The On-Commit Database Trigger

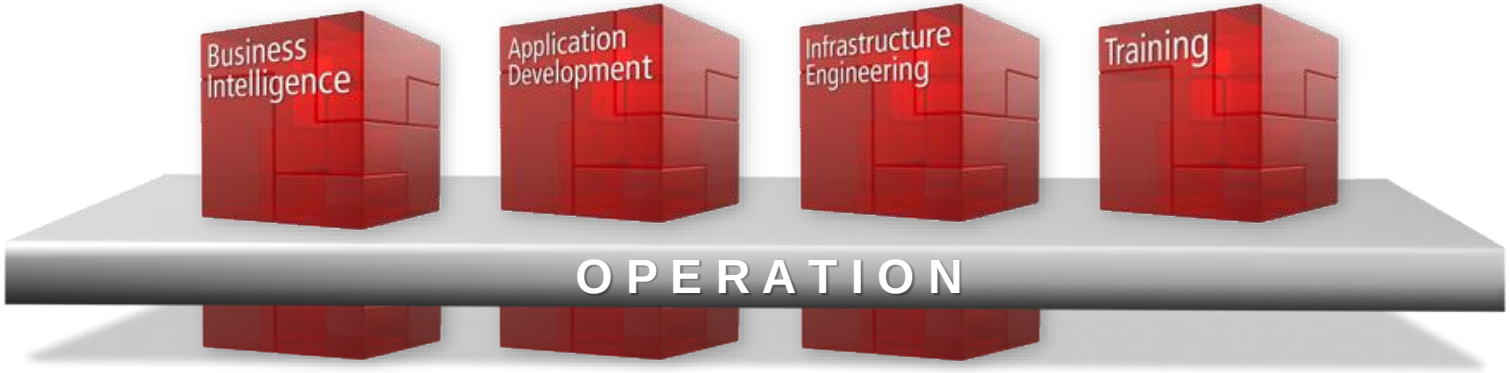
Philipp Salvisberg

@phsalvisberg # DOAG2017

trivadis
makes IT easier.

Our Company

Trivadis is a market leader in IT consulting, system integration, solution engineering and the provision of IT services focusing on ORACLE and Microsoft technologies
in Switzerland, Germany, Austria and Denmark. We offer our services in the following strategic business fields:



Trivadis Services takes over the interacting operation of your IT systems.

trivadis makes IT easier.

With over 600 Specialists and IT Experts in Your Region

- 14 Trivadis branches and more than 600 employees
- 200 Service Level Agreements
- Over 4,000 training participants
- Research and development budget: CHF 5.0 million
- Financially self-supporting and sustainably profitable
- Experience from more than 1,900 projects per year at over 800 customers

trivadis makes IT easier.

About Me

- Trivadian since April 2000
  - Senior Principal Consultant, Partner
  - Member of the Board of Directors
  - @phsalvisberg
  - https://www.salvis.com/blog
  - https://github.com/PhilippSalvisberg
- Database centric development with Oracle database
- Model Driven Software Development
- Author of free SQL Developer Extensions PL/SQL Unwrapper, PL/SQL Cop, plscope-utils, oddgen and Bitemp Remodeler

trivadis makes IT easier.

Agenda

1. Use Case
2. Message Queuing Essentials
3. Database Setup for PL/SQL and JMS
4. PL/SQL Application Using JMS
5. Java Application Using JMS
6. Core Messages

Use Case

Transactional Notification of Salary Changes

ENAME	JOB	SAL
SMITH	CLERK	800
ALLEN	SALESMAN	1700
WARD	SALESMAN	1350
JONES	MANAGER	2975
MARTIN	SALESMAN	1550
BLAKE	MANAGER	2850
CLARK	MANAGER	2480
SCOTT	ANALYST	3200
KING	PRESIDENT	5000
…	…	…

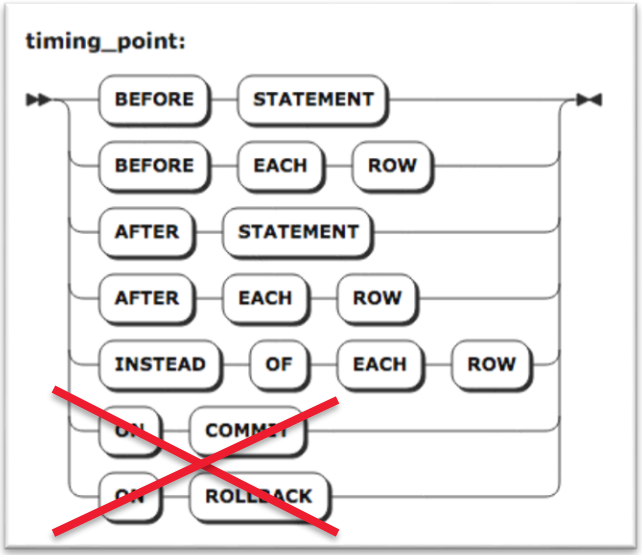
- Update emp set sal = sal * 2;
- Rollback;
- Update emp set sal = sal + 100 where job = 'SALESMAN';
- Update emp set sal = sal + 200 where ename in ('MARTIN' ,'SCOTT');
- Commit;
- Notify aggregated changes, e.g.
  - +300 for MARTIN
  - +200 for SCOTT

"Desired" Implementation

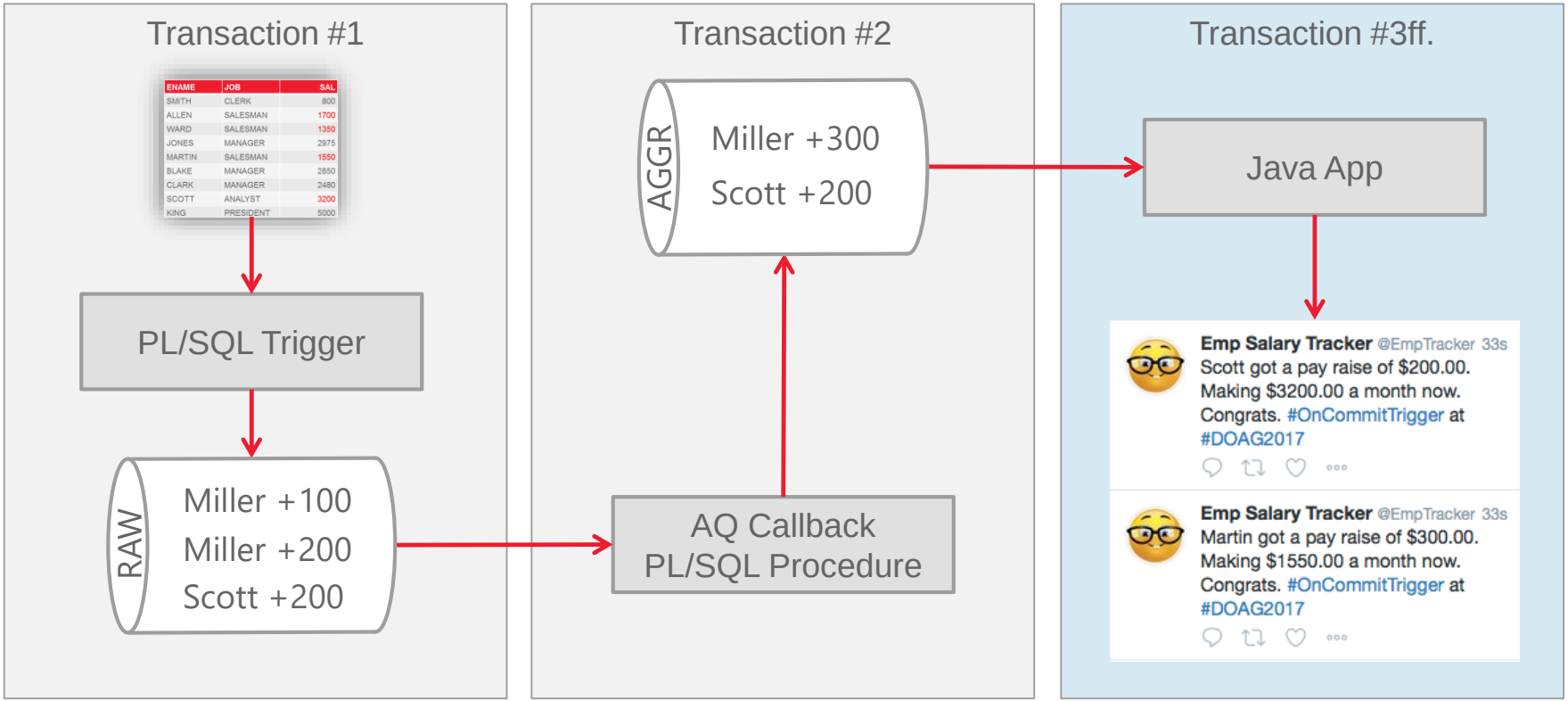
```
CREATE OR REPLACE TRIGGER emp_trg FOR UPDATE OF sal ON emp
   COMPOUND TRIGGER
      t_emp_log t_emp_log_type := t_emp_log_type();

   AFTER EACH ROW IS
   BEGIN …
   END AFTER EACH ROW;

   ON COMMIT IS
   BEGIN
      FOR r IN (…)
      LOOP
         emp_notify(emp_log_type(r.log_id,
            r.ename, r.old_sal, r.new_sal));
      END LOOP;
      t_emp_log.delete;
   END ON COMMIT;
END;
```



Solution Architecture



Why Oracle Advanced Queuing (AQ)?

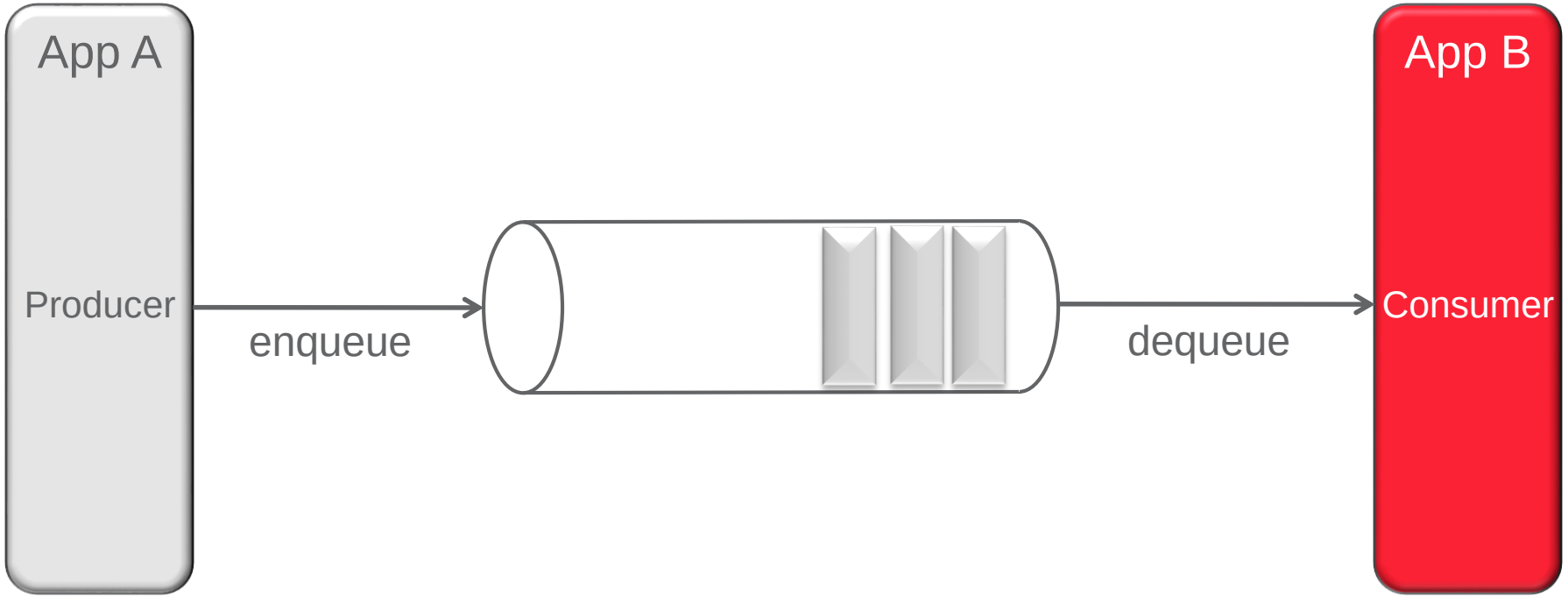
- Transactions
  - Message processing is part of the database transaction
- Asynchronous
  - Limit the impact on original transaction (performance, robustness)
- Control
  - Limit the number of concurrent dequeue processes
  - Set message priority to override default FIFO behaviour
- Comprehensive API
  - PL/SQL and Java (JMS)

HTTP Calls from PL/SQL in Transaction #3ff.?

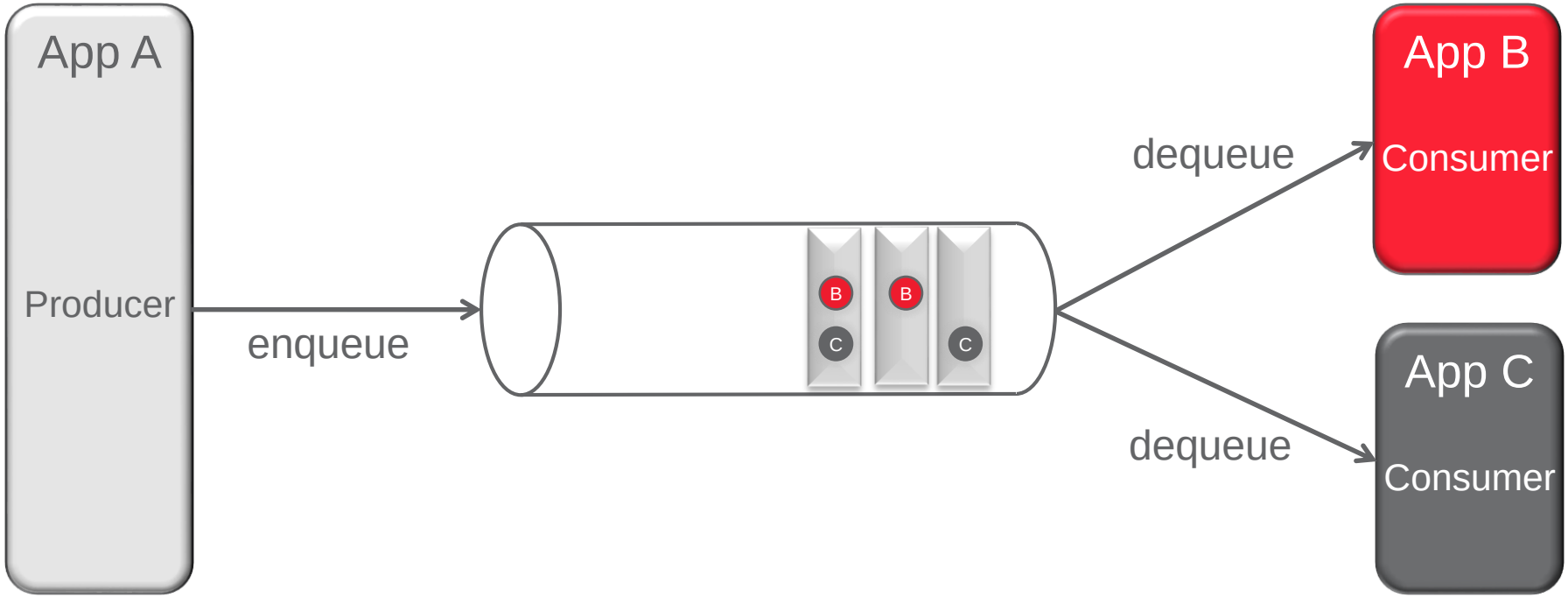
- Possible
  - But not necessarily simpler
- Options
  - UTL_HTTP, low-level API, challenging over SSL
  - APEX_WEB_SERVICE, easy to use after network ACL setup
- No options
  - UTL_DBWS, challenging to install and buggy (e.g. Bug 18705437 : UTL_DBWS REQUESTS LARGER THAN 65599 ARE TRUNCATED)
  - HttpUriType, super easy to use, but limited to GET calls over HTTP

Message Queuing Essentials

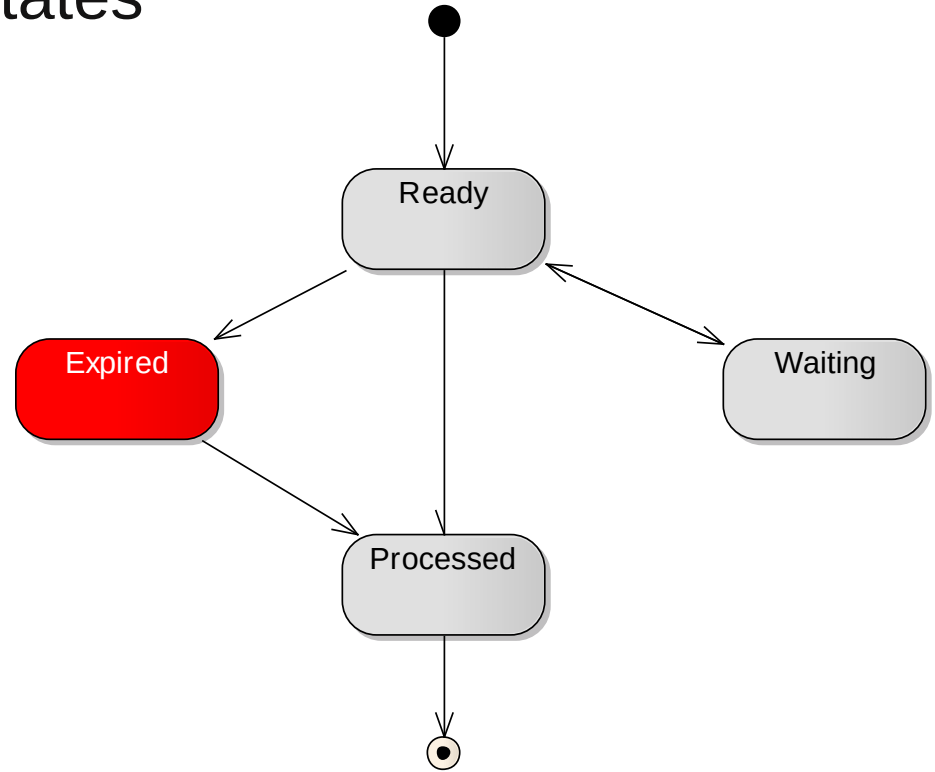
Single Consumer (Point-to-Point / Queue)



Multi-Consumer (Publish-Subscribe / Topic)



Message States*

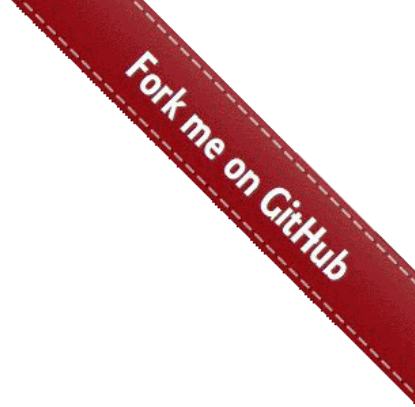


* = additional sub-states exist, e.g. for ready: “in memory”, “deferred”, “spilled”, “deferred spilled”

Database Setup for PL/SQL and JMS

Source Code

- Examples in this presentation are simplified code excerpts
- Complete source code is on GitHub
  - https://github.com/PhilippSalvisberg/emptracker



Create Queue Tables

```
BEGIN
   dbms_aqadm.create_queue_table (
      queue_table         => 'REQUESTS_QT',
      queue_payload_type => 'SYS.AQ$_JMS_TEXT_MESSAGE',
      sort_list           => 'PRIORITY,ENQ_TIME',
      multiple_consumers => TRUE,
      message_grouping    => dbms_aqadm.transactional
   );
   dbms_aqadm.create_queue_table (
      queue_table         => 'RESPONSES_QT',
      queue_payload_type => 'SYS.AQ$_JMS_TEXT_MESSAGE',
      sort_list           => 'PRIORITY,ENQ_TIME',
      multiple_consumers => TRUE
   );
END;
```

sys.aq$_jms_text_message ≈ javax.jms.TextMessage

- header — aq$_jms_header
  - replyto — sys.aq$_agent
    - name — varchar2(10)
    - address — varchar2(1024)
    - protocol — number
  - type — varchar(100)
  - userid — varchar(100)
  - appid — varchar(100)
  - groupid — varchar(100)
  - groupseq — int
  - properties — varray(100) of aq$_jms_userproperty
    - name — varchar(100)
    - type — int
    - str_value — varchar(2000)
    - num_value — number
    - Java_type — int

- text_len — int
- text_vc — varchar2(4000)
- text_lob — clob

Some member procedures/functions:

- set_text / get_text
- set_replyto / get_replyto
- set_string_property / get_string_property
- set_int_property / get_int_property
- set_long_property / get_long_property
- set_double_property / get_double_property

Create Queues

```
BEGIN
   dbms_aqadm.create_queue (
      queue_name          => 'REQUESTS_AQ',
      queue_table         => 'REQUESTS_QT',
      max_retries         => 1,
      retry_delay         => 2, -- seconds
      retention_time      => 60*60*24*7 -- 1 week
   );
   dbms_aqadm.create_queue (
      queue_name          => 'RESPONSES_AQ',
      queue_table         => 'RESPONSES_QT',
      max_retries         => 1,
      retry_delay         => 2, -- seconds
      retention_time      => 60*60*24*7 -- 1 week
   );
END;
```

Start Queues

```
BEGIN
   dbms_aqadm.start_queue(
      queue_name => 'REQUESTS_AQ'
     ,enqueue    => TRUE
     ,dequeue    => TRUE
   );
   dbms_aqadm.start_queue(
      queue_name => 'RESPONES_AQ'
     ,enqueue    => TRUE
     ,dequeue    => TRUE
   );
END;
```

PL/SQL Application Using JMS

Subscribe Topics

```
BEGIN
   -- for messages created in trx #1 to be processed in trx #2
   dbms_aqadm.add_subscriber(
      queue_name => 'requests_aq',
      subscriber => sys.aq$_agent('RAW_ENQ', 'REQUESTS_AQ', 0),
      rule => q'[tab.user_data.get_string_property('msg_type') = 'RAW']'
   );
   -- for response/logging messages produced in trx #2 and #3ff.
   dbms_aqadm.add_subscriber(
      queue_name => 'responses_aq',
      subscriber => sys.aq$_agent('ALL_RESPONSES', 'RESPONSES_AQ', 0)
   );
END;
```

Database Trigger

```
CREATE OR REPLACE TRIGGER emp_au_trg
   AFTER UPDATE OF sal ON emp FOR EACH ROW
DECLARE
   PROCEDURE enqueue IS …
      l_jms_message     sys.aq$_jms_text_message :=
                           sys.aq$_jms_text_message.construct;
   BEGIN …
      sys.dbms_aq.enqueue(queue_name         => 'requests_aq',
                          enqueue_options    => l_enqueue_options,
                          message_properties => l_message_props,
                          payload            => l_jms_message,
                          msgid              => l_msgid); …
BEGIN
   IF :old.sal != :new.sal THEN
      enqueue;
   END IF;
END;
```

Register Callback Procedure

```
BEGIN
   -- call raw_enq_callback procedure for every message produced in trx #1
   dbms_aq.register(
      reg_list => sys.aq$_reg_info_list(
                     sys.aq$_reg_info(
                        name      => 'REQUESTS_AQ:RAW_ENQ',
                        namespace => dbms_aq.namespace_aq,
                        callback  => 'plsql://raw_enq_callback?PR=0',
                        context   => NULL
                     )
                  ),
      reg_count => 1
   );
END;
```

Callback Procedure

```
CREATE OR REPLACE PROCEDURE raw_enq_callback (
   context  IN RAW,
   reginfo  IN SYS.AQ$_REG_INFO,
   descr    IN SYS.AQ$_DESCRIPTOR, -- queue_name, consumer_name, msg_id, …
   payload  IN RAW,
   payloadl IN NUMBER
) IS
   …
BEGIN
   dequeue_all;
   copy_id_to_jms_messages;
   enqueue_aggr_messages;
EXCEPTION
   …
END raq_enq_callback;
```

```
WITH
   msg AS (
      SELECT m.get_int_property('id') AS id,
             m.get_string_property('ename') AS ename,
             m.get_double_property('old_sal') AS old_sal,
             m.get_double_property('new_sal') AS new_sal
        FROM TABLE(t_jms_message) m
   ),
   base AS (
       SELECT MAX(id) OVER(PARTITION BY ename) AS id,
              ename,
              FIRST_VALUE(old_sal) OVER(PARTITION BY ename ORDER BY id) AS old_sal,
              LAST_VALUE(new_sal) OVER(PARTITION BY ename ORDER BY id
                 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS new_sal
         FROM msg
   )
SELECT DISTINCT id, ename, old_sal, new_sal
  FROM base
 WHERE old_sal != new_sal
```

Synchronous Call

```
CREATE OR REPLACE FUNCTION tweet(in_text IN VARCHAR2) RETURN VARCHAR2 IS
   PRAGMA AUTONOMOUS_TRANSACTION; …
FUNCTION dequeue(in_correlation IN VARCHAR2) RETURN VARCHAR2 IS …
   l_dequeue_options.wait           := 5;
   l_dequeue_options.correlation    := in_correlation; …
   dbms_aq.dequeue(dequeue_options => l_dequeue_options, …);
   l_jms_message.get_text(l_text);
   l_msg_type := l_jms_message.get_string_property('msg_type');
   COMMIT;
   RETURN l_msg_type || ': ' || l_text; …
END dequeue;
BEGIN
   l_correlation := enqueue(in_text => in_text);
   RETURN dequeue(in_correlation => l_correlation);
END tweet;
```

Java Application Using JMS

Spring Boot Application

Define application

```
@SpringBootApplication
public class EmptrackerApplication {
   public static void main(String[] args) {
      SpringApplication.run(EmptrackerApplication.class, args);
   }
}
```

Run it

```
mvn spring-boot:run
```

Application Properties

```
# EmpTracker
db.url=jdbc:oracle:thin:@localhost:1521:odb
db.user=emptracker
db.password=emptracker
…

# Logging
logging.level.com.salvis.emptracker=INFO
logging.level.org.springframework=ERROR
logging.level.twitter4j=INFO

# Twitter account, default in ${user.home}/emptracker.properties
# create OAuth keys via https://apps.twitter.com/
#twitter4j.oauth.consumerKey=
#twitter4j.oauth.consumerSecret=
#twitter4j.oauth.accessToken=
#twitter4j.oauth.accessTokenSecret=
```

Message Listener Container

```
@Configuration public class AppConfig { …
   @Bean public DefaultMessageListenerContainer messageListenerContainer() {
      DefaultMessageListenerContainer cont = new DefaultMessageListenerContainer();
      cont.setMessageListener(new TextMessageListener());
      cont.setConnectionFactory(
         AQjmsFactory.getTopicConnectionFactory(messageDataSource()));
      cont.setDestinationName("requests_aq");
      cont.setPubSubDomain(true);
      cont.setSubscriptionName("EmpTracker");
      cont.setSubscriptionDurable(true);
      cont.setMessageSelector("msg_type IN ('AGGR', 'TWEET')");
      cont.setSessionAcknowledgeMode(Session.SESSION_TRANSACTED);
      cont.setSessionTransacted(true);
      cont.setConcurrency("1-4");
      cont.setMaxMessagesPerTask(20);
      cont.setReceiveTimeout(10);
      return cont;
   } …
}
```

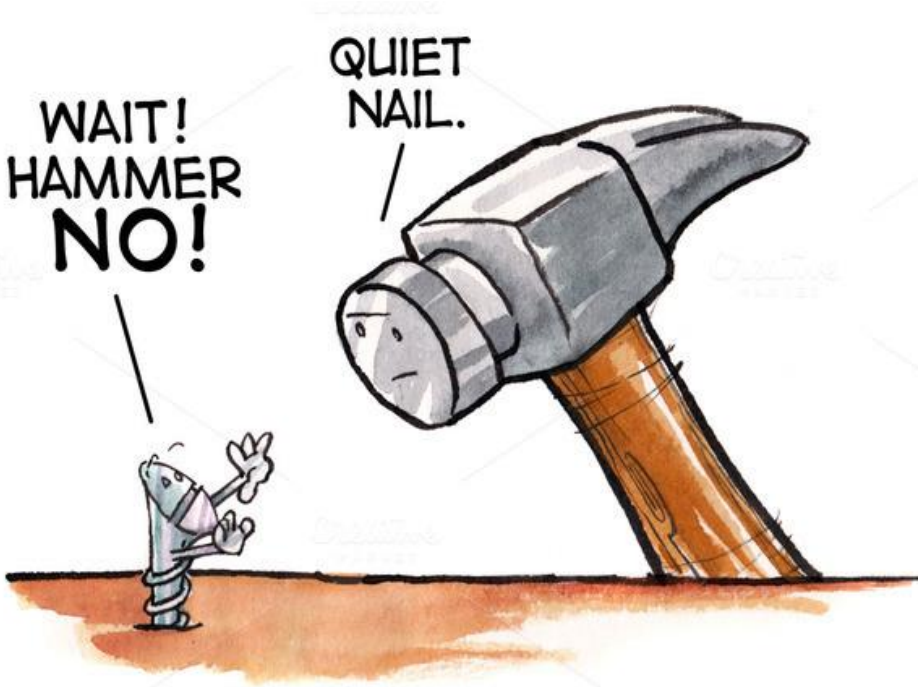
Message Listener


```
@Component
public class TextMessageListener implements SessionAwareMessageListener<TextMessage> {…
   public void onMessage(final TextMessage request, final Session session) {
      try {
         String text = request.getText();
         if (text == null || text.isEmpty()) {
            String ename = request.getStringProperty("ename");
            Double oldSal = request.getDoubleProperty("old_sal");
            Double newSal = request.getDoubleProperty("new_sal");
            text = getText(ename, oldSal, newSal);
         }
         Status status = twitter.updateStatus(text);
         String screenName = status.getUser().getScreenName();
         sendResponse(request, session, screenName + ": " + text, "INFO");
         session.commit();
      } catch (Exception e) {…}
   }
}
```

Core Messages

Just Because You Can Doesn't Mean You Should

- Consider carefully what belongs into an Oracle Database
- Oracle AQ is a cost free option and part of every database edition – just use it



Happy Queuing

- Make the granularity of a message a part of your design considerations, that's the key to improve overall messaging performance
- Use dedicated queue tables and queues for topics with high throughput requirements
- Messages are processed row by row, contention issues are not unusual, mitigation is similar to conventional tables (e.g. increase PCTFREE)
- Just be aware that AQ has a limited scalability

Questions and answers…

Philipp Salvisberg
Senior Principal Consultant

Tel. +41 58 459 52 31
philipp.salvisberg@trivadis.com

trivadis
makes IT easier.

Trivadis @ DOAG 2017 #OpenCompany

- Booth: 3rd Floor – next to the escalator
- We share our Know how!
  Just come across, Live-Presentations and documents archive
- T-Shirts, Contest and much more
- We look forward to your visit