ALLES FLIESST: TEMPORALE DATENHALTUNG IN RELATIONALEN DATENBANKEN

Dass „alles fließt", wusste bereits Heraklit im antiken Griechenland. Alle Dinge sind ständiger Veränderung unterworfen, auch die Informatik. Dieser Artikel beschäftigt sich mit dem Management von sich verändernden (temporalen) Daten. Nach der Vorstellung der temporalen Kernkonzepte wird erläutert, wie diese Konzepte in relationalen Datenbanken umgesetzt werden können. Ein Blick über den Tellerrand vermittelt, wie aktuelle Datenbanksysteme temporale Konzepte unterstützen, und zeigt weitere Lösungsansätze auf.

der autor

Christoph Schmidt-Casdorff
(c.schmidt-casdorff@iks-gmbh.com)
ist als IT-Berater bei der iks Gesellschaft für Informations- und Kommunikationssysteme tätig. In großen Kundenprojekten beschäftigt er sich mit dem Thema der modellgetriebenen Softwareentwicklung und mit flexiblen Softwarearchitekturen mit JEE, Spring und OSGi.

Die Behandlung des Zeitbezugs (Temporalität) von Informationen und Daten ist ein komplexes Thema der Datenhaltung. Sind die Informationen gültig? Wann haben sie sich gegebenenfalls geändert? Inwieweit muss das System auf Änderungen der Wirklichkeit reagieren und zu welchem Zeitpunkt tut es dies? Anhand eines einfachen Adressverwaltungs-Systems möchte ich in diese Problematik einführen.

Die allseits bestens bekannte Frau Erika Mustermann lebt in Köln und wird unter ihrer dortigen Adresse im System verwaltet. Am 15.02.2011 zieht sie nach Berlin. Dieser Umzug wird am 23.02.2011 in das System eingepflegt. Ihren nächsten Umzug bereitet Frau Mustermann gut vor und informiert über den am 20.10.2011 bevorstehenden Umzug nach Dortmund rechtzeitig. Bereits am 01.10.2011 kann in das System der bevorstehende Umzug eingepflegt werden (siehe Tabelle 1). Auf Basis dieser Datenlage beleuchten die folgenden Szenarien unterschiedliche Aspekte der temporalen Datenhaltung.

Wirksamkeit von Informationen

Wo wohnte Frau Mustermann im August 1990? Bis zum 15.02.2011 lebte sie in Köln, ab dem 15.02.2011 in Berlin. Die Information über den Wohnsitz ist also zeitabhängig (temporal) und kann nur durch zusätzliche Angabe eines Zeitpunkts ermittelt werden.

Frau Mustermanns Wohnsitz besitzt mehrere temporale Ausprägungen (*Versionen*). Jede dieser Versionen ist während eines bestimmten Zeitraums wirksam (*Wirksamkeit* oder *effectivity*, wonach dieses Konzept in [Fow] referenziert wird), d. h. für jeden Zeitpunkt aus diesem Zeitraum ist die Information gültig. Jede Version erhält einen Zeitstempel in Form eines Zeitintervalls, der die Wirksamkeit dieser Version beschreibt.

Datum	
15.02.2011	Umzug nach Berlin
23.02.2011	Neuer Wohnsitz Berlin eingepflegt
01.10.2011	Zukünftiger Wohnsitz Dortmund eingepflegt
20.10.2011	Umzug nach Dortmund

Tabelle 1: Chronologie der Wohnsitzänderungen von Erika Mustermann.

	Wohnsitz	actual from	actual to	record from	record to
1	Köln	01.01.1990	offen	01.01.1990	23.02.2011
2	Köln	01.01.1990	15.02.2011	23.02.2011	offen
3	Berlin	15.02.2011	offen	23.02.2011	01.10.2011
4	Berlin	15.02.2011	20.10.2011	01.10.2011	offen
5	Dortmund	20.10.2011	offen	01.10.2011	offen

Tabelle 2: Protokoll der Wohnsitzänderungen in beiden Zeitebenen.

Wirklichkeit und System

Am 21.02.2011 wurde an Frau Mustermann ein Brief aus der Adressverwaltung versendet. Dieser Brief ging nach Köln, denn der Umzug war dem System nicht bekannt, obwohl er in Wirklichkeit bereits stattgefunden hatte. Ereignisse sind dem System häufig nicht unmittelbar bekannt, sondern es gibt einen Unterschied zwischen dem Ereignis (in der Realität) und dem Zeitpunkt, zu dem das System darüber informiert wird. Ein Zeitpunkt aus der Realität (*actual time*, vgl. [Fow]) ist von einem Zeitpunkt zu unterscheiden, zu dem das System von einer Information Kenntnis erlangt (*record time*, vgl. [Fow]). Bitemporalität zeichnet sich dadurch aus, dass diese beiden Zeitebenen gepflegt werden. Jede Version erhält zwei Zeitstempel, einen für jede Zeitebene. Um eine Aussage über das System zu erhalten, ist immer ein Zeitpunkt als Kombination beider Zeitebenen notwendig. Dieser Zeitpunkt heißt bitemporal.

An dieser Stelle sollen einige Festlegungen getroffen, die das Rüstzeug für den weiteren Artikel bilden: Ein bitemporaler Zeitpunkt wird im Weiteren in der Form (*record, actual*) angegeben. Der Beginn des Zeitintervalls der *record time* wird mit *record from* notiert, das Ende mit *record to*. Entsprechendes gilt für die *actual time*.

Ein Zeitintervall ist im Beginn abgeschlossen und im Ende offen. Eine Angabe des Endes wird als *offen* bezeichnet, wenn kein bekanntes Endedatum vorliegt und die

ID	Vorname	Nachname	actual from	actual to	record from	record to
2	Erika	Mustermann	01.01.1990	01.12.2011	07.12.2012	offen
3	Erika	Mustermann	01.11.2012	offen	07.12.2012	offen

Tabelle 3: Zeitweise Deaktivierung von Erika Mustermann.

Information bis auf weiteres gültig ist. In der Literatur (vgl. [Sno99], [Joh10]) sind auch andere Begriffe für die beiden Zeitebenen gebräuchlich. *Actual time* wird auch als *valid time* oder *effective time* und *record time* auch als *transaction time* oder *assertion time* bezeichnet.

Nachvollziehbarkeit

Frau Mustermann wundert sich, warum ein Brief mit Poststempel vom 20.02.2011 an sie falsch adressiert wurde. Das erklärt sich daraus, dass am 20.02.2011 die neue Adresse nicht bekannt war. Um nachvollziehen zu können, warum das System so entschieden hat, muss das System seine vollständige Entscheidungsgrundlage zum 20.02.2011 rekapitulieren können. Dazu ist es wichtig, dass bei der Änderung einer Information die ursprüngliche unverändert mit Änderungsdatum dokumentiert wird.

Zukünftige Wirksamkeit

Frau Mustermann hat aus den Schwierigkeiten bei ihrem letzten Umzugs gelernt und informiert über ihren am 20.10.2011 bevorstehenden Umzug nach Dortmund rechtzeitig. Bereits am 01.10.2011 kann in das System der bevorstehende Umzug eingepflegt werden. Die neue Version des Wohnsitzes hat den 01.10.2011 als *record time* und den 20.10.2011 als *actual time*.

Um eine Mailing-Aktion zu starten, muss der Druckerei im Vorfeld eine Liste aller Adressaten übermittelt werden. Eine Mailing-Aktion, die am 10.10.2011 gestartet wurde, konnte schon die zukünftige Adresse von Frau Mustermann berücksichtigen. Wird jede Änderung protokolliert und mit *actual* und *record time* versehen, so ergibt sich das Änderungsprotokoll aus Tabelle 2.

Es gibt also auch Informationen mit zukünftiger Wirksamkeit. In diesen Fällen liegt der Beginn der *actual time* in der Zukunft. Da die *record time* beschreibt, wann das System Kenntnis über ein Ereignis erhalten hat, liegt deren Beginn niemals in der Zukunft. Dies würde der Bedeutung der *record time* widersprechen.

Das System wurde am 01.01.1990 aufgesetzt – daher ist der 01.01.1990 der initiale Zeitstempel.

Rückwirkende Neuberechnung

Lassen Sie sich mit mir auf folgendes Gedankenspiel ein. Es liegt ein Lebenslauf von Frau Mustermann vor, der am 17.02.2011 erstellt wurde. Dort wurde als aktueller Wohnsitz Köln ermittelt. Frau Mustermann wünscht nun einen aktuellen Lebenslauf. Welchen Wohnsitz ermittelt ein aktueller Lebenslauf zum 17.02.2011?

In bitemporaler Weise formuliert, soll das System auf Basis des heutigen Wissensstands den Wohnsitz von Frau Mustermann am 17.02.2011 bestimmen (es ist Berlin). Der bitemporale Auswertungszeitpunkt hat das heutige Datum als *record time* und den 17.02.2011 als *actual time*. Dieses Szenario der rückwirkenden Neuberechnung spielt in der Versicherungsbranche oder Telekommunikation eine Rolle, wo Verträge nach einer rückwirkenden Änderung neuberechnet werden.

Zeitweilige Deaktivierung

Angenommen, Frau Mustermann ruht zeitweise im Adressverwaltungssystem. In Tabelle 3 wird am 07.12.2012 vermerkt, dass Frau Mustermann zwischen dem 01.12.2011 und dem 01.11.2012 nicht im System aktiv war. Nach dem 01.11.2011 wurde sie reaktiviert. Dieses Szenario tritt häufig in jeder Form von Vertragsverwaltungen auf. Verträge ruhen vorübergehend, müssen aber gegebenenfalls zu einem späteren Zeitpunkt wieder aktiviert werden können.

Interpretation der Bitemporalität

Wie hat man sich die Zeitebenen *actual time* und *record time* und ihr Zusammenwirken vorzustellen? Folgende Interpretation liefert eine intuitive Vorstellung von Bitemporalität. *Durch die Wahl eines (Referenz-)Zeitpunkts der „record time" wird der Wissensstand des Systems festgelegt. Dieser Wissensstand besteht aus allen Versionen, die bezüglich der „record time" zu diesem Zeitpunkt wirksam sind. Auf Basis dieses Wissensstands kann das System nun Entscheidungen treffen.*

Am 30.03.2011 wusste das System nur über die Wohnsitze in Berlin und Köln und um die jeweiligen Umzüge Bescheid. Ab dem 01.10.2011 wusste das System zusätzlich um den Umzug nach Dortmund. Folgende Beispiele sollen das Zusammenspiel von *record time* und *actual time* demonstrieren:

- Um festzustellen, wo Frau Mustermann am 17.02.2011 gewohnt hat, wird der

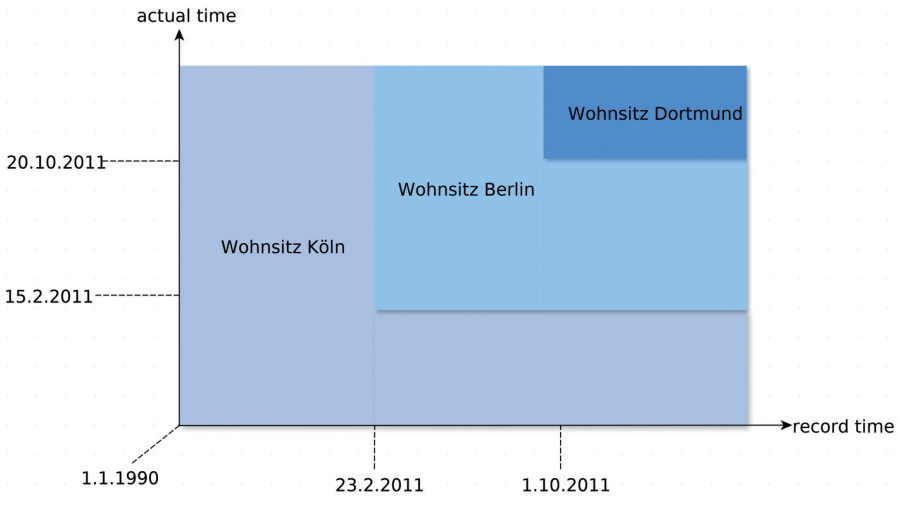

Abb. 1: Wirksamkeit des Wohnsitzes in Abhängigkeit vom bitemporalen Zeitpunkt.

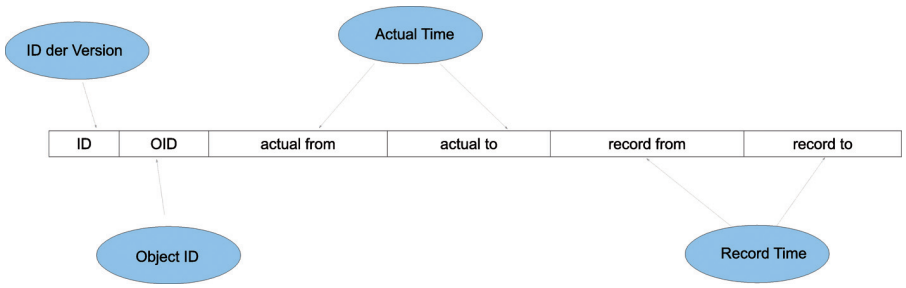

Abb. 2: Spalten zur Unterstützung des bitemporalen Konzepts.

heutige Tag als *record time* gewählt. Aus heutiger Sicht sind die Umzüge von Köln nach Berlin und von dort nach Dortmund bekannt. Das System stellt Berlin als Wohnsitz zum 17.02.2011 fest.

- Um eine Entscheidung des Systems zu einem Zeitpunkt nachzuvollziehen, wird dieser Zeitpunkt als *record time* gewählt. Der sich daraus ergebende Wissenstand diente als Entscheidungsgrundlage. In einem auf den 20.02.1011 datierten Lebenslauf war als aktueller Wohnsitz Köln angegeben. Um zu prüfen, ob das System den Wohnsitz aus seiner damaligen Sicht korrekt ermittelt hat, wird der 20.02.2011 als Referenzzeitpunkt für die *record time* gewählt. Zu diesem Zeitpunkt wusste das System noch nichts über den Umzug nach Berlin und hat daher auf Basis der damals vorliegenden Informationen korrekt entschieden.

Abbildung 1 zeigt die Wirksamkeit des Wohnsitzes in Abhängigkeit des Zeitpunkts, bestehend aus *record time* und *actual time*.

Möglichkeiten des bitemporalen Konzepts

Mit Hilfe des bitemporale Konzepts können die folgenden Aufgaben erledigt werden:

- Die Entscheidungsgrundlage des Systems kann zu jedem Zeitpunkt nachvollzogen und wiederhergestellt werden.
- Es können Datensätze angelegt werden, die erst zu einem zukünftigen Zeitpunkt gültig sein werden.
- Auf Basis zukünftiger Datenstände können Prognosen erstellt werden.
- Zu jedem Zeitpunkt in der Vergangenheit können Prognosen mit gleichem Ergebniswiederholt werden.

Die Anforderungen eines Systems an Temporalität können vielfältig sein. Es mag sein, dass der Zeitbezug ignoriert werden kann, weil immer nur der aktuelle Datenbestand ohne Sicht in die Zukunft oder Vergangenheit notwendig ist. Es gibt auch Anwendungsfälle, für die eine der beiden Zeitebenen ausreicht. Der Umfang temporaler Aspekte eines Systems hängt stark von den jeweiligen Anforderungen ab.

Temporale Invarianzen

Im Alltag ist die Identität eines Objekts in der Regel offensichtlich. Die Abstraktion von der Wirklichkeit durch IT-Modelle macht diese Entscheidung in IT-Systemen oft schwierig. Im temporalen Umfeld ist zu unterscheiden, wann es sich um ein neues Objekt oder um eine neue Version eines bereits vorhandenen Objekts handelt (*continuity*, vgl. [Fow]). Diese Unterscheidung ist nur fachlich zu treffen, indem bestimmte Attribute als temporal-invariant erklärt werden. Sie zeichnen sich dadurch aus, dass sie für alle Versionen eines Objekts denselben Wert haben. Ändert sich der Wert des Attributs, so führt diese Änderung nicht zu einer Version, sondern zu einem neuen Objekt. Ein Objekt wird durch all seine temporal-invarianten Attribute definiert.

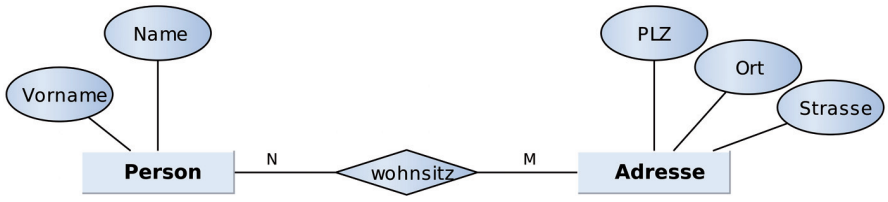

Abb. 3: ERM des Adressverwaltungssystems.

Temporalität und relationale Datenbanken

Trotz neuer Verfahren (*non-SQL*) trifft man heutzutage in der Regel *relationale Datenbanksysteme (RDBMS)* als System zur Datenhaltung an. Daher ist die Umsetzung von (Bi-)Temporalität auf RDBMS von besonderer Bedeutung.

Der naheliegende Ansatz heißt: *Jede Version ein Datensatz.*

Bitemporale Tabellen enthalten die Spalten *actual from* bis *record to*. Neben diesen wird eine eindeutige ID für die Version und ein Identifikator-OID für das zu Grunde liegende Objekt (siehe „Temporale Invarianzen") unterstützt. Mit der Erweiterung um die in Abbildung 2 beschriebenen Spalten kann eine Tabelle bitemporal gepflegt werden.

Hier ein Beispiel: Das *Entity Relationship Model (ER-Model)* aus Abbildung 3 beschreibt das Adressverwaltungssystem. Dabei ist das Modell stark vereinfacht und spiegelt natürlich nicht die Realität wider. In Tabelle 4 sind die Daten der Tabellen „Person" und „Adresse" zu erkennen.

Die Entität „Adresse" wird als eigenständig angesehen. Ihr Lebenszyklus ist unabhängig von demjenigen der Entität der „Person". Die Verbindung beider wird durch „Wohnsitz" beschrieben. Zeitliche Veränderungen im Adressverwaltungssystem werden durch einen Umzug ausgelöst, der durch Änderungen der Tabelle „Wohnsitz" dokumentiert wird. Die in Tabelle 2 dokumentierten Umzüge können daher in die Tabelle „Wohnsitz" überführt werden.

Temporale Integrität

In diesem Abschnitt werden zwei entscheidende Integritätskriterien vorgestellt, denen temporale Daten genügen müssen.

Temporale Objektintegrität

Eine wichtige Konsistenzforderung an temporale Daten ergibt sich intuitiv. *Es darf zu*

ID	OID	Vorname	Nachname	actual from	actual to	record from	record to
1	1	Erika	Mustermann	01.01.1990	offen	01.01.1990	offen

Tabelle 4: Beispielhafter Datenstand der Tabelle „Person".

einem Zeitpunkt höchstens eine Version eines Objekts geben. Das gilt auch für bitemporale Zeitpunkte. Eine Verletzung dieser Konsistenzforderung würde bedeuten, dass zu einem Zeitpunkt zwei unterschiedliche Informationen über ein und dasselbe Objekt vorlägen – welch ein Chaos.

Mit der oben vorgestellten Interpretation der Bitemporalität können wir die temporale Objektintegrität überprüfen, indem wir einen beliebigen Zeitpunkt der *record time* wählen. Zu diesem Zeitpunkt der *record time* dürfen sich die Intervalle der *actual time* aller Versionen desselben Objekts nicht überschneiden.

Temporale referenzielle Integrität

Referenzielle Integrität für nicht-temporale Beziehungen sichert die Existenz referenzierter Objekte zu.

In temporalen Beziehungen reicht es nicht, die Existenz des referenzierten Datensatzes zuzusichern. Auch dessen Wirksamkeit zu jedem Zeitpunkt der Wirksamkeit des referenzierenden Datensatzes ist zu prüfen. Diese Forderung heißt temporale referenzielle Integrität und ist eines der Kernkonzepte temporaler Datenhaltung.

Relationen im temporalen Umfeld

Wir befinden uns im Umfeld von RDBMS. Was ist also naheliegender, als die Beziehung zwischen temporalen Tabellen über Fremdschlüsselbeziehungen zwischen Versionen herzustellen? Ändert sich die PLZ, so wird eine neue Version des Adress-Datensatzes angelegt (ID=5). Die bisherige Version wird gesichert (ID=3) und anschließend mit den neuen Gültigkeiten (ID=4) aktualisiert.

Tabelle 7 zeigt die neue Version, die durch die PLZ-Änderungen der Adresse „Dortmund, Poststr. 3" entstanden ist. Die PLZ-Änderung wurde am 01.11.2011 wirksam und am 07.11.2011 eingepflegt. Man beachte, dass sich die OID nicht verändert, denn es wird lediglich eine neue Version des Objekts mit der OID 3 erzeugt. Um die folgende Argumentation zu vereinfachen, wird lediglich die *record time* untersucht und für die *actual time* als fester Referenzzeitpunkt der 10.11.2011 gewählt. Adresse 4 spielt keine Rolle, da diese zum 10.11.2011 bezüglich der *actual time* nicht wirksam ist. Daher ist diese in Tabelle 7 ausgegraut.

Welchen Einfluss hat diese Änderung auf die Tabelle „Wohnsitz"? Würde sich nichts ändern, so würde „Wohnsitz 5" am 10.11.2011 auf „Adresse 3" verweisen (siehe Tabelle 6). Diese ist allerdings zu diesem Zeitpunkt nicht mehr wirksam. Durch die Änderung der Adresse verliert dieser Verweis die temporale referenzielle Integrität.

Abbildung 4 demonstriert die Verletzung der temporalen Integrität (die Graphik beschränkt sich der Übersichtlichkeit halber auf die *record time*). Im rot unterlegten Zeitraum verweist „Wohnsitz 5" zwar per Fremdschlüssel auf „Adresse 3", diese ist aber nicht wirksam.

Um die temporale referenzielle Integrität wiederherzustellen, muss der Datensatz „Wohnsitz 5" auf den neuen Datensatz „Adresse 5" verweisen. Dazu muss eine neue Version von „Wohnsitz 5" angelegt werden, die ihrerseits auf „Adresse 5" verweist (siehe Abbildung 5). Das Ergebnis dieser Versionierung ist in Tabelle 8 dokumentiert.

Um temporale referenzielle Integrität zu erhalten, führt die Änderung eines Satzes, auf den eine Fremdschlüssel-Beziehung verweist, zur Versionierung der referenzierenden Sätze. In unserem Fall bedingt die Änderung der Adresse die Versionierung des Wohnsitzes. Dieser Effekt kann sich fortpflanzen. Wird ein aus diesem Grund versionierter Datensatz selbst referenziert, so ist für den referenzierenden Datensatz ebenfalls eine Versionierung durchzuführen. Die Nachteile dieses Ansatzes sind die durch die kaskadierende Versionierung bedingte Datenredundanz, der komplexe Algorithmus und die schlechte Skalierung. Es ist nicht selten, dass die Änderung eines Datensatzes – je nach Datenmodell und Umfang des Datenbestands – Auswirkungen auf hunderte von mittelbar betroffenen Datensätzen hat.

Dieses Konzept kann zum Tragen kommen,

- wenn die Änderungsfrequenz nicht hoch, aber ein effizientes Lesen mit SQL-Mitteln erwünscht ist,
- wenn sich der Fortpflanzungseffekt der Versionierung nicht auswirkt, weil die Relationen keine transitiven Abhängigkeiten enthalten,
- wenn der Bestand nicht zu groß ist, sodass redundante Daten akzeptiert werden können,
- wenn komplexe Abfragen über mehrere temporale Tabellen ausgeführt werden.

Ansonsten ist dieser Ansatz (gerade bei komplexen Datenstrukturen oder großen Datenbeständen) nicht tragfähig. Im fol-

ID	OID	PLZ	Ort	Straße	actual from	actual to	record from	record to
1	1	50823	Köln	Karlstr. 16a	01.01.1990	offen	01.01.1990	offen
2	2	15566	Berlin	Veilchenweg 2	01.01.1990	offen	01.01.1990	offen
3	3	44137	Dortmund	Poststr. 3	01.01.1990	offen	01.01.1990	offen

Tabelle 5: Beispielhafter Datenstand der Tabelle „Adresse".

ID	ID Person	ID Adresse	actual from	actual to	record from	record to
1	1	1	01.01.1990	offen	01.01.1990	23.02.2011
2	1	1	01.01.1990	15.02.2011	23.02.2011	offen
3	1	2	15.02.2011	offen	23.02.2011	01.10.2011
4	1	2	15.02.2011	20.10.2011	01.10.2011	offen
5	1	3	20.10.2011	offen	01.10.2011	offen

Tabelle 6: „Wohnsitz" dokumentiert die zeitabhängige Beziehung zwischen „Person" und „Adresse".

ID	OID	PLZ	Ort	Straße	actual from	actual to	record from	record to
3	3	44137	Dortmund	Poststr. 3	01.01.1990	offen	01.01.1990	07.11.2011
4	3	44137	Dortmund	Poststr. 3	01.01.1990	01.11.2011	07.11.2011	offen
5	3	44138	Dortmund	Poststr. 3	01.11.2011	offen	07.11.2011	offen

Tabelle 7: Datenlage der Tabelle „Adresse" nach Änderung der PLZ.

genden Abschnitt wird eine Lösung aus diesem Dilemma aufgezeigt.

Alternative Umsetzungen in RDBMS

Werden Relationen zwischen Versionen gebildet, ändern sich referenzierter und referenzierender Datensatz in gegenseitiger Abhängigkeit. Dieser Effekt ist unerwünscht und widerspricht dem herkömmlichen Verständnis referenzieller Integrität.

Der in [Sno99] vorgestellte Ansatz schlägt daher vor, dass die referenzierte Rolle in einer Beziehung nicht durch eine Version, sondern durch das Objekt (siehe Abschnitt „Temporale Invarianzen") eingenommen wird.

Für den Aufbau einer Relation wird die Objekt-Identität (eine Teilmenge der temporal invarianten Attribute) genutzt. Häufig wird für die Objekt-Identität ein *surrogate key* eingeführt.

Objekte sind in der Datenbank nicht als Entität modelliert, sondern werden durch die Menge alle Versionen zu gleicher Objekt-Identität repräsentiert. Wird die Beziehung zwischen „Adresse" und „Wohnsitz" gemäß diesem Konzept modelliert, hat die Änderung von „Adresse" aus obigem Beispiel (siehe Tabelle 7) keinen Einfluss auf den zugehörigen Datensatz aus der Tabelle „Wohnsitz". Der Inhalt von „Wohnsitz 5" nach Änderung ist in Tabelle 9 dokumentiert. Die Beziehung zu „Adresse" wird über die Objekt-Identität (mittels OID) aufgebaut. Wie aus Abbildung 6 ersichtlich, kennt der referenzierende Datensatz („Wohnsitz 5") nicht mehr die Version, sondern nur noch das Objekt, auf das er sich bezieht.

Es besteht kein expliziter Zusammenhang zwischen zwei Versionen, der sich im Datenmodell ausdrückt. Dieser Zusammenhang muss zu jedem Zeitpunkt explizit hergestellt werden, indem unter den möglichen Versionen (des referenzierten Objekts) mittels Einschränkungen der Zeitebenen die wirksame Version ermittelt wird.

Nach obigem Beispiel liefert die Auflösung der Beziehung von „Wohnsitz 5" zur Tabelle „Adresse" zwei mögliche Versionen: Je nach Zeitpunkt muss entschieden werden, ob „Adresse 3" oder „Adresse 5" wirksam ist. Dieser Ansatz reduziert deutlich die Datenmenge, da keine kaskadierenden Versionierungen von abhängigen Datensätzen vorgenommen werden. Der Zugriff auf Relationen mit SQL-Mitteln ist allerdings recht aufwändig (detaillierte Informationen hierzu finden sich in [Sno99]).

Da Objekte keine Datenbank-Entitäten sind, kann in einem RDBMS die Relation nicht als Fremdschlüssel-Beziehung mit entsprechenden Zusicherungen modelliert werden. Mit Mitteln eines RDBMS können also weder Zusicherungen über die Existenz eines Objekts noch über die einer Version gemacht werden.

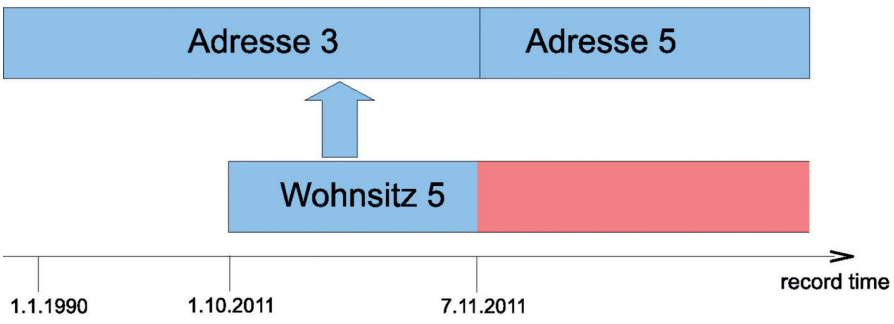

Abb. 4: Verletzung der temporalen referenziellen Integrität.

Temporale Aspekte relationaler Datenbanken

Sollen bitemporale Konzepte in relationalen Datenbanken umgesetzt werden, so sind dabei einige technische Aspekte zu beachten.

Darstellung von „offen"

Für die Darstellung des logischen Wertes

„offen" für ein Zeitintervall findet man in der Literatur (z. B. [Sno99]) drei unterschiedliche Ansätze: Entweder wird dieser Zustand durch den Wert „NULL" (NULL-Ansatz) beschrieben, durch einen maximalen Wert (z. B. 31.12.9999) (MAX-Ansatz) identifiziert oder dadurch kenntlich gemacht, dass Beginn und Ende identisch sind (POINT-Ansatz).

Alle Ansätze haben ein unterschiedliches Verhalten bezüglich Indizes über die Spalten der Zeitstempel. Vor allem haben sie aber Einfluss auf die Komplexität von Abfragen, ob ein Zeitpunkt innerhalb eines Wirksamkeitsintervalls liegt (siehe unten).

Datenzugriff in SQL

Jede Abfrage auf Versionen muss zusätzlich die Wirksamkeit der Version zu einem gegebenen Zeitpunkt prüfen. Dies muss in beiden Zeitebenen und – je nach Wahl der Darstellung von „offen" – mit einer Sonderbehandlung für offene Intervalle geschehen (in [Sno99] wird dieses Thema detailliert beschrieben).

Entsprechend komplex ist die Abfrage einer Relation, wenn diese auf „Version zu Objekt" basiert. In diesem Fall muss die Prüfung der Wirksamkeit für beide Seiten der Relation durchgeführt werden. Sind die Abfragen bekannt, so kann durch parametrisierte *SQL-Views* die Komplexität verborgen werden.

Granularität der Zeitstempel

Die Granularität der Zeitstempel definiert den zeitlichen Mindestabstand zwischen zwei Änderungen. Werden Zeitstempel taggenau gewählt, so kann es höchstens eine Version eines Satzes pro Tag geben. Werden Zeitstempel sekundengenau gewählt, muss zwischen zwei Änderungen entsprechend mindestens eine Sekunde liegen.

In vielen Anwendungen kann die *actual time* taggenau gewählt werden, denn aus fachlicher Sicht spiegelt eine feinere Genauigkeit nicht die Realität wider. Eine zu feine Granularität kann bei der Bestimmung der Versionen zu falschen Ergebnissen führen. Die *record time* wird aus technischen Gründen zumindest sekundengenau gewählt, um keine unerwünschten Einschränkungen auf die Änderungsfrequenz eines Satzes zu erfahren.

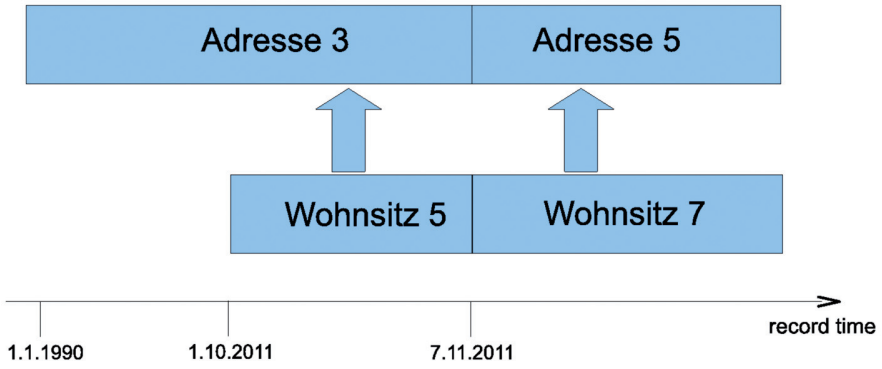

Abb. 5: Situation nach Wiederherstellung der temporalen referenziellen Integrität.

Temporale Normalisierung

Tabelle 7 dokumentiert die durch die Änderung der PLZ an der Dortmunder Adresse bedingten neuen Versionen („Adresse 4/5"). Die neue Versionen enthalten alle nicht durch die Versionierung betroffenen Spalten, wie z. B. Ort, Straße, mit unveränderten Werten.

Alle Spalten einer Tabelle werden gemeinsam versioniert, auch wenn sich einige Spalten nicht verändert haben. So kann es mehrere Versionen geben, die für eine Spalte einen unveränderten Wert enthalten. Diese Werte werden für jede Version redundant übernommen und führen zu einer nicht notwendigen Datenflut. Dieser Effekt wirkt sich vor allem dann stark aus, wenn sich einige Attribute der Spalten sehr häufig, andere dagegen selten ändern. Werden Zeiträume mit gleichen Spaltenwerten zusammengefasst, so wird dies als temporale Normalisierung bezeichnet. Um dies in RDBMS zu erreichen, müssen im Design der Datenbank alle Spalten, die sich gemeinsam verändern, in eine eigene Tabelle ausgegliedert werden.

Temporale Unterstützung in RDBMS

In diesem Artikel habe ich Konzepte der temporalen Datenhaltung und deren

ID	ID Person	ID-Adresse	actual from	actual to	record from	record to
5	1	3	20.10.2011	offen	01.10.2011	07.11.2011
6	1	4	20.10.2011	01.11.2011	07.11.2011	offen
7	1	5	01.11.2011	offen	07.11.2011	offen

Tabelle 8: Datenlage der Tabelle „Wohnsitz" nach Teilung.

ID	OID Person	OID-Adresse	actual from	actual to	record from	record to
5	1	3	20.10.2011	offen	01.10.2011	07.11.2011

Tabelle 9: Datenlage der Tabelle „Wohnsitz" nach Änderung der PLZ (gemäß Tabelle 7).

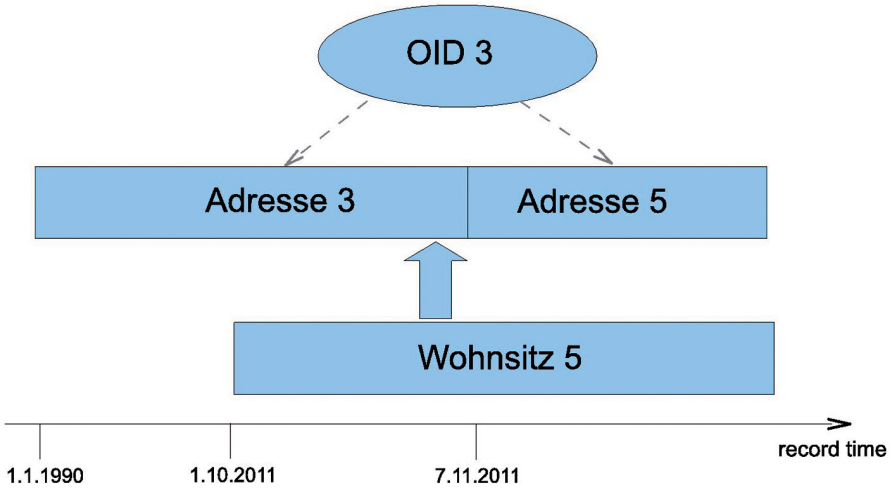

Abb. 6: Referenzierung via Objekt-Identität (nur „record time").

Umsetzung in RDBMS vorgestellt, ohne spezifische temporale Erweiterungen von RDBMS zu nutzen. Alle Ansätze, bitemporale Konzepte in Datenbanken umzusetzen, basieren auf diesen Konzepten, verfolgen aber unterschiedliche Strategien, diese umzusetzen. Bei temporalen Datenbanken (z. B. [Ter]) ist Bitemporalität integraler Bestandteil und ein immanenter Aspekt des Systems.

Andere Ansätze erweitern herkömmliche RDBMS um temporale Aspekte und erweitern SQL. Der bekannteste Ansatz ist „TSQL2" (vgl. [Sno96-a] und [Sno96-b]). Leider ist TSQL2 noch nicht in den SQL-Standard eingeflossen und das entsprechende Standardisierungsprojekt wird nicht weitergeführt.

Namhafte RDBMS wie „Oracle 11" (mit dem „Oracle Workspace Manager", vgl. [Ora]), „DB2" oder „PostGres" (vgl. [Tem]) bieten temporale Erweiterungen mit unterschiedlichem Funktionalitätsumfang an. Der „Oracle Workspace Manager" umfasst beispielsweise die Funktionalität von TSQL2.

Ein interessanter Ansatz wird in [Tim] verfolgt. Dort wird nicht auf einen erweiterten Umfang von SQL gesetzt, sondern temporale Statements à la TSQL2 werden durch ein Framework in Standard-SQL-Statements überführt. Diese werden gegen eine nicht-temporale Datenbank ausgeführt.

Ein dritter Ansatz behandelt Temporalität in Frameworks, die die Temporalität mit jeweils eigenen Konzepten umsetzen. Das zu Grunde liegende RDBMS wird in keiner Weise temporal erweitert. Innerhalb des Frameworks wird ein temporales Datenmodell gepflegt, das einerseits eine Programmierschnittstelle anbietet und andererseits die Persistenz dieses Modells in einem RDBMS übernimmt. Das Datenbank-Datenmodell muss dazu um spezifische temporale Spalten erweitert werden, die durch das Framework gepflegt werden. Gegen die Datenbank werden Standard-SQL-Statements abgesetzt (vgl. [Ass10]).

Fazit und Ausblick

Mit dem Management temporaler Daten in einer Datenbank ist es nicht getan. Ebenso wichtig ist es, temporale Konzepte auf die Ebene der Anwendung zu heben und einen Zugriff auf temporale Konzepte innerhalb einer Anwendung bereitzustellen.

Es gibt interessante Arbeiten auf dem Gebiet der temporalen Muster (z. B. [Fow], [And98], [Lan98]). Mit deren Hilfe ist es möglich, temporale Konzepte in tragfähige Daten- und Programmiermodelle umzusetzen. Allerdings geht die Diskussion temporaler Muster über den Rahmen dieses Artikels hinaus.

Eine Umsetzung temporaler Konzepte besteht daher immer aus dem Management temporaler Daten in einem RDBMS und einem Programmiermodell für die temporalen Konzepte. Beide Aspekte sind zu beachten.

Leider ist Temporalität im Umfeld relationaler Datenbanken noch nicht standardisiert, sodass jeder Anbieter unterschiedliche Funktionalität mit unterschiedlichen Konzepten anbietet. Derzeit sind wir noch weit von einer nahtlosen, standardisierten Integration temporaler Konzepte in RDBMS entfernt. Selbst für Datenbanksysteme mit weitreichender Unterstützung ist derzeit die Umsetzung bitemporaler Konzepte nicht ohne den Einsatz eines Frameworks zu empfehlen. Dieses Framework stellt die temporalen Konzepte innerhalb einer Anwendung bereit. ■

Literatur & Links

[And98] F. Anderson, A Collection of History Patterns, 1998, siehe: http://hillside.net/plop/plop98/final_submissions/P63.pdf

[Ass10] Asserted Versioning LLC, The Asserted Versioning Framework, 2010, siehe: http://www.assertedversioning.com/ProductsAndServices.asp

[Fow] M. Fowler, Development of Further Patterns of Enterprise Application Architecture, siehe: http://martinfowler.com/eaaDev/

[Joh10] T. Johnston, R. Weis, Managing Time in relational databases, Morgan Kaufmann 2012

[Lan98] M. Lange, Time Patterns, 1998, siehe: http://hillside.net/plop/plop98/final_submissions/P04.pdf

[Ora] Oracle, Orcale Workspace Manager (OWM), siehe: http://www.oracle.com/technetwork/database/enterprise-edition/index-087067.html

[Sno96-a] R.T. Snodgrass, M.H. Böhlen, C.S. Jensen, A. Steiner, Adding Valid Time to SQL/Temporal, 1996, siehe: http://www.timeconsult.com/Publications/ansi-96-501.pdf

[Sno96-b] R.T. Snodgrass, M.H. Böhlen, C.S. Jensen, A. Steiner, Adding Transaction Time to SQL/Temporal, 1996, siehe: http://www.timeconsult.com/Publications/ansi-96-502.pdf

[Sno99] R. Snodgrass, Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann Publishers Inc. 1999

[Tem] Temporal Postgres, siehe: http://temporal.projects.postgresql.org/

[Ter] Teradata, siehe: http://www.teradata.com/database/teradata-temporal/

[Tim] TimeConsult, siehe: http://www.timeconsult.com/Software/Software.html