# Ablaufkontrolle in der Datenbank bei externem Scheduling

**Dr. Kurt Franke**
**Cellent Finance Solutions GmbH**
**Stuttgart**

**Schlüsselworte**

Batch-Verarbeitung, Scheduling, Parallelverarbeitung, Locks

## Einleitung

Wenn das Scheduling von Batch-Verarbeitungen in der Datenbank selbst erfolgt, kann über eine Abhängigkeits-Steuerung sichergestellt werden, dass für jeden beliebigen Zeitpunkt ein Scheduling nur für solche Verarbeitungen erfolgt, für die kein Hinderungsgrund in Form fehlender Voraussetzungen oder gerade laufender Änderungen vorliegt, bei denen gleichzeitiges Laufen zu unbrauchbaren Daten führen würde.

Anders ist es, wenn das Scheduling von außerhalb der Datenbank erfolgt. Das ist meist beschränkt auf Zeitsteuerung und eine serielle Abfolge in Job-Ketten. Eine durchgehende Überprüfung der betroffenen Datenbank auf laufende und abgebrochene Jobs mit Auswirkungen auf das Scheduling gibt es eher selten.

In diesem Vortrag wird beschrieben, wie auch bei externem Scheduling sichergestellt werden kann, dass jede Verarbeitung nur dann läuft, wenn es keinen Hinderungsgrund irgendwelcher Art dafür gibt.

## Was bietet uns die Datenbank als Grundlage für die Implementierung eines Verfahrens ?

Um datenbank-seitig eine Möglichkeit zu Ablauf-Kontrolle zu implementieren, wird ein Mechanismus benötigt, der bei Vorliegen eines Hinderungsgrunds für eine Verarbeitung entweder wartet, bis dieser Hinderungsgrund entfällt, oder andernfalls sofort mit einem Fehler abbricht. Die Entscheidung,, welcher Variante der Vorzug zu geben ist, ist eher fachlich begründet und teilweise verschieden nach Art der Verarbeitung und sollte deshalb nicht in einem vor-implementierten Mechanismus fixiert werden, sondern dem Aufrufer des Mechanismus zur Wahl stehen.

Generell bieten Locks und damit auch PL/SQL-Userlocks nur durch den Einsatz der beiden möglichen Wartezeiten 0 (zero, NOWAIT) und ∞ (unendlich, WAIT) genau ein derartiges Verhalten, wenn ein zu allokierender Lock bereits von einer anderen Session allokiert wurde. Das Oracle-Datenbank-Package **`dbms_lock`** stellt dazu die notwendigen grundlegenden Utilities bereit.

Einschränkend ist hier noch anzumerken, dass der Lock-Mechanismus immer nur ein Handling für einen bestimmten Lock bereitstellt, jedoch kein Lock-übergreifendes Handling.

## Lock-übergreifendes Handling und Zuordnung von Eigenschaften zu Locks

Zur Vermeidung datenbank-seitiger Probleme durch Verarbeitungen, die gerade nicht laufen sollten, bietet sich also der Einsatz von PL/SQL-User-Locks über das dbms_lock-Package mit Verwendung

von Locknamen in den Verarbeitungs-Packages an. Durch dbms_lock wird die Möglichkeit geboten, dass ein Lock-Request nicht erfolgreich ist, wenn der Lockname bereits in einer anderen Session allokiert wurde. Es gibt hier jedoch keine Wechselwirkung verschiedener Locknamen. Diese voneinander völlig unabhängigen User-Locks lassen sich über ein geeignetes Handling-Package in beliebige Zusammenhänge und Abhängigkeiten der Locks untereinander bringen. Für komplexere Zusammenhänge kann es dazu notwendig sein, die unterstützten Locknamen vorzudefinieren, damit jederzeit die anderen möglichen Locknamen bekannt sind. Für die Abbildung der notwendigen Funktionalität werden den Locknamen dann verschiedene Eigenschafts-Kombinationen zugewiesen. Parallele Verarbeitungen gleich strukturierter unabhängiger Einheiten werden durch Erweiterung der Locknamen um die jeweilige Unit-ID möglich, Unit-übergreifende Sperren dabei durch speziell definierte Pseudo-Unit-ID´s. Die verschiedenen vordefinierten Eigenschaften müssen letztlich dazu führen, dass eine vom Scheduling gestartete Verarbeitung den zugehörigen Lock nicht allokieren kann, wenn nicht alle Voraussetzungen zum Laufen der Verarbeitung erfüllt sind, und dann - je nach zur Situation passendem Handling - entweder auf den Lock wartet oder abbricht.

Für das vorliegende System müssen die Locks folgende Anforderungen abbilden:

1. Daten-Änderungen in verschiedenen Units sind unabhängig voneinander. Aufgrund der Notwendigkeit des Abschaltens von Foreign-Key-Constraints während der Daten-Integration durch Partition-Exchange in den Batchverarbeitungen wird jedoch für dieses Verarbeitungs-Abschnitt eine Unit-übergreifende Serialisierung benötigt.
2. Innerhalb einer Unit darf es zu jedem beliebigen Zeitpunkt nur entweder genau eine daten-ändernde Batchverarbeitungen oder alternativ eine beliebige Anzahl von Datenpflege-Operationen via GUI geben.
3. Nicht daten-ändernde Batchverarbeitungen (z. B. Exporte) dürfen nicht gleichzeitig mit daten-ändernden Batchverarbeitungen in derselben Unit laufen. Ansonsten können verschiedene nicht daten-ändernde Batchverarbeitungen in einer Unit beliebig parallel laufen, wobei jedoch gleichzeitige Mehrfachläufe einer bestimmten nicht daten-ändernden Batchverarbeitung in einer Unit verboten sind.
4. Das Clearing von Log-Tabellen erfolgt Unit-übergreifend und darf von keiner Verarbeitung mit Bezug zu einer Unit geblockt werden.

Daraus wurde eine Klassifizierung der Locks nach den Typen IMPORT, API, EXPORT, PROC_CNTRL abgeleitet, wobei der Typ IMPORT für daten-ändernde Batchverarbeitungen verwendet wird, der Typ API für Daten-Änderung durch Datenpflege, der Typ EXPORT für nicht daten-ändernde Batchverarbeitungen, und der Typ PROC_CNTRL für das Unit-übergreifende Log-Clearing.

Die meisten Locks haben den Lock-Level MAIN zugeordnet, was bedeutet, dass sie auf Ebene der Verarbeitungen angesiedelt sind. Nur ein solcher MAIN-Lock darf in einer Session zu einem bestimmten Zeitpunkt allokiert sein. Zur Abbildung einer Unit-übergreifende Serialisierung wurde der Lock-Level SUB eingeführt, der zusätzlich zu einem MAIN-Lock allokiert werden darf. Die Allokierung erfolgt jedoch in allen Units für die gleiche Pseudo-Unit.

Damit alle Verarbeitungen und Datenpflege-Aktionen in einer Unit von jeder beliebigen daten-ändernden Batchverarbeitung (Lock-Typ IMPORT) blockiert werden, ist der Lock-Typ IMPORT so implementiert, dass für die betreffende Unit immer implizit Locks für alle definierten Locknamen im Exclusive-Mode allokiert werden. Umgekehrt werden diese dann bei einem release des explicit allokierten Locks auch wieder freigegeben. Locks vom Typ PROC_CNTRL sind hier nicht betroffen, weil ihre tatsächliche Verarbeitungs-Allokierung immer für eine Pseudo-Unit erfolgt. Dieses spezielle Handling mit Allokierung zusätzlicher Locks gibt es für die anderen Lock-Typen nicht.

Damit es auch eine Blockierung zwischen einem Unit-übergreifenden SUB-Lock gegen die Lock-Typen EXPORT und API gibt, wird für diese Typen automatisch der lock auch für die betreffende Pseudo-Unit allokiert, jedoch im Shared Mode, um gegenseitige Unit-übergreifenden Sperren je Lockname zu verhindern.

Locks, die für Batchverarbeitungen vorgesehen sind – das sind alle außer diejenigen mit Locktype API – werden mit Session Duration allokiert. Bei den Locks mit Locktype API wird dies anders gehandelt. Da hier die DB-Sessions normalerweise nicht beendet werden und nach Bedarf für Aufgaben aus verschiedenen Anwender-Logins verwendet werden, ist es hier sinnvoll, die Locks mit Transaction Duration zu allokieren. Ein Commit führt dann automatisch dazu, dass ein solcher Lock freigegeben wird.

Da Locks für die Datenpflege via GUI unabhängig von der Art der Datenpflege verwendet werden und diese Datenpflege für eine beliebige Anzahl gleichzeitig möglich sein muss, müssen Locks vom Locktype API im Shared Mode allokiert werden.

Zur vereinfachten Identifizierung bei einer Auflistung von Locks – nur mit DBA-Rechten möglich – erhalten die Locknamen auch noch einen Prefix. der sich wie folgt zusammensetzt:

'RUN_LOCK_DEPENDANTS.' || current_schema || '.'

Der Einsatz des Schemanamens erlaubt die gleichzeitige Verwendung dieses Mechanismus in mehreren Datenbank-Schemata. Da Locknamen nicht einem Datenbankschema zugeordnet sind, wäre das sonst nicht einfach möglich.

Da nicht alle allokierten Locks direct einer Verarbeitung zugeordner sind, sondern oft nur dem Blockieren anderer Verbeitungen dienen, wird für Locks, die direct einer Verarbeitung zugeordner sind, ein zusätzlicher Lock mit dem um 'RUNNING.' erweiterten Prefix allokiert. Dieser erlaubt dann auch eine Identifizierung der laufenden Verarbeitungen über eine Auflistung der Locks.

| LOCK_NAME | LOCK_TYPE | LOCK_LEVEL | ALLOKATION | DURATION |
|---|---|---|---|---|
| GEPARD-SYNC-DELTA | IMPORT | MAIN | EXCLUSIV | SESSION |
| GEPARD-SYNC-FULL | IMPORT | MAIN | EXCLUSIV | SESSION |
| GEPARD-SYNC-FULL-WITH-ELEM | IMPORT | MAIN | EXCLUSIV | SESSION |
| NEU-BEWERTUNG | IMPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-AKTIONSLISTE | EXPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-AKTIONSLISTE2 | EXPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-P24C_MELDUNGEN | EXPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-MDS_ZUSATZINFO | EXPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-JP_NO_WE | EXPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-JP_KB_VERTRETUNGSORGAN | EXPORT | MAIN | EXCLUSIV | SESSION |

| | | | | |
|---|---|---|---|---|
| EXPORT-LAENDER_LISTE | EXPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-BETEILIGTE_FIRMEN | EXPORT | MAIN | EXCLUSIV | SESSION |
| API-CALL | API | MAIN | SHARED | TRANSACTION |
| PROC-CNTRL-LOG-CLEARING | PROC_CNTRL | MAIN | EXCLUSIV | SESSION |
| FILL-PARTNER-INFO | EXPORT | MAIN | EXCLUSIV | SESSION |
| SWITCH_GAE_BACK_TO_CONSISTENT | IMPORT | MAIN | EXCLUSIV | SESSION |
| FLASHBACK_GAE | IMPORT | MAIN | EXCLUSIV | SESSION |
| RESET_GAE | IMPORT | MAIN | EXCLUSIV | SESSION |
| WRITE_ACTUAL_TO_GEPARD_FILES | IMPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-KO_PARTNER | EXPORT | MAIN | EXCLUSIV | SESSION |
| EXPORT-MAN_RISK_OVERRIDDEN | EXPORT | MAIN | EXCLUSIV | SESSION |
| OVERRIDE-MANRISK | IMPORT | MAIN | EXCLUSIV | SESSION |
| SERIALIZE-FK-REBUILD | IMPORT | SUB | EXCLUSIV | SESSION |

*Tabelle 1: Liste der vordefinierten Locknamen mit einigen Eigenschaften*

**Was ist beim Einsatz des dbms_lock Packages zu beachten ?**

Die Verwendung von Lock-Namen zur wohldefinierten Zuordnung von bestimmten Verarbeitungen setzt den Einsatz der Prozedure dbms_lock.allocate_unique() voraus, um ein Verknüpfung des entwickler-definierten Lock-Namens mit einem internen Lock-Handle herzustellen. Diese Prozedure führt jedoch grundsätzlich ein COMMIT aus und schließt damit die aktuelle Transaktion ab. Da dies sowohl für Lock-Handling als auch die durch die Locks kontrollierten Verarbeitungen nicht gewünscht ist, wird dieser Aufruf in eine lokale Prozedure mit AUTONOMOUS_TRANSACTION eingeschlossen, so dass die Aufrufumgebung vor diesem COMMIT geschützt ist.

Wenn Locks in großer Anzahl in sehr kurzen Zeitabständen allokiert werden, führt dies eventuell zu einer "Überforderung" des dbms_lock-Mechanismus. Das äußert sich dann darin, dass Deadlocks als Fehler erscheinen, obwohl es eigentlich keine Ursache dafür gibt. Wenn die Zeitabstände der Allokierung vergrößert werden, verschwinden diese Deadlocks wieder. Dies kann z. B. durch Aufrufe von dbms_lock.sleep() erreicht werden.

Da eine Prüfung, ob ein bestimmter anderer Lock allokiert ist oder nicht, auch über einen Allokierungsversuch (mit NOWAIT) und ggf. anschließend ein sofortiges release() erfolg, kann eine derartige Prüfung das System auch zum Kippen bringen, insbesondere wenn bei jeder Lock-Allokation noch eine solche Prüfung stattfindet. Da die Locks auch sicherstellen sollen, dass bei einem laufenden Versions-Update keine Verarbeitungen gestartet werden können, findet jedoch genau eine solche Überprüfung statt. Diese wird dadurch entschärft, dass zuerst in einer speziellen Tabelle überprüft wird, ob ein Update begonnen wurde. Und nur wenn das der Fall ist, erfolgt auch die Lock-Überprüfung.

**Integration anderer Prüfungen ins Lock-Handling Package**

Wenn andere Bedingungen / Zustände vor Beginn jeder Verarbeitung geprüft werden müssen, bietet sich an, diese Prüfung ins Lock-Handling-Package zu integrieren. Dadurch kann vermieden werden, dass dies für jede Verarbeitung separat erfolgen muss.

Für jede Unit wird der Konsistenz-Zustand als J/N-Eigenschaft hinterlegt und aktiv während der Verarbeitungen nach Bedarf angepasst. Eine einzige Verarbeitung – die mit dem Locknamen SWITCH_GAE_BACK_TO_CONSISTENT – ist dafür zuständig, den konsistenten Zustand wiederherzustellen, wenn dieser nach Abschluß eine anderen Verarbeitung nicht mehr besteht. Diese eine Verarbeitung darf dann im inkonsistenten Zustand gestartet werden, alle anderen jedoch nicht. Umgekehrt darf diese ein Verarbeitung jedoch nicht im konsistenten Zustand gestartet werden, dafür aber alle anderen. Genau dieses Verhalten wurde in die Lock-Allokation integriert. Wenn die Prüfung zeigt, dass der gewünschte Lock wegen unpassender Konsistenz nicht zugeteilt werden darf, wird vom Lockhandling-Package ohne die Möglichkeit des Wartens eine Exception erzeugt. Die Prüfung erfolgt üblicherweise für den Konsistenzstatus der Unit, für die ein Lock allokiert werden soll. Wenn jedoch eine Lock-Allokation für eine Pseudo-Unit erfolgen soll, erfolgt die Prüfung über alle Units. Wenn dann auch nur eine einzige Unit inkonsistent ist, gilt diese Inkonsistenz für das ganze System.

**Zusammenfassung**

Der Einsatz von PL/SQL Userlocks mit Locknamen bietet die grundsätzliche Funktionalität, um eine Verarbeitung entweder zu blockieren oder abzubrechen, wenn nicht alle notwendigen Bedingungen für diese Verarbeitung erfüllt sind. Dazu ist es jedoch erforderlich, ein Lock-Handling-Package zu implementieren, das die notwendigen Zusammenhänge / Abhängigkeiten zwischen den Locks abbildet. Diese Abhängigkeiten können auch weitaus komplexer sein als im betrachteten Fall. Dann ist es natürlich sinnvoll, in einer Tabelle zu hinterlegen, welche Locks von welchen anderen Locks abhängig sind. Die Auswirkung des automatischen Commit in dbms_lock.allocate_unique() auf das Lockhandling und den Applikationscode läßt sich durch Einschließen in eine lokale Prozedure mit AUTONOMOUS_TRANSACTION im Lock-Handling-Package vermeiden. Massive Lock-Allokation pro Zeiteinheit kann zu massivem Auftreten von Deadlocks führen – hier hilft nur eine Vermeidungsstrategie zur Reduzierung der Lock Requests pro Zeiteinheit. Das Handling von Statuswerten der Applikation mit ähnlichen Auswirkungen wie sperrende Locks sollte am besten mit ins Lock-Handing-Package integriert werden.

**Kontaktadresse:**

Dr. Kurt Franke
Cellent Finance Solutions GmbH
Calwer Straße 33
D-70173 Stuttgart

| | |
|---|---|
| Telefon: | +49 (0) 711-222992-676 |
| Fax: | +49 (0) 711-222992-899 |
| E-Mail | Kurt.Franke@cellent-fs.de |
| Internet: | www.cellent-fs.de |