Oracle Streams AQ

Lessons From the Trenches

Should I Slip Out?

- Brief into to asynchronous processing
- Brief history, overview of Oracle Streams AQ
- Will dive deeply into single-consumer queues
- Will cover real-world traps encountered and their solutions
- No time spent on multiple-consumer queues or esoteric corners of AQ
- So this session is for novice and intermediate AQ user and DBA (should be PL/SQL literate)

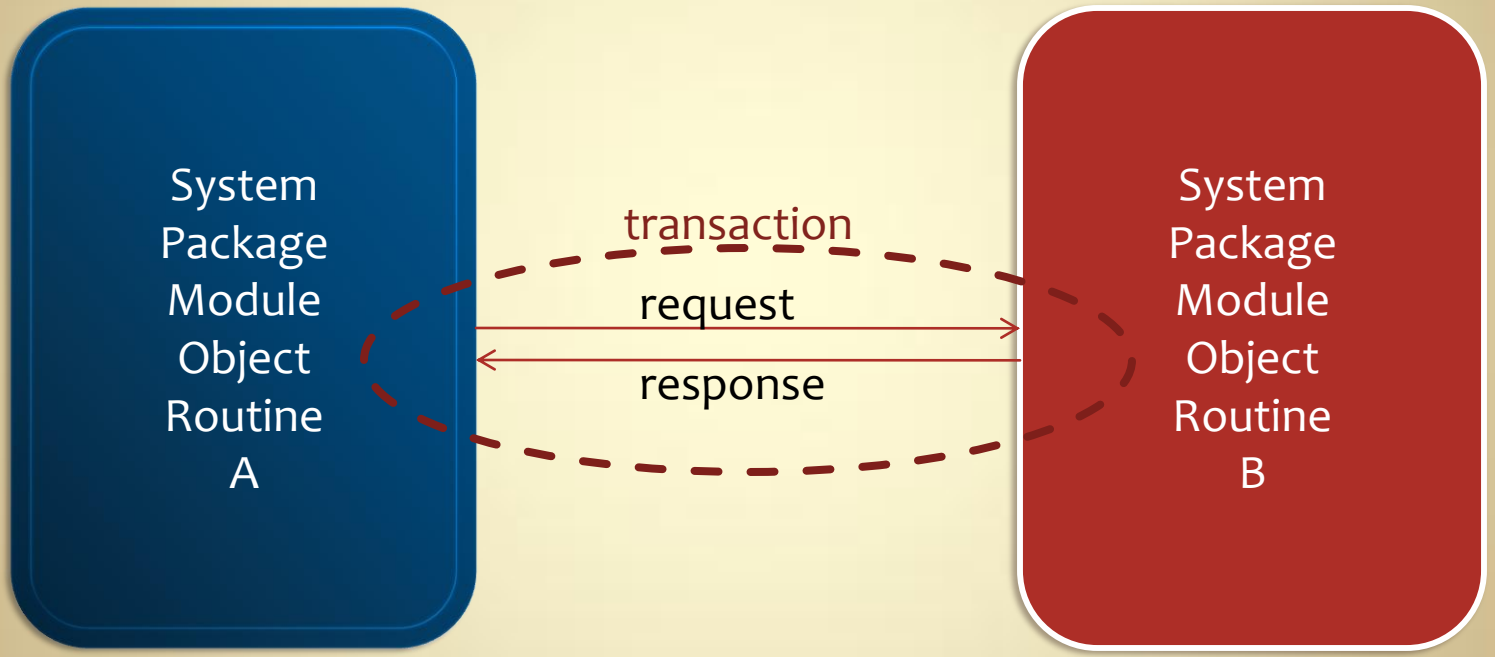
Agenda

- Asynchronous Processing vs. Synchronous
- Middleware
  - CPI-C, RPC, MOM
  - MQM
- Oracle Streams AQ
  - History and Features
  - Setup
  - Design
  - Create
  - Use (Enqueue and Dequeue)
  - Maintain & Troubleshoot
- >> Hard Lessons <<

Synchronous Processing

- Typical communication model employed in most programming languages
- Call and wait
  - Similar to live, interactive phone call
- Structured
  - Routine A calls Routine B, which queries the database and returns control to Routine A
- OO
  - ObjectA.method1 sends a message to an ObjectB.method3, which inspects the data it controls, and returns an answer to ObjectA

Synchronous Processing



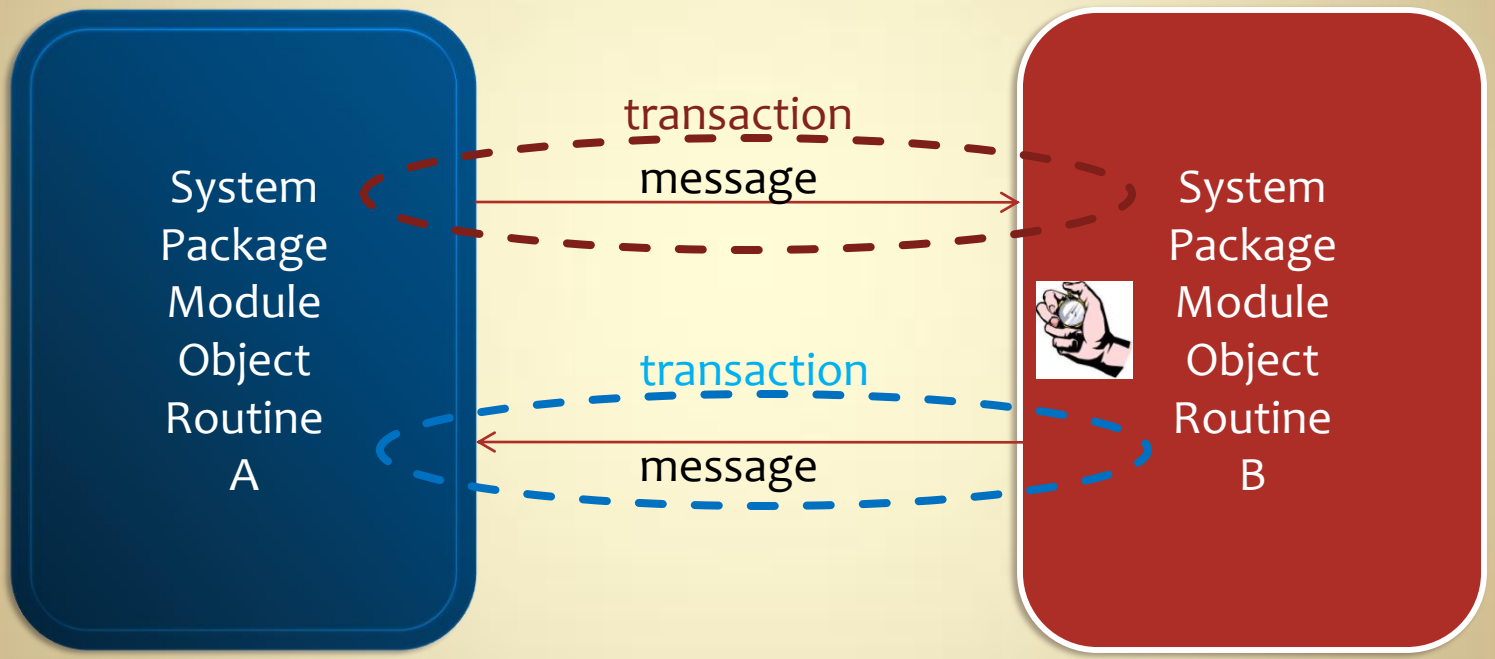
Problems with Syncronous

- Dependencies on undependable things
  - Length of execution
  - Uncertainty of completion
- Event-driven processes
  - Sensitive to response time
- Transaction management
  - Lost work if trouble on other end
- Resource usage
  - Idle time, resources wasted while waiting (either end)

Asynchronous Processing

- No hard link to the remote resource
- Leave message and hang-up
  - Callee will return call when they can
  - Similar to leaving a message in voicemail
- Structured and OO Programming:
  - Client sends message and goes on with life
  - Message receiver eventually processes the message and leaves a message for the client in return.
- Great for things like workflows, publish/subscribe communication/notification, progress meters, email handlers and more.

Asynchronous Processing



Asynchronous Processing

- Other end can be slow, undependable; no longer affects our end
- Event-driven processes
  - Now have the appearance of responsiveness as work was delegated
- Transaction management
  - Previous work retained if other end fails
- Resource usage
  - Resources efficiently utilized

Agenda

- Asynchronous Processing vs. Synchronous
- Middleware
  - CPI-C, RPC, MOM
  - MQM
- Oracle Streams AQ
  - History and Features
  - Setup
  - Design
  - Create
  - Use (Enqueue and Dequeue)
  - Maintain & Troubleshoot
- >> Hard Lessons <<

Middleware

- CPI-C
  - Common Programming Interface for Communication
  - Older. Mainframe and minis. MVS, OS/400, OS/2
- RPC
  - Remote Procedure Call
  - OO: Known as remote invocation
  - Slightly less old. Unix, Microsoft, CORBA, others
- MOM
  - Message Oriented Middleware
  - Newer. Many vendors and flavors and implementations
  - MQM is most popular flavor of MOM
    - Message Queuing Middleware

Agenda

- Asynchronous Processing vs. Synchronous
- Middleware
  - CPI-C, RPC, MOM
  - MQM
- Oracle Streams AQ
  - History and Features
  - Setup
  - Design
  - Create
  - Use (Enqueue and Dequeue)
  - Maintain & Troubleshoot
- >> Hard Lessons <<

Oracle Streams AQ

- Oracle's MQM solution
  - Implemented using… what else?...the Oracle database
  - Inherits the security, backup, transactional integrity, scheduling and other benefits of using the world's best database
- Oracle Advanced Queuing (8.0)
  - Queue Monitor processes (ora_qmn_* processes)
  - Job_queue_processes manually set
- Oracle Streams AQ (10.1)
  - Queue Monitor Coordinator (ora_qmnc_* processes)
  - Automatically adjusted

Oracle Streams AQ

- Single-consumer queues
- Multi-consumer queues (for pub/sub)
- Non-persistent messages (now called buffered)
- Message ordering, prioritization, grouping, navigation, selection, inspection, delay, retention, and expiration
- SQL-based access to queue, message metadata, message payload
- Various interfaces including PL/SQL, C++ and Java
- Rich payload typing model. Scalar, user-defined and XML.
- Much, much more

Agenda

- Asynchronous Processing vs. Synchronous
- Middleware
  - CPI-C, RPC, MOM
  - MQM
- Oracle Streams AQ
  - History and Features
  - Setup
  - Design
  - Create
  - Use (Enqueue and Dequeue)
  - Maintain & Troubleshoot
- >> Hard Lessons <<

AQ Setup

- AQ already installed and free to use
- As DBA…
  - QSCHEMA wants to create a queue

```
GRANT EXECUTE ON sys.dbms_aqadm TO QSCHEMA;
GRANT EXECUTE ON sys.dbms_aq TO QSCHEMA;
```

  - CUSTSCHEMA wants to enqueue

```
GRANT EXECUTE ON sys.dbms_aq TO CUSTSCHEMA;
```

  - CLIENTSCHEMA wants to dequeue

```
GRANT EXECUTE ON sys.dbms_aq TO CLIENTSCHEMA;
```

  - If app/svc connected to CLIENTSCHEMA will use JMS

```
GRANT EXECUTE ON sys.dbms_aqin TO CLIENTSCHEMA;
GRANT EXECUTE ON sys.dbms_aqjms TO CLIENTSCHEMA;
```

Design

- Design message payload
  - Identifiers
  - Content and format
- Design queue
  - Payload type?
  - How many messages per minute/hour/day? Spikes?
  - Multiple clients allowed to pull the message?
  - How to handle errors? Notify anyone?
  - Retries allowed? How many?
  - Delay needed to fix problems?
  - Is Oracle RAC involved?
  - Need to browse or inspect messages?
  - Grouping, sorting, tagging, priority needed?

Create

(as QSCHEMA)

1. Create queue table
2. Create queue
3. Start queue
4. Grant enqueue/dequeue permissions

Create: Start Clean

- Cleanup Script

```
SET SERVEROUTPUT ON
DECLARE
   l_queue_name VARCHAR2(30) := 'MY_Q';
   l_queue_table_name VARCHAR2(30) := 'MY_SQT';
   lx_queue_is_not EXCEPTION;
   lx_queue_running EXCEPTION;
   lx_queue_tab_is_not EXCEPTION;
   PRAGMA EXCEPTION_INIT(lx_queue_is_not,-24010);
   PRAGMA EXCEPTION_INIT(lx_queue_running,-24011);
   PRAGMA EXCEPTION_INIT(lx_queue_tab_is_not,-24002);
BEGIN
   BEGIN
      dbms_aqadm.drop_queue(queue_name => l_queue_name);
   EXCEPTION
      WHEN lx_queue_is_not THEN
         dbms_output.put_line(l_queue_name||' does not exist. Check spelling.');
      WHEN lx_queue_running THEN
         dbms_output.put_line('Stopping '||l_queue_name);
         dbms_aqadm.stop_queue(queue_name => l_queue_name);
         dbms_output.put_line('Dropping '||l_queue_name);
         dbms_aqadm.drop_queue(queue_name => l_queue_name);
   END;
   BEGIN
      dbms_aqadm.drop_queue_table(queue_table => l_queue_table_name, force=>TRUE);
   EXCEPTION
      WHEN lx_queue_tab_is_not THEN
         dbms_output.put_line(l_queue_table_name||' does not exist. Check spelling.');
   END;
END;
```

Create: Queue Table

- Create queue table

```
BEGIN
    dbms_output.put_line('Creating MY_SQT');
    dbms_aqadm.create_queue_table(
       queue_table         => 'MY_SQT'
      ,queue_payload_type => 'SYS.AQ$_JMS_MESSAGE'
      ,storage_clause => 'PCTFREE 0 PCTUSED 99'
      ,multiple_consumers => FALSE
      ,comment => 'My Queue Table: Supports the blah,
blah...');
END;
```

Create: Queue

- Create queue and start it
- Name limited to 24 characters

```
BEGIN
   dbms_output.put_line('Creating MY_Q');
   dbms_aqadm.create_queue(
      queue_name => 'MY_Q'
     ,queue_table => 'MY_SQT'
     ,comment => 'My Queue: Routes the messages
from...');

   dbms_aqadm.start_queue(queue_name=>'MY_Q');
END;
```

Create

- That's it! You now have a running queue, waiting for messages.
- In addition, Oracle created two "hidden" views on top of your queue table:
  - AQ$*queue_table*
    - Very useful for monitoring and maintenance
    - Nice to grant to schemas and roles that need to peer into queue
  - AQ$*queue_table*_F
    - Not sure why it exists… yet. No documentation.

Create: Grant Privileges

- In order for anyone else to use the queue, permissions must be granted.

```
BEGIN
   dbms_output.put_line('Granting enqueue privs');
   dbms_aqadm.grant_queue_privilege(
      privilege    => 'ENQUEUE' -- also DEQUEUE or ALL
     ,queue_name   => 'MY_Q'
     ,grantee      => 'CUSTSCHEMA'
     ,grant_option => FALSE);
END;

BEGIN
   dbms_output.put_line('Granting dequeue privs');
   dbms_aqadm.grant_queue_privilege(
      privilege    => 'DEQUEUE'
     ,queue_name   => 'MY_Q'
     ,grantee      => 'CLIENTSCHEMA'
     ,grant_option => FALSE);
END;
```

Use: Enqueue

- Now use the appropriate programmatic interface to enqueue or dequeue
- PL/SQL example (as CUSTSCHEMA):

```
DECLARE
   l_msg            sys.aq$_jms_message;
   l_queue_options  dbms_aq.enqueue_options_t;
   l_msg_props      dbms_aq.message_properties_t;
   l_msg_id         RAW(16);
BEGIN
   l_msg :=
sys.aq$_jms_message.construct(dbms_aqjms.jms_text_message);
   l_msg.set_text('<useful message here>');
   dbms_aq.enqueue(
      queue_name          => 'QSCHEMA.MY_Q'
     ,enqueue_options     => l_queue_options
     ,message_properties  => l_msg_props
     ,payload             => l_msg
     ,msgid               => l_msg_id);
   COMMIT; -- very important; won't enqueue without commit
END;
```

Use: Dequeue

- Pulls the first message off the queue by default
- Many modes and options and design decisions here
  - By query, by identifiers, by grouping, browse mode, etc.
- Will rarely see messages in the queue table
  - Unless dequeue transaction is failing
  - Or sender requested a dequeue delay
  - Or table created with retry_delay value
- Messages will be READY, PROCESSED or EXPIRED
- Dequeue request is a blocking operation

Use: Dequeue

- Using the PL/SQL API:

```
DECLARE
   l_msg            sys.aq$_jms_message;
   l_msg_text       VARCHAR2(100);
   l_queue_options  dbms_aq.dequeue_options_t;
   l_msg_props      dbms_aq.message_properties_t;
   l_msg_id         RAW(16);
BEGIN

   dbms_aq.dequeue(
      queue_name          => 'QSCHEMA.MY_Q'
     ,dequeue_options     => l_queue_options
     ,message_properties  => l_msg_props
     ,payload             => l_msg
     ,msgid               => l_msg_id);

   l_msg.get_text(l_msg_text);
   dbms_output.put_line('Dequeued message text: ' ||
      CHR(10) || l_msg_text);
   COMMIT;
END;
```

Agenda

- Asynchronous Processing vs. Synchronous
- Middleware
  - CPI-C, RPC, MOM
  - MQM
- Oracle Streams AQ
  - History and Features
  - Setup
  - Design
  - Create
  - Use (Enqueue and Dequeue)
  - Maintain & Troubleshoot
- >> Hard Lessons <<

Maintaining a Queue

- Queues and queue tables are self-maintaining
- You can stop a queue and alter it
- Administer through OEM and DBMS_AQADM

- Will generally be empty, unless nothing is dequeuing, or dequeue transactions are failing
- If not empty, the system doing the dequeue must be investigated, not the queue

Troubleshooting a Queue

- Useful message metadata in AQ$*queue_table* view
  - Can query, but cannot perform DML on the queue table

```
SELECT queue
      ,enq_timestamp
      ,enq_user_id
      ,msg_state
      ,retry_count
      ,original_queue_name
      ,expiration_reason
      ,user_data
  FROM aq$my_sqt t
 ORDER BY 2 DESC;
```

```
SELECT t.queue
      ,t.enq_timestamp
      ,t.enq_user_id
      ,t.msg_state
      ,t.retry_count
      ,t.original_queue_name
      ,t.expiration_reason
      -- good to convert if message is numeric
      ,TO_NUMBER(t.user_data.text_vc) customer_id
  FROM aq$my_sqt t
 ORDER BY 2 DESC;
```

- Oracle data dictionary queue views, like [G]V$AQ, user/all/dba_queues and user/all/dba_queue_tables

```
SELECT dq.owner
      ,dq.name
      ,dq.queue_type
      ,g.*
  FROM gv$aq g
  JOIN dba_queues dq
    ON dq.qid = g.qid;
```

Troubleshooting a Queue

- Expired or failed messages moved to the exception queue, a queue table created by Oracle and named AQ$*queue_table*_E
- Cannot enqueue directly to exception queue
- But can dequeue from it, allowing one to re-process or re-enqueue failed messages
  - Must formally "start" it and enable dequeuing

```
BEGIN
   -- Start the default exception queue as well so we can dequeue from it.
   dbms_output.put_line('Starting AQ$_MY_SQT_E exception Q');
   dbms_aqadm.start_queue(queue_name => 'AQ$_MY_SQT_E', enqueue => FALSE, dequeue => TRUE);
END;
```

Troubleshooting a Queue

- After that, the queue table view can tell us about messages that are now in exception
  - Using the query seen 2 slides ago:

	QUEUE	ENQ_TIMESTAMP	ENQ_USER_ID	MSG_STATE	RETRY_COUNT	ORIGINAL_QUEUE_NAME	EXPIRATION_REASON	CUSTOMER_ID
1	SCRN_LSNR_Q	06-JAN-11 04.24.26.469421 PM	SCRN_PTC	READY	0			312449
2	SCRN_LSNR_Q	06-JAN-11 04.14.39.007230 PM	SCRN_PTC	READY	0			330941
3	SCRN_LSNR_Q	06-JAN-11 01.25.39.578126 PM	SCRN_PTC	READY	0			336028
4	SCRN_LSNR_Q	06-JAN-11 12.32.44.070731 PM	SCRN_PTC	READY	0			344285
5	SCRN_LSNR_Q	06-JAN-11 12.05.35.649095 PM	SCRN_PTC	READY	0			341867
6	SCRN_LSNR_Q	06-JAN-11 11.33.36.130970 AM	SCRN_PTC	READY	1			341702
7	SCRN_LSNR_Q	06-JAN-11 11.26.18.746358 AM	SCRN_PTC	READY	1			282635
8	AQ$_SCRN_LSNR_SQT_E	01-DEC-10 03.13.24.340569 PM	ICSAJA	EXPIRED	4	SCRN_LSNR_Q	MAX_RETRY_EXCEEDED	348527
9	AQ$_SCRN_LSNR_SQT_E	08-NOV-10 03.23.52.487292 PM	SCRN_PTC	EXPIRED	4	SCRN_LSNR_Q	MAX_RETRY_EXCEEDED	345735

Troubleshooting a Queue

- Tried notify/fix on entry into exception queue. Fail.
- Prefer to trap, notify and fix problem messages during the retry_delay * max retries window
  - We had the need to know about errors the second they happened.
  - We attached an after update trigger to the queue table that looks for any change in retry_count, and sends an email with message context.

```
CREATE OR REPLACE TRIGGER my_sqt_au_trg
   AFTER UPDATE ON my_sqt
   FOR EACH ROW
DECLARE
BEGIN

   IF (:old.retry_count <> :new.retry_count) THEN
      my q_mgr.handle_retry(:old.user_data.text_vc);
   END IF;

END my_sqt_au_trg;
```

  - Created a package for this notification routine, and other common queue-related operations.

<switch to PL/SQL Developer to show package>

Agenda

- Asynchronous Processing vs. Synchronous
- Middleware
  - CPI-C, RPC, MOM
  - MQM
- Oracle Streams AQ
  - History and Features
  - Setup
  - Design
  - Create
  - Use (Enqueue and Dequeue)
  - Maintain & Troubleshoot
- >> Hard Lessons <<

Hard Lessons

"Too Many Cooks in the Kitchen"

- Lots of developers running local Tomcat with copy of the app, each with their own listener dequeuing from the same queue on the shared Dev database.
- Random who ended up with the message
- Failures would retry the default 5 times < 1 second and immediately go to exception. Default delay is 0 seconds. No time to diagnose. Frustrating.
- We bumped delay to 3600 seconds, and limited to 4 attempts:

```
BEGIN
   dbms_aqadm.alter_queue(
      queue_name => 'MY_Q'
     ,max_retries => 4
     ,retry_delay => 3600);
END;
```

Hard Lessons

"Double the Fun!"

- Basic tenet of queuing is that each message will be processed once and only once. In 10.2.0.4, try twice and often twice!
- Bug (5590163) in Oracle allows messages in our single-consumer queue to be dequeued twice.
- Logs showed the two nodes of the app server each dequeuing same message in same second.
- Processing didn't see the other transaction, and tried to create duplicate records in downstream system.
- AQ was acting like it had never heard of ACID transactions.
- Oracle's "fix" created bug 7393292. Truly fixed in 10.2.0.5?

Hard Lessons

"Crusty Queue"

- Our system dequeuing did too much: too many queries and DML statements before deciding to finish the transaction. Too much stuff to go wrong.
- Lots of errors during initial months of dev and testing. Queue table became encrusted with old, failed messages. Needed to clean it out.
- Purge with DBMS_AQADM interface:

```
DECLARE
   l_purge_opt dbms_aqadm.aq$_purge_options_t;
BEGIN
   l_purge_opt.block := TRUE;
   dbms_aqadm.purge_queue_table(
      queue_table     => 'MY_SQT'
     ,purge_condition => 'queue IN (''AQ$_MY_SQT_E'',''MY_Q'')'
     ,purge_options   => l_purge_opt);
END;
```

Hard Lessons

"Crusty Queue"

- Also possible to pinpoint the messages to remove using the purge_condition parameter, which operates on the columns found in the queue table.
- Alias "qtview." required for access to attributes of the user_data column.

```
DECLARE
   l_purge_opt dbms_aqadm.aq$_purge_options_t;
BEGIN
   l_purge_opt.block := FALSE; -- don't block enqueue or dequeue attempts (this is the default)
   dbms_aqadm.purge_queue_table(
      queue_table     => 'MY_SQT'
     ,purge_condition => 'queue = ''MY_Q'' AND qtview.user_data.text_vc = ''hello world'''
     ,purge_options   => l_purge_opt
   );
END;
```

Hard Lessons

"F view Fail"

- Different project got error on dequeue:

```
ORA-00942 table or view does not exist at this DBMS: sys.DBMS_AQIN line 651
```

- Run as queue owner: Good
- Run as other schema accessing the queue: Error
- Had to run trace to find missing priv
- Found that if the system dequeues in BROWSE mode, the queue owner must grant SELECT access on the AQ$*queue_table*_F view to dequeuing schema.

Hard Lessons

"AQ$_JMS_MESSAGE Massage"

- During upgrade project, half DBs 10g, other half 9i.
- Found that enqueue script written for 10g didn't work on 9i.
- Turns out AQ$_JMS_MESSAGE has multiple constructors in 10g, and only one in 9i.
- 9i version that takes an integer (message type constants defined in DBMS_AQ package spec) worked great on both versions.
- 10g constructors can accept a variable of the message type, like SYS.AQ$_JMS_TEXT_MESSAGE, but is more complex to use (3 more lines of code)

Questions?

- Bill Coulam
  - bcoulam@yahoo.com
  - http://www.dbartisans.com

  - Open Source PL/SQL “Starter” Application Framework
    - http://plsqlframestart.sourceforge.net