Gunter Saake
Kai-Uwe Sattler
Andreas Heuer

Datenbanken

Konzepte und Sprachen

Sechste Auflage

Dieses pdf-Kapitel ist eine kostenlose Ergänzung zum oben genannten Buch, das 2018 bei MITP erschienen ist.

Bitte beachten: Verweise auf Umgebungen B-X beziehen sich auf Teile innerhalb dieses Download-Kapitels. Verweise, die mit Kapitelnummern beginnen, wie 11-31, beziehen sich auf Umgebungen innerhalb des obigen Lehrbuches. Verweise auf Seiten vor 738 beziehen sich ebenfalls auf das zugrundeliegende Lehrbuch.

Literaturhinweise in diesen pdf-Kapiteln beziehen sich auf ein erweitertes Literaturverzeichnis zum Lehrbuch, das auch als kostenlose Ergänzung an derselben Stelle zum Download bereitsteht.

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Bei der Herstellung des Werkes haben wir uns zukunftsbewusst für umweltverträgliche und wiederverwertbare Materialen entschieden. Der Inhalt ist auf elementar chlorfreiem Papier gedruckt.

ISBN 978-3-95845-776-8
6. Auflage 2018

www.mitp.de
E-Mail: mitp-verlag@sigloch.de

Telefon: +49 7953/7189-079
Telefax: +49 7953/7189-082

Lektorat: Sabine Janatschek, Ernst-Heinrich Pröfener
Sprachkorrektorat: Jürgen Dubau, Astrid Langen
Covergestaltung: Christian Kalkert, www.kalkert.de
Bildnachweis Cover: iStock.com/Shawn Hempel | fotolia.com/karpenko_ilia
Satz: Gunter Saake, Magdeburg; Kai-Uwe Sattler, Ilmenau; Andreas Heuer, Rostock
Druck: Westermann Druck Zwickau GmbH

B.6 ECA-Regeln und aktive Datenbanken

In Kapitel 13 des diesem Abschnitt zugrundeliegenden Lehrbuchs haben wir das Trigger-Konzept kennengelernt, das unter anderem die Sicherung der Integrität der Datenbank realisieren kann. Komplexe Integritätsbedingungen konnten mithilfe von Triggern überwacht werden, Techniken für Updates auf Sichten mit Triggern umgesetzt werden (Kapitel 15). Einige Ansätze erweitern nun das Trigger-Konzept um zusätzliche Aspekte. Die entstehenden Systeme sind unter dem Schlagwort *aktive Datenbanken* bekannt. Aktive Datenbanken benutzen sogenannte *ECA-Regeln*. Die Buchstaben E, C und A stehen für die drei Bestandteile einer ECA-Regel: *Event*, *Condition* und *Action*:

Event: Analog zu Triggern wird ein auslösendes *Ereignis* angegeben. In ECA-Regeln können dies neben Datenbankmodifikationen wie in Triggern aber auch sogenannte Zeitereignisse oder Anwendungsereignisse sein.

Zeitereignisse können explizite Zeitpunktangaben, periodische Zeitangaben (jede volle Stunde) oder relative Zeitangaben (drei Stunden nach Löschen des Tupels) sein. Anwendungsereignisse könnten zum Beispiel Aufrufe von Anwendungsmethoden oder Ereignisse der Benutzerschnittstelle sein. Weitere mögliche Ereignisse sind Ereignisse der Transaktionssteuerung (Beginn einer Transaktion, Ende einer Transaktion, Abbruch einer Transaktion).

Condition: Dies beschreibt eine *Bedingung*, die zum Ausführen der Regelaktion erfüllt sein muss. Neben Datenbankanfragen kann dies eine Bedingung über Parameter des feuernden Ereignisses sein.

Action: Dieser Teil gibt die auszuführende *Aktion* an, in der Regel eine Folge von Datenbankmodifikationen bzw. ein Abbruch der Transaktion.

Syntaktisch wird eine ECA-Regel vereinfacht in der folgenden Form notiert:

```
on Ereignis if Bedingung do Aktion
```

Die Entwicklung von Datenbank-Triggern hin zu aktiven Datenbanken lässt sich als eine Entwicklung hin zu offenen Datenbanksystemen charakterisieren: Nicht allein Datenbankereignisse sind Auslöser von Regeln (oder werden durch Regeln angestoßen), sondern auch Ereignisse anderer Softwarekomponenten wie der Benutzeroberfläche oder der Uhr des Betriebssystems sind in das Regelsystem integriert.

Durch die angesprochenen Erweiterungen öffnen sich einige interessante Problemfelder, die aktuell in der Forschung und Entwicklung von Prototypen bearbeitet werden.

B.6.1 Entwurf von ECA-Regeln

ECA-Regel verführen genau wie Trigger zu einem unsauberen Programmierstil, da die Aktivierung von ansonsten unabhängigen Datenbankaktionen zu ähnlichen Effekten wie der Einsatz von **`goto`**-Anweisungen in einer imperativen Programmiersprache führen kann. Andererseits ermöglichen sie strukturiert eingesetzt einen sehr mächtigen regelbasierten Programmierstil zur Erzwingung von Integritätsbedingungen. Ihr Einsatz ist somit wünschenswert, und die im Ansatz inhärenten Gefahren müssen durch einen *sauberen Entwurf* und geeignete *Analyseverfahren* beherrscht werden. Leider ist bisher noch keine umfassende Entwurfs- und Analysemethodik entwickelt worden, wie sie beispielsweise im Datenbankstrukturbereich durch die Transformation von ER-Modellen in relationale Schemata und deren Normalisierung bekannt ist.

Spezielle Probleme, die von einer derartigen Methodik behandelt werden müssen, sind unter anderem:

- *Konfluenz*: Ein Regelsystem heißt *konfluent*, wenn der Effekt auf die Datenbank bei gleichzeitig aktivierten Regeln immer unabhängig von der Reihenfolge der Abarbeitung dieser Regeln ist.

- *Terminierung*: Ein Regelsystem terminiert bei einer gegebenen Ausgangsdatenbank, wenn ein Zustand erreicht wird, in dem keine weiteren Regeln aktiviert sind.

Insbesondere die Terminierung ist natürlich im Allgemeinen nicht entscheidbar, so dass geeignete Restriktionen beim Einsatz von ECA-Regeln beachtet werden müssen.

B.6.2 Zusammengesetzte Ereignisse

In vielen Anwendungen hängt die Aktivierung einer Regel nicht von einem einzelnen atomaren Ereignis ab, sondern vom Eintreten von sogenannten *zusammengesetzten Ereignissen* (engl. *Composite Events*). Zusammengesetzte Ereignisse können als Kombinationen von (atomaren oder selbst zusammengesetzten) Ereignissen definiert werden, etwa als *„Ereignis A gefolgt von Ereignis B“*.

Typische Kombinatoren zur Konstruktion von komplexeren Ereignissen basierend auf anderen Ereignissen, sind etwa (angelehnt an den Vorschlag in [GD94, GD93, DG96]):

and: Die *Konjunktion* zweier Ereignisse tritt ein, wenn zwei Ereignisse A und B in beliebiger Reihenfolge auftreten:

$$A \textbf{ and } B$$

or: Die *Disjunktion* zweier Ereignisse A und B wird angezeigt, falls eins von beiden eintritt:

$$A \texttt{ or } B$$

then: Der **then**-Operator (oft auch als ; notiert) modelliert die *Sequenz* zweier Ereignisse, etwa A gefolgt von B:

$$A \texttt{ then } B$$

Mittels Sequenz und Disjunktion kann die Konjunktion wie folgt als abgeleiteter Operator definiert werden:

$$A \texttt{ and } B := (A \texttt{ then } B) \texttt{ or } (B \texttt{ then } A)$$

not _ in (_ , _): Die *Negation* eines Ereignisses kann nur bezüglich eines Intervalls definiert werden. Die folgende Bedingung wird wahr, falls im Intervall zwischen den Ereignissen B und C das Ereignis A nicht eingetreten ist:

$$\texttt{not } A \texttt{ in } (B,C)$$

_ times _ in (_ , _): Der **times**-Operator zählt das Eintreten von Ereignissen in einem Intervall, zum Beispiel ob das Ereignis A im Intervall von B bis C genau fünfmal eingetreten ist:

$$5 \texttt{ times } A \texttt{ in } (B,C)$$

Ein abgewandelter Operator erkennt ein Ereignis beim n-ten Eintreten eines Ereignisses nach einem Startereignis B:

$$n \texttt{ times } A \texttt{ after } B$$

all _ in (_ , _): Die bisherigen Operatoren reichen aus, um die Zeitpunkte des *Signalisierens* eines zusammengesetzten Ereignisses festzulegen. In der Regelverarbeitung wird aber oft auch auf die Parameter von Ereignissen zugegriffen. Der **all**-Operator sammelt alle Ereignisse inklusive ihrer Parameterwerte innerhalb eines Intervalls auf, so dass diese in der Regelaktion verarbeitet werden können.

Die angegebenen Operatoren definieren eine *Ereignisalgebra* zur Konstruktion zusammengesetzter Ereignisse. Sowohl für die Benennung als auch für die Auswahl der sinnvollen Konstruktoren ist noch keine allgemeine Übereinstimmung erzielt worden; aus diesem Grund haben wir hier einige naheliegende Konstruktoren in einer syntaktischen Notation verwendet, die an die englische Sprache angelehnt ist. Auf konkrete Sprachvorschläge wird in Abschnitt 13.8 dieses Kapitels verwiesen.

B.6.3 Zeitereignisse und Echtzeitanforderungen

Sowohl absolute (am Freitag um 12:00 Uhr) als auch relative (10 Minuten nach Änderung des Kontostands) Zeitereignisse sind im Datenbankbetrieb problematisch, da die Aktivierung außerhalb einer Transaktion erfolgen kann. Das Datenbankmanagementsystem muss in diesen Fällen selber Transaktionen starten, um die angestoßenen Aktionen auszuführen. Transaktionen können mehrfach zurückgesetzt und neu gestartet werden, wenn der Mehrbenutzerbetrieb und damit verbundene Zugriffskonflikte dies erfordern. Aus prinzipiellen Gründen können Zeitereignisse in einer derartigen Architektur darum nicht für Echtzeitanforderungen eingesetzt werden – insbesondere können ECA-Regeln, deren aktivierendes Ereignis ein Zeitereignis ist, unter diesen Umständen nicht im **`immediate`**- oder **`deferred`**-Modus behandelt werden.

B.6.4 Verschiedene Kopplungsmodi

Bereits in der Diskussion von Integritätsregeln wurden sogenannte *Kopplungsmodi* diskutiert. Mit der Kopplung bezeichnet man die zeitliche Beziehung zwischen einem Ereignis und der von ihm aktivierten Aktion. Die klassischen Kopplungsmodi sind **`immediate`** (unmittelbar nach Eintreten des Ereignisses) und **`deferred`** (bis zum Ende einer Transaktion verzögert). Da in aktiven Datenbanken auch Ereignisse auftreten können, die nicht dem Transaktionsprinzip der Datenbank unterliegen (etwa nicht zurückgesetzt werden können), werden hier auch weitere Kopplungsmodi diskutiert, beispielsweise die folgenden zusätzlichen Modi aus [Buc94]:

`detached independent:` Angestoßene Aktionen werden in einer *unabhängigen* Transaktion ausgeführt – das Zurücksetzen der ursprünglichen Transaktion beeinflusst die Aktionsausführung nicht.

`detached but causally dependent:` Die Aktion wird zu einer Transaktion gemacht, die in Abhängigkeit vom Ergebnis der triggernden Transaktion ausgeführt wird. Spezielle Abhängigkeiten wären **`parallel`** (Synchronisation beim erfolgreichen Ende der triggernden Transaktion), **`sequential`** (Start der Transaktion nach erfolgreichem Ende der triggernden Transaktion) und **`exclusive`** (Start nur nach dem *Abbruch* der triggernden Transaktion).

Die letzteren beiden Modi machen insbesondere Sinn in offenen Umgebungen, wo Ereignisse stattfinden und Aktionen ausgeführt werden können, die beim Abbruch der Haupttransaktion *nicht zurückgesetzt* werden können und somit eine Behandlung außerhalb des ACID-Transaktionskonzeptes erfordern.

B.6.5 Literatur

Zu ECA-Regeln gibt es diverse Forschungsprojekte. Das Projekt HIPAC [DBB+88] prägte viele Begriffe aktiver Datenbanken und ist bereits abgeschlossen. Das Samos-Projekt von Dittrich et al. zeichnet sich durch eine Event-Sprache für zusammengesetzte Ereignisse aus [GD92, GD93]. Das Projekt REACH wird in [BZBW95] beschrieben. Algorithmen zur Terminierung und Konfluenz von ECA-Regeln finden sich beispielsweise in [WH95, Wei97].

Die Integration von ECA-Regeln in objektorientierte Datenbanksysteme ist ein aktuelles Forschungsgebiet. Eine Übersicht wird von Buchmann in [Buc94] gegeben.

Die Dissertation von Türker [Tür99] behandelt speziell die Integration von Integritätsbedingungen im Rahmen der Datenbankföderation, enthält aber auch detailliertere Literaturaufarbeitungen zu andern Aspekten der Integritätsüberwachung.