

Zeit-Maschine

Temporale Datenhaltung

Achim Demelt

In vielen Anwendungen muss festgehalten werden, in welchem zeitlichen Bereich ein Datensatz gültig ist. Dieser Zeitbezug ist aufwändig und kompliziert zu pflegen. Der Artikel zeigt allgemeine Konzepte und eine konkrete Implementierung temporaler Daten mit Hilfe eines generativen Ansatzes.

▶ Bei herkömmlichen Anwendungen werden Datensätze aus einer Datenbank gelesen (bzw. neu erzeugt), durch das Programm verändert und mit den neuen Werten zurück in die Datenbank geschrieben. Die ehemaligen Werte gehen dabei verloren und die neuen sind sofort nach dem Schreiben gültig.

Für viele Anwendungen reicht dieses Schema der Datenverarbeitung aus. In anderen Fällen jedoch ist es zwingend notwendig, dass die Historie der Daten nachvollzogen werden kann. Hier ergeben sich beispielsweise Fragestellungen wie: „Welche Adresse(n) hatte eine Person am 17. April 2002?“ , oder: „Welchen Basispreis hatte ein Artikel zu dessen Markteinführung?“ . Bestimmte Datensätze haben also einen bestimmten Gültigkeitszeitraum. Oft ist es auch notwendig, Datensätze rückwirkend zu ändern. So kann es beispielsweise vorkommen, dass eine Person ihren Wohnsitz erst mehrere Wochen nach dem Umzug ummeldet.

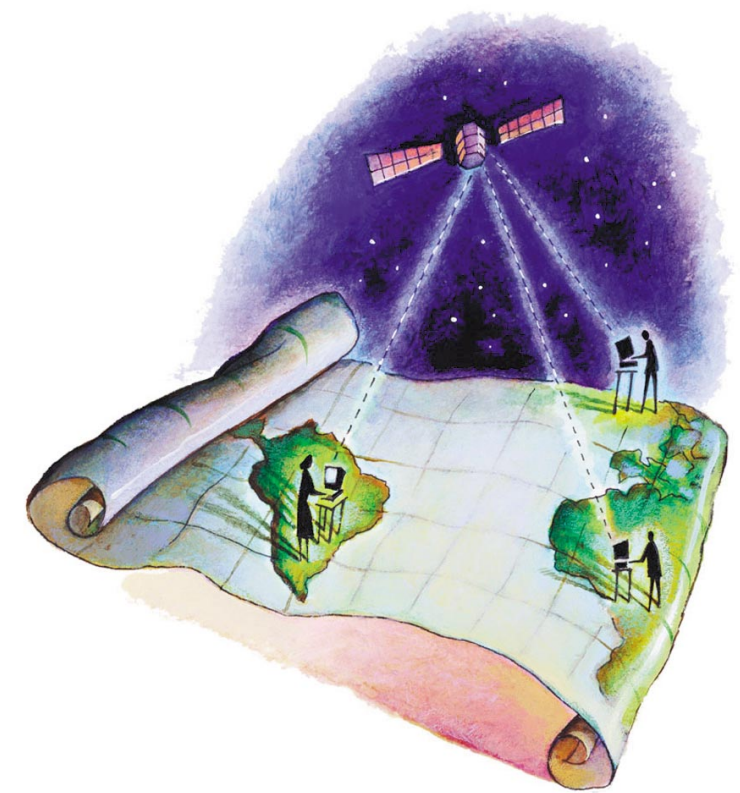
Die Zeit

Im Kontext temporaler (auch *historisiert* genannter) Daten werden drei Arten der Zeit unterschieden:

- ▼ **Benutzerdefinierte Zeit:** Hierbei handelt es sich um einen losgelösten, fest stehenden Zeitstempel, z. B. das Geburtsdatum einer Person.
- ▼ **Gültigkeitszeit:** Der Zeitpunkt bzw. -bereich, zu dem ein bestimmter Datensatz in der realen Welt gültig ist, z. B. der Preis eines Artikels, der vom 1. Mai 2003 bis zum 1. Juni 2003 gilt.
- ▼ **Bearbeitungszeit:** Der Zeitpunkt, an dem ein Datensatz in der Datenbank gespeichert wurde, z. B. wurde der Preis eines Artikels am 1. April 2003 bearbeitet.

Benutzerdefinierte Zeiten sind für die temporale Datenhaltung nicht relevant. Sie werden ausschließlich von der Anwendung und nicht vom historisierten Datenspeicher interpretiert. Anders verhält es sich mit Gültigkeits- und Bearbeitungszeit. Jede dieser beiden Zeiten kann vom Datenspeicher temporal verwaltet werden. In manchen Fällen ist es sogar wünschenswert, *beide* Varianten gleichzeitig zu führen.

Die Bearbeitungszeit kann hierbei in vielen Fällen mit recht einfachen Mechanismen verwaltet werden. Dazu zählen beispielsweise Logging von Benutzeraktionen oder die Speicherung des bearbeitenden Anwenders und des Bearbei-



tungszeitstempels durch Trigger in der Datenbank. Für die Gültigkeitszeit ist dieses Vorgehen allerdings nicht ausreichend. Die hierfür notwendigen Konzepte werden nun im Folgenden dargestellt.

Begriffsdefinitionen

Für relationale Datenbanken gibt es bereits seit mehreren Jahren Konzepte für temporale Datenhaltung (s. z. B. [Sno00]). Diese werden allerdings (vermutlich auf Grund der Komplexität) kaum oder nur zögerlich von den Datenbankherstellern umgesetzt. Viel gravierender ist jedoch die Tatsache, dass solche Konzepte im OO-Bereich noch völlig vernachlässigt werden. Daher werden hier zunächst einige Begrifflichkeiten geklärt.

Ein temporales *Objekt* im Kontext dieses Artikels ist ein Objekt, das in mehreren *Versionen* vorliegt, die jeweils eine Gültigkeitsdauer besitzen. Der Begriff Objekt umfasst hierbei also die Gesamtheit aller Versionen.

Die Gültigkeitsdauern der Versionen eines Objekts müssen disjunkt sein, d. h. sie dürfen sich nicht überlappen, und dürfen ferner keine Lücken im Lebenszyklus aufweisen. Ein Objekt hat genau einen definierten Gültigkeitsbeginn seines Lebenszyklus. Ein Objekt ohne Gültigkeitsende ist unbegrenzt gültig. Eine Objektlöschung ist in der Regel nicht möglich. Stattdessen wird ein Objekt *terminiert*, indem ein Gültigkeitsende gesetzt wird und keine Nachfolgeversion mehr existiert. Abbildung 1 zeigt Beispiele solcher Objekt/Versionskonstellationen.

Eine Version ist am Gültigkeitsende *nicht* mehr gültig, da zu diesem Zeitpunkt bereits die Gültigkeitsdauer der Nachfolgeversion beginnt. Ist also eine Version V_1 gültig von T_1 bis T_2 , so ist ihre Gültigkeitsdauer das Intervall $[T_1, T_2)$.

Änderung an Objekten

Sollen historisierte Objekte verändert werden, so ist es wichtig, dass der Gültigkeitsbereich jeder Änderung angegeben wird. Zur Illustration dieses Aspekts soll ein Objekt O_1 dienen. In der Version V_1 ist dies gültig vom Zeitpunkt T_1 bis zum Zeitpunkt T_3 . Danach ist die Version V_2 ohne Begrenzung gültig.

Eine Anwendung arbeite nun mit der Version V_1 zum Zeitpunkt T_2 , der zwischen T_1 und T_3 liegt. Eine Änderung des Objektzustands wird immer zu diesem Zeitpunkt T_2 gültig. Das Gültigkeitsende der Version V_1 wird dabei vom Zeitpunkt T_3 auf den Zeitpunkt T_2 zurückgesetzt. Abhängig vom Gültigkeitsende der Änderung kann die Aktion allerdings drei unterschiedliche Ergebnisse haben:

- ▼ **Gültigkeitsende bei T_3 :** Eine neue Version V_3 mit Gültigkeit von T_2 bis T_3 wird erzeugt. V_2 beginnt nach wie vor zum Zeitpunkt T_3 (s. Abb. 2).
- ▼ **Gültigkeitsende vor T_3 :** Eine neue Version V_3 beginnt bei T_2 und endet zum definierten Zeitpunkt T_{3-} . Im Zeitraum zwischen T_{3-} und T_3 wird eine weitere Version V_4 erzeugt, die den Wert von V_1 besitzt (s. Abb. 3, Teil 1).
- ▼ **Gültigkeitsende nach T_3 :** Eine neue Version V_3 beginnt bei T_2 und endet zum definierten Zeitpunkt T_{3+} . Der Gültigkeitsbeginn von V_2 wird auf T_{3+} nach hinten verschoben (s. Abb. 3, Teil 2).

Relationen

Relationen sind orthogonal zu ihren Bezugsobjekten ebenfalls historisiert. Das bedeutet, dass jede Beziehung zwischen zwei Objekten ihrerseits Gültigkeitsgrenzen besitzt und nur in diesem Zeitrahmen gültig ist. Die konkreten Zeitgrenzen einer Beziehung ergeben sich durch die Zeitgrenzen, innerhalb derer die Objekte assoziiert wurden. Die Assoziation bezieht sich jedoch immer auf Objekte, nicht auf deren Versionen.

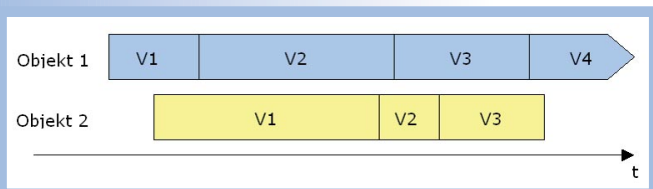


Abb. 1: Temporaler Lebenszyklus von Objekten

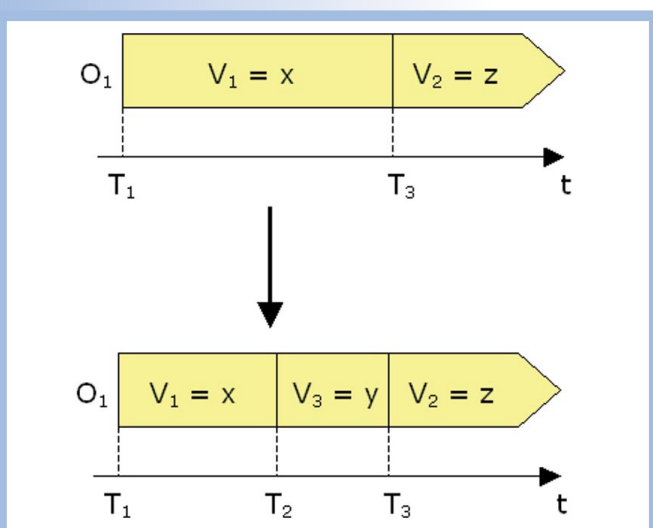


Abb. 2: Objektänderung zum Zeitpunkt T_3

Abbildung 4 zeigt eine solche Konstellation. Als Beispiel für historisierte Relationen kann eine Ehepartnerbeziehung zwischen zwei Personen dienen. Im wahrsten Sinne des Wortes entstehen hier Beziehungen während des Lebenszyklus verschiedener Objekte. Während die Objekte selbst – also die Personen – Änderungen unterworfen sind (z. B. Adressänderungen), werden unabhängig davon Beziehungen geknüpft und auch wieder getrennt.

Abgrenzung

Die Historisierung der Daten erfolgt unter rein zeitlichen Gesichtspunkten. Die Gültigkeit einer Version ist durch einen Zeitraum definiert. Das impliziert die Tatsache, dass zu einem beliebigen Zeitpunkt maximal eine einzige Version eines bestimmten Objektes existiert.

Anders stellt sich die Versionierung von Objekten dar. Dort hat eine Version eines Objektes nur eine eindeutige Versionsnummer. Es ist möglich, ausgehend von einer Version mehrere Nachfolgeversionen zu erstellen. Dies stellt eine Verzweigung (Branching) dar und resultiert in einem Versionsbaum. Solche Konstrukte sind mit dem hier vorgestellten Historisierungskonzept nicht möglich.

Datenspeicherung

Die Speicherung temporaler Objekte erfolgt zweckmäßigerweise mit O/R (objektrelationalen) Mappern. Anders als beispielsweise EJB EntityBeans ermöglichen sie ein schlankes, objektorientiertes Modell mit allen Freiheiten bezüglich Vererbung und Polymorphie. Mit diesen Konzepten lassen sich die Aspekte temporaler Datenhaltung sehr elegant vereinheitlichen.

Da man nicht davon ausgehen kann, dass die zu Grunde liegende Datenbank selbständig in der Lage ist, historisierte Daten zu speichern, müssen die Geschäftsobjekte und die Datenbanktabellen anderweitig um entsprechende Informationen ergänzt werden. Im Detail sind dies die folgenden Felder:

- ▼ **VERSION:** Die Versionsnummer dient zur eindeutigen Kennzeichnung der einzelnen Versionen eines Objekts. Dies ist nur ein interner Schlüssel, der der Anwendung nicht zugänglich ist.
- ▼ **VALID_FROM:** Der Gültigkeitsbeginn der Version.
- ▼ **VALID_TO:** Das Gültigkeitsende der Version. Ist dies nicht gesetzt, so hat die Version unendliche Gültigkeit.

Der Primärschlüssel der Tabelle muss um die Spalte VERSION erweitert werden. Tabelle 1 zeigt beispielsweise die historisierte Verwaltung von Mehrwertsteuersätzen, Tabelle 2 dasselbe für Artikeldaten. Referenzen zwischen Objekten werden über eine Zwischentabelle mit den Zeitinformationen der Relation abgebildet. In dieser Tabelle werden nur die IDs der assoziierten Objekte ohne die VERSION-Spalte abgebildet, wie in Tabelle 3 zu sehen ist. Welche konkrete Version referenziert wird, wird erst zur Laufzeit anhand des Datums festgestellt. Als Folge können keine klassischen Fremdschlüssel-Constraints auf der Datenbank definiert werden.

Fehlende Fremdschlüssel und die Auslagerung aller Relationen in eigene Tabellen haben leider noch einen weiteren gravierenden Nachteil: Beziehungen können nicht mehr vom O/R Mapper gemanagt werden. Vielmehr müssen künstliche „Objekte“ eingeführt werden, die der Persistenzmechanismus auf die Zwischentabellen abbilden kann.

Historisierung in der Praxis

Die Verwaltung temporaler Daten erfolgt zweckmäßigerweise in einem Framework. Dort müssen im Wesentlichen zwei Aufgaben bewältigt werden: Zum einen die (statische) Erweiterung der Geschäftsobjekte um Zeitstempel sowie um Relationsobjekte, und zum anderen die (dynamische) Steuerung der zeitbezogenen Datenzugriffe, und das sowohl lesend als auch schreibend. Als Beispiel für eine mögliche Implementierung soll hier das „pleXX“-Framework dienen [Dem03a]. Es erfüllt beide oben genannten Aufgaben und besteht dementsprechend auch aus zwei Teilen: Einem Codegenerator, der die Implementierung der Business-Objekte auf die Historisierung vorbereitet, sowie einer Laufzeitumgebung, die alle Datenzugriffe überwacht und die temporalen Aspekte beachtet.

Der generative Ansatz erlaubt es dem Entwickler, sich bei der Modellierung der Geschäftsobjekte ausschließlich auf die Fachlichkeit zu konzentrieren. Das Objektmodell wird als Zustand zu *genau einem Punkt in der Zeit* entworfen. Die historisierten Teile müssen lediglich als solche markiert werden. So kann beispielsweise eine 1-zu-1 Beziehung von der Person zu dessen Ehepartner modelliert werden, denn die Geschäftslogik besagt, dass man zu einem Zeitpunkt nur einen Ehepart-

| STEUER | | | | | |
|--------|---------|------------|------------|----------|------|
| ID | VERSION | VALID_FROM | VALID_TO | NAME | WERT |
| 1 | 1 | 01.01.1975 | | Ermäßigt | 7% |
| 2 | 1 | 01.01.1975 | 01.01.1986 | Normal | 14% |
| 2 | 2 | 01.01.1986 | 01.01.1992 | Normal | 15% |
| 2 | 3 | 01.01.1992 | | Normal | 16% |

Tabelle 1: Temporale Mehrwertsteuerdaten

| ARTIKEL | | | | |
|---------|---------|------------|----------|-----------|
| ID | VERSION | VALID_FROM | VALID_TO | NAME |
| 1 | 1 | 01.01.1975 | | Milch |
| 2 | 1 | 01.01.1975 | | Fernseher |

Tabelle 2: Temporale Artikeldaten

| ARTIKEL_STEUER_RELATION | | | | |
|-------------------------|------------|------------|------------|-----------|
| ID | VALID_FROM | VALID_TO | ARTIKEL_ID | STEUER_ID |
| 1 | 01.01.1975 | | 1 | 1 |
| 2 | 01.01.1975 | 01.01.1980 | 2 | 1 |
| 2 | 01.01.1980 | | 2 | 2 |

Tabelle 3: Temporale Relation

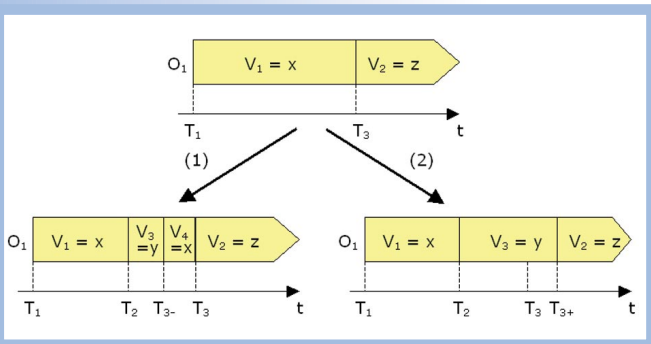


Abb. 3: Objektänderung vor und nach T₃

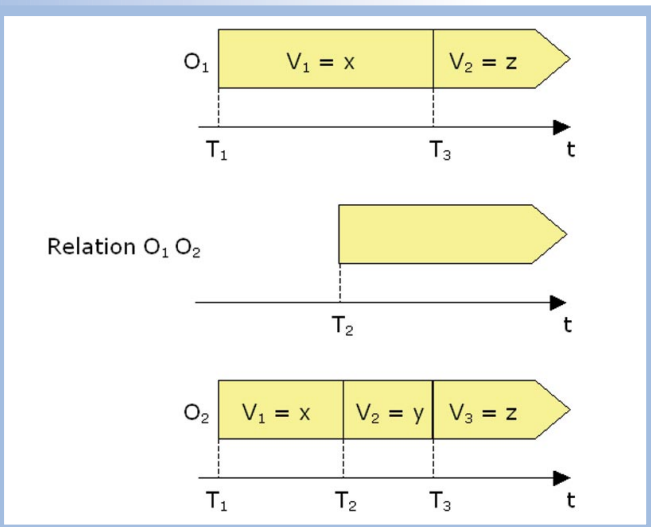


Abb. 4: Historisierung einer Beziehung

ner haben darf. Die Kennzeichnung als historisierte Beziehung führt dazu, dass der Codegenerator ein entsprechendes Relationsobjekt erzeugt. Dieses Objekt wird später allerdings nur intern und zur Abbildung durch den O/R Mapper verwendet. Die Anwendung selbst wird davon nichts bemerken.

Kernaspekt der Laufzeitumgebung ist, dass alle Objektanfragen niemals direkt an den O/R Mapper gerichtet werden dürfen, sondern nur über das Framework. Ferner müssen alle Transaktionen ebenfalls über das Framework abgewickelt werden. Dabei hat jede Transaktion genau definierte Zeitgrenzen für die Gültigkeit der dort durchgeführten Änderungen. Diese Zeitgrenzen steuern die Sichtbarkeit von bestehenden Objektversionen und die Gültigkeitsgrenzen neuer Versionen, die durch Objektänderungen innerhalb der Transaktion entstehen. Nur so kann gewährleistet werden, dass die Objektlebenszyklen stets disjunkt und überlappungsfrei sind. Bei freier und unkontrollierter Manipulation von Gültigkeitsgrenzen kann andernfalls sehr schnell ein Objekt zu einem Zeitpunkt doppelt oder gar nicht mehr existieren. Daneben bietet die Laufzeitumgebung natürlich noch Möglichkeiten zur Navigation in der Zeit, zum Abfragen von konkreten Objekt- und Relationsversionen zu bestimmten Zeitpunkten bzw. -räumen, etc.



Seine ganze Mächtigkeit spielt das Framework allerdings erst in Kombination mit einem weiteren Feature aus: der Constraintprüfung [Dem03b]. Dabei werden im Geschäftsmodell Gültigkeitsregeln definiert, die prüfen, ob ein konkretes Objektnetz fachlich korrekt ist. Genau wie bei der Modellierung der Geschäftsobjekte selbst können auch diese Constraints als Prüfungen zu genau einem Zeitpunkt definiert werden. Bei Änderungen an historisierten Objekten sorgt nun das Framework dafür, dass z. B. auch bei rückwirkender Gültigkeit alle relevanten Constraints zu allen relevanten Zeitpunkten überprüft werden. Dafür wird falls nötig das Objektnetz im Hintergrund auf der Zeitachse von Punkt zu Punkt verschoben, bis alle erforderlichen Stellen abgearbeitet sind.

Fazit

Die Speicherung von temporalen Daten wird weder von Datenbanken noch von O/R Mappern in vollem Umfang inhärent unterstützt. Zur effizienten und fehlerfreien Behandlung historisierter Geschäftsmodelle empfiehlt sich ein zweiteiliges Framework, bestehend aus einem Codegenerator und einer Laufzeitumgebung. Der Einsatz eines solchen Frameworks hat sich in der Praxis bereits in mehreren Projekten bewährt.

Literatur

- [Dem03a] A. Demelt, Ein Framework für Business-Objekte, in: JavaSPEKTRUM, 01/2003
- [Dem03b] A. Demelt, Objektvalidierung mit Constraints, in: JavaSPEKTRUM 04/2003
- [Sno00] R. T. Snodgrass, Developing Time-Oriented Database Applications in SQL, Morgan Kaufman Publishers, 2000



Dipl.-Inf. (FH) Achim Demelt ist Softwarearchitekt und Projektleiter bei der eXcellent solutions GmbH in Ulm. Einer seiner Schwerpunkte ist die Steigerung der Effizienz bei der Implementierung datengetriebener Anwendungen in Java.
E-Mail: a.demelt@excellent.de.

Weiterführende Informationsquellen <http://www.excellent.de>