

Ablaufkontrolle in der Datenbank bei externem Scheduling

Dr. Kurt Franke

Cellent Finance Solutions GmbH

Kurt.Franke@cellent-fs.de, Kurt-Franke@web.de

Agenda

- cfs Kurzvorstellung
- DB-interne Ablaufkontrolle Scheduling intern/extern
- Anforderung an Datenbank-Mechanismus
- Lockübergreifendes Handling
- Systemspezifische Anforderungen an Lockhandling
- daraus abgeleitetes Design des Lockhandlings
- Auflisten und Identifizierung der Locks
- Grenzen des dbms_lock Packages
- Integration weiterer Prüfungen ins Handling-Package
- API Calls im Handling-Package
- Zusammenfassung

Zahlen und Fakten auf einen Blick

Firmensitz:	Stuttgart
Geschäftsstellen:	Frankfurt am Main, München
Branchenerfahrung:	seit 1988
Umsatz:	€ 30 Mio.
Anzahl Mitarbeiter:	210
Rechtsform:	Gesellschaft mit beschränkter Haftung
Geschäftsführer:	Thomas Wild
Gesellschafter:	100 % Landesbank Baden-Württemberg (LBBW)
Aufsichtsratsvorsitz:	Dr. Martin Setzer



SMARAGD Compliance Suite



- Mehr als 1.600 Institute in über 50 Ländern vertrauen auf SMARAGD Produkte
- Unter den 50 Top Banken weltweit nutzen 9 Banken SMARAGD Lösungen
- Die ersten 7 der Top 10 Banken in Deutschland setzen auf die SMARAGD Compliance Suite
- Die deutsche und die österreichische Zentral-Bank nutzen SMARAGD Lösungen
- Seit 1999 stehen wir mit unseren Lösungen der Finanzwirtschaft und seit 2011 der Industrie zur Verfügung

Referenzen SMARAGD (Auszug)

DekaBank
 DEUTSCHE BUNDESBANK EUROSISTEM
 OESTERREICHISCHE NATIONALBANK EUROSISTEM
ENB
 TELEKOM AUSTRIA GROUP
ING
DiBa
Deutsche Bank
Allianz
BILFINGER
Munich RE
COMMERZBANK
DAB bank
HypoVereinsbank UniCredit Group
SBERBANK
UniCredit Bank
TARGO BANK Die Initiativbank
FFB
 valartisbank+
EQUENS
IKB Deutsche Industriebank
bwin.com
DZ BANK Zusammen geht mehr.
 fidor bank ag
Atos
HSBC
Trinkaus
clearstream DEUTSCHE BÖRSE GROUP
SAB Sächsische AufbauBank
e@sy Credit
BHF BANK PRIVAT SEIT 1854
PAY BOX
Raiffeisen Landesbank Oberösterreich
EFIS FINANCIAL SOLUTIONS
LB BW
VOLKSBANK INTERNATIONAL
Haspa Hamburger Sparkasse
Bayern LB
SüdLeasing Man least viel Gutes über uns.
Aurubis
finanzinformatik
KOMERCIJALNA BANKA Sa nama je lakše
KGAL REAL INVESTMENTS

Datenbank-interne Ablaufkontrolle

- simple bei Scheduling in der Datenbank
 - Scheduling-Mechanismus prüft jede Verarbeitung
 - auf mögliche Hinderungsgründe
 - fehlende Voraussetzungen
 - gerade laufender Änderungen
 - Scheduling nur, wenn es keine Hinderungsgründe gibt
- bei Scheduling von außerhalb der Datenbank
 - Verarbeitung wird angestartet
 - meist nur nach Zeitsteuerung
 - und Abfolge in Job-Ketten
 - genauere Prüfung auf Hinderungsgründe
 - innerhalb Verarbeitung am Beginn notwendig
 - Prüfungsmechanismus muss je nach gerade sinnvollem Handling
 - Blockieren, bis Verarbeitung erlaubt ist
 - oder mit Fehler abbrechen

Anforderung an Datenbank-Mechanismus

- als Implementierungsgrundlage
 - Prüfung aktuelle Verarbeitung auf Hinderungsgründe
 - fehlende Voraussetzungen
 - gerade laufende Änderungen
 - Handling-Möglichkeiten bei Vorliegen von Hinderungsgründen
 - Blockierung der Verarbeitung bis zum Wegfall
 - sofortiger Abbruch der Verarbeitung mit geeigneter Fehlermeldung
 - Auswahl der Handling-Variante durch Aufrufer
 - abhängig von fachlicher Anforderung für aktuelle Verarbeitung
- passende Funktionalität durch Locks in der Datenbank
 - speziell PL/SQL-Userlocks – Funktionalität mit **dbms_lock** Package
 - Wartezeit 0 (zero, NOWAIT), Lock nicht erhalten → Fehlerabbruch
 - Wartezeit ∞ (unendlich, WAIT), Lock nicht erhalten → Blockierung
 - Lockübergreifendes Handling muss implementiert werden

Lockübergreifendes Handling

- pro Verarbeitungsart Zuordnung eines Locknamens
- Allokierung im Exklusivmode
 - kann nur einmal in der Datenbank erhalten werden
- Sperren anderer Locks / Verarbeitungen
 - nicht direkt möglich
 - dazu sind Beziehungen zwischen Locks notwendig
 - die zu abhängigen Exklusiv-Lock-Allokierungen führen
 - die nur als mittel zur Sperre zusätzlich allokiert werden
 - Implementierung als Applikations-spezifischer Mechanismus
 - wegen spezifischer Locks
 - und spezifischer Beziehungen / Abhängigkeiten zwischen Locks
 - mit Einbindung von `dbms_lock`-Aufrufen
 - keine direkte Verwendung von `dbms_lock`-Aufrufen außerhalb

Anforderungen speziell für vorliegendes System

- Daten-Änderungen in verschiedenen Units sind unabhängig
- Batch-Daten-Integration durch Partition-Exchange
 - mit Abschalten von Foreign-Key-Constraints
 - → Unit-übergreifende Serialisierung für diesen Schritt
- pro Unit
 - maximal eine daten-ändernde Batch-Verarbeitung
 - oder beliebige Anzahl von Daten-Pflege-Operationen via GUI
- nicht daten-ändernde Batch-Verarbeitung für gleiche Unit
 - gesperrt durch daten-ändernde Batch-Verarbeitung
 - beliebig parallel mit nicht daten-ändernden Batch-Verarbeitungen
 - aber nicht die gleiche mehrfach
- Clearing von Log-Tables unit-übergreifend
 - keine Sperre durch Batch-Verarbeitungen auf Unit-Ebene

Abgeleitet aus Anforderungen: Klassifizierung nach Typen

- Lock-Type IMPORT – Allokierung im Exklusiv-Mode
 - Locks für daten-ändernde Batchverarbeitungen
- Lock-Type API – Allokierung im Shared-Mode
 - Locks für Daten-Änderung durch Datenpflege via GUI
- Lock-Type EXPORT – Allokierung im Exklusiv-Mode
 - Locks für nicht daten-ändernde Batchverarbeitungen
- Lock-Type PROC_CNTRL – Allokierung im Exklusiv-Mode
 - Locks für das Unit-übergreifende Log-Clearing
 - Allokierung immer für Pseudo-Unit 0
 - Keine Wechselwirkung mit Verarbeitungen auf Unit-Ebene
 - Nur zur Verhinderung gleichzeitiger Mehrfachläufe

Klassifizierung nach Typen – Details Lock-Type IMPORT

- Lock-Type IMPORT – Allokierung im Exklusiv-Mode
 - Locks für daten-ändernde Batchverarbeitungen
 - muss jede andere Datenänderung für die Unit verhindern
 - Batchverarbeitungen und Datenpflege
 - dazu verwendete Methode:
 - alle anderen Locks im Exklusiv-Mode für die Unit mit allokierten
 - als reine Sperr-Locks
 - für alle Batch-Verarbeitungen und GUI-Datenpflege
 - für die gleiche Unit
 - „alle anderen Locks“ bedeutet wenig Aufwand
 - auch bei Erweiterung um zusätzliche Lock-Namen
 - ist Voraussetzung für erfolgreiche Allokierung des gewünschten Locks
 - dazu müssen die möglichen Lock-Namen vorher bekannt sein
 - bis auf die Erweiterung durch die Unit
 - dadurch wird auch jeder mögliche API-Type Lock Exklusiv allokiert
 - und sämtliche GUI-Daten-Änderungen blockiert

Klassifizierung nach Typen – Details Lock-Type API

- Lock-Type API – Allokierung im Shared-Mode
 - Locks für Daten-Änderung durch Datenpflege via GUI
 - Shared-Mode ermöglicht:
 - gleichzeitige Verwendung durch mehrere Sessions
 - für beliebige Daten-Änderungs-Aktionen
 - Lock kann auch für neue Datenpflege-Aktionen verwendet werden
 - ohne die Liste der Locks zu erweitern
 - Lock-Type API blockiert Lock-Type IMPORT
 - weil dafür der API-Type Lock nicht exklusiv allokiert werden kann
 - Lock-Type IMPORT blockiert Lock-Type API
 - weil dafür der API-Type Lock bereits exklusiv allokiert wurde
 - andere Lock-Typen werden nicht von Lock-Type API blockiert
 - zusätzliche Allokierung für Pseudo-Unit -1 im Shared-Mode
 - als Blockierungsmechanismus für Unit-übergreifende Sperren
 - mit Lock-Type IMPORT

Klassifizierung nach Typen – Details Lock-Type EXPORT

- Lock-Type EXPORT – Allokierung im Exklusiv-Mode
 - Locks für nicht daten-ändernde Batchverarbeitungen
 - dient zur Verhinderung gleichzeitiger Mehrfachläufe pro Unit
 - Lock-Type EXPORT blockiert Lock-Type IMPORT
 - weil dafür der EXPORT-Type Lock nicht exklusiv allokiert werden kann
 - Lock-Type IMPORT blockiert Lock-Type EXPORT
 - weil dafür der EXPORT-Type Lock bereits exklusiv allokiert wurde
 - andere Lock-Typen werden nicht von Lock-Type EXPORT blockiert
 - zusätzliche Allokierung für Pseudo-Unit -1 im Shared-Mode
 - als Blockierungsmechanismus für Unit-übergreifende Sperren
 - mit Lock-Type IMPORT

Abgeleitet aus Anforderungen: verschiedene Lock-Level

▪ Lock-Level MAIN

- Alle Locks, die direkt einer Verarbeitung zugeordnet sind
- Es kann nur ein Lock mit Level MAIN direkt allokiert werden
 - Abgesichert im Lock-Handling-Package
 - Aber intern als reine Sperr-Locks beliebig viele

▪ Lock-Level SUB

- Locks, die nicht direkt einer Verarbeitung zugeordnet sind
- Zusätzliche Locks für daten-ändernde Batchverarbeitungen
 - Immer Lock-Type IMPORT
 - keine Einschränkung für Allokierung verschiedener SUB Locks
- Für zusätzliche Unit-übergreifende Sperren
 - Für bestimmte Teile einer Verarbeitung
 - Durch generelle Allokierung für Pseudo-Unit -1
 - wirksam auch gegen Lock-Type API und Lock-Type EXPORT
 - wegen deren Zusatz-Allokierung für Pseudo-Unit -1 im Shared-Mode

Abgeleitet aus Anforderungen: verschiedene Lebensdauer

▪ Lock-Duration SESSION

- Alle Locks für Batch-Verarbeitungen
- Sessions werden extra für eine bestimmte Verarbeitung erzeugt
 - und danach sofort beendet
 - im normalen Betrieb der Applikation
 - keine übrig bleibenden Locks zu erwarten
 - auch nicht bei Programmierfehlern
 - kann beim Testen anders sein
 - wenn dann übrig bleibende Locks gefunden werden
 - kann der vergessene `release()`-Aufruf eingebaut werden

▪ Lock-Duration TRANSACTION

- Alle Locks für Daten-Änderung durch Daten-Pflege via GUI
- Sessions aus Session-Pool werden beliebig verwendet
 - aber für jede Änderungs-Aktion gibt es ein COMMIT
 - Lock wird am Ende einer Aktion garantiert freigegeben

Identifizierung der Locks beim Auflisten

- Auflisten der Locks zur Problem-Analyse
 - nur mit DBA-Rechten möglich
- Locks erhalten einen Prefix zur Selection
 - Der Prefix setzt sich so folgendermaßen zusammen
 - 'RUN_LOCK_DEPENDANTS.' || current_schema || '.'
 - Schema-Name, in dem das Handling-Package installiert ist
 - erlaubt mehrfache Verwendung in verschiedenen DB-Schemata
 - wäre sonst nicht möglich
 - weil Locknamen keinem DB-Schema zugeordnet sind
- weiterer Lock mit erweitertem Prefix
 - für alle direkt einer Verarbeitung zugeordneten Locks
 - 'RUN_LOCK_DEPENDANTS.' || current_schema || '.RUNNING.'
 - nicht für reine Sperr-Locks und SUB-Locks
 - zur einfacheren Identifizierung aktiver Verarbeitungen

Liste der Lock-Namen

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

Grenzen des dbms_lock Packages

- `dbms_lock.allocate_unique()`
 - Abbildung Lock-Name auf Lock-Handle
 - COMMIT bei jedem Aufruf
 - Vermeidung Außenwirkung des COMMIT
 - durch Wrapper Procedure im Handling Package
 - `dbms_lock.allocate_unique()`
 - mit `PRAGMA AUTONOMOUS_TRANSACTION`
 - Duration `TRANSACTION` sonst nicht möglich
- viele Lock-Requests in kurzen Zeitabständen
 - in verschiedenen Sessions Quasi-Parallel
 - kann `dbms_lock`-Mechanismus „überfordern“
 - es gibt dann Deadlock-Fehler ohne äußere Ursache
 - Reduzierung Anzahl der „gleichzeitigen“ Lock-Requests
 - Deadlock-Fehler verschwinden wieder

Grenzen des dbms_lock Packages II

- viele Prüfungen auf allokierte andere Locks
 - implementiert als request NOWAIT und ggf. sofortiges release
 - sind damit auch Lock-Requests in kurzen Zeitabständen
- Einsatz Locks für Sperren bei Versions-Update
 - Update gegen aktive Verarbeitungen und vice versa
 - Update-Lock wirkt auf gesamte Applikation
 - nicht beschränkt auf eine Unit
 - Prüfung Update-Lock bei jeder Lock-Request
 - würde Deadlock-Szenario auslösen
 - Vermeidung durch vorherige Prüfung eines Tabellen-Eintrags
 - zur Kennzeichnung eines aktiven Updates
 - Lock-Prüfung nur, wenn Tabelleneintrag gefunden wird
 - dadurch massive Reduzierung der Lock-Prüfungen
 - Deadlock-Problem wird vermieden

Integration weiterer Prüfungen ins Handling-Package

- eine Unit kann inkonsistenten Zustand haben
- Konsistenzprüfung erfolgt im Lock-Handling-Package
 - Lock request() für inkonsistente Unit erzeugt Exception
 - außer bei request eines speziellen Locks
 - SWITCH_GAE_BACK_TO_CONSISTENT
 - für Verarbeitung zur Bereinigung der Inkonsistenz
 - erzeugt Exception bei request für konsistente Unit
- Notwendigkeit einer separaten Prüfung entfällt

API Calls im Handling-Package

```
SUBTYPE lock_name_t IS VARCHAR2(128);
TYPE request_lock_ret_t IS RECORD
  ( indicator INTEGER, lock_name lock_name_t );
-- indicator = 0: angeforderter lock bereits allokiert
-- indicator = 1: angeforderter lock wurde inclusive aller
  eventueller Zusatzlocks für Sperren erhalten
-- indicator = 2: angeforderter lock wurde erhalten, aber
  es wurden nicht alle eventuellen Zusatzlocks
  für Sperren erhalten -
  kann nur bei Locktype IMPORT vorkommen
-- zurückgegebener lock_name ist um unit-part erweitert
FUNCTION request_lock(nam lock_name_t, gae_cdd_id INTEGER)
  RETURN request_lock_ret_t;

-- erster Aufruf für Lockname und Unit immer NOWAIT
-- nachfolgender Aufruf für gleiche Lockname und Unit immer WAIT
-- solange kein request für andere Lockname und Unit erfolgt ist
-- Aufrufer kann entscheiden, ob abgebrochen oder gewartet wird
```

API Calls im Handling-Package II

```
PROCEDURE release_lock(nam lock_name_t, gae_cdd_id INTEGER);
```

```
FUNCTION test_allocated(name lock_name_t, gae_cdd_id INTEGER)  
    RETURN INTEGER;
```

```
-- Return 1: Lock wurde bereits in aktueller Session allokiert
```

```
-- Return 0: Lock wurde bereits in anderer Session allokiert
```

```
-- Return -1: Lock wurde noch nicht allokiert
```

```
FUNCTION test_running_allocated(name lock_name_t,  
                                gae_cdd_id INTEGER)
```

```
    RETURN INTEGER;
```

```
PROCEDURE request_cdd_update_lock;
```

```
PROCEDURE release_cdd_update_lock;
```

Zusammenfassung

- PL/SQL Userlocks mit Locknamen
 - grundsätzliche Funktionalität zum blockieren oder abubrechen
 - wenn nicht alle notwendigen Bedingungen erfüllt sind
- Lock-Handling-Package
 - Zusammenhänge / Abhängigkeiten zwischen den Locks
 - komplexere Abhängigkeiten über Tabellen-Einträge definieren
 - erlaubt beliebige lock-spezifische Abhängigkeiten
 - Auswirkungen Commit in `dbms_lock.allocate_unique()` vermeiden
 - durch wrapper Procedure mit Autonomous Transaction
- Massive Lock-Allokation pro Zeiteinheit vermeiden
 - wegen möglichen internen Deadlock-Problemen
- Applikations-Stati-Handling im Lock-Handling-Package
 - bei analogen Auswirkungen wie Sperr-Locks

Fragen & Antworten

Vielen Dank für Ihre Aufmerksamkeit

Kontakt: **Dr. Kurt Franke**

Cellent Finance Solutions GmbH, Calwer Str. 33, D-70173 Stuttgart

Email: Kurt.Franke@cellent-fs.de , Kurt-Franke@web.de

Phone: +49-(0)711-222992676 , Mobil: +49-(0)171-7963089