Tipps & Tricks: Advanced Queuing

Bereich:	PL/SQL, DBA	Erstellung:	03/2003 HA
Versionsinfo:	8.1.7, 9.2, 10.2, 11.1	Letzte Überarbeitung:	05/2009 HA

Advanced Queuing

Advanced Queuing dient der Kommunikation, beispielsweise zwischen verschiedenen Applikationen, wobei die zu verschickenden Meldungen permanent in der Datenbank gespeichert werden. Sie verschwinden erst aus den zugehörigen Tabellen, wenn sie abgearbeitet wurden. Die Queues und zugehörigen Tabellen können sogar mit exportiert und importiert werden. Der Sender braucht daher keine Rückmeldung eines Empfängers abwarten, und der Empfänger kann die Nachrichten zu einer beliebigen Zeit abholen und weiterverarbeiten.
Eine erschöpfende Beschreibung der Möglichkeiten dieses Features würde den Rahmen dieses Tipps sprengen, deshalb wird hier nur der prinzipielle Weg der Implementierung vorgestellt.

Der Administrator, der den Queuing-Mechanismus implementiert, braucht entweder die Rolle AQ_ADMINISTRATOR_ROLE oder die darin enthaltenen Berechtigungen, soweit notwendig. Wichtig sind in diesem Zusammenhang vor allem die EXECUTE-Rechte auf DBMS_AQ und DBMS_AQADM sowie SELECT-Rechte auf DD-Views. Zum Versenden und Empfangen von Meldungen braucht ein User bzw. Anwendungsentwickler zumindest das EXECUTE-Recht auf DBMS_AQ.

Um Messages über Advanced Queuing austauschen zu können, sind folgende Schritte nötig:

1. Anlegen einer Queue table
2. Einrichten einer Queue in dieser Tabelle
3. Starten der Queue

Daneben muss mindestens ein SNP-Prozess laufen, d.h., der init.ora-Parameter job_queue_processes muss > 0 eingestellt sein.

Anlegen einer Queue Table

Angegeben werden müssen hier der Name der Tabelle und der Datentyp der zu versendenden Messages, i.a. Objekttypen.
Wird nichts weiter angegeben, so sind alle Queues dieser Tabelle sog. single consumer queues, d.h., genau ein Empfänger kann die Message verarbeiten. In diesem Fall braucht kein (impliziter oder expliziter) Empfänger der Message angegeben werden, und der Empfänger braucht sich auch nicht auszuweisen.

Alternativ kann der Parameter multiple_consumers auf TRUE gesetzt werden; dann kann es mehr als einen Empfänger geben. Diese werden entweder implizit ausgewiesen als sog. subscribers, also standardmäßige Empfänger aller Messages einer Queue (unten nicht gezeigt), oder explizit als sog. recipients, die direkt beim Versenden (enqueuing) einer Nachricht angegeben werden (unten gezeigt). Werden explizit recipients angegeben, so erhalten etwaige subscribers die Message nicht.
Hier muss sich der Empfänger namentlich ausweisen, um die Nachricht abrufen zu können.

Auf den Inhalt der Queue table kann mit normalen SELECT-Befehlen zugegriffen werden. Dies sollte aber nur zu Kontrollzwecken geschehen, nicht als Alternative zum Queuing-Mechanismus! Jede Queue table beinhaltet (mindestens) eine Exception queue, die automatisch zusammen mit der Tabelle angelegt wird.

Versenden von Nachrichten

Beim Starten einer Queue werden standardmäßig Equeuing (Versenden) und Dequeuing (Empfang) freigegeben; es kann aber auch explizit nur eins von beiden ermöglicht werden.

Nachrichten können zu Gruppen zusammengefasst werden, falls dies beim Anlegen der Tabelle ermöglicht wurde, und es können Prioritäten vergeben werden. Eine Weiterleitung auf eine andere Queue (Propagation) und ggf. eine Umwandlung des Datentyps (Transformation) können auch eingebaut werden.

Beim Versenden einer Nachricht kann u.a. auch eine Verzögerung eingestellt werden, ab wann diese erst abrufbereit ist, und ein "Verfallsdatum", ab wann sie nicht mehr abgerufen werden kann. Gibt es bei einer multiconsumer queue keine subscribers, und es wird beim Versenden auch kein recipient angegeben, so tritt ein Fehler auf (ORA-24033).
Beim Abrufen wird prinzipiell so lange gewartet, bis eine Nachricht erhalten wurde, es sei denn, es wird eine maximale Wartezeit angegeben. Nach Ablauf dieser Zeit wird eine Exception ausgelöst (ORA-25228). Es kann auch gefiltert werden, welche Nachrichten erhalten werden sollen, z.B. ist die Angabe der Message-Id, der Priorität(sbereiche) oder von Inhalten möglich.

Weitere Einzelheiten finden Sie in der Oracle-Dokumentation.

Data Dictionnary Views:

- DBA_QUEUE_TABLES
- DBA_QUEUES

Datentyp für nachfolgende Beispiele:

```
CREATE TYPE queue_type AS OBJECT(subject VARCHAR2(100),
                                 text    VARCHAR2(2000));
/
```

Beispiel für single consumer queue:

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(queue_table =>'queuetab',
                          queue_payload_type => 'queue_type');
   DBMS_AQADM.CREATE_QUEUE(queue_name => 'queue_1',
                          queue_table =>'queuetab');
   DBMS_AQADM.START_QUEUE('queue_1');
END;
/

CREATE PROCEDURE SEND_QUEUE
   (p_subj IN VARCHAR2,
    p_msg  IN VARCHAR2)
IS
   v_options   DBMS_AQ.enqueue_options_t;
   v_props     DBMS_AQ.message_properties_t;
   v_handle    RAW(16);
   v_msg       queue_type;
BEGIN
   v_msg := queue_type(p_subj, p_msg);
   DBMS_AQ.ENQUEUE('queue_1', v_options, v_props, v_msg, v_handle);
   COMMIT;
END;
/
```

```
CREATE PROCEDURE RECEIVE_QUEUE IS
   v_options   DBMS_AQ.dequeue_options_t;
   v_props     DBMS_AQ.message_properties_t;
   v_handle    RAW(16);
   v_msg       queue_type;
BEGIN
   DBMS_AQ.DEQUEUE('queue_1', v_options, v_props, v_msg, v_handle);
   COMMIT;
   --DBMS_OUTPUT.PUT_LINE('Betreff war: '|| v_msg.subject);
   --DBMS_OUTPUT.PUT_LINE ('Meldung war: '|| v_msg.text);
   -- Weiterverarbeitung
END;
/
```

Beispiel für multi consumer queue mit einem recipient:

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(queue_table =>'queuetab_multi',
                           queue_payload_type => 'queue_type',
                           multiple_consumers => TRUE);
   DBMS_AQADM.CREATE_QUEUE(queue_name => 'queue_multi',
                           queue_table => 'queuetab_multi');
   DBMS_AQADM.START_QUEUE('queue_multi');
END;
/

CREATE PROCEDURE SEND_QUEUE_MULTI
   (p_subj IN VARCHAR2,
   p_msg   IN VARCHAR2)
IS
   v_options   DBMS_AQ.enqueue_options_t;
   v_props     DBMS_AQ.message_properties_t;
   v_handle    RAW(16);
   v_msg       queue_type;
   v_recipient DBMS_AQ.aq$_recipient_list_t;
BEGIN
   v_msg := queue_type(p_subj, p_msg);
   v_recipient(1) := SYS.aq$_agent ('dummy', null, null);
   v_props.recipient_list := v_recipient;
   DBMS_AQ.ENQUEUE('queue_multi', v_options, v_props, v_msg, v_handle);
   COMMIT;
END;
/

CREATE PROCEDURE RECEIVE_QUEUE_MULTI IS
   v_options   DBMS_AQ.dequeue_options_t;
   v_props     DBMS_AQ.message_properties_t;
   v_handle    RAW(16);
   v_msg       queue_type;
BEGIN
   v_options.consumer_name := 'dummy';
   DBMS_AQ.DEQUEUE('queue_multi', v_options, v_props, v_msg, v_handle);
   --DBMS_OUTPUT.PUT_LINE('Betreff war: '|| v_msg.subject);
```

```
   --DBMS_OUTPUT.PUT_LINE ('Meldung war: '|| v_msg.text);
   COMMIT;
   --Weiterverarbeitung
END;
/
```

Beispiele für Zugriff auf die queue table:

```
SELECT q_name,  priority, state, TO_CHAR(enq_time, 'dd.mm.rr hh24:mi:ss') enq_time
FROM queuetab;
SELECT t.user_data.subject subject, t.user_data.text text
FROM queuetab t;
```

Aufruf:

```
SQL> EXEC send_queue('Applikation A', 'Ergebnis: 4711')
PL/SQL-Prozedur wurde erfolgreich abgeschlossen.

SQL> SELECT q_name,  priority, state, TO_CHAR(enq_time, 'dd.mm.rr hh24:mi:ss')
enq_time FROM queuetab;

Q_NAME       PRIORITY      STATE ENQ_TIME
---------- ---------- ---------- -----------------
QUEUE_1             1          0 19.03.03 09:23:01

SQL> SELECT t.user_data.subject subject, t.user_data.text text
FROM queuetab t;

SUBJECT                TEXT
-------------------- --------------------
Applikation A        Ergebnis: 4711

SQL> EXEC RECEIVE_QUEUE
PL/SQL-Prozedur wurde erfolgreich abgeschlossen.

SQL> SELECT q_name,  priority, state, TO_CHAR(enq_time, 'dd.mm.rr hh24:mi:ss')
enq_time FROM queuetab;
Es wurden keine Zeilen ausgewählt
```