for queries with following pattern

```
SELECT      /*+ FIRST_ROWS(1) */
          tab.ROWID,
          …
           tab.user_data
      FROM <queue_table_name>
    WHERE q_name = :1 AND (state = :2 and …
ORDER BY q_name, …
FOR UPDATE SKIP LOCKED;
```

FOR UPDATED SKIP LOCKED

- Undocumented feature
  - Secret sauce for AQ
  - Non-blocking SELECT FOR UPDATE
  - Only selects rows that are not currently locked
    - New messages
- Appears to only lock rows when fetched (unconfirmed)

More Stuff

- AQ manages space, performs COALESCE
  - May be performed manually
- AQ can propagate messages via external protocols and gateways
- AQ can be accessed via SOAP
- AQ can retain the entire history of a message for non-repudiation, logging, etc.

Summary

- Oracle Advance Queuing provide a full featured messaging platform, supporting all common and desired asynchronous messaging capabilities