Asynchrone Replikation - Projekt oder Produkt

Lukas Grützmacher
AIS Automation Dresden GmbH

Schlüsselworte
Replikation, Oracle® Standard Edition, LogMiner

Einleitung (Abstrakt)
Sie brauchen eine schlanke, schnelle Datenbank für die Produktion?
Trotzdem müssen die Daten über einen langen Zeitraum zur Verfügung stehen?
Und Sie (oder Ihr Kunde) kann sich eine Oracle® Enterprise Edition und/oder GoldenGate® nicht leisten?
Dann haben Sie das gleiche Problem, welches auch wir vor langer Zeit hatten.

In dem Vortrag wollen wir zeigen, was es mit der Idee der Asynchronen Replikation auf sich hat und wie man mit Mitteln der Standard Edition eine projektspezifische Replikation oder - wie in unserem Fall - eine eigene Replikationslösung erstellen kann.

Motivation
In den Jahren 2005 und 2006 erarbeitete sich meine Firma ein neues Geschäftsfeld im Bereich Fabrikautomation. Bei dieser Anwendung fallen sehr viele Daten aus der Produktion an: Messdaten, Prozessprotokoll, Arbeits- und Wartungspläne, Rezeptinformation, etc.
Sehr schnell zeigte sich, dass Abfragen über große Datenmassen die Produktionssteuerung beeinflussen können. Um dies zu vermeiden wurde entschieden, den Datenzugriff auf zwei Datenbank-Instanzen zu verteilen. Dies sollte transparent für die Produktionssteuerungssoftware erfolgen. Somit war eine Replikationslösung notwendig.
Unsere Projekte im Bereich der Produktionssteuerung waren seinerzeit eher bei mittelständischen Firmen angesiedelt. Damit war der Einsatz einer Oracle® Enterprise Edition meist nicht finanzierbar und alle Replikationslösungen von Oracle waren aus dem Spiel.
(Bezahlbare) Lösungen anderer Anbieter waren uns zu diesem Zeitpunkt nicht bekannt. Als Software-Entwicklungsfirma mit langjähriger Erfahrung mit Oracle®-Datenbanken entschieden wir uns, versuchsweise eine einfache Replikationslösung mit den Mitteln einer Oracle® Standard Edition zu entwickeln. Ich erhielt den Auftrag dafür.
Innerhalb weniger Wochen war eine kundenspezifische Lösung auf Basis des Oracle®-Pakets LogMiner fertiggestellt und ging produktiv.
Durch die Fortentwicklung der Lösung für mittlerweile über 30 verschiedene Kunden ist aus dem Projekt mittlerweile ein Produkt geworden: FabEagle®replication. Es bietet sich heute als Ablösung für die von Oracle abgekündigten Lösungen (Data Guard, Streams) an.
Von den Erfahrungen mit dem LogMiner und wie damit eigene Projekte realisiert werden können, wird in diesem Vortrag berichtet.

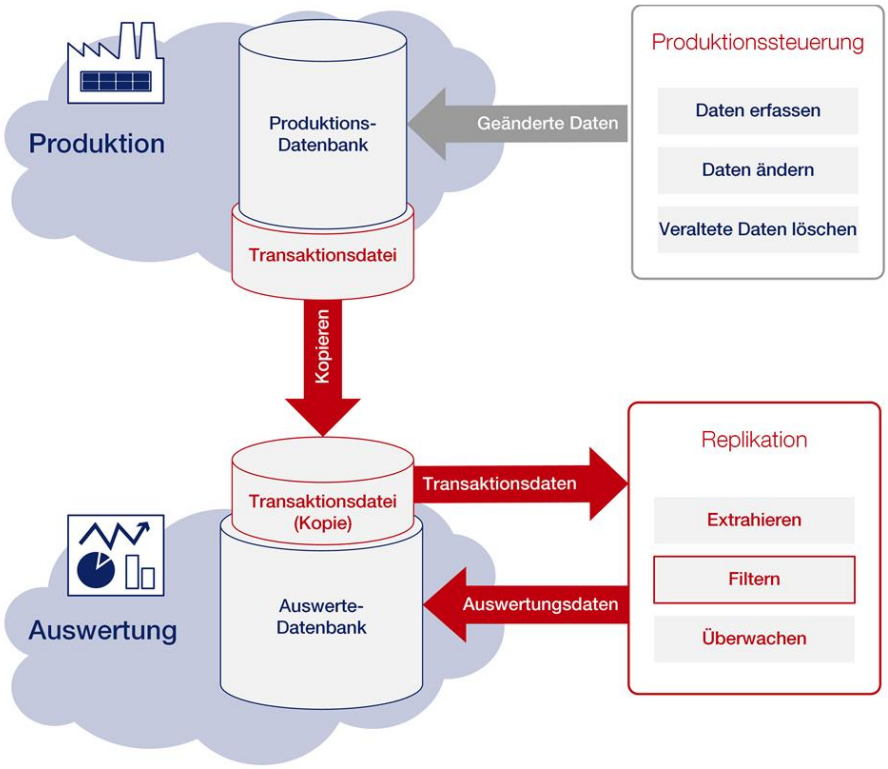

Abbildung 1 Überblick der Replikation

LogMiner – Was ist das?
Jede Oracle®-Datenbank (seit Version 8) in jeder Edition (Standard Edition One, Standard Edition „Classic“, Standard Edition Two, Enterprise Edition) enthält das Paket LogMiner. Es besteht aus einigen Packages und Views.[1]
Der LogMiner erlaubt die Analyse aller Transaktionprotokolldateien, sowohl REDOLOG-Dateien als auch ARCHIVELOG-Dateien. Ergebnis der Analyse ist die lesbare Abfolge von Änderungen, die in der Datenbank vorgenommen wurden. Besonders interessant ist dabei die Darstellung eines SQL-Statements, welches der Änderung entspricht. Dazu kommen Kontextinformationen über Benutzer, Anmeldung, Transaktionskontext, etc.
Werden die generierten SQL-Statements auf einer zweiten Datenbank ausgeführt, so werden Stück für Stück alle Änderungen nachgeführt und man erhält eine Datenbank-Kopie.

Ist doch alles ganz einfach, oder?
Auf den ersten Blick sieht eine Replikation basierend auf dem LogMiner sehr einfach aus. Die Dokumentation erklärt einiges über die Anwendung des Paketes und zeigt an einigen Beispielen, wie die wichtigsten Optionen funktionieren.[2]

---

[1] 10g Package: https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_logmnr.htm

[2] 12c Einführung: https://docs.oracle.com/database/121/SUTIL/GUID-3417B738-374C-4EE3-B15C-3A66E01AE2B5.htm

Hier ein kleines Beispiel von mir.

```
insert into TEST1(A) values (1);
```

Abbildung 2 Einfaches INSERT, welches in der Quelle ausgeführt wurde.

```
set transaction read write;
insert into "LG"."TEST1"("A","B") values ('1',NULL);
commit;
```

Abbildung 3 Ergebnis der LogMiner-Analyse

Aber in der Beschreibung des LogMiner steht an keiner Stelle, dass er für eine Replikation geeignet ist. So gibt es auch einige Stolpersteine, über die ich im Laufe der Zeit gestiegen oder auch mal zu Fall gekommen bin.
Von über 200 Service Requests bei Oracle hatten bislang 32 mit dem LogMiner zu tun, 16 davon identifizierten aktuelle oder sogar neue Bugs. Während der Vorbereitung dieses Vortrags kam ein weiterer Bug dazu. Manche Versionen waren für die Replikation erst einmal gar nicht verwendbar. Andere Probleme bereiteten uns Designentscheidungen. So wird manchmal ungültiges SQL erzeugt, obwohl die Dokumentation sagt, dass das Ergebnis valid sein sollte. Dies wurde von Oracle aber nicht als Fehler anerkannt.

COMMIT und ROLLBACK
Um erfolgreich eine Replikation basierend auf den Transaktionsdaten zu etablieren, müssen wir tiefer in die technischen Details einsteigen.
Dass jede Änderung an den Daten grundsätzlich zu einem Eintrag im Transaktionsprotokoll führt, muss ich sicherlich niemandem erklären. Aber für das Verständnis dessen, was der LogMiner letztendlich ausgibt, sind doch ein paar Informationen notwendig.
1. Mit jeder Änderung an den Daten (INSERT, UPDATE, DELETE) wird automatisch auch eine Transaktion begonnen.
2. Die gewünschte Änderung eines Datenblocks wird direkt in die REDO-Liste eingetragen (und später vom Database Writer in die Datenbankdateien übernommen).
3. Wird ein ROLLBACK ausgeführt, so werden die offenen Änderungen zurückgerollt, die selbst wieder als Änderung (invers) protokolliert werden.
4. Wird das COMMIT ausgeführt, so werden die Änderungen als dauerhaft markiert.

Daraus resultiert die Tatsache, dass alle Änderungen, ob letztlich abgeschlossen oder nicht, für den LogMiner, welcher ja die Transaktionsprotokolle lesbar macht, auch sichtbar werden.

Größere Transaktionen
Für die Aufgabe der Replikation ist letztlich nur interessant, was wirklich „committed“ wurde. Dafür stellt der LogMiner eine Funktion zur Verfügung, die dafür sorgt, dass nur Änderungen ausgegeben werden, die auch mit einem COMMIT abgeschlossen wurden. Alles, was zurückgerollt oder noch nicht abgeschlossen ist, wird ausgeblendet.
Diese Funktion war sehr naheliegend, so dass ich damit auch loslegte. Erst später stellte sich bei einigen kleinen Details heraus, dass diese Funktion, so schön einfach sie auch ist, doch ein paar Probleme bereitet, nämlich dann, wenn die Transaktion etwas länger dauert.

Der LogMiner zeigt nur dann die Änderungen der Transaktion an, wenn Beginn und Ende der Transaktion in den Protokolldateien enthalten sind, welche für die Analyse konfiguriert wurden. Fehlt der Anfang oder das Ende, so sieht man gar keine Änderungen dieser Transaktion.

Selbst bei sehr kleinen Transaktionen kann es nun vorkommen, dass die Änderungen auf zwei verschiedene Protokolldateien verteilt werden. Wir müssen also immer mehrere Dateien gleichzeitig analysieren. Später gehe ich darauf nochmals im Detail ein.
Das Problem potenziert sich, wenn wir mit wirklich langlaufenden Transaktionen konfrontiert werden. Leider kann dies in einigen Fällen auch dazu führen, dass der LogMiner den Dienst versagt und mit der Nachricht „Nicht ausreichend Speicher verfügbar“ abbricht. Daher muss man dafür Sorge tragen, dass die Änderungen auf der Quelldatenbank in überschaubare Häppchen verteilt werden.

```
-- ungünstig
UPDATE TEST1 SET A = 123 WHERE A IS NULL;

-- besser
DECLARE
BEGIN
  LOOP
    UPDATE TEST1 SET A = 123
     WHERE COLUMN1 IS NULL AND ROWNUM <= 1000;
    EXIT WHEN SQL%ROWCOUNT < 1000;
    COMMIT;
  END LOOP;
  COMMIT;
END;
/
```

Abbildung 4 Varianten umfangreicher Updates

Tabellenänderungen
Nun verändert sich die Datenbankstruktur einer Anwendung in der Regel während der Lebenszeit. Um das Update auf den Datenbanken zu vereinfachen ist es wünschenswert, dass auch Strukturänderungen repliziert werden.
Der LogMiner kann diesem Wunsch nachkommen, weil die Datenbankstrukturen selbst als Einträge in den internen Tabellen der Datenbank gespeichert werden und eine Änderung beispielsweise des Tabellennamens auch zu einem UPDATE im sogenannten Data Dictionary führt. Diese Änderung erkennt der LogMiner und generiert daraus ein Statement, welches die Tabelle umbenennt. Die generierten Änderungen können dann genauso erneut auf der Zieldatenbank ausgeführt werden und deren Struktur wird aktualisiert.
Aber auch hier steckt die Tücke im Detail. Um den LogMiner zu befähigen, die Strutkuränderungen korrekt zu interpretieren, braucht er eine konsistente Kopie des Data Dictionary. Diese Kopie muss für diese Funktion in die Protokolldateien geschrieben werden. Diese Kopie wird dann bei der Analyse ausgelesen und mit jeder gefundenen Strukturänderung aktualisiert.
Daraus folgt aber auch, dass eine Analyse immer mit einer Transaktionsdatei beginnen muss, die eine solche Kopie enthält.
Damit die Analyse nicht immer umfangreicher wird, muss die Kopie in regelmäßigen Abständen neu erstellt werden. Dies ist zwar ein gewisser Overhead, hilft aber dabei, in eben diesen Abständen wieder einen konsistenten Ansatzpunkt zu finden.

Die richtigen Protokolldateien
Aus den beiden vorgenannten Problemdimensionen ergibt sich die Strategie, wie die Protokolldateien für die Analyse ausgewählt werden.
1. Jede Analyse beginnt bei einer Datei mit Data-Dictionary-Kopie.
2. Um in der Analyse einen neuen Startpunkt zu erhalten müssen alle Transaktiondateien analysiert werden, bis die nächste Dictionary-Kopie erreicht wurde.

3. Um Transaktionen zu sehen, die eine gewisse Zeit in Anspruch nehmen, müssen die Protokolldateien mit einer Überlappung analysiert werden, welche der erwarteten Maximaldauer einer Transaktion entspricht.

Daraus folgt, dass die längste Liste von Protokolldateien von einer Datei mit Dictionary-Kopie bis zur übernächsten Datei mit Kopie reicht.

Archivelog-Datei	Erste SCN	Dictionary-Beginn	Dictionary-Ende
.../thread_1_seq_12540.1744.922608195	1152105535	YES	NO
.../thread_1_seq_12541.1724.922611617	1152106753	NO	YES
.../thread_1_seq_12542.1730.922612697	1152131387	NO	NO
.../thread_1_seq_12543.1911.922612729	1152240173	NO	NO
.../thread_1_seq_12544.1813.922612741	1152241237	NO	NO
.../thread_1_seq_12545.1864.922616337	1152243158	NO	NO

Abbildung 5 Informationen über die Archivelog-Dateien liefert der View V$ARCHIVED_LOG

Besonders spannend wird diese Geschichte, wenn wir eine RAC-Datenbank als Quelle verwenden. Jeder RAC-Knoten schreibt seine eigenen Protokolldateien, unabhängig voneinander. Bei der Analyse müssen aber alle Protokolldateien in der richtigen Reigenfolge analysiert werden. Die Details der richtigen Auswahl lasse ich hier außen vor, weil sie den Rahmen sprengen würden. Es sei aber so viel verraten, dass dies eines der komplexesten Aufgaben unserer Replikationslösung geworden ist…

Asynchrone Replikation

An dieser Stelle haben wir eine vollständige Replikation hergestellt. Wir haben nun zwei Datenbanken zur Verfügung, welche – mit leichtem Zeitversatz – die gleichen Daten bereitstellen. Die Quelldatenbank wird von der Produktionssteuerung mit Daten beliefert. In der Zieldatenbank können Auswertungen ausgeführt werden, ohne die Performance der Quelldatenbank und damit der Produktionssteuerung zu beeinflussen.

Mit steigender Datenmenge wird die Performance der Quelldatenbank aber trotzdem leiden, weil die produktionsrelevanten Daten in der Menge der historischen Daten schwer zu identifizieren sind. Wie eingangs erwähnt haben wir bei unseren Kunden in der Regeln nicht die Möglichkeit, Features der Enterprise Edition – allen voran das Partitioning – zu nutzen. Es liegt daher nahe, die Quelldatenbank von historischen Daten zu befreien. Ein Löschjob ist schnell geschrieben.

Die Zieldatenbank soll die historischen Daten aber weiterhin bereitstellen, weil sie für die Qualitätssicherung von großer Bedeutung sind. Wir müssen also dafür sorgen, dass die Löschoperationen nicht repliziert werden. Genau dies ist der Kern dessen, was wir Asynchrone Replikation nennen.

Im Strom der vielen Änderungen sind also die Löschoperationen zu identifizieren und vor der Replikation aus der Liste zu löschen. Es gibt verschiedene Ansätze, diese Operationen zu erkennen, denn es ist leider zu einfach, jede DELETE-Operation zu streichen, die in der Quelle ausgeführt wurde, weil beispielsweise ein versehentlich angelegter Artikel aus Quell- und Zieldatenbank entfernt werden soll.

In unseren Projekten hat sich überwiegend etabliert, dass der Löschjob durch einen separaten Datenbank-Nutzer ausgeführt wird. Da der LogMiner neben den Informationen über betroffene Tabellen und ihre Besitzer liefert, sondern auch über den ausführenden Nutzer. Leider schlagen auch hier gelegentlich Bugs des LogMiners zu und erfordern Fine-Tuning der Filter-Regeln.

Die beschriebene Replikation ist eine logische Replikation. Damit bietet sie einen weiteren Vorteil: Die Struktur der Quell- und Zieldatenbank muss nicht identisch sein. Gerade wenn der Datenumfang der beiden Datenbank unterschiedlich ist, kann es hilfreich sein, die technische Infrastruktur jeweils entsprechend anzupassen. So können die Datenbanken in unterschiedlichen Netzwerken entsprechend

ihrer Nutzer angebunden sein. Die können – in gewissen Grenzen – auf unterschiedlichen Plattformen installiert sein. Für Testzwecke können unterschiedliche Oracle-Versionen installiert sein. Nicht zuletzt kann beispielsweise für die mächtige Zieldatenbank eine Enterprise Edition mit Partitioning verwendet werden, ohne dass auch die Quelldatenbank eine Enterprise Edition sein muss.

Fazit

Das Paket LogMiner der Oracle®-Datenbank bietet die Möglichkeit, sämtliche Transaktionen einer Datenbank in lesbare und wieder ausführbare SQL-Statements umzuwandeln. Alle Statements nacheinander erneut auf einer zweiten Datenbank ausgeführt resultieren in einer datentechnisch identischen Kopie.
Um den Datenumfang der beiden Datenbanken ihren jeweiligen Aufgaben anzupassen, können Daten aus der Quelldatenbank gelöscht werden, die Replikation der Löschoperationen aber wird unterbunden. So erhalten wir eine Replikation, die wir als Asynchron bezeichnen.

Kontaktadresse:

Lukas Grützmacher
AIS Automation Dresden GmbH
Otto-Mohr-Straße 6
D-01237 Dresden

Telefon:	+49 (0) 351 2166 0
Fax:	+49 (0) 351 2166 3210
E-Mail	lukas.gruetzmacher@ais-automation.com
Internet:	www.ais-automation.com
LinkedIn:	https://de.linkedin.com/in/lukasgruetzmacher
XING:	https://www.xing.com/profile/Lukas_Gruetzmacher