Universität Rostock

Traditio et Innovatio

Masterarbeit

Temporale Aspekte und Provenance-Anfragen im Umfeld des Forschungsdaten-managements

Frank Meyer
Matrikelnummer: 209204737
Abgabedatum: 13.09.2016

Gutachter
Prof. Dr. Andreas Heuer
Dr. Ralf Prien

Betreuer
Dipl.-Inf. Ilvio Bruder
Dipl.-Inf. Susanne Jürgensmann
Dr. Susanne Feistel

Universität Rostock
Institut für Informatik
Lehrstuhl für Datenbank- und Informationssysteme
Albert-Einstein-Straße 22
18059 Rostock

Abstract

Zur Verbesserung der Qualität wissenschaftlicher Arbeiten besteht Interesse an der Rückverfolgbarkeit und Nachnutzung von Forschungsdaten. Die Erhebung von diesen Daten erfolgt einerseits manuell, andererseits automatisiert durch Instrumente und Sensoren der im Forschungsgebiet genutzten Geräte. Das Leibniz-Institut für Ostseeforschung Warnemünde betreibt eine instrumentierte Messplattform (GODESS) zur Untersuchung von physikalischen und chemischen Umweltparametern der Ostsee in verschiedenen Tiefen. Durch den modularen Aufbau und Konfiguration der Plattform ergeben sich Änderungen in den Datenformaten sowie Anpassungen der Auswerte- und Analysefunktionen. Im Kontext von Datenbanken entspricht dies Änderungen des relationalen Datenbankschemas sowie von SQL-Funktionen. In dieser Arbeit wird untersucht, unter welchen Voraussetzungen veränderte SQL-Funktionen auf veränderte Schemata angewendet werden können. Dazu wird im Konzept gezeigt, wie Veränderungen von SQL-Funktionen durch Techniken der temporalen Datenhaltung aufgezeichnet werden können. Das Festhalten und Verfolgen von Schemaänderungen wird zusätzlich skizziert. Danach erfolgt die Untersuchung der Anwendbarkeit von Funktionen auf sich verändernde Schemata sowie die Anwendbarkeit von Funktionen bei Funktionsänderungen. Die Ergebnisse werden zusammenfassend in Form von Tabellen präsentiert. Ferner wird auf Möglichkeiten der automatisierten Kompensation der Änderungen zur Wahrung der weiteren Anwendbarkeit der Funktionen mittels Schemaintegrationstechniken eingegangen. Danach wird der Zusammenhang der Herkunft von Daten und Aspekten der temporalen Datenhaltung beschrieben. Es folgt eine prototypische Umsetzung der Versionierung von SQL-Funktionen. Abschließend wird die Arbeit zusammengefasst und ein Ausblick auf weitere mögliche und weiterführende Themen gegeben.

Abstract

To improve the quality of scientific work it is necessary to trace research data and provide it for a subsequent use. The data is acquired manually as well as automatically, generated with instruments and sensors of the involved devices in the particular scientific discipline. The Leibniz Institute for Baltic Sea Research Warnemünde runs a profiling monitoring station (GODESS) to measure physical and chemical environmental data at several depths of the Baltic Sea. The modular structure and configuration of the platform leads to changes in the data formats and the functions by which the data is analyzed. In the context of databases these changes correspond to changes in the relational schemata and SQL-functions. This thesis analyses the requirements to use changed SQL-functions on changed schemata. For that purpose it will be demonstrated how changes of SQL-functions could be recorded with techniques of temporal data management. Furthermore the recording of relational schema changes will be outlined. The applicability of SQL-functions on changed schemata and the applicability of SQL-functions on changed functions will be analyzed. The results will be summarized and presented in tables. Moreover options to compensate the changes to reuse the function are shown with techniques of schema integration. The link between data provenance and temporal data management will be described. A prototype to record changes of SQL-functions over a period of time is proposed. The thesis will be summarized and provides an outlook of possible future topics and further papers.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

SQL-Anweisungen und Anfragen

Begriffsdefinitionen & Abkürzungsverzeichnis

IOW Leibniz-Institut für Ostseeforschung Warnemünde

GODESS *Gotland Deep Environmental Sampling Station*

PIP *Profiling Instrumentation Platform*

CTD *Conductivity, Temperature, Depth*. Eine Sonde für Tiefseeuntersuchungen mit Sensoren für die benannten abgekürzten Einheiten.

DBMS Datenbankmanagementsystem

UDF *User Defined Function*. Eine im DBMS abgespeicherte benutzerdefinierte Funktion in SQL-Syntax.

STP *Stored Procedure*. Eine im DBMS abgespeicherte benutzerdefinierte Prozedur in SQL-Syntax. Unterschiede zur UDF liegen in der zusätzlichen Unterstützung von Ausgabeparametern sowie der Aufruf über explizite SQL-Schlüsselwörter. Die Nutzung innerhalb von SQL-Anfragen ist gegenüber von UDFs hier nicht möglich. [SSH13]

Constraint Einschränkung, Restriktion, Integritätsbedingung, deren Einhaltung vom DBMS garantiert wird.

DDL *Data Definition Language*. Sprache um Datenbankelemente, wie Tabellen oder Indexe anzulegen, zu verändern, zu löschen. Beinhaltet Befehlsklauseln wie CREATE, DROP, ALTER.

DML *Data Manipulation Language*. Sprache um Daten zu handhaben. Beinhaltet Befehlsklauseln wie INSERT, UPDATE, MERGE, DELETE, TRUNCATE.

Audit Verfahren zur Untersuchung von Prozessen/Operationen/Prozeduren, Anforderungen, Richtlinien oder Ereignissen. Durch chronologische Aufzeichnung aller Aktivitäten entsteht ein Audit-Log.

Wrapper-Funktion Eine Funktion, die eine andere Funktion umschließt und diese aufruft. Dient zur Adaption von unterschiedlichen Schnittstellen und zur Delegation von Funktionsaufrufen.

XML Extensible Markup Language

1 Einführung

Dieses Kapitel soll an das Thema dieser Arbeit heranführen. Es wird knapp beschrieben, warum es notwendig ist, die unterschiedlichen Themengebiete zu untersuchen. Als Kontext dient in dieser Arbeit das Leibniz-Institut für Ostseeforschung Warnemünde (IOW) und der dort im Betrieb befindlichen *Gotland Deep Environmental Sampling Station* (GODESS). Es wird beschrieben, worum es sich bei dieser Plattform handelt sowie anhand eines Ausschnitts der von ihr gelieferten Daten ein Beispiel konstruiert, auf das in der Arbeit zurückgegriffen wird. Abschließend folgt der Aufbau der Arbeit.

1.1 Motivation

Die stetig wachsenden Datenmengen im Bereich der Forschung und Wissenschaft erfordern neue Ansätze der Speicherung und Verwaltung von Forschungsdaten. Sensoren und Messinstrumente tragen dazu bei, dass täglich neue Mengen an Daten anfallen. Ob es sich dabei um Maschinen in der Größenordnung des LHC[1] und seinen Detektoren handelt oder um kleinere Sensoren, die in ihrer Umgebung verschiedene physikalische und chemische Werte messen oder gar um textuelle Berichte, Befragungen und Auswertungen dieser. Das Grundproblem bleibt: die Entwicklung technischer Möglichkeiten die unterschiedlichen Formate langfristig zu speichern sowie die Bereitstellung der (Forschungsroh-)Daten für die Nachnutzung mittels Auswertungs- und Analysemethoden. Diese Themen und welche Herausforderungen mit ihnen einhergehen, werden im Abschnitt 3.1 über das Forschungsdatenmanagement näher beschrieben.

Darüber hinaus muss die Nachvollziehbarkeit von wissenschaftlichen Ergebnissen sichergestellt werden. Dazu werden die Herkunft und der Verarbeitungsweg der Forschungsdaten von der Aufnahme dieser bis zum finalen Ergebnis aufgezeichnet. Neben den eigentlichen Daten sollen zusätzlich die Funktionen, die zur Auswertung und Analyse genutzt wurden, mit erfasst werden um die Rekonstruierbarkeit des Ergebnisses zur gewährleisten. Was hierbei die Herausforderungen sind und was für Techniken existieren, wird in Abschnitt 3.2 beschrieben.

Ein Beispiel für das Thema Forschungsdatenmanagement bietet das IOW. Dieses beschäftigt sich mit der interdisziplinären Meeresforschung in Küsten- und Randmeeren, wobei der Schwerpunkt

[1] Large Hadron Collider (Teilchenbeschleuniger) - https://home.cern/topics/large-hadron-collider

in der Erforschung des Ökosystems der Ostsee liegt [Lei16]. In der Sektion der Meereschemie ist die Arbeitsgruppe „Chemische in situ Sensoren“ angesiedelt, welche die profilierende Verankerung *Gotland Deep Environmental Sampling Station* (GODESS) betreibt. Messinstrumente bzw. Sensoren auf dieser profilierenden Plattform erfassen verschiedene physikalische und chemische Werte. Bei der Verwaltung und Speicherung dieser (Forschungsroh-)Daten ist das IOW an performanten, sicheren und rückverfolgbaren Verfahren interessiert.

1.2 GODESS

Der folgende Abschnitt beschreibt die GODESS Plattform und die von ihr erzeugten Daten. Dabei werden der Aufbau der Plattform und die verwendeten Technologien dargestellt. Synonym zu GODESS wird in dieser Arbeit je nach Kontext von der (Mess-)Station, der (profilierenden) Verankerung oder der (Mess-)Plattform gesprochen. Des Weiteren wird auf die aktuelle Vorgehensweise bei der Verarbeitung und Analyse der Sensordaten eingegangen sowie aus der Struktur der vorliegenden Daten ein Beispiel konstruiert, welches sich fortlaufend durch diese Arbeit ziehen wird. Die folgenden Informationen und Daten in diesem Abschnitt stammen, sofern nicht anders gekennzeichnet, aus [PSB16].

1.2.1 Aufbau, Technik und Funktionsweise

Die gesamte Messstation ist aus mehreren Komponenten aufgebaut. Diese werden im Folgenden von unten (Meeresgrund) nach oben aufgezählt und beschrieben. Der Aufbau der Station ist in Abbildung 1.1 ersichtlich.

Unten auf dem Meeresboden befindet sich das Grundgewicht, welches als Anker fungiert und dafür sorgt, dass die Station an Ort und Stelle bleibt. Zusätzlich besitzt dieses eine Grundleine zur Notfall-Bergung, falls der höher gelegene Auslösemechanismus für die Bergung der Station nicht ordnungsgemäß funktioniert. Etwa 35 m höher befindet sich dieser akustische Auslösemechanismus mit einer 400 m langen Bergungsleine auf einer Rolle. Weitere 10 m höher befindet sich die Unterwasserwinde mit Auftriebskörpern. Diese besitzt vier Batterien als Energiequelle und sorgt dafür, dass die eigentliche (profilierende instrumentierte) Plattform (*Profiling Instrumentation Platform* (PIP)) in der Wassersäule hoch- und heruntergefahren werden kann. Die PIP sitzt als letzte Komponente ganz oben und verfügt über Instrumente mit physikalischen und chemischen Sensoren und eigene Auftriebskörper. Sie operiert in einer Tiefe von ca. 180 m bis hoch zu 30 m unter der Meeresoberfläche, das Grundgewicht selbst ist etwas tiefer bei ca. 220 m gelegen.

Das Aussetzen der Station vom Schiff aus sowie die Zeit, die die Station im Einsatz ist, wird *Deployment* genannt und erhält von den Forschern des IOW immer eine eindeutige fortlaufende

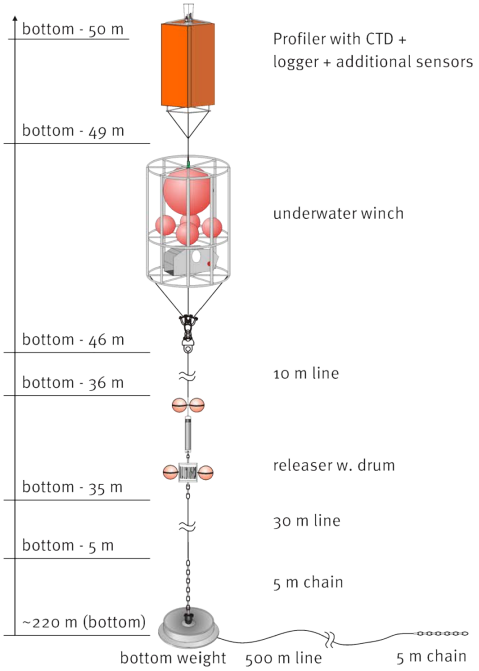


Abbildung 1.1: GODESS-Aufbau nach [PSB11]

Nummer. Der Zeitraum von einem Deployment reicht dabei von mehreren Wochen bis zu mehreren Monaten, je nach Konfiguration. Während eines Deployment werden mehrere sogenannte Profile gefahren. Die Anzahl hängt wieder von der Konfiguration und der Programmierung ab. Ein Profil besteht aus dem Aufstieg der PIP bis unter die Meeresoberfläche und den Abstieg zurück zur Unterwasserwinde. Während des Aufstiegs nehmen die angebrachten Sensoren auf der PIP die verschiedenen Messdaten auf. Darunter sind u. a. elektrische Leitfähigkeit (Conductivity), Temperatur, Druck (daraus ermittelte Tiefe, Depth), Trübung, Sauerstoffgehalt, Strömung oder der ph-Wert. Aufgrund der Anfangsbuchstaben der englischen Bezeichnungen für die ersten drei genannten physikalischen Größen wird so ein Instrument auch kurz CTD genannt. Profile werden zu fest einprogrammierten Zeiten gefahren. Ist der einprogrammierte Zeitpunkt erreicht, löst die Unterwasserwinde die Sperre, damit sich die Leine abwickeln kann. Durch die Auftriebskörper der PIP wird die Leine kontrolliert abgespult. Das Aufnehmen der Sensordaten durch die PIP muss mit dem Entsperren der Unterwasserwinde synchronisiert werden. Da es keine Kommunikationsverbindung von Unterwasserwinde zur PIP gibt, werden dafür die internen Echtzeituhren der beiden Geräte verwendet. Aufgrund der niedrigeren Präzision der Uhren kommt es zu einer sogenannten Drift, so dass die PIP zwei Minuten bevor die Unterwasserwinde auslöst mit der Aufzeichnung der Sensorwerte beginnt.

Durch den modularen Aufbau der Plattform kann die PIP in Zukunft mit zusätzlichen oder neueren Sensoren bestückt werden. Dies ist ein wichtiger Fakt, der bei dem Format der Sensordaten, der Auswertung sowie der Problemstellung in Kapitel 2 eine Rolle spielt.

1.2.2 Sensordaten und Datenformate

Während einer Profilmessung werden die Sensordaten zur Aufzeichnung an einen zentralen Logger geschickt, welcher sich auf der PIP befindet. Dieser schreibt kontinuierlich die Daten in mehrere Textdateien, die abhängig vom Instrument und deren Sensoren unterschiedliche Formate besitzen. Dabei erstellt der Logger jeweils Dateien für die CTD (A.2), den Strömungsmesser (A.3), sowie den Spektraldatensensor (A.1). Darüber hinaus werden einige Daten des Strömungsmessers und des Spektraldatensensors auch in die CTD-Datei A.2 geschrieben. In dieser Arbeit wird nur auf die Datei für die CTD, welche u. a. die Werte wie Leitfähigkeit, Temperatur oder Druck enthält, eingegangen. Diese Dateien werden pro Tag generiert und können Daten mehrerer Profile enthalten. Des Weiteren existieren Metadaten, welche einerseits die Datei selbst beschreiben, z. B. Dateiname und Datum, andererseits das Format der Nutzdaten charakterisieren.

Alle Dateien der unterschiedlichen Sensoren liegen als ASCII-Textdateien in einem proprietären Format vor. Die Metadaten der CTD-Dateien beginnen dabei mit dem @-Symbol, Nutzdaten starten mit einem vordefinierten String und sind mittels Trennzeichen, konkret durch Kommata, separiert. Dateien des Strömungsmessers besitzen keine Metadaten und bestehen hauptsächlich aus einem Datum und Uhrzeit gefolgt von Rohdaten im hexadezimalen Format. Die Dateien des Spektralsensors sind in Abschnitte unterteilt, welche durch Bezeichner in eckigen Klammern getrennt sind. Diese enthalten sowohl Meta- als auch Nutzdaten.

Des Weiteren wird pro Deployment eine XML-Datei (A.4) erzeugt, welche Metadaten über das Deployment selbst sowie Kalibrierungswerte bzw. Polynom-Koeffizienten für die einzelnen Sensoren beinhaltet. Die Sensoren selbst liefern Rohwerte, welche mittels eines n-stelligen Polynoms in eine SI-Einheit umgerechnet werden. Die Koeffizienten für diese Polynome stammen dabei aus dieser XML-Datei.

1.2.3 Verarbeitung, Analyse und Ergebnisse der Daten

Die Filterung, Umrechnung und Auswertung der Sensordaten erfolgt mit von den beteiligten Wissenschaftlern erstellten Matlab-Skripten. Dazu werden die Rohdateien und die XML-Konfiguration eingelesen, gegebenenfalls zusammengeführt, die Sensorwerte der Abstiege der Profile herausgefiltert sowie mittels der Koeffizienten aus der XML-Datei in SI-Einheiten umgerechnet. Zum Schluss erfolgen verschiedene Vergleiche sowie das Erzeugen von verschiedenen Diagrammen, welche Messkurven der chemischen oder physikalischen Werte über die Zeit abbilden. Ein Diagramm, welches die Temperaturen aller Profile, also eines gesamten Deployments, über die Zeit und Tiefe darstellt, ist in Abbildung 1.2 zu sehen.

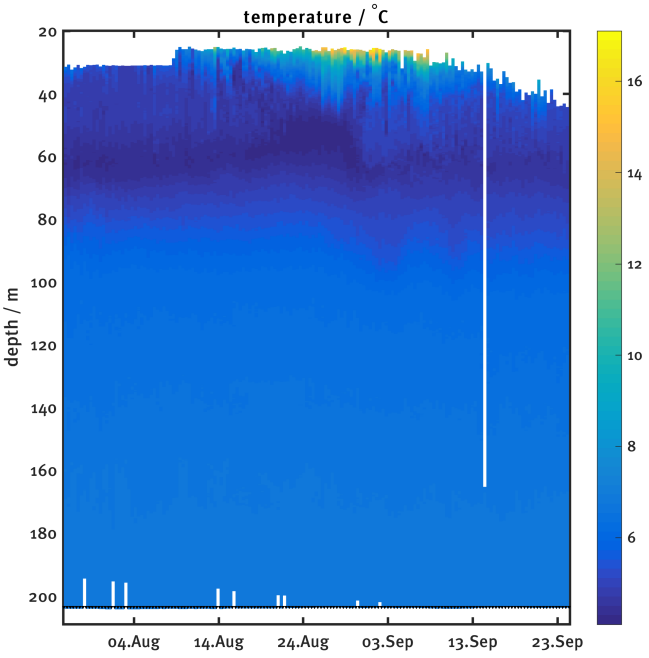


Abbildung 1.2: Darstellung der Temperatur über die Tiefe und die Zeit während Deployment 15 aus dem Jahr 2015

Zu sehen ist auch ein weißer Streifen. Gemäß Provenance-Fragestellungen kann nun die Ursache bzw. die Herkunft dafür erfragt werden. Was Provenance ist und welche Typen und Arten von Anfragen und Antworten es gibt, wird in Abschnitt 3.2 beschrieben.

1.2.4 Fortlaufendes Beispiel

Anhand der Struktur der CTD-Sensordaten und angelehnt an das Datenbankschema aus [Mö16] wird ein Beispiel konstruiert, welches fortlaufend in der Arbeit Verwendung findet. Es soll aus zwei Tabellen bestehen: die *deployment*-Tabelle 1.1, welche einige (Meta-)Daten für Deployments enthält sowie die *payload*-Tabelle 1.2, welche die eigentlichen Nutzdaten, also die CTD-Sensordaten, beinhaltet. Um das Beispiel nicht zu überladen und übersichtlich zu halten, wurden nur die Sensoren für die elektrische Leitfähigkeit, Temperatur und Druck mit aufgenommen. Des Weiteren wurden reale Sensordaten aus den Textdateien von Deployment 14 respektive 15 importiert. Zusätzlich sind die Werte der Übersicht halber mittels der hinterlegten Koeffizienten aus der XML-Konfigurationsdatei in SI-Einheiten umgewandelt worden. Die SQL-Anweisungen zur Erzeugung dieser Tabellen sind im Anhang A.1.2 zu finden. Ferner sind dort auch temporale Attribute mit enthalten, welche hier nicht mit abgebildet sind. Zusätzlich sind weitere Metadaten in Form von Kommentaren an den Attributen hinterlegt.

deployment_id	name	deployed_ts	recoverd_ts	lon	lat
14	D14	2015-04-21 17:00:00	2015-07-18 09:00:00	57.32277	20.13333
15	D15	2015-07-26 16:00:00	2015-09-24 11:00:00	57.32083	20.13667

Tabelle 1.1: Tabelle *deployment* enthält GODESS-Deployment (Meta-)Daten

id	deployment_id	profile_ts	pressure	temperature	conductivity
1	14	2015-06-02 18:01:46	201.459	6.866	14.456
2	14	2015-06-02 18:01:47	201.116	6.864	14.454
3	14	2015-06-02 18:01:48	201.065	6.865	14.454
		...			
2942	15	2015-09-13 02:57:33	202.939	6.869	14.547
2943	15	2015-09-13 02:57:34	202.952	6.867	14.548
2944	15	2015-09-13 02:57:35	202.952	6.867	14.547
		...			
14685	15	2015-09-14 05:57:35	202.817	6.847	14.466
14686	15	2015-09-14 05:57:36	202.813	6.848	14.467
14687	15	2015-09-14 05:57:37	202.808	6.848	14.467
		...			
23057	15	2015-09-15 00:00:02	202.719	6.854	14.471
23058	15	2015-09-15 00:00:03	202.724	6.853	14.47
23059	15	2015-09-15 00:00:03	202.724	6.853	14.47
		...			

Tabelle 1.2: Tabelle *payload* enthält GODESS-Nutzdaten

1.3 Aufbau der Arbeit

Kapitel 1 dient zur Einführung und Motivation. Es wird erklärt, warum die Aspekte der temporalen Datenhaltung, der Provenance sowie des Forschungsdatenmanagements benötigt werden. Dazu wird der Kontext dieser Arbeit, nämlich das Leibniz-Instituts für Ostseeforschung Warnemünde, konkret die dort eingesetzte GODESS-Plattform, beschrieben. In Kapitel 2 wird zur Problemstellung hingeführt und beschrieben, welche Untersuchungen Gegenstand der vorliegenden Arbeit sind. Das Kapitel 3 beschreibt die grundlegenden Begriffe des Forschungsdatenmanagements, der Provenance sowie der temporalen Datenhaltung. Es wird erklärt, worum es sich jeweils handelt und wie der Stand der Forschung ist. Der Bereich, die Aufgaben und langfristigen Ziele des Forschungsdatenmanagements werden in Abschnitt 3.1 beschrieben. Im Provenance Abschnitt 3.2 wird auf die unterschiedlichen Provenance-Anfragen sowie -Antworten anhand eines Beispiel eingegangen. Im Abschnitt 3.3 der temporalen Datenhaltung wird aufgezeigt, welche drei Methoden der Speicherung zeitbezogener Daten existieren. In Kapitel 4 folgt das Konzept. Dort wird in Abschnitt 4.1 auf eine Methode der Versionierung von SQL-Funktionen eingegangen. Ferner werden zwei Varianten der Schemaversionierung skizziert sowie festgelegt, welche Variante in der vorliegenden Arbeit angenommen wird. Es folgt in Abschnitt 4.2 die Untersuchungen von sich verändernden Funktionen auf sich verändernden Schemata. Im Schlussteil des Konzepts in Abschnitt 4.3 wird auf den Zusammenhang zwischen Provenance und temporalen Daten eingegangen und beschrieben, welche Herausforderungen dort existieren. In Kapitel 5 erfolgt eine prototypische Umsetzung des Versionierungskonzepts für SQL-Funktionen unter Einsatz von PostgreSQL. Danach folgt in Kapitel 6 die Zusammenfassung sowie ein Ausblick, in dem beschrieben wird, welche weiteren Möglichkeiten und Ansätze in den in dieser Arbeit behandelten Themenbereichen existieren. Weiterhin werden Fragestellungen zur weiteren Bearbeitung in zukünftigen Arbeiten aufgeworfen.

Auf der beiliegenden CD sind alle Bilder, Grafiken sowie die zitierte Literatur in Form von PDF-Dateien enthalten. Weiterhin sind alle SQL-Funktionen und Tabellen, während der Erstellung des Prototypen erzeugt wurden, vorhanden.

2 Problemstellung

In diesem Kapitel wird beleuchtet, welche Folgen und damit auftretenden Probleme die in Abschnitt 1.2.1 erwähnten Aspekte haben. Es werden die aktuelle Vorgehensweise und der aktuelle Stand beschrieben. Dabei werden zwei wesentliche Fragestellungen aufgeworfen, die in dieser Arbeit untersucht werden.

2.1 GODESS Konfigurationsänderungen und aktuelle Vorgehensweise

Die in Abschnitt 1.2.1 erwähnten Konfigurationsänderungen der GODESS-Plattform, z. B. durch das Hinzufügen neuerer höher auflösender Sensoren oder das Entfernen von Sensoren, verursachen eine Strukturänderung der Datenformate. Diese Änderungen spiegeln sich in den Log-Dateien, welche die Sensordaten beinhalten, derart wider, als dass neue Spalten, Metadaten sowie (teils genauere) Werte für die betreffenden Sensoren hinzukommen. Darüber hinaus führt dies zur Anpassung der zugehörigen Analyse- und Auswertefunktionen in den Matlab-Skripten, sodass diese auf den neuen Datenformaten arbeiten können.

Zurzeit existiert kein Prozess, der automatisch diese Änderungen der Zusammenstellung der Instrumente und Sensoren der PIP und damit die Strukturänderungen der Log-Dateien aufzeichnet bzw. protokolliert. Die aktuelle Vorgehensweise in diesem Fall ist, dass die Matlab-Skripte von Deployment zu Deployment kopiert und dann jeweils per Hand angepasst werden. Somit können nur die jeweiligen Sensordaten mit den dazugehörigen passenden Skripten ausgewertet werden. Des Weiteren findet eine Kalibrierung der Sensoren nach jedem Deployment statt. Die daraus erzeugten Kalibrierungswerte werden in einer Excel-Datei erfasst und in eine XML-Datei überführt. Somit bilden diese Kalibrierungswerte die (neuen) Koeffizienten für die Umrechnung der Sensorrohwerte in SI-Einheiten mittels des Polynoms n-ten Grades für den jeweiligen Sensor.

Ferner liegen alle zugehörigen Dateien in lokalen Ordnerstrukturen auf den Computern der beteiligten Forscher. Dies schränkt die Weitergabe und Nachnutzung der (Forschungs-)Rohdaten unter dem Gesichtspunkt des Forschungsdatenmanagement aus Abschnitt 3.1 ein. Des Weiteren gibt es keine Zugriffskontrolle auf die Daten und die Matlab-Skripte unterliegen keinerlei Versionskontrolle. Aufgrund dieser dezentralen Speicherung und Verwaltung soll in Zukunft die Migration in eine Datenbank erfolgen.

In [Mö16] wurde bereits untersucht, wie die Beziehungen zwischen Deployment, der verwendeten Sensoren sowie der Sensordaten in ein relationales Datenbankschema abgebildet werden können. Darüber hinaus müssen die Analyse- und Auswertefunktionen aus den Matlab-Skripten in benutzerdefinierte SQL-Funktionen (*User Defined Function* (UDF)) bzw. in SQL abgespeicherte Prozeduren (*Stored Procedure* (STP)) umgeschrieben werden. Das dies nur unter Einschränkungen automatisiert möglich ist, wurde in [Weu16] für die Programmiersprache *R* gezeigt. Die Untersuchung wurde nur auf einige ausgewählte Operatoren und Funktionen beschränkt, da nicht jede *R*-Funktion auf SQL abgebildet werden kann. Weiterhin ist es schwierig, semantische Zusammenhänge über mehrere Ausdrücke hinweg zu erkennen, obwohl dies möglicherweise zu stark vereinfachten SQL-Anweisungen hätte führen können [Weu16]. Auf die Problematik der automatisierten Übersetzung bestehender Auswerte- und Analysemethoden in SQL-Funktionen und -Anweisungen wird daher in dieser Arbeit nicht weiter eingegangen. Die Versionierung dieser SQL-Funktionen wurde dagegen bereits in [Mö16] für das DBMS MySQL gezeigt. In dieser Arbeit wird sich jedoch an dem DBMS PostgreSQL orientiert, da es MySQL an temporalen Eigenschaften fehlt. (siehe dazu Abschnitt 3.3 über temporale Datenhaltung & aktueller Stand 3.3.4) Die Erkenntnisse und Ergebnisse aus [Mö16] wurden zum Stand dieser Arbeit noch nicht praktisch umgesetzt.

2.2 Untersuchungsgegenstand

Unter Betrachtung der Strukturänderungen der oben angesprochenen Dateiformate im Datenbankkontext entspricht dies Änderungen am relationalen Datenbankschema. Wurde das Schema erstmalig bezüglich der zu dem damaligen Stand gültigen Dateiformaten entworfen, so muss dieses aufgrund der (Konfigurations-)Änderungen gleichermaßen angepasst und weiterentwickelt werden. Im Datenbankkontext wird die Änderung der Struktur der Daten - das Schema - als Schemaevolution bezeichnet. Auf das Thema der Schemaevolution wird in dieser Arbeit explizit nicht weiter eingegangen. Zusätzlich zu den Schemaänderungen müssen auch die Auswerte- und Analysemethoden angepasst werden, damit diese auf dem neuen Schema operieren können. Somit ergeben sich zwei wesentliche Untersuchungsgegenstände dieser Arbeit:

1. Erfassen der Änderungen der Auswerte- und Analysemethoden über die Zeit, sowie die Fragestellung, unter welchen Bedingungen eine veraltete Funktion auf ein neues Schema bzw. eine neue Funktion auf ein altes Schema angewendet werden können. Dabei soll betrachtet werden, welche Schema- und Funktionsänderungen unter Berücksichtigung des praktischen Aspekts bezüglich des IOW möglich sind und welche Kriterien dabei eine Rolle spielen.

2. Untersuchung der Zusammenhänge zwischen Provenance und temporalen Aspekten

Abbildung 2.1 verdeutlicht den Sachverhalt aus Punkt 1. Hierzu werden die Auswerte- und Analysefunktionen vereinfachend als Anfragen „*Queries*“ (Q) behandelt. Die durchgezogenen Linien von Q auf ein Schema S entsprechen der Anwendung der Anfrage Q auf das aktuelle Schema S. Daten werden durch das Rechteck mit den Buchstaben *d* symbolisiert. Dabei wird zwischen den vorherigen Daten aus dem vorherigen Schema und den Daten im aktuellen Schema mittels Trennzeichen unterschieden. Dies entspricht der temporalen Datenhaltung, die in Abschnitt 3.3 behandelt wird. Die Schemaänderungen (S, S', S") sind durch die dicker dargestellte durchgezogene Linien veranschaulicht. Die gestrichelten und gepunkteten Linien beschreiben die unter Punkt 1 genannten Fragestellung nach der Anwendung von geänderten Anfragen, z. B. Q', auf veränderte Schemata, hier S und S". Die grauen Pfeile zwischen den Anfragen (Q, Q', Q") symbolisieren den Verlauf der Versionierung der Anfragen bzw. Funktionen.

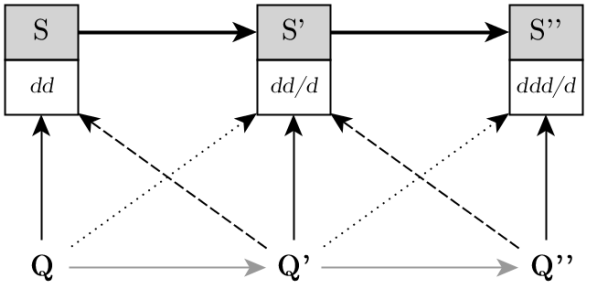


Abbildung 2.1: Schemaänderungen (S), temporale Daten (*d*) und Anfragen (Q)

Die Versionierung von Funktionen (Q) muss nicht zwangsläufig stattfinden, da zu jedem Zeitpunkt neue Funktionen passend zum jeweiligen, zu dem Zeitpunkt gültigen, Schema erstellt werden können. Das bedeutet konkret, dass z. B. die Anfrage Q' nicht durch Versionierung von Anfrage Q entstanden sein muss. Dieser und weitere Fälle werden im Konzept in Abschnitt 4.2 betrachtet. Das Erfassen der Schemaänderungen, also die Versionierung von Schemata, ist nicht Bestandteil dieser Arbeit, wird aber in Abschnitt 4.1.2 kurz skizziert. Ferner stellt die Abbildung 2.1 nur einen Ausschnitt dar. Es können weitere Versionen von Schemata und Funktionen links von S und rechts von S" existieren. Weiterhin ist es möglich, dass Funktionen auch Versionsübergreifend auf Schemata angewendet werden können, z. B. von Q" auf S. Diese Verbindungen sind in der Abbildung der besseren Darstellung halber nicht eingezeichnet, sind aber erlaubt.

Punkt 2 aus obiger Liste beleuchtet, ob temporale Techniken (siehe dazu Abschnitt 3.3) die Provenance von Daten (siehe dazu Abschnitt 3.2) unterstützen können und was dabei die Herausforderungen sind.

3 Grundlagen

Dieses Kapitel erklärt die Begriffe des Forschungsdatenmanagements, der Provenance sowie die temporale Datenhaltung.

3.1 Forschungsdatenmanagement

Das Forschungsdatenmanagement beschäftigt sich mit der Speicherung, Verwaltung und Bereitstellung von Forschungsdaten, um den wissenschaftlichen Nutzen dieser zu erhalten, langfristig verfügbar und nachnutzbar zu machen. Der Begriff der Forschungsdaten sowie deren Verwaltung sind stark von der jeweiligen wissenschaftlichen Disziplin geprägt und daher immer in Bezug zur jeweiligen Fachdisziplin zu setzen [NSO+12]. Die Daten selbst stammen aus unterschiedlichen Quellen und werden mit verschiedenen Instrumenten erfasst, erhoben oder generiert. Forschungsdaten sind nach [BHM11, NSO+12] z. B. Messdaten aus Instrumenten wie einem Teleskop, Rohdaten aus einem Massenspektrometer, empirische Daten, Quelldaten oder Forschungsrohdaten, digitale Karten und Volltexte. Weiter gehören die Erhebung von Proben, die Befragungen von Personen, die Beobachtung von sozialen Situationen oder die Analyse von kulturellen Artefakten dazu. Aufgrund der stetig wachsenden Menge an Daten, die durch neue Verfahren und Methoden gewonnen werden, ist es erforderlich, Informationssysteme zu entwickeln, die diese Daten langfristig speichern und für die weitere (Nach-)Nutzung zugänglich machen [BHM11].

Zu den Aufgaben des Forschungsdatenmanagement gehören nach [BHM11]:

- Das Bereitstellen eines transparenten und nachvollziehbaren Prozesses, welcher die Transformationen, Speicherung und Übermittlung dieser Daten beschreibt.
- Das nachvollziehbare Dokumentieren dieses (Verarbeitungs-)Prozesses.
- Das Ermöglichen von Datenzugriff und -auswertung unabhängig vom Erzeuger.
- Die integrierte Verarbeitung und Darstellung heterogener Daten.
- Die Sicherstellung der Zuverlässigkeit der Daten.
- Die Erhaltung der wissenschaftlichen Aussagekraft von Forschungsdaten.
- Die Sicherung der Nachweiskette der verarbeiteten Daten.

Anhand dieser Liste ist zu erkennen, dass das Forschungsdatenmanagement ein komplexer dyna-

mischer Prozess ist, der je nach betrachteter wissenschaftlicher Disziplin völlig verschieden aussehen kann. So wird z. B. in der Biologie gegenwärtig viel mit Excel-Dateien gearbeitet, wohingegen in mathematisch-physikalisch geprägten Disziplinen Rohdaten in Form von vom Hersteller des Messinstruments vorgegebenem Dateiformat erzeugt werden. Die Speicherung selbst erfolgt häufig lokal auf den Rechnern der beteiligten Wissenschaftler anstatt auf den durch das Institut bereitgestellten Netzlaufwerken. Die Verarbeitung erfolgt dann mittels Dritthersteller-Software, z. B. Matlab, oder durch von beteiligten Wissenschaftlern entwickelten Programmen.

Ein wichtiger Punkt ist die Verwaltung der heterogenen Daten und deren Formate. Oft liegen die Daten in den einzelnen Instituten und Einrichtungen, seien es Universitäten oder Forschungszentren, unstrukturiert und in teils proprietären Dateiformaten vor, welche es erschweren, die Daten im Sinne der Nachvollziehbarkeit und der langfristigen Speicherung zu handhaben. So entstehen immer wieder Insellösungen, die allein für sich genommen, den Alltag des Wissenschaftlers in seiner Umgebung erleichtern, es aber erschweren, dass Austausch und Weitergabe von Forschungsdaten abseits von Publikationen, global stattfinden kann. Zusätzlich ist es notwendig, Richtlinien zu entwickeln, um den Ablauf zu beschreiben, auf welche Art und Weise und an welchem Zeitpunkt (Roh-)Daten veröffentlicht bzw. weitergegeben werden können. Der Datenaustausch kann so mit Policies, Datenschutz Aspekten und Ebenen des Zugriffsschutzes geregelt werden [BHM11].

In [NSO+12] wurden elf Wissenschaftsdisziplinen und ihren Umgang mit Forschungsdaten analysiert und verglichen. Als Ergebnis werden dort Erkenntnisse und Thesen formuliert, welche die Bedeutung der Langzeitarchivierung aus wissenschaftlicher Sicht hervorheben sowie auf konzeptionelle und operative Sachverhalte verweisen. Im folgenden Absatz erfolgt eine aus [NSO+12] (Kapitel 16) unvollständige und verkürzte Zusammenfassung, um den gegenwärtigen Stand, die Problematik und Bedeutung des Forschungsdatenmanagement zusammenzufassen.

Die Wichtigkeit von Forschungsdaten und die langfristige Archivierung und Bereitstellung wird von allen Wissenschaftsdisziplinen betont. Dabei sind die verschiedenen Ansätze in den unterschiedlichen Gebieten nicht Ausdruck eines mangelnden Kooperationswillens über die Disziplingrenzen hinweg, sondern ergeben sich aus der logischen Konsequenz der unterschiedlichen Anforderungen und ausgeübten Verfahren innerhalb der jeweiligen Disziplin. Darüber hinaus werden (Forschungs-)Datenzentren als ideale Lösung angesehen, die Verfügbarkeit und damit effiziente Nachnutzung erheblich zu verbessern und langfristig zu sichern. Eine offene Fragestellung hierbei ist die Vertrauenswürdigkeit und die Überprüfung dieser. Wie bereits angeführt, wird auch hier bestätigt, dass jede Wissenschaftsdisziplin eigene (Meta-)Datenformate sowie zahlreiche fachspezifische und proprietäre Formate benutzt. Dabei ist die Vielzahl an Datenformaten fast unüberschaubar. Um dieser Vielfalt zu begegnen, greifen einige Disziplinen auf Richtlinien zurück, welche das Format der Daten bestimmen. Trotzdem ist es nicht möglich, alle Formate per se festzulegen, da aufgrund der unterschiedlichen genutzten Geräte und Software die Datenformate vom jeweiligen Hersteller vorgegeben werden. Diskussionspunkt ist hier die Stan-

dardisierung unter Berücksichtigung von etablierten Formaten aus der kommerziellen Industrie. Die Einschränkung der Datenformate unterstützt die technische Datensicherung und erleichtert die Nachnutzung der Daten. Dabei kann die Integrität der Daten unabhängig von Datei- und Metaformaten gesichert werden. Dies führt aber dazu, dass die inhaltliche Nachnutzung der Forschungsdaten nicht gewährleistet werden kann. Die Nachnutzung ist wichtig für die Kooperation innerhalb von Forschungsprojekten, für externe Wissenschaftler oder bei Publikationen sowie für die breite (Fach-)Öffentlichkeit. Vor der Bereitstellung für die Nachnutzung der Daten müssen aber noch einige Punkte, wie Rechteverhältnisse und datenschutzrechtliche Aspekte, geklärt werden.

Ein Hauptbeweggrund für die Langzeitarchivierung von Forschungsdaten ist die langfristige Zitierbarkeit und Referenzierbarkeit. Darüber hinaus spielen auch wissenschaftliche Standpunkte eine Rolle. Die Kosten für das Forschungsdatenmanagement sind im Vergleich zur Datenerhebungen in Großforschungsprojekten, z. B. Polarexpeditionen oder Bohrungen, gering [BHM11]. Trotzdem muss differenziert werden: sind die Kosten bei den genannten Beispielen punktuell hoch, so sind im Vergleich zum Forschungsdatenmanagement die Kosten für die Verwaltung und Archivierung der Forschungsdaten über einen längeren Zeitraum zu betrachten und können somit ähnliche Ausmaße annehmen. Nach wie vor besteht ein dringender Handlungsbedarf, die Kosten und Kostenfaktoren für die einzelnen Dienstleistungen der Langzeitarchivierung zu klären. Dazu muss diese Kostenabschätzung, Personal und Material, Teil der Aufwandsplanung eines Forschungsprojekts werden. Weiterhin muss qualifiziertes Personal, welches im Bereich der Langzeitarchivierung sowohl im theoretischen als auch im konzeptionellen Bereich geschult ist, vorhanden sein. Vorgeschlagen wird, dass das Thema „Langzeitarchivierung digitaler Forschungsdaten“ in Studiengängen oder Studienschwerpunkten als methodische Basisqualifizierung mit einfließt. So entstehen völlig neue Arbeitsfelder wie „Data Librarian“ oder „Data Curator“. Abschließend wird auf die gesellschaftliche Bedeutung des Forschungsdatenmanagements eingegangen. Denn gesellschaftspolitische Entscheidungen werden in steigendem Maße von wissenschaftlichen Ergebnissen beeinflusst. Um diese Entscheidung zu einem späteren Zeitpunkt überprüfen zu können, werden eben jene Forschungsdaten benötigt, die zu den Entschlüssen geführt haben. Des Weiteren können mittels archivierter und langfristig verfügbarer Forschungsdaten Verstöße gegen die „gute wissenschaftliche Praxis“ in Publikationen aufgeklärt werden.

Dieser Abschnitt sollte einen Einstieg und Überblick über das Forschungsdatenmanagement geben. Es zeigt sich, dass es noch viele offene Punkte über die Machbarkeit und die Umsetzung gibt. Neben all den technischen Entwicklungen muss ein Bewusstsein für die Forschungsdaten und deren Verwaltung geschaffen und verbreitet werden. Perspektivisch betrachtet führt das weg von der gängigen „small science“ über „data driven science“ hin zu „enhanced science“ (eScience), in der eine globale Zusammenarbeit in der Wissenschaft angestrebt wird, sowie auch die dafür benötigte Infrastruktur bereitsteht und dies ermöglicht. ([BHM11])

3.2 Provenance

Im folgenden Abschnitt wird der Begriff der Provenance geklärt. Ausgehend vom allgemeinen Begriff wird der Bezug zur Informatik, im Speziellen zum Datenbankkontext, aufgezeigt. Es wird auf das Provenance-Management sowie auf die Notwendigkeit und Bedeutung von Provenance eingegangen. Darüber hinaus werden die Typen von Provenance, die Anfragen sowie die Arten der Antworten beschrieben. Abschließend wird der aktuelle Stand skizziert.

3.2.1 Begriffsklärung

Das Wort „Provenance“ entstand aus dem lateinischen Verb „provenire“ und bedeutet „entstehen“ oder „herauskommen“. Es wurde im späten 18. Jahrhundert im Französischen als „von etwas abstammen“ benutzt [Dic16a]. Vermischt mit englischen Elementen [Dic16b] entstand das heutige Wort Provenance. Es beschreibt die Herkunft oder das erste Auftreten von (antiken) Artefakten und wurde meist im Zusammenhang von Kunstwerken benutzt [Sva16]. Heutzutage wird es in einer weiten Spanne unterschiedlichster Anwendungsgebiete, wie z. B. der Archäologie, in Archiven, bei (gedruckten) Büchern sowie in der Informatik, verwendet [Gla14, Wik16].

In letzterer Disziplin, der Informatik, bezeichnet Provenance die Informationen über den Entstehungsprozess und die Herkunft der Daten [Gla14]. Weiterhin wird damit der Prozess der Nachverfolgbarkeit und der Aufzeichnung der Ursprünge von Daten beschrieben [BKT00]. Als Anwendungsgebiete in der Informatik selbst, sind u. a. Datenbanken, Softwaretechnik oder verteilte Systeme zu nennen [Gla14]. Im Kontext von Datenbanken wird untersucht, durch welchen Prozess Daten in die eigentliche Datenbank gelangen [BKT00] und wie dieser Prozess beschrieben werden kann. Weiterer Untersuchungsgegenstand ist die Verfolgung der Verschiebe- und Kopier-Operationen von Daten zwischen verschiedenen Datenbanken [BKT00] sowie die Aufzeichnung der angewendeten Transformationen auf diesen [Gla14]. In dieser Arbeit wird sich auf die Data-Provenance bzw. Provenance-Management bezüglich Datenbanken und Datenbankmanagement-Systemen bezogen.

Provenance-Management selbst setzt sich aus der Verwaltung der Provenance-Daten, den Auswertungsprozessen sowie Anfragen und Ergebnisse auf Provenance-Daten zusammen. Ziel ist die Überprüfung der Nachvollziehbarkeit und Rekonstruierbarkeit mittels Anfragen auf den Provenance-Daten. Ergebnisse der Anfragen auf Provenance-Daten sind Auswertungen in Form von Tabellen, Artikeln oder Berichten [Heu16]. Dadurch wird sichergestellt, dass vom Ergebnis eines beliebigen Auswertungsprozesses von Forschungsdaten, die Analyse bis zu den Originaldaten zurückzuverfolgen ist und die gesamte Analyse mit demselben Ergebnis reproduziert werden kann [BKM+16]. Dazu existieren verschiedene Qualitätsstufen der Rückverfolgbarkeit [Heu16, BKM+16], beschrieben in der Reihenfolge von schwacher nach starker Qualität:

Plausibilität
Diese Qualitätsstufe bedeutet, dass das Ergebnis aus den vorliegenden Metadaten bzw. Primärforschungsdaten unter Berücksichtigung der Auswertungsprozesse sowie vorliegenden Sekundärforschungsdaten hätte entstehen können. Dabei können mehrere plausible Ergebnisse auftreten. Die Originaldaten liegen unvollständig oder gar nicht vor.

Nachvollziehbarkeit
Im Gegensatz zur Plausibilität müssen hier die Originaldaten vorhanden sein, um prüfen zu können, ob ein daraus ermitteltes Ergebnis aus diesen hätte entstehen können.

Rekonstruierbarkeit
Bei Vorhandensein der originalen Experimentaldaten und der darauf ausgeführten Auswertung als Ergebnis muss diese Auswertung jederzeit eineindeutig aus den Originaldaten rekonstruierbar sein, wenn die Auswertungsprozesse bekannt und eindeutig festgelegt sind.

Reproduzierbarkeit (Rekonstruierbarkeit ohne Medienbruch)
Diese Qualitätsstufe unterscheidet sich von der reinen Rekonstruierbarkeit dadurch, dass die Reproduzierbarkeit vollautomatisch und ohne Eingriff von außen im IT-System abläuft.

Die Notwendigkeit von Provenance folgt zwangsläufig aus dem zuvor erläuterten Forschungsdatenmanagement in Abschnitt 3.1. Dazu ergänzt [BT07]: Viele wissenschaftliche Daten sind ein Resultat von komplexen Analysen oder Simulationen. Daher ist es wichtig, den kompletten Lebensweg eines Datensatzes sowie die angewandten Berechnungen auf diesen festzuhalten. Dies stellt sowohl die Reproduzierbarkeit als auch das Katalogisieren des Resultats sicher. Parallel dazu wird doppelte Arbeit vermieden und es können die Ursprungsdaten aus den Ausgabe- bzw. Ergebnisdaten wiederhergestellt werden. Durch Provenance kann somit die Qualität von Daten sichergestellt werden, da ihr Entstehungsprozess und Verarbeitungsweg bekannt sind. Dennoch ist das Problem der Bewertung der Qualität von Daten sowie das Vertrauen in diese zu komplex, als das es allein durch Provenance gelöst werden kann [Gla14]. Trotzdem kann - allein durch das Vorhandensein von qualitativ hochwertigen und vertrauenswürdigen Daten - Provenance dazu genutzt werden, um ein Maß für die Integrität und die Qualität der Ergebnisse von Transformationen zu berechnen [Gla14].

Im Zusammenhang mit dem Forschungsdatenmanagement spielt Provenance eine technische Rolle bezüglich der Infrastruktur. Beherrscht die Infrastruktur, in dem konkreten Fall also das Datenbankmanagementsystem (DBMS), die Technik des Provenance-Managements, so wird die wissenschaftliche Aussagekraft der Daten vollkommen erhalten und die Nachweiskette wird automatisch im Hintergrund gesichert (vgl. Aufgaben des Forschungsdatenmanagements in Abschnitt 3.1). So ist es dann jederzeit möglich, Ergebnisse von wissenschaftlichen Arbeiten anhand der Originaldaten nachzuvollziehen, zu validieren und ggf. erneut auszuwerten [BT07].

3.2.2 Provenance-Typen, -Anfragen und -Antworten

Nach [Gla14] bezeichnet Provenance verschiedene Arten von Informationen. Unter Verallgemeinerung und Abstraktion der Anwendungen von Analysen und Modifikationen auf Daten ist zu erkennen, dass es ein immer wiederkehrendes Prinzip ist: Anwenden von Transformationen auf existierenden Daten und dadurch Erzeugung von neuen Datensätzen. Diese Abstraktion erlaubt es, verschiedene Typen von Provenance-Informationen zu klassifizieren und zu bestimmten Transformationen in Beziehung zu setzen. Die Einteilung der Provenance-Informationen ist nach [Gla14] folgende:

Daten

Welche Daten wurden von Transformationen benutzt, um neue Daten zu erhalten.

Transformationen

Welche Transformationen wurden angewendet, um diese neuen Daten zu generieren.

Beteiligte, Akteure und Umgebung

Welche weiteren Informationen waren an der Erzeugung der neuen Daten beteiligt. Hierzu gehören z. B. der Nutzer, der eine bestimmte Transformation durchgeführt hat oder die (physische) Umgebung - also Maschine oder Softwaresystem - in der die Transformation stattgefunden hat.

Provenance spielt, wie oben beschrieben, eine wichtige Rolle, um zu entscheiden, woher ein digitales Artefakt, z. B. ein Tupel einer Ergebnisrelation einer Datenbankanfrage, stammt. Des Weiteren soll sichtbar gemacht werden, durch welche Operationen die Tupel manipuliert wurden und warum diese im Ergebnis enthalten sind. Dies führt zu den *Where*-, *Why*- und *How*-Provenance-Anfragen. Diese Arten von Anfragen gehören nach [BT07] zu der feingranularen Provenance. Die *Where*- und *Why*-Anfragen wurden in [BKWC01] eingeführt. Die *How*-Anfrage wurde erstmals in [GKT07] beschrieben [Sva16]. Es folgt eine knappe Zusammenfassung aus [Sva16]. Wie die Provenance-Anfragen formalisiert und spezifiziert sind, kann detailliert in [Sva16] bzw. [BKWC01, GKT07] und [CCT09] nachgelesen werden. Die Semantiken der Anfragen beziehen sich dabei auf die in [Gla10] und [CCT09] definierten Modelle.

Bevor auf die Provenance-Anfragen eingegangen wird, müssen die unterschiedlichen Typen von Provenance-Antworten beleuchtet werden: *extensionale* und *intensionale* Antworten.

3.2.2.1 Extensionale Antwort

Die in einer Datenbank abgelegten Daten entsprechen extensionalen Informationen [Sva16]. Eine extensionale Antwort besteht folglich aus einer (Teil-)Menge dieser extensionalen Informationen. Die Repräsentation der extensionalen Antwort ist eine Menge an Tupeln [YP99], welche in

Form einer (Ergebnis-)Relation zurückgegeben werden [Mot94]. Diese Menge an Tupeln genügt den Bedingungen der gestellten Anfrage und kann sortiert, gruppiert [Mot94] und/oder durch Selektions-Prädikate eingeschränkt sein. Die extensionale Antwort in Form von Relationen ist die konventionelle Form von Antworten eines DBMS [Mot94] und ist den meisten Anwendern vertraut.

Als Beispiel soll der Ort des Deployments 15 gesucht werden. Mit Anweisung 3.1 auf Tabelle 1.1 wird dann die extensionale Antwort 3.1 erzeugt.

```
SELECT lon, lat
FROM deployment
WHERE deployment_id = 15
```

Anweisung 3.1: Ort von Deployment 15

lon	lat
57.32083	20.13667

Tabelle 3.1: extensionale Ergebnisrelation von Anweisung 3.1

3.2.2.2 Intensionale Antwort

Unter der isolierten Betrachtung der extensionalen Antwort ist zu erkennen, dass diese keinerlei Bedeutung (Intension) aufweist. Die Anfrage selbst gibt diesen Daten, z. B. durch die Selektions-Prädikate, erst einen Sinn. Somit ist die Anfrage bereits eine intensionale Information bzw. Anweisung [Mot94]. Sie beschreibt die Bedeutung der von ihr zurückgelieferten Daten. Eine intensionale Antwort kann als eine Art der Interpretation oder Erklärung von extensionalen Antworten angesehen werden [YP99]. Sie liefert Informationen über die Daten in der Datenbank und nicht die Daten selbst [Sva16]. Die Repräsentation gestaltet sich im Vergleich zu extensionalen Antworten aufgrund der erhöhten Komplexität [Mot94] schwierig. Einerseits bestehen die intensionalen Informationen aus dem Relationenmodell [Sva16] und allen darin enthaltenen Elementen, wie Relationen, Views oder Integritätsbedingungen [Mot94], andererseits können zu den Daten weitere Metadaten, wie Annotationen existieren, welche die weitere Bedeutung dieser Daten bezüglich einer konkreten Fachdisziplin beschreiben. Ferner ist es schwierig, intensionale Antworten zu berechnen. In [YP99] und zusammengefasst in [Sva16] werden sogenannte Konzepthierarchien eingeführt, um eine semi-automatisierte Methode zur Generierung von intensionalen Antworten aus einer großen Menge von extensionalen Informationen bereitzustellen.

Als Beispiel wird wieder Anweisung 3.1 betrachtet. Als eine Art von intensionalen Informationen, können die hinterlegten Kommentare als Metadaten der Attribute *lon* und *lat* im DBMS selbst betrachtet werden (siehe dazu die SQL-Anweisung A.1 im Anhang). Diese intensionale Antwort in Form einer extensionalen Relation ist in Tabelle 3.2 zu sehen. Wie diese Relation entstanden ist, ist im Anhang in Abfrage A.3 zu sehen.

Andererseits können weitere Metadaten derart hinterlegt sein, dass das DBMS anhand der Attributnamen erkennt, dass es sich um geographische Informationen bezüglich eines Ortes handelt

name	value	info
lon	57.32083	The longitude given in decimal degree of the location where this deployment has operated.
lat	20.13667	The latitude given in decimal degree of the location where this deployment has operated.

Tabelle 3.2: extensionale Antwort erweitert um Metadaten

und welche Maßeinheit diese besitzen. Weiterhin könnte ein Dienst existieren, welcher anhand dieser geographischen Informationen den konkreten Namen des gesuchten Ortes heraussucht. Somit könnte das zu einer textuellen für Menschen lesbaren intensionalen Antwort führen:

> „Der Name des Ortes von Deployment 15 an der geographischen Länge (lon=57.32083) und Breite (lat=20.13667) ist ‚östliches Gotlandbecken'".

Diese Art von Antwort verdeutlicht die bereits angesprochenen Schwierigkeiten und Komplexität bei den intensionalen Antworten: das DBMS muss „wissen", dass die Attribute *lon* und *lat* geographische Angaben sind und in welchem Format bzw. Maßeinheit diese vorliegen. Die Schwierigkeit hierbei ist die Repräsentation dieses Wissens: Charakterisierung, Spezifizierung und Wahl des Format. Die nächste Hürde besteht darin, das bestehende DBMS um diesen Mechanismus des „Wissens" zu erweitern und während des laufenden Betriebes darauf zurückzugreifen, um gegebenenfalls gestellte Provenance-Anfragen intensional beantworten zu können. Das Kontaktieren von Diensten, die weitere Informationen über die gelieferten und um Metadaten angereicherten Tupel aus dem DBMS bereitstellen, ist dabei außerhalb des DBMS anzusiedeln bzw. wenn überhaupt nur mit speziellen Erweiterungen zu handhaben. Ferner muss die zurückgelieferte Information derart an das DBMS übergeben werden, als das dieses dann dem Nutzer die textuell aufbereitete (intensionale) Antwort geben kann. Das wirft wiederum die Frage auf, wie intensionale Antworten repräsentiert werden sollten: textuell, in Tabellenform, grafisch oder als Kombinationen davon.

Bei den folgenden Provenance-Anfragen und den dazugehörigen Beispielen sind die Antworten immer extensional. Ausgangspunkt ist die Anweisung 3.2, welche die Durchschnittstemperaturen aller Deployments liefert. Untersuchungsgegenstand in den folgenden Provenance-Anfragen ist das blau hinterlegte Tupel {15, D15, 6.335741...} in der Ergebnisrelation 3.3.

```
SELECT d.deployment_id, d.name, AVG(p.temperature) AS avg_temp
FROM deployment d, payload p
WHERE d.deployment_id = p.deployment_id
GROUP BY d.deployment_id, d.name
```

Anweisung 3.2: Durchschnittstemperaturen aller Deployments

deployment_id	name	avg_temp
14	D14	6.026723...
15	D15	6.335741...

Tabelle 3.3: Ergebnisrelation von Anweisung 3.2

3.2.2.3 Why-Provenance

Diese Provenance-Anfrage beantwortet, warum sich ein bestimmtes Tupel in der Ergebnisrelation befindet. Zur Beantwortung dieser Frage ist es notwendig, dass die Information der Herkunft für jedes Ergebnistupel mitgeführt werden muss. Das kann u. a. auch bedeuten, dass die komplette Ursprungsrelation dafür übernommen werden kann [Sva16]. Nach [Mü15] entspricht *Why*-Provenance der Semantik der WHERE-Klausel in einer SQL-Abfrage. Weiterhin existiert die Negation dieser Anfrage, die sogenannte *Why-Not*-Provenance, die beantworten soll, warum ein Tupel gerade nicht in der Ergebnisrelation auftaucht. Zusätzlich kann die Antwort hier die minimalste Änderung der Anfrage enthalten, damit das gewünschte Tupel doch im Ergebnis auftaucht. Zu klären ist aber, was mit „minimalste Änderung“ gemeint ist, da hier unterschiedliche Distanzfunktionen und Metriken gemeint sein können [Heu16]. Nach [Her11] gehört die *Why-Not*-Anfrage zur Kategorie *Missing Data* Provenance und kann detailliert in [CJ09] und [HH10] nachgelesen werden.

Ausgehend von der Ergebnisrelation 3.3 wird nun untersucht, warum das Tupel {15, D15, 6.335741...} in dieser auftaucht. Als Antwort wird folgende Tupel-Menge geliefert. Die Granularität entspricht hier einzelner Attributwerte.

$$
\begin{aligned}
&\{t2.deployment\_id, t2.name, \{\\
&\quad t100.deployment\_id, t100.temperature,\\
&\quad t101.deployment\_id, t101.temperature,\\
&\quad t102.deployment\_id, t102.temperature,\\
&\quad ...,\\
&\quad t1000.deployment\_id, t1000.temperature,\\
&\quad t1001.deployment\_id, t1001.temperature,\\
&\quad t1002.deployment\_id, t1002.temperature,\\
&\quad ...,\\
&\quad t10000.deployment\_id, t10000.temperature,\\
&\quad t10001.deployment\_id, t10001.temperature,\\
&\quad t10002.deployment\_id, t10002.temperature,\\
&\quad ...\\
&\}\}
\end{aligned}
$$

Um diese Menge optisch zu veranschaulichen, werden die Attributwerte aus der Ausgangsrelation hellblau eingefärbt und es ergeben sich die Tabellen 3.4 und 3.5. Aufgrund der Durchschnittsbe-

rechnung wurden alle Tupel der Ursprungstabelle 1.2 genutzt. Somit tauchen diese auch in der Antwort auf.

	deployment_id	name	deployed_ts	recoverd_ts	lon	lat
t1	14	D14	2015-04-21 17:00	2015-07-18 09:00	57.32277	20.13333
t2	15	D15	2015-07-26 16:00	2015-09-24 11:00	57.32083	20.13667

Tabelle 3.4: Erweiterte deployment-Tabelle mit Tupel-Bezeichner

	id	deployment_id	profile_ts	pressure	temperature	conductivity
t3	1	14	2015-06-02 18:01:46	201.459	6.866	14.456
t4	2	14	2015-06-02 18:01:47	201.116	6.864	14.454
t5	3	14	2015-06-02 18:01:48	201.065	6.865	14.454
			...			
t100	2942	15	2015-09-13 02:57:33	202.939	6.869	14.547
t101	2943	15	2015-09-13 02:57:34	202.952	6.867	14.548
t102	2944	15	2015-09-13 02:57:35	202.952	6.867	14.547
			...			
t1000	14685	15	2015-09-14 05:57:35	202.817	6.847	14.466
t1001	14686	15	2015-09-14 05:57:36	202.813	6.848	14.467
t1002	14687	15	2015-09-14 05:57:37	202.808	6.848	14.467
			...			
t10000	23057	15	2015-09-15 00:00:02	202.719	6.854	14.471
t10001	23058	15	2015-09-15 00:00:03	202.724	6.853	14.47
t10002	23059	15	2015-09-15 00:00:03	202.724	6.853	14.47
			...			

Tabelle 3.5: Erweiterte payload-Tabelle mit Tupel-Bezeichner

3.2.2.4 Where-Provenance

Diese Anfrage gibt die Orte, aus denen die Tupel für die Antwort herauskopiert wurden, zurück. Orte sind je nach gewählter Granularität: ganze Datenbanken, Schemata, Relationen und/oder einzelne Attribute. Es wird eine Beziehung zwischen Eingabeort (Relation, Tupel, Attribut) und dem Ergebnis (Ausgabeort) hergestellt. Der Unterschied zur *Why*-Anfrage ist der, dass diese den Zusammenhang zwischen Quell- und Zieltupel beschreibt, während die *Where*-Anfrage die Beziehung zwischen Quell- und Zielorten, z. B. die Attribute einer Relation, wiedergibt [Her11].

Zur Beantwortung der *Where*-Provenance des Attributwerts {15} (*deployment_id*, dunkelblau hinterlegt) in der Ergebnisrelation 3.3, wird hier auf Granularitätsebene von Attributen nur das Attribut {*t2.deployment_id*} aus der erweiterten deployment-Tabelle 3.4 geliefert. Das Attribut *deployment_id* aus der payload-Tabelle 3.5 wird hier nicht zur Beantwortung der *Where*-Provenance-Anfrage herangezogen, da in der zugrundeliegenden Anweisung 3.2 die Projektion auf das Attribut *deployment_id* der deployment-Tabelle 3.4 stattfindet. Somit besagt die Antwort

der *Where*-Provenance-Anfrage, dass für den Attributwert {15} das Attribut *deployment_id* aus der Tabelle *deployment* 3.4 genutzt und dort der Attributwert herauskopiert wurde.

Die Provenance-Informationen der *Why*- und *Where*-Anfragen können wiederum zusätzlich als extensionale Ergebnisrelation separat oder an die bereits erzeugte Ergebnisrelation als weitere „künstliche“ Attribute angehängt und so dem Benutzer repräsentiert werden. Diese Vorgehensweise wird z. B. vom PERM-System genutzt, welches in Abschnitt 3.2.3 beschrieben wird

3.2.2.5 How-Provenance

Diese Art der Anfrage gibt die Informationen zurück, auf welche Art und Weise aus den Ursprungstupeln der Eingabe die Ergebnisrelation entstanden ist. Sie beschreibt also die angewendeten Transformationen bzw. wie die Eingabetupel an der Erzeugung der Ausgabetupel beteiligt sind. Die *Why*-Provenance lässt sich stets aus der *How*-Provenance ableiten, die Gegenrichtung gilt dabei aber nicht [Her11].

Die Antwort auf diese Art von Provenance wird üblicherweise als Polynom angegeben bzw. beschrieben [Her11]. Dabei sind folgende Operatoren in Verwendung: Die Multiplikation ($*$) entspricht der Kombination durch einen Verbund. Die Addition ($+$) ist die Vereinigung von Tupeln. Das Tensorprodukt ($\otimes$) entspricht der Multiplikation des Provenance-Polynoms mit dem gesuchten Attributwert. Ausgehend von Ergebnisrelation 3.3 ist die Frage zu beantworten, auf welche Art und Weise das blau hinterlegte Tupel $\{15, D15, 6.335741...\}$ - ausgehend von der *Why*-Provenance - entstanden ist. Dabei werden wie bei der *Why*-Provenance alle beteiligten Tupel aufgesammelt. Zusätzlich benötigt die Durchschnittsberechnung die Anzahl der beteiligten Tupel. Als Basis für das folgende Provenance-Polynom p dienen wieder die erweiterten deployment- und payload-Tabellen 3.4 und 3.5:

$$\begin{aligned} p &= [(t2 * t100) \otimes 6.869 + (t2 * t101) \otimes 6.867 + ... + (t2 * tn) \otimes \#.\#\#\#]/[t100 + t101... + tn] \\ &= [(t2 * t100) \otimes 6.869 + (t2 * t101) \otimes 6.867 + ... + (t2 * tn) \otimes \#.\#\#\#]/n \end{aligned}$$

3.2.3 Techniken & aktueller Stand

In diesem Abschnitt erfolgt ein kurzer Abriss darüber, welche theoretischen Techniken existieren, um Provenance und die dazugehörigen Anfragen formal zu beschreiben sowie ein aktueller Stand der Forschung. Des Weiteren werden Systeme benannt, die Provenance bereits implementieren und anbieten.

Nach [Heu15] konzentrieren sich viele Arbeiten auf dem Gebiet des Provenance-Managements auf die Reproduzierbarkeit der Ergebnisse aus den Rohdaten, d.h. sie liefern eine Vorwärtsverfolgung. Daneben gibt es weitere Ansätze, die zusätzlich die Nichtfälschbarkeit sowie den Urhebernachweis berücksichtigen. [BKT00] führt dazu aus, dass gegenwärtige DBMS derzeit keine Unterstützung

zur Aufzeichnung von Provenance-Informationen bieten. Es fehlt die Möglichkeit, Erläuterungen und zusätzliche (Meta)-Informationen in Form von Anmerkungen den Daten ad hoc hinzuzufügen. Dies wird aber oft benötigt, um Provenance aufzeichnen zu können. Schließlich stellt sich die Frage, was die technischen Herausforderungen bei der Betrachtung von Data Provenance sind.

Wird die unterste Ebene von DBMS betrachtet, so sind es die einzelnen Operatoren (SQL, relationale Anfragealgebra) und Modelle, auf die sich ein Teil der Forschung konzentriert, um Rohdaten rekonstruieren zu können [CAB+14, Fre09]. Es fehlt aber eine durchgängige und automatisierte Vorgehensweise. Des Weiteren erfolgte die Charakterisierung und Formalisierung von Provenance-Eigenschaften, leider aber nicht für die automatische Rekonstruktion der Originaldaten rückwärts [BKWC01, CMDZ10, PIW11] zitiert nach [Heu15].

Um eine solche Rekonstruktion, also von einer Ergebnisrelation zu den Ursprungsdaten, vornehmen zu können, werden Techniken der inversen Schemaabbildungen und inverser Analyseoperatoren gebraucht [Heu15]. Die Anwendung von inversen Anfragen, die, angewendet auf den Ursprungsdaten, alle diejenigen Tupel zurückliefern, welche an der Erzeugung der Ergebnisrelation beteiligt waren, wird in [BKT00] vorgeschlagen. Zeitgleich werden aber auch die Grenzen dieses Ansatzes aufgezeigt: sobald die Negation oder eine Form der Aggregation auftauchen, schlägt die formale Definition mittels positiver relationaler Algebra[1], nämlich dass ein Eingabe-Tupel an der Entstehung eines Ausgabe-Tupels genau dann beteiligt ist, wenn es in irgendeiner Form dazu genutzt wurde, das Ausgabe-Tupel zu erzeugen, fehl.

Parallel dazu existieren temporale Datenbanken und Überwachungstechniken durch Loggen. Wie temporale Datenspeicherung in Datenbanken funktioniert, ist in Abschnitt 3.3 beschrieben. Hier folgt nur eine knappe Zusammenfassung aus [Gla14], warum diese Technik allein nicht ausreicht, um Provenance sicher zu stellen. Zwar findet durch die temporale Speicherung eine Änderungsverfolgung von Tupeln statt, dennoch bietet sie keine Hilfe bei der Verfolgung der Provenance der Anfrage selbst. Weiterhin kann nicht festgehalten werden, wie neue Datensätze im DBMS erzeugt worden sind, z. B. durch die Kombination bereits vorhandener Daten. Schlussendlich bieten temporale Datenbanken nur einen Zugriff auf bereits dauerhaft gespeicherte Versionen von Tupeln. Temporäre Versionen, welche während einer Transaktion entstehen, werden verworfen. Bei einigen Anwendungsfällen, z. B. dem *Audit*, kann es aber entscheidend sein, diese Zwischenversionen einsehen zu können. Das Überwachen der Anfragen an das DBMS mittels Loggen kann als Typ von Transformations-Provenance (vgl. 3.2.2) angesehen werden. Dennoch kann damit nicht festgestellt werden, welche Transformation auf welche Daten stattfanden. Transformations-Provenance dagegen zeichnet genau diese Information auf: welche Transformation wurde auf welche Daten angewendet. [Gla14] kommt zu dem Schluss, dass das Loggen durch Erweitern mittels temporaler Techniken die Nachteile in gewissem Maße kompensieren kann.

Die in Abschnitt 3.2.2.2 behandelten Provenance-Anfragen sind in [BKWC01, GKT07, CCT09]

[1] relationale Algebra ohne den Differenz-Operator (auf Mengen) [Sir09]

formal spezifiziert und charakterisiert. [BKWC01] formalisierte die *Where*- und *Why*-Provenance unter Verwendung von sogenannten *witnesses*. Diese beschreiben diejenige Menge von Tupeln, die zur Bildung eines Tupels aus der Ergebnisrelation beigetragen haben. Ferner führte [GKT07] die Berechnung der *Why*-Provenance für Anfragen aus der positiven relationalen Algebra auf die algebraische Struktur der Halbringe zurück, deren symbolische Darstellung gewöhnlichen Polynomen entspricht. Zeitgleich wurden die Grenzen von *Why*-Provenance aufgezeigt, nämlich dass bei mehreren möglichen Eingangs-Tupeln nicht unterschieden werden kann, welches konkret durch welche Kombination zum Ergebnis beigetragen hat [GKT07]. Dies machte die Einführung der *How*-Provenance notwendig, welche formal durch Polynome beschrieben werden kann. In [CCT09] werden die drei Haupt-Notationsweisen von Provenance beschrieben, verglichen, Beziehungen zwischen ihnen hergestellt sowie Anwendungsgebiete aufgezeigt.

Aufbauend auf die verschiedenen Arten der Beschreibung von Provenance hat Boris Glavic in [Gla10] die „Perm Influence Contribution Semantics“ (PI-CS) entwickelt, welche in das Provenance Management System PERM (*Provenance Extension of the Relational Model*) eingeflossen ist [GMA13]. PERM selbst steht als Open Source[2] zur Verfügung und unterstützt *Why*-, *Where*- und *How*-Provenance [GMA13]. Das System wurde als Erweiterung eines relationalen DBMS, konkret PostgreSQL, entwickelt und benutzt eine sogenannte SQL-PLE[3] Spracherweiterung [GA09] (siehe dazu Beispielanweisung 4.11). Schlüsselmerkmal von PERM ist die Repräsentation der Provenance zusammen in einer einzelnen Ergebnisrelation. Diese Darstellung wird durch das Umschreiben der ursprünglichen Anfrage in eine um Provenance-Informationen erweiterte Anfrage (*query rewrites*) erzeugt. Durch diese Vorgehensweise kann PERM von den bereits in das DBMS implementierten Techniken der Anfrageverarbeitung, Optimierung und Daten-Speicherung profitieren [GMA13]. Nachfolgersystem von PERM ist GProM (*Generic Provenance database Middleware*), welches auch auf dem Konzept der *query rewrites* basiert und auf die algebraische Graph-Darstellung von Datenbank-Operationen angewendet wird [AGR+14]. Eine Auswahl weiterer Systeme und Frameworks, die Provenance-Informationen bereitstellen sind in [Sva16] zu finden.

In diesem Abschnitt wurde beschrieben, was Provenance bzw. Provenance-Management ist und welche Qualitätskriterien existieren. Ferner wurde gezeigt, welche Provenance-Anfragen und dazugehörige Antworttypen vorhanden sind. Dazu wurden eine Beispielanfrage und die dazugehörigen extensionalen Antworten verwendet, um die Unterschiede herauszuarbeiten. Danach wurde ein Überblick über die bestehenden Techniken gegeben und Formalismen für die Beschreibung von Provenance in DBMS beschrieben.

[2] siehe http://permdbms.sourceforge.net/ und https://sourceforge.net/projects/permdbms/
[3] *Provenance Language Extension*

3.3 Temporale Datenhaltung

In diesem Abschnitt wird die temporale Datenhaltung beschrieben. Diese wurde bereits detailliert in [Mey15] wieder gegeben, daher soll an dieser Stelle nur ein knapper Ausschnitt mit Beispielen erfolgen.

Das Konzept der temporalen Datenhaltung entspricht dem Abbilden von zeitlichen Zusammenhängen zwischen Datensätzen. So kann ein Verlauf sowie die Änderungen von Daten über Zeiträume hinweg erfasst werden. Dazu werden temporale Datentypen als Attribute verwendet, um einen Zeitintervall zu charakterisieren. Anhand des Anwendungskontextes muss entschieden werden, welche Granularität, z. B. DATE oder TIME mit unterschiedlichen Genauigkeiten in der Nachkommastelle, zum Einsatz kommt. Ein Zeitintervall wird in der temporalen Datenhaltung als *closed-open* definiert, d.h. der Startzeitpunkt liegt im Intervall, der Endzeitpunkt dagegen nicht mehr [SGM00, SSH13]. Das Festhalten des zeitlichen Verlaufs von Daten im DBMS wird auch als *Historisierung* oder im weiteren Verlauf dieser Arbeit auch als *Versionierung* bezeichnet. Es werden dabei keine Tupel physikalisch gelöscht, sondern es erfolgt mittels der temporalen Attribute eine Kennzeichnung derart, dass ein „gelöschter“ Datensatz nicht mehr in aktuell gültigen Daten auftaucht. Gültige Datensätze sind dabei diejenigen, welche durch das größtmögliche Datum des DBMS als Endzeitpunkt im Intervall repräsentiert werden. Diese Vorgehensweise ist die am häufigsten in der Literatur beschriebene Methode und ermöglicht in der WHERE-Klausel das aktuelle Datum zu nutzen, um aktuelle und damit gültige Datensätze zu ermitteln. Die native Unterstützung von grundlegenden Aspekten der Speicherung temporaler Daten wurde mittels des SQL:2011-Standards eingeführt. Dazu wurde die Anfragesprache SQL um temporale Konzepte erweitert, so dass Zeitdimensionen automatisch verwaltet, manipuliert und Integritätsbedingungen berücksichtigt werden können [SSH13]. So kann bereits im CREATE-Statement für Tabellen angegeben werden, dass es ein Zeitintervall gibt und aus welchen temporalen Attributen es sich zusammensetzt (vgl. dazu Anweisung 3.3 in Abschitt Gültigkeitszeit 3.3.1). Zusätzlich werden neue temporale Prädikate zum Abfragen über Zeitintervalle in einem SFW-Block bereitgestellt: CONTAINS, OVERLAPS, EQUALS, PRECEDES, SUCCEEDS, IMMEDIATELY RECEDES und IMMEDIATELY SUCCEEDS [KM12, SSH13].

Die zeitbezogene Speicherung von Daten kann auf drei Arten erfolgen: mittels benutzerdefinierter Gültigkeitszeit, mittels Transaktionszeiten des DBMS oder durch eine Kombination aus Gültigkeitszeit und Transaktionszeit, der sogenannten bitemporalen Speicherung. Die reine Anwesenheit eines temporalen Attributs, z. B. eines Geburtstages, heißt aber nicht gleich automatisch, dass diese Tabelle eine temporale Tabelle im Sinne der oben genannten drei Arten ist [SGM00]. In folgenden Beispielen werden ungültige, d.h. historisierte, Tupel in den jeweiligen Tabellen hellgrau hinterlegt, um eine bessere Lesbarkeit zu gewährleisten.

3.3.1 Relationen mit Gültigkeitszeit

Diese Art von Tabellen enthalten vom Anwender und dessen Kontext definierte Zeitintervalle. Das heißt, dass es dem Benutzer obliegt, wie und mit welcher Granularität die Attribute definiert und belegt werden. Das Zeitintervall beschreibt dabei die Gültigkeit von Fakten aus der modellierten Realität [SGM00]. Als Beispiel dient die aus Anweisung 3.3 erzeugte Relation 3.6 mit einigen Beispieldaten. Zwei Bedingungen müssen hier erfüllt sein: der Endzeitpunkt muss vor dem Startzeitpunkt liegen [SSH13] und das Zeitintervall, welches im Primärschlüssel Verwendung findet, darf sich mit keinem anderen Zeitintervall gleicher *id* überlappen. In Relation 3.6 ist zu erkennen, dass das Enddatum der historisierten und das Startdatum der neu hinzugekommenen Zeile mit *id=1* identisch sind und dies vermeintlich eine Überlappung und damit eine Verletzung der Primärschlüsselbedingung darstellt. Aufgrund der Definition der Zeitintervalle als *closed-open* ist dies aber nicht der Fall.

```
CREATE TABLE sensors (
  id INTEGER NOT NULL,
  name VARCHAR(100) NOT NULL,
  vt_begin DATE NOT NULL DEFAULT CURRENT_DATE,
  vt_end DATE NOT NULL DEFAULT DATE '9999-12-31',
  PERIOD FOR ValidTime (vt_begin, vt_end),
  PRIMARY KEY (id, ValidTime WITHOUT OVERLAPS)
)
```

Anweisung 3.3: Erzeugung einer temporalen Gültigkeitstabelle

id	name	vt_begin	vt_end
1	TempSens1000	1960-01-01	1988-10-31
1	TempSens3000	1988-10-31	9999-12-31
2	CondSens2000	1978-06-09	9999-12-31

Tabelle 3.6: Tabelle mit Gültigkeitszeit

Die Gültigkeitszeit beantwortet die Frage, wann ein Fakt aus der Realität gültig war, bzw. noch ist. An den Beispieldaten ist zu sehen, dass ein neuer Sensor am 31.10.1988 hinzugekommen ist und gleichzeitig der Sensor „TempSens1000“ als nicht mehr gültig deklariert wurde, indem das Enddatum auf 31.10.1988 gesetzt wurde. Im Anwendungskontext betrachtet würde dies bedeuten, dass der alte Sensor durch den neuen ersetzt wurde. Anhand der *id* kann der zeitliche Verlauf dieses Sensors nachvollzogen werden.

3.3.2 Relationen mit Transaktionszeit

Bei dieser Art von Tabellen wird das Zeitintervall vom DBMS verwaltet. Der Anwender hat dabei keine Möglichkeiten, das Zeitintervall zu manipulieren. Dies betrifft hauptsächlich Tupel, die in der Vergangenheit liegen: der Anwender darf keine Gelegenheit besitzen, diese zu ändern. Des Weiteren wird hier vom DBMS sichergestellt, dass jede UPDATE- oder DELETE-Anweisung automatisch den alten Zustand der betreffenden Tupel erhält, bevor diese durchgeführt wird. Dies stellt sicher, dass ein exakter Verlauf von Datenänderungen aufgezeichnet wird. Als Beispiel wird mittels Anweisung 3.4 die Tabelle 3.7 erzeugt. Diese enthält die beiden *transaction-time*-Attribute (tt) *tt_start* und *tt_stop*, welche das Intervall der Transaktionszeit des DBMS widerspiegeln. Es wurde ein TIMESTAMP mit höchster Präzision gewählt, aber in der Abbildung der Tabelle 3.7 der Übersicht halber nicht mit angegeben.

```
CREATE TABLE sensors (
  id INTEGER NOT NULL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  tt_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
  tt_stop TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
  PERIOD FOR SYSTEM_TIME (tt_start, tt_stop)
) WITH SYSTEM VERSIONING
```

Anweisung 3.4: Erzeugung einer temporalen Transaktionszeittabelle

id	name	tt_start	tt_stop
1	TempSens1000	2014-06-09 09:00:00	2016-02-05 14:00:00
1	TempSens3000	2016-02-05 14:00:00	9999-12-30 00:00:00
2	CondSens2000	2015-09-10 11:00:00	9999-12-30 00:00:00

Tabelle 3.7: Tabelle mit Transaktionszeit

Der Primärschlüssel ist hier das *id*-Attribut. Hintergrund ist der, dass die Primärschlüsselbedingung nur auf aktuell gültige Datensätze angewendet wird. Aktuell gültige Datensätze sind dabei diejenigen, deren Systemzeit-Intervall den aktuellen Zeitpunkt enthält. Alle anderen Tupel gelten demnach als historisiert. In der Beispiel-Tabelle 3.7 ist der Datensatz „TempSens1000“ ungültig, also historisiert. Auf diesen muss also die Primärschlüsselbedingung nicht mehr angewendet werden, obwohl das *id*-Attribut den gleichen Wert wie der aktuell gültige Datensatz „TempSens3000“ hat [KM12] zitiert nach [Mey15].

3.3.3 Bitemporale Relation

Wie eingangs erwähnt, setzen sich bitemporale Tabellen aus zwei Zeitintervallen, nämlich der Gültigkeitszeit und der Transaktionszeit zusammen. Diese Art der temporalen Datenhaltung kann für die lückenlose Dokumentation der Datenänderungen und Entwicklung der Daten über die Zeit hinweg genutzt werden. Als Beispiel dient Anweisung 3.5, welche die bitemporale Tabelle 3.8 erzeugt.

```
CREATE TABLE sensors (
  id INTEGER NOT NULL,
  name VARCHAR(100) NOT NULL,
  vt_begin DATE NOT NULL DEFAULT CURRENT_DATE,
  vt_end DATE NOT NULL DEFAULT DATE '9999-12-31',
  tt_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
  tt_stop TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
  PERIOD ValidTime (vt_begin, vt_end),
  PERIOD SYSTEM_TIME (tt_start, tt_stop),
  PRIMARY KEY (id, ValidTime WITHOUT OVERLAPS)
)
```

Anweisung 3.5: Erzeugung einer bitemporalen Tabelle

id	name	vt_begin	vt_end	tt_start	tt_stop
1	TempSens1000	1960-01-01	9999-12-31	2010-06-09 20:00:00	2016-02-05 14:00:00
1	TempSens1000	1960-01-01	1988-10-31	2016-02-05 14:00:00	9999-12-30 00:00:00
1	TempSens3000	1988-10-31	9999-12-31	2016-02-05 14:00:00	9999-12-30 00:00:00
2	CondSens2000	1978-06-09	9999-12-31	2015-09-10 11:00:00	9999-12-30 00:00:00

Tabelle 3.8: Bitemporale Tabelle

Es ist zu erkennen, dass die erste Zeile versioniert wurde, da die Transaktionszeit angibt, dass diese Zeile ab dem 05.02.2016 um 14:00 Uhr als gelöscht gekennzeichnet wurde und somit als Information nicht mehr zur Verfügung steht. Die zweite Zeile sagt aus, dass der Datensatz vom 01.01.1960 bis 30.10.1988 (*closed-open*) im Sinne der Gültigkeitszeit gültig war. Sie ist also Nachfolger der ersten versionierten Zeile. Die dritte Zeile ist sowohl gültig im Sinne der Gültigkeitszeit als auch der Transaktionszeit und würde als Ergebnis der Anfrage nach aktuell gültigen Daten mit Selektion auf den aktuellen Zeitpunkt geliefert werden. Die letzte Zeile wurde nicht verändert. Der Primärschlüssel setzt sich wie bei den Gültigkeitszeit-Tabellen in Abschnitt 3.3.1 aus der *id* und dem Gültigkeitszeitintervall (vt) ohne Überschneidungen zusammen, wird aber auch nur auf aktuelle Systemzeit-Tupel angewendet. Andernfalls würde die erste Zeile die Primärschlüsselbedingungen verletzen, falls auf diese trotz der Ungültigkeit diese Primärschlüsselbedingungen angewendet werden würden.

3.3.4 Aktueller Stand der Forschung

Die Erforschung von temporaler Datenhaltung und temporalen Datenbanken erfolgt schon seit einigen Jahrzehnten [SSH13]. In [Sno95] wurde die *Temporal Structured Query Language* (TSQL2) erarbeitet und in [SGM00] ist beschrieben, wie die Entwicklung von zeitbezogenen Datenbank-Anwendungen aus praktischer Sicht erfolgen kann. Die in SQL:2011 neu eingeführten Erweiterungen erlauben, zeitbezogene Daten und Zeitdimensionen zu speichern, Integritätsbedingungen einzuhalten sowie temporale Konzepte in Anfragen zu berücksichtigen. Trotzdem verhält sich SQL:2011 genau entgegengesetzt zu den ersten Ansätzen der Forschung von [Sno95]. Von [Sno95] wurde vorgeschlagen, dass die temporalen Attribute vor dem Anwender versteckt werden, so dass auch in den SQL-Anfragen keine weiteren Zeitinformationen enthalten sein sollen, um auf aktuellen Daten arbeiten zu können. Dies ist in SQL:2011 nicht der Fall, denn die Attribute sind sichtbar. Um auf aktuellen Gültigkeitszeit-Tupeln zu arbeiten, muss immer der aktuelle Zeitpunkt selektiert werden [SSH13]. Hintergrund für diese Vorgehensweise ist der, dass es für die Hersteller und deren bestehende DBMS-Produkte einen minimalen Aufwand für die Implementierung des Standards geben sowie auf tiefgreifende Eingriffe verzichtet werden sollte. Für spätere Versionen des SQL-Standards sind zusätzliche Erweiterungen, wie Verbunde von temporalen Relationen, Gruppierungen oder Aggregate sowie Anwendung von mehr als einer Gültigkeitszeit pro Relation in Vorbereitung [SSH13].

Der SQL:2011-Standard selbst wurde bisher nur von einer Minderheit der Datenbankhersteller umgesetzt - und das auch nur teilweise. So besitzt DB2 V10 von IBM die sogenannten „time travel“-Anfragen sowie weitere Konzepte, die direkt vom SQL:2011-Standard abgeleitet wurden, um temporale Datenhaltung zu unterstützen [NS12]. Zur Umsetzung von Transaktionszeit-Tabellen wird dazu eine zweite Tabelle genutzt, welche nur die bereits historisierten Daten enthält. Somit umfasst die ursprüngliche „eigentliche“ Transaktionszeit-Tabelle nur aktuelle gültige Daten. Werden Tupel aus der Vergangenheit mittels temporaler Anfrage ermittelt, so wird erst dann die zweite Tabelle mit einbezogen. Weitere kommerzielle Systeme, die temporale Funktionalitäten zur Verfügung stellen, sind u. a. Oracle 12c mit dem sogenannten *Temporal Validity Support* und der *Flashback Technology* [MK16] oder der Microsoft SQL Server 2016 [Mic16]. Unter Betrachtung von Open-Source DBMS, ist PostgreSQL zu nennen. Dort wird über die Einführung temporaler Konzepte diskutiert [Pos16e], während es bereits einen Wiki-Eintrag für SQL:2011 gibt [Pos16a]. Parallel dazu entstehen temporale Erweiterungen von Open-Source-Entwicklern [Pos16b]. Durch die Einführung neuer *Range*-Datentypen in PostgreSQL 9.2 können nun viele temporale Konzepte transparenter und unkomplizierter umgesetzt werden. Aus der Bachelorarbeit [Mey15] ist bekannt, dass es zwar möglich ist, ohne den SQL:2011-Standard in MySQL temporale Datenhaltung zu betreiben, es aber bedeutet, jede einzelne Anfrage und DML (*Data Manipulation Language*)-Anweisungen dementsprechend in mindestens fünf einzelne Anweisungen zu zerlegen, um die Funktionsweise nachzubilden. Diese künstliche Erhöhung der Komplexität ist fehleranfällig und kompliziert. Aus diesen Gründen wird MySQL in dieser Arbeit nicht weiter

betrachtet. Ferner sind reine temporale Datenbanken ein aktives Forschungsgebiet und in der Praxis so gut wie nicht vorhanden. Als Ausnahme sei hier *Teradata* genannt [SSH13].

Der SQL:2011-Standard bietet heute bereits temporale Konzepte, die sonst nur durch externe Anwendungsprogrammierung seitens der DBMS Anwender umgesetzt wurden. Durch die aktive Forschung und Weiterentwicklung sowohl bei den kommerziellen Anbietern als auch im Open-Source-Bereich sind somit schon heute temporale Konzepte und temporale Datenhaltung verfügbar.

4 Konzept

In diesem Kapitel wird ein Konzept entwickelt, das beschreibt, wie sowohl Veränderungen von Datenbankschemata als auch Veränderungen von darauf angewendeten Funktionen versioniert werden können. Im Anschluss folgen Untersuchungen über die Anwendbarkeitsvorrausetzungen für Funktionen auf veränderten Schemata sowie von veränderten Funktionen auf gleichbleibenden Schemata. Darüber hinaus werden Möglichkeiten zur Kompensation von Funktions- und Schemaänderungen beschrieben. Abschließend wird der Zusammenhang von Provenance und temporaler Datenhaltung skizziert sowie die dabei auftretenden Herausforderungen aufgezeigt.

4.1 Änderungen und Versionierung

Es erfolgt die Beschreibung der Versionierung mittels temporaler Datenhaltung von Veränderungen von Funktionen. Dabei wird angenommen, dass der SQL:2011-Standard zur Verfügung steht. Anhand eines Beispiels wird aufgezeigt, welche Anweisungen ausgeführt werden müssen, um eine temporale Datenhaltung zu gewährleisten. Danach erfolgt eine Beschreibung, wie eine Versionierung von Schemata aussehen kann.

4.1.1 Versionierung von Funktionsänderungen

Im Kontext dieser Arbeit entsprechen die benutzerdefinierten SQL-Funktionen (UDFs/STPs) den Auswerte- und Analyseverfahren auf Forschungsdaten. Sie sind nach den Aufgaben des Forschungsdatenmanagements 3.1 neben den Rohdaten ein wichtiger Bestandteil für die Rückverfolgbarkeit und das Zustandekommen von Ergebnissen in wissenschaftlichen Arbeiten. Die Begriffe (SQL-)Funktion(en) sowie Auswerte- und Analyseverfahren werden in diesem Kapitel synonym verwendet.

Wie in Abschnitt 2.1 bereits erwähnt, wurde in [Mö16] ein Konzept zur Versionierung von UDFs/STPs in MySQL präsentiert. Vorgeschlagen wurde, dass die Funktionen und deren Parameter in bitemporalen Tabellen parallel zur *information_schema*-Datenbank[1] versioniert werden.

[1] Datenbank mit Sichten (*Views*), welche Informationen über alle anderen Datenbanken und deren Objekte (Tabelle, Attribute, Funktionen), die auf demselben MySQL-Server laufen, bereitstellt [MyS16]

Werden Funktionen geändert oder gelöscht, müssen diese Änderungen zuvor in die Versionierungstabellen eingepflegt werden. Erst danach sollen die Änderungen im DBMS selbst stattfinden. Das bedeutet, dass die alten Funktionen nur in den Versionierungstabellen aber nicht mehr im DBMS selbst gespeichert sind. Das DBMS enthält nur aktuelle Funktionen.

Im Gegensatz dazu wird in dieser Arbeit ein anderes Versionierungskonzept vorgestellt, welches sich von dem in [Mö16] in dem Punkt unterscheidet, als dass die SQL-Funktionen und deren Parameter, Rückgabewerte sowie Funktionskörper im DBMS durch geeignete Umbenennung belassen und nicht physisch gelöscht werden. Die Umbenennung der Funktion im DBMS ist notwendig, da die meisten DBMS keine Funktionen mit gleicher Signatur, d.h. gleichen Namens und gleichen Parametern und deren Typen, zulassen, obwohl sich der Funktionskörper unterscheidet. Das Belassen der SQL-Funktionen im DBMS sorgt dafür, dass zur Verwendung dieser Funktionen, diese später nicht umständlich im DBMS wiederhergestellt werden müssen. Somit können veraltete Funktionen unkompliziert mittels des neuen Namens und der ursprünglichen Parameter aufgerufen werden. Daneben wird eine eigene Tabelle *routines_history* für die Versionierung von an verschiedenen Zeitpunkten stattgefundenen Funktionsänderungen mit benutzerdefinierten Gültigkeitszeiträumen anstelle der bitemporalen Tabellen von [Mö16] angelegt. Das CREATE-Statement für die *routines_history*-Tabelle zur Versionierung von SQL-Funktionen ist im Anhang unter A.4 zu finden. Es wurde die Gültigkeitszeit gewählt, damit bestehende aber trotzdem veraltete Auswerte- und Analyseverfahren im Nachhinein erfasst werden können.

Als Beispiel soll eine Funktion *raw_to_units_press* dienen, die Werte aus den Sensor-Rohdaten in SI-Einheiten (*presssure*, Druck) mittels eines Polynoms umrechnet. Diese Funktion existiert in zwei Versionen. Die veraltete Funktion nutzt ein Polynom dritten Grades (siehe Prototyp-Kapitel Anweisung 5.1), die Nachfolger-Funktion ein Polynom vierten Grades (siehe Prototyp-Kapitel Anweisung 5.2). Es ändert sich somit die Anzahl der Parameter sowie die Berechnungsvorschrift. Der Name dagegen bleibt gleich. Darüber hinaus nimmt die Versionierungstabelle (siehe z. B. Tabelle 4.1) weitere Informationen, wie die Anzahl und die Typen der Parameter, den eigentlichen Funktionskörper sowie die Rückgabewerte auf. Dennoch sind diese Informationen redundant weiterhin im DBMS selbst vorhanden, da die Funktion nur umbenannt, aber nicht gelöscht wurde. Aufgrund der eingeführten Redundanz kann es zur Inkonsistenz zwischen den vom DBMS verwalteten Daten über die SQL-Funktionen und den Daten in der Versionierungstabelle kommen. Dies wird bewusst in Kauf genommen. Die zusätzlichen Informationen in der Versionierungstabelle bieten die Möglichkeit der Wiederherstellung der Funktionen bei einer (ungewollten) physischen Löschung aus dem DBMS. Zur Vorbeugung einer solchen ungewollten Löschung muss der Prozess des Anlegens und des Versionierens, der weiter unten beschrieben wird, eingehalten werden. Darüber hinaus ist eine Kapselung des Prozesses in einer externen Anwendung möglich. Diese Vorgehensweise unterstützt die Minimierung von Fehlern bei der Ausführung der Schritte zur Sicherstellung der Versionierung. Weiterhin können dadurch die Zusatzinformationen in der Versionierungstabelle und damit die Rückverfolgbarkeit der Funktionen und deren Unterschiede an einer Stelle eingesehen werden.

Bei der Darstellung der Versionierungstabelle in den folgenden Beispielen werden die Attribute *routine_result* und *tt* der Übersichtlichkeit halber weggelassen (vgl. dazu Anweisung 5.5). Zusätzlich werden die Attributwerte gekürzt (durch drei Punkte verdeutlicht) dargestellt. Darüber hinaus wurde bei der Darstellung auf die erste Normalform zugunsten der Übersichtlichkeit verzichtet (vgl. dazu das *routine_arguments*-Attribut z. B. in Tabelle 4.1, welches kommasepariert die Namen und Typen der Parameter der Funktion als Attributwerte enthält). In der Praxis würde dagegen eine eigene Relation für die Parameter sowie eine 1:n Zuordnung zwischen Funktion und Parameter existieren. Ein weiterer Grund ist die Verwendung von PostgreSQL bei der Erstellung des Prototypen und der damit verbundenen Anwendung PostgreSQL spezifischer Funktionen. Durch die Funktion *pg_get_function_identity_arguments* werden die Parameter und deren Datentypen in genau jenem kommaseparierten Format zurückgegeben.

Für die Versionierung von Funktionen werden folgende drei Fälle unterschieden: das Anlegen einer neuen Funktion, das Ersetzen und das Löschen von bestehenden Funktionen. Dabei entspricht die Versionierung im Wesentlichen dem Konzept der temporalen Datenhaltung mittels Gültigkeitszeit aus Abschnitt 3.3.1. Weiterhin wird angenommen, dass nur eine Version einer Funktion zur selben Zeit gültig ist. Das bedeutet, dass es im Allgemeinen keine Überschneidungen der Gültigkeitszeiträume mehrerer Versionen einer einzelnen Funktion gibt.

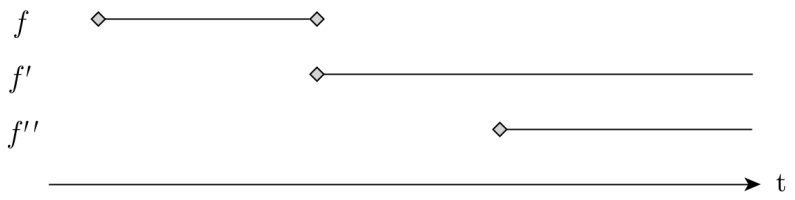


Abbildung 4.1: Versionierung der Funktion f mit Nachfolgerfunktionen f' und f''

Trotzdem existieren in der Wissenschaft Analyse- und Auswertefunktionen, die trotz unterschiedlicher Parameter und Berechnungsvorschriften das gleiche Resultat berechnen. Diese Funktionen sind dabei nicht zwangsläufig Nachfolgerversionen im Sinne der Versionierung. Das bedeutet, dass die Funktionen unabhängig voneinander sind und unterschiedliche Startzeitpunkte der Gültigkeitszeit besitzen. Abbildung 4.1 soll dies verdeutlichen. Funktion f' ist Nachfolgerversion von Ursprungsfunktion f. Dies entspricht dem Konzept der Versionierung. Die Funktionen f' und f'' sollen im selben Kontext anwendbar sein und das gleiche Resultat berechnen. Funktion f' ist dabei die ältere Funktion aufgrund des früheren Startzeitpunkts. Zu erkennen ist, dass Funktion f'' nicht Nachfolgerversion von f' im Sinne der Versionierung ist. Sie wird als neue eigenständige Funktion parallel im DBMS angelegt und versioniert. Diese Vorgehensweise resultiert aus der jeweiligen wissenschaftlichen Disziplin, in der ältere Funktionen, hier f', nicht zwangsläufig durch neuere Funktionen, hier f'', abgelöst werden. Dadurch können ggf. ältere Daten mit der neuen Funktion und neuere Daten mit der alten Funktion ausgewertet und verglichen werden.

Der Vorteil dieser Vorgehensweise ist, dass Überschneidungen zwischen älterer und aktuellerer Funktion zugelassen werden. Der Nachteil ist der, dass es keinen Zusammenhang zwischen den beiden Funktionen gibt. Das bedeutet, obwohl die aktuelle Funktion Nachfolgerfunktion der älteren ist, besitzt diese keinen Bezug zur älteren Funktion im Datenbankkontext, z. B. durch ein gleiches *id*-Attribut. Demnach handelt es sich von außen betrachtet um zwei beliebige, eigenständige, voneinander unabhängige Funktionen, die unterschiedliche Berechnungen durchführen und unterschiedliche Resultate liefern können. In dieser Arbeit wird dieser Fall weder im Konzept noch im Prototyp behandelt, trotzdem soll kurz skizziert werden, wie die Zusammengehörigkeit von Funktionen hergestellt werden kann. Durch die Einführung eines weiteren Attributs, welches die „verwandten" Funktionen gruppiert, kann eine Beziehung zwischen verschiedenen Versionen von Funktionen hergestellt werden. Somit würden z. B. alle Funktionen und deren Versionen zur Berechnung des Salzgehaltes in eine Gruppe gehören.

Da sich die verfügbaren DBMS in der Speicherung und Verwaltung der angelegten UDFs/STPs unterscheiden, wird an dieser Stelle nur generisch veranschaulicht, welche Schritte in welcher Reihenfolge ausgeführt werden müssen, um die Versionierung der Funktionen sicherzustellen. In Kapitel 5 wird dies prototypisch in PostgreSQL umgesetzt.

4.1.1.1 Anlegen einer neuen Funktion

Existiert noch keine Funktion zur Auswertung, kann diese im DBMS mit entsprechend bereitgestellten CREATE-Anweisungen angelegt werden (siehe z. B. im Prototyp-Kapitel Anweisung 5.1 in PostgreSQL-Syntax). Im nächsten Schritt werden diese neue Funktion und die dazugehörigen Informationen, wie z. B. Funktionsname, Typ und Anzahl der Parameter, Funktionskörper, mittels einer INSERT-Anweisung in die Versionierungstabelle eingefügt. Auch hier gilt wieder, das Aufsammeln der Informationen über gespeicherte SQL-Funktionen ist DBMS spezifisch und wird hier nicht gezeigt. Unter der Annahme, dass diese Funktion ab dem 01.01.1960 verwendet wurde und die restlichen Informationen zur Beschreibung dieser zur Verfügung stehen, entsteht die Tabelle 4.1. Es ist zur erkennen, dass zum Einfüge-Zeitpunkt die gerade angelegte Funktion gültig ist, da der aktuelle Zeitpunkt im Gültigkeitszeitintervall liegt.

4.1.1.2 Ersetzen einer bestehenden Funktion

Angenommen es soll die soeben angelegte Funktion durch eine neue ersetzt werden, welche eine andere Anzahl an Parametern oder eine geringfügig veränderte Berechnungsvorschrift aber den gleichen Namen besitzt.

Dazu ist es im ersten Schritt notwendig, dass die alte Funktion im DBMS mit entsprechend bereitgestellten spezifischen DB-Funktionen umbenannt wird. Der neue Name muss dabei eindeutig bezüglich des aktuellen Zeitpunkts, als auch bezüglich zukünftiger Zeitpunkte sein. Es wird hierzu der aktuelle Unix-Zeitstempel im Integer-Format an das Ende des alten Funktionsnamens in Verbindung mit einem Unterstrich angehängt. Somit können gleich lautende Funktionen ohne

Probleme versioniert werden. Der Unix-Zeitstempel simuliert dabei eine hochzählende Sequenz, ist also eindeutig, und steht in jedem DBMS zur Verfügung. Das jeweilige Erzeugen des Unix-Zeitstempels ist aber DBMS spezifisch.

Der zweite Schritt besteht darin, den neuen Namen der alten Funktion in die Versionierungstabelle einzutragen. Dafür wird die UPDATE-Anweisung 4.1 verwendet.

```
UPDATE routines_history
  SET routine_new_name = 'raw_to_units_press_1469545200'
WHERE id = 1
AND vt_end = '9999-12-31 23:59:59'
```

Anweisung 4.1: Aktualisierung des Namens während des Ersetzens einer bestehenden Funktion

Der nächste Schritt besteht im Anlegen der neuen Funktion im DBMS mittels CREATE-Anweisung (siehe Anweisung 5.2 im Prototyp-Kapitel). Die neue Funktion erhält dabei den Namen der alten Funktion ohne den zusätzlichen Zeitstempel.

Im vierten und letzten Schritt muss nach dem Anlegen die Versionierungstabelle aktualisiert werden. Dies geschieht im Gegensatz zum Anlegen einer neuen Funktion nicht mit einer INSERT-Anweisung sondern mit einer temporalen UPDATE-Anweisung 4.2. (Der Übersicht halber wurden auch dort die textuellen Werte gekürzt.)

```
UPDATE routines_history
  FOR PORTION OF BUSINESS_TIME FROM '1978-06-09 00:00:00' TO '9999-12-31 23:59:59'
  SET
    routine_name = 'raw_to_units_press',
    routine_new_name = '',
    routine_arguments = 'p_deployment_id integer, a0 numeric, a1 numeric, ...',
    routine_declaration = 'CREATE FUNCTION raw_to_u...'
WHERE id = 1
```

Anweisung 4.2: Temporale Aktualisierung einer bestehenden Funktion

Mittels der temporalen Aktualisierung, gegeben durch die temporale FOR PORTION OF-Klausel, wird die alte Funktion im Sinne der Gültigkeitszeit als ungültig markiert. Zusätzlich wird ein neuer Datensatz erzeugt, welcher den Namen und die neue Gültigkeitszeit der Nachfolgerfunktion enthält. Nach den beiden Operationen enthält die Versionierungstabelle 4.2 zwei Datensätze. Der erste Datensatz ist versioniert und enthält den ursprünglichen Funktionsnamen sowie den aktuellen Namen mit Zeitstempel, wie er im DBMS vorhanden ist. Im Gegensatz zum ersten Datensatz ist der zweite nun der gültige Datensatz. Anhand des *id*- und des *routine_name*-Attributs kann die Versionierungsgeschichte der Funktionen nachvollzogen werden.

4.1.1.3 Löschen einer bestehenden Funktion

Das Löschen entspricht dem Beenden der Gültigkeit dieser Funktion in der Versionierungstabelle. Dazu wird mittels der temporalen DELETE-Anweisung 4.3 der Endzeitpunkt auf einen beliebigen Zeitpunkt größer als der Startzeitpunkt der zu löschenden Funktion gesetzt. Üblicherweise wird dazu der aktuelle Zeitpunkt zur Ausführung der Anweisung genutzt.

```
DELETE FROM routines_history
  FOR PORTION OF BUSINESS_TIME FROM CURRENT_DATE TO '9999-12-31 23:59:59'
WHERE id = 1
```

Anweisung 4.3: Temporales Löschen einer bestehenden Funktion

Das DELETE-Statement ist so zu interpretieren, dass alles von „heute“ bis zum größtmöglichen Endzeitpunkt gelöscht wird. Somit wird der Endzeitpunkt auf den aktuellen Zeitpunkt der Ausführung gesetzt. In Tabelle 4.3 ist zu erkennen, dass der Endzeitpunkt des zweiten Datensatzes auf den Zeitpunkt der Ausführung, hier der 25.07.2016 um 15:30 Uhr, gesetzt wurde und somit ungültig im Sinne der Gültigkeitszeit ist. Es ist aber zu beachten, dass die eigentliche SQL-Funktion weiterhin im DBMS bestehen bleibt. Soll diese wirklich physisch gelöscht werden, muss die entsprechende DROP-Anweisung des DBMS aufgerufen werden. Da die Funktion in der Versionierungstabelle historisiert ist, kann diese jederzeit im DBMS durch in der Versionierungstabelle zusätzlich hinterlegten Informationen neu erzeugt werden.

id	routine_name	routine_new_name	routine_arguments	routine_declaration	vt_begin	vt_end
1	raw_to_u...		p_deployment_id integer, a0 nume...	CREATE FUNCTION raw_to_u...	1960-01-01 00:00:00	9999-12-31 23:59:59

Tabelle 4.1: Versionierung von UDFs/STPs: Anlegen einer neuen Funktion

id	routine_name	routine_new_name	routine_arguments	routine_declaration	vt_begin	vt_end
1	raw_to_u...	raw_to_u...s_1469545200	p_deployment_id integer, a0 nume...	CREATE FUNCTION raw_to_u...	1960-01-01 00:00:00	1978-06-09 00:00:00
1	raw_to_u...		p_deployment_id integer, a0 nume...	CREATE FUNCTION raw_to_u...	1978-06-09 00:00:00	9999-12-31 23:59:59

Tabelle 4.2: Versionierung von UDFs/STPs: Ersetzen einer bestehenden Funktion

id	routine_name	routine_new_name	routine_arguments	routine_declaration	vt_begin	vt_end
1	raw_to_u...	raw_to_u...s_1469545200	p_deployment_id integer, a0 nume...	CREATE FUNCTION raw_to_u...	1960-01-01 00:00:00	1978-06-09 00:00:00
1	raw_to_u...		p_deployment_id integer, a0 nume...	CREATE FUNCTION raw_to_u...	1978-06-09 00:00:00	2016-07-25 15:30:00

Tabelle 4.3: Versionierung von UDFs/STPs: Löschen einer bestehenden Funktion

4.1.2 Versionierung von Schemaänderungen

Um später anhand der Provenance feststellen zu können, wie welche Daten zu welchem Zeitpunkt mittels der zu dem Zeitpunkt gültigen Funktionen und Schemata entstanden sind, ist es notwendig, die Informationen über die Veränderungen eines Schemas aufzuzeichnen. Im Sinne des Forschungsdatenmanagements muss dafür gesorgt werden, dass eine lückenlose Dokumentation der Schemaänderungen vorhanden ist, um jederzeit jeden Zustand aus der Vergangenheit wieder herstellen zu können.

Es können folgende Schemaänderungen auftreten:

- Hinzufügen eines Attributs/einer Relation
- Entfernen eines Attributs/einer Relation
- Umbenennung eines Attributs/einer Relation
- Änderung des Datentyps eines Attributs
- Teilen eines Attributs/einer Relation in zwei oder mehrere einzelne Attribute/Relationen
- Zusammenführen zwei oder mehrerer Attribute/Relationen zu einem einzigen Attribut/einer einzigen Relation

Zwei Aspekte gilt es bei der Versionierung zu beachten: einerseits das Speichern der Umformungsschritte von einer Änderung zur nächsten, andererseits die Zeiträume, von wann bis wann welches Schema gültig war. Ein Beispiel sei das Teilen eines Attributs, z. B. *name*, in zwei weitere Attribute, z. B. *vorname* und *nachname*. Hierbei muss festgehalten werden, aus welchem Attribut die neuen Attribute entstanden sind und wie die ursprünglichen Daten in die zwei neuen Attribute transformiert wurden - z. B. durch das Teilen des vollen Namens anhand des Leerzeichens in Vor- und Nachname. Wurde diese Änderung ausgeführt und gespeichert, so ist eine neue Version eines Schemas entstanden. Dieser Zeitpunkt muss im Sinne der temporalen Datenhaltung festgehalten und mit der Änderungsverfolgung gespeichert werden.

Um unerwünschte Nebeneffekte auszuschließen, werden Integritäts- und Konsistenzbedingungen nicht betrachtet. Als Beispiel eines Nebeneffekts sei das Löschen eines Attributs aus der Eltern-Relation genannt, welches als Fremdschlüssel in der Kind-Tabelle einer Eltern-Kind-Beziehung benutzt wird. So können durch das Entfernen des Attributs mittels Einsatz strikter Integritätsbedingungen unabsichtlich Datensätze aus der Kind-Tabelle gelöscht werden, welches u. U. nicht der Intention des Anwenders bei der Änderung des Schemas war.

Im Folgenden werden zwei Möglichkeiten vorgestellt, wie Schemata versioniert werden können: die *Snapshot-* und die *Forschungsdatenmanagement-Variante*.

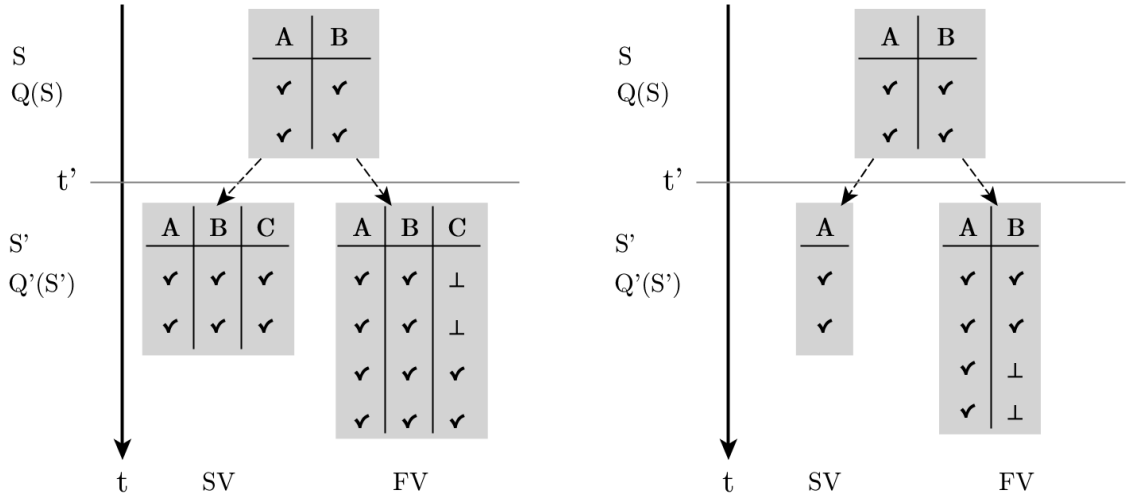
Die *Snapshot*-Variante erfasst die Schemaänderungen, welche durch DDL-Anweisungen[2] entste-

[2] Data Definition Language

hen, in einem Log-Protokoll. Zusätzlich werden Informationen über Gültigkeitszeiträume von Schemata in einer temporalen Relation gespeichert. Da die meisten DBMS ohnehin schon Mechanismen zur Protokollierung von Anweisungen bereitstellen, können diese Protokollierungen genutzt werden, ohne einen eigenen Log parallel pflegen zu müssen. Daneben muss weiterhin eine Relation oder eine Art „Blackbox“ bzw. ein Mechanismus existieren, der die temporale Versionierung vornimmt. So muss vor Ausführung einer DDL-Anweisung die Gültigkeit des aktuellen Schemas beendet werden und eine neue Version beginnen. Dabei entspräche der Zeitpunkt für das Ende der Gültigkeit sowie der Startzeitpunkt der neuen Version dem Ausführungszeitpunkt der DDL-Anweisung. Dadurch „gehört“ die Anweisung zur neuen Version und es ist möglich, die Entstehung der neuen Version mittels der DDL-Anweisung anhand des Startzeitpunkts nachzuvollziehen.

Eine Herausforderung, der an dieser Stelle begegnet werden muss, ist die Behandlung mehrerer DDL- und DML-Anweisungen. Dabei ändert eine Anweisung das Schema und mindestens eine weitere Anweisung transformiert die Daten. Da es nun mehrere Zeitpunkte von Änderungen gibt, muss eine andere Möglichkeit der Synchronisation bzw. Verknüpfung zwischen Log und temporaler Relation gefunden werden.

Bei der Snapshot-Variante werden die DDL-Anweisungen ausgeführt, d.h. es erfolgt eine physische Veränderung des Schemas. So wird z. B. ein Attribut physisch gelöscht und steht danach folglich nicht mehr zur Verfügung. Weiterhin bedeutet dies, dass Funktionen, die nach einer Schemaänderung angelegt worden sind, auch nur dasjenige Schema „sehen“, das während ihrer Anlage gültig im Sinne der Gültigkeitszeit war. Abbildung 4.2 soll dies verdeutlichen.



(a) Hinzufügen eines Attributs (b) Entfernen eines Attributs

Abbildung 4.2: Schemaversionierung: Unterschiede zwischen der *Snapshot*-Variante (SV) und der *Forschungsdatenmanagement-Variante* (FV)

In beiden Abbildungen 4.2a und 4.2b ist eine Schemaänderung an Zeitpunkt t' von S nach S' dargestellt. In Abbildung 4.2a wird ein Attribut C hinzugefügt, in Abbildung 4.2b das Attribut B entfernt. Daraus entstehen die Relationen SV und FV.

Der Nachfolgerfunktion Q'(S') stehen nach der Schemaänderung bei der *Snapshot*-Variante nur aktuelle und gültige Daten aus den Wertebereichen der Datentypen der Attribute A, B, C aus der Beispielrelation SV des Datenbankschemas S' zur Verfügung (symbolisiert durch das Häkchen). Das u. U. unabsichtliche Auftreten von NULL-Werten ($\perp$) und deren Verarbeitung innerhalb von Funktionen wird damit unterbunden (FV-Relation). Dadurch wird der Name der *Snapshot*-Variante erklärt: sie liefert einen zu einem Zeitpunkt gültigen Ausschnitt des Schemas. Diese Variante wird zukünftig als vorhanden angenommen und in dieser Arbeit benutzt.

Bei der *Forschungsdatenmanagement-Variante* werden Schemaänderungen im Kontext vom Forschungsdatenmanagement betrachtet. Bei der Anwendung von Methoden des Forschungsdatenmanagements wird davon ausgegangen, dass Daten - und in diesem Fall Informationen über Schemata, Relationen, Attribute - physisch nicht gelöscht werden dürfen. Das nicht physische Entfernen eines Attributes B ist in Abbildung 4.2b durch den Übergang der Ausgangsrelation in Schema S zur Relation FV in Schema S' dargestellt. Es wird deutlich, dass das Attribut nur als gelöscht gekennzeichnet wurde, aber weiterhin physisch vorhanden ist. Weiterhin ist zu erkennen, dass zukünftige Werte durch NULL-Werte ($\perp$) repräsentiert werden. Diese Änderungen in Form von Löschen durch Kennzeichnung müssen protokolliert und durch temporale Datenhaltung festgehalten werden.

Bei beiden vorgestellten Varianten der Versionierung von Schemata muss zudem sichergestellt werden, dass die UDFs/STPs, die z. B. auf ein verändertes Attribut zugegriffen haben, auch angepasst werden. Bei der Snapshot-Variante würden die Funktionen gar nicht mehr funktionieren und würden Fehler erzeugen. Bei der Forschungsdatenmanagement-Variante würden diese Funktionen weiterhin funktionieren, weil das Attribut physisch noch existiert. Die Werte des Attributs sind aber ab dem Löschzeitpunkt als NULL-Werte im Datenbankkontext zu betrachten und würden somit die Auswertung verfälschen bzw. unbrauchbar machen. Daher ist es bei dieser Vorgehensweise, welche das nicht physische Löschen von Attributen betrifft, notwendig, dass die Funktionen bei einer Schemaänderung angepasst werden, bzw. so modifiziert werden, dass diese nur auf vorhandenen Attributen im Sinne der Kennzeichnung der Löschung arbeiten.

Zusätzlich spielt bei beiden Varianten wieder der Aspekt der Synchronisierung der Zeitpunkte von Schemaänderung, Protokollierung der DDL/DML-Anweisungen im Log, sowie die temporale Versionierungstabelle der UDFs/STPs eine Rolle. Es ist zu klären, ob und wie zeitliche Änderungen zueinander gruppiert und als eine einzige Änderung aufgefasst werden können, die zu einer neuen Version eines Schemas führen.

Dieser Abschnitt sollte kurz skizzieren, welche Möglichkeiten bei der Versionierung von Schemata

bestehen und hat einige Probleme aufgezeigt, die zur Umsetzung der Versionierung gelöst werden müssen. In dieser Arbeit wird angenommen, dass die *Snapshot*-Methode zur Verfügung steht und die Informationen liefert, die für die folgenden Untersuchungen der Anwendbarkeit von Funktionen und der späteren Provenance gebraucht werden.

4.2 Untersuchung der Anwendbarkeit von Funktionen

In diesem Abschnitt werden Kriterien zur Anwendung von sich verändernden Funktionen auf sich verändernde Schemata untersucht. Dabei werden die Begriffe Funktionen und Abfragen synonym verwendet und wie in Abbildung 2.1 mit Q bezeichnet. Unter Schemaänderungen fallen Attribut- und Relationenänderungen, wie z. B. das Hinzufügen, das Löschen, das Umbenennen oder die Vereinigung bzw. Teilung von Attributen oder Relationen. Weiterhin wird immer nur eine Schemaänderung pro Zeitpunkt betrachtet. Das heißt, es handelt sich um zwei aufeinanderfolgende Schemaänderungen, wenn z. B. ein neues Attribut hinzukommt und ein anderes entfernt wird. Funktionsänderungen betreffen z. B. die Parameter sowie deren Typen, Änderung der Berechnungsvorschrift oder veränderte Rückgabewerte. Ferner gehört auch das Erzeugen einer komplett neuen Funktion dazu. Das Löschen einer Funktion wird nicht untersucht, weil nur bei einer physisch vorhandenen Funktion die Anwendbarkeit dieser untersucht werden kann. Darüber hinaus nutzt eine Anfrage Q nur eine Teilmenge der zur Verfügung stehenden Relationenschemata R aus dem Datenbankschema $S = \{R_1, R_2, ..., R_n\}$

$$\begin{aligned} Q_1(S_1) \quad &| \quad S_1 \subseteq S, \\ Q_2(S_2) \quad &| \quad S_2 \subseteq S, \\ &... \\ Q_n(S_n) \quad &| \quad S_n \subseteq S, \end{aligned}$$

Das bedeutet, dass die in dem jeweiligen Fall betrachteten Schemaänderungen nur diejenige Teilmenge an Schemata betrifft, die von der Funktion benutzt wird. Darüber hinaus werden nur lesende Funktionen bzw. Anfragen auf der Ebene der Originaldaten behandelt. Das bedeutet, dass die Zwischenergebnisse, die z. B. durch Aggregationen innerhalb der Auswerte- und Analysemethoden entstanden sind, nicht betrachtet und somit nicht dauerhaft im DBMS abgespeichert werden.

Darüber hinaus wird eine Begrifflichkeit eingeführt: die Richtung der Versionierung. Diese Richtung ist durch die temporale Datenhaltung und damit durch den Ablauf der Zeit vorgegeben. Wird in diesem Abschnitt von der Betrachtung der Richtung der Versionierung gesprochen, so ist damit einerseits die Vorwärtsrichtung als auch die Richtung rückwärts gemeint. In welche Richtung der jeweilige Fall gerade untersucht wird, geht aus dem Kontext hervor oder wird explizit erwähnt.

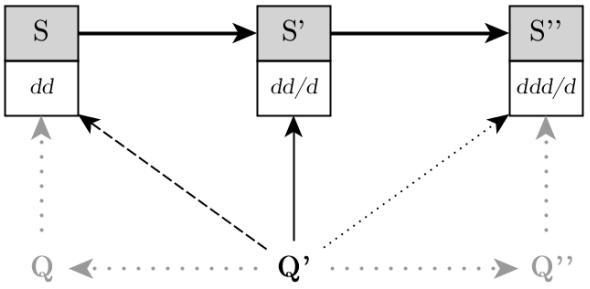


Abbildung 4.3: Perspektiven von Schema- (S) und Anfrageänderungen (Q)

Für folgende Ausführungen wird Abbildung 4.3 (abgeleitet aus Abbildung 2.1) zur Verdeutlichung genutzt. Bei der Untersuchung der Veränderungen der Funktionen und Schemata ergeben sich zwei Perspektiven: aus Sicht einer Funktion kann sich das Schema ändern, aus Sicht eines Schemas kann sich die Funktion ändern. Es soll dazu zunächst aus Sicht von Funktion Q' eine beispielhafte Schemaänderungen betrachtet werden. Funktion Q' stellt zusammen mit dem dazugehörigen Schema S' einen Zustand aus der Versionierung zu einem vergangenen Zeitpunkt dar. Unter der Annahme, dass beispielhaft stetig ein Attribut pro Schemaänderung von S über S' nach S" dazugekommen ist, hat sich aus Sicht von Q' sowohl das in der Vergangenheit liegende Schema S als auch das neue Schema S" verändert. In Bezug zu Schema S stellt sich die Veränderung als Entfernen eines Attributs, in Bezug zu Schema S" als Hinzufügen eines Attributs dar. Beim umgekehrten Fall, also bei stetigem Entfernen eines Attributs pro Schemaänderung, gilt dasselbe nur umgekehrt. Aus diesem Beispiel geht hervor, dass die Umkehrfunktion des Hinzufügens eines Attributs das Löschen desselben ist und umgekehrt. Es existieren für die in dieser Arbeit behandelten Schemaänderungen eindeutige Umkehrfunktionen (siehe Tabelle 4.4). So ist z. B. die Umkehrung („rückgängig machen“) der Teilung einer Relation, die Zusammenführung der aus der Teilung entstandenen Relationen. Als weiteres Beispiel sei hier noch die Umbenennung erwähnt, deren Umkehrfunktion die Umbenennung selbst ist. Aus dieser Symmetrie folgt, dass aus Sicht von Funktionen einige Fälle der Schemaänderungen ohne Rücksicht auf die Richtung der Versionierung betrachtet werden können. Ob die Richtung in dem jeweiligen zu untersuchenden Fall eine Rolle spielt, wird explizit genannt.

Analog kann die Perspektive von Schemata auf Funktionsänderungen behandelt werden. Beispielhaft soll stetig ein neuer Parameter von Q über Q' nach Q" hinzugefügt worden sein. Vom Standpunkt von S und S" betrachtet, hat sich die Funktion Q' verändert. Dies stellt aus Sicht von S" auf Q' aber das Entfernen eines Parameters dar. Auch hier gilt der umgekehrte Fall, nämlich dass bei stetigem Entfernen eines Parameters, dies aus Sicht von S" auf Q' als Hinzufügen eines Parameters gewertet werden würde. Unter Ausnutzung dieser Symmetrie ist auch hier die Richtung im Sinne der Versionierung, aus der Schemata eine Funktionsänderung betrachten, ohne Relevanz.

4.2.1 Anwendbarkeit von Funktionen bei Schemaänderungen

Es folgen die Beschreibungen und Untersuchungen der Anwendungsvoraussetzung aus der Liste in Abschnitt 4.1.2. Dabei können einige Fälle gemeinsam behandelt werden, da diese die gleichen Anwendungsvoraussetzungen besitzen. Zu jedem Fall werden die Anwendungsvoraussetzungen angegeben, die beschreiben, welche Kriterien erfüllt sein müssen, um die Anwendbarkeit der Funktion zu gewährleisten. Weiterhin werden mögliche Aktionen angegeben, welche unter Berücksichtigung der Bedingungen angewendet werden können, um die Anwendbarkeit der Funktionen sicherzustellen. Wie diese Aktionen konkret aussehen können und welche Techniken dabei zur Verfügung stehen, wird in Abschnitt 4.2.3 aufgezeigt. Zusätzlich wird ein Resultat angegeben, welches unter Betrachtung der Anwendungsvoraussetzung und der Aktion die Auswirkungen bzw. Folgen angibt. Tabelle 4.4 fasst die Untersuchungen noch einmal kompakt und übersichtlich zusammen.

4.2.1.1 Hinzufügen eines neuen Attributs & Hinzufügen einer neuen Relation

Das Hinzufügen eines Attributs zu einer Relation oder das Hinzufügen einer neuen Relation zur Menge der im Datenbankschema vorhandenen Relationen beeinflusst das Verhalten einer existierenden Funktion nicht. Alle bestehenden Informationen bleiben erhalten und es findet eine Erhöhung der Informationskapazität statt. Dieser Fall ist unabhängig von der Betrachtung der Richtung der Versionierung.

Als Beispiel sei hier der Übergang von Schema S' nach S" in Abbildung 4.3 betrachtet. Funktion Q' kann weiterhin auf das neue Schema S" ohne Einschränkungen angewendet werden. Ferner kann das neue Attribut oder die neue Relation als Bestandteil der Verarbeitung einer neuen Funktion Q", die nicht zwingend aus der alten Funktion Q' als Folge der Versionierung entstehen muss, verwendet werden. Das Auftreten von NULL-Werten im neuen Attribut wird durch Verwendung der Snapshot-Variante zur Versionierung von Schemata aus Abschnitt 4.1.2 unterbunden. Die neue Funktion Q" sieht somit nur aktuelle gültige Daten aus dem Wertebereich des Datentyps des neu eingefügten Attributs der Relation im neuen Schema S".

Anwendungsvoraussetzung: keine
Aktion: keine; ggf. Erzeugung einer Nachfolgerfunktion mittels Versionierung zur Nutzung des neuen Attributs/der neuen Relation.
Resultat: Versionierte und neue Funktionen können uneingeschränkt (weiter) verwendet werden.

4.2.1.2 Entfernen eines Attributs & Entfernen einer Relation

Durch das Entfernen bzw. Löschen von Attributen oder ganzen Relationen erfolgt eine Verringerung der Informationskapazität. Dies bedeutet nicht zwangsläufig einen Informationsverlust. In einigen Fällen sind Daten redundant gespeichert oder können aus anderen weiterhin vorhandenen Informationen errechnet werden. Dadurch ist es möglich, dass, durch Kompensation

	Schemaänderungen					
	Hinzufügen Attribut/Relation	**Entfernen Attribut/Relation**	**Umbenennung Attribut/Relation**	**Änderung des Datentyps eines Attributs**	**Teilen Attribut/Relation**	**Vereinigung Attribute/Relationen**
Anwendungs-voraussetzung	keine	Kompensation des Informationsverlusts durch in anderen Attributen/Relationen gespeicherte Informationen oder kontextbasiert möglich.	ggf. Kompensation durch mögliche Anwendung der Umkehroperation.	Kompatible Datentypen oder verlustfreie explizite Typumwandlung zwischen verschiedenen Datentypen möglich.	Teilung von Attributen: Kompatible Datentypen oder verlustfreie explizite Typumwandlung zwischen verschiedenen Datentypen möglich. Teilung von Relationen: Verbundtreue muss gegeben sein.	Vereinigung von Attributen: Kompatible Datentypen oder verlustfreie explizite Typumwandlung zwischen verschiedenen Datentypen möglich. Vereinigung von Relationen: natürlicher Verbund durch entsprechende Verbundattribute
Aktion	keine; ggf. Erzeugung einer Nachfolgerfunktion mittels Versionierung zur Nutzung des neuen Attributs/der neuen Relation.	Wenn Kompensation möglich: Anpassung und Nutzung der Funktion (Versionierung) bzw. kontextbasierter Aufruf der Vorgängerfunktion. Wenn keine Kompensation möglich: Löschen durch Kennzeichnung mittels Beenden der Gültigkeitszeit der Funktion.	Erstellen einer neuen Funktion, die entweder den neuen Namen oder die Umkehroperation, welche in diesem Fall die Umbenennung selbst ist, benutzt.	Wenn kompatible Datentypen, dann implizite Typumwandlung durch DBMS, andernfalls explizite Typumwandlung. Keine Typumwandlung möglich: Löschen durch Kennzeichnung mittels Beenden der Gültigkeitszeit der Funktion.	Erstellen einer neuen Funktion, die entweder die neuen Attribute bzw. Relationen oder die Umkehroperation benutzt, welche in diesem Fall die Vereinigung der Attributen bzw. der Relationen ist.	Erstellen einer neuen Funktion, die entweder die neuen Attribute bzw. Relationen oder die Umkehroperation benutzt, welche in diesem Fall die Teilung der Attributen bzw. der Relationen ist.
Resultat	Versionierte und neue Funktionen können uneingeschränkt (weiter) verwendet werden.	Wenn Kompensation möglich: Anwendung und Nutzung der neuen Version der Funktion bzw. Verwendung durch Kontext gegeben. Wenn keine Kompensation möglich: Keine Anwendbarkeit der Funktion gegeben.	Nutzung der neuen Version der Funktion.	Weiternutzung der alten Funktion bei impliziter Typumwandlung, andernfalls Nutzen der neuen Version der Funktion mit expliziter Typumwandlung. Keine Typumwandlung möglich: Keine Anwendbarkeit der Funktion gegeben.	Neue Versionen der Funktionen können uneingeschränkt angewendet werden.	Neue Versionen der Funktionen können uneingeschränkt angewendet werden.

Abbildung 4.4: Zusammenfassung der Anwendbarkeit von Funktionen bei Schemaänderungen

mittels anderer Attribute oder Relationen oder deren Kombination, die Funktion weiterhin auf das ursprüngliche Schema angewendet werden kann. Bei diesem Fall werden die Richtungen der Versionierung separat betrachtet, weil trotz gleicher Kriterien unterschiedliche Resultate in Erscheinung treten.

Zunächst wird die allgemeine (Vorwärts-)Richtung der Versionierung, konkret der Übergang von Schema S' nach S", betrachtet. Die Fragestellung lautet, unter welchen Kriterien kann Q' auf S" unter Berücksichtigung des Löschens eines Attributs oder einer Relation angewendet werden. Liegt ein Informationsverlust derart vor, dass die wegfallenden Informationen nicht durch andere Daten berechnet bzw. durch geeignete Abfragen kompensiert werden können, ist Q' nicht anpassbar bzw. nicht versionierbar. Daraus folgt das Löschen von Q' durch Beenden der Gültigkeitszeit. Ist der Informationsverlust dagegen durch die im ersten Absatz genannten Möglichkeiten kompensierbar, kann Q' angepasst werden, was der Anwendung des Konzepts der Versionierung entspricht. Es entsteht Q", welche die Nachfolgerfunktion von Q' ist und auf das Schema S" angewendet werden kann.

Als nächstes folgt die Untersuchung der Gegenrichtung, also der Übergang von Schema S' nach S unter Wegfall eines Attributs oder einer Relation. Es existieren hier zwei Varianten, die betrachtet werden müssen:

1. die Existenz einer Vorgängerfunktion Q
2. die Existenz einer semantisch gleichwertigen aber syntaktisch unterschiedlichen Funktion F

Punkt 1 spiegelt die Versionierung von Q nach Q' wider. Wird die Vorwärtsrichtung von S nach S' betrachtet, so ist zu erkennen, dass ein Attribut bzw. eine Relation hinzugekommen ist. Unter Anwendung der Untersuchungen des Hinzufügens von Attributen bzw. Relationen aus Abschnitt 4.2.1.1 ist die Anwendbarkeit von Q auf S' ohne Einschränkungen gegeben. Trotzdem existiert eine Nachfolgerfunktion Q', welche nur auf S' aber nicht auf S anwendbar ist. Wie bei der Betrachtung der Vorwärtsrichtung der Versionierung muss das Löschen des Attributs bzw. der Relation durch bestehende Daten kompensierbar sein, um eine Anwendbarkeit der Funktion Q' auf S zu ermöglichen. Ist die Kompensation nicht möglich, kann die Funktion nicht auf das ältere Schema angewendet werden. Im Gegensatz zur Vorwärtsrichtung wird die Funktion aber nicht im Sinne der Gültigkeitszeit beendet, sondern bleibt bestehen. Ist die Kompensierbarkeit gegeben, muss die Funktion wieder angepasst werden. Dazu wird eine neue Version von Q' erzeugt. Dies ist zu jedem Zeitpunkt möglich, unabhängig, ob eine Schemaänderung vorliegt oder nicht. Diese Funktion sei (Q')' genannt und wird parallel als neue eigenständige Funktion angelegt (vgl. dazu Abschnitt 4.1). So ist es möglich, dass Q' weiterhin auf S' angewendet werden kann. Die neue Funktion (Q')' wird dabei nur auf Anwendung auf Schema S hin konzipiert und kann nicht mehr auf S' angewendet werden. Abbildung 4.5 soll die Ausführungen veranschaulichen. Dort bedeutet der durchgezogene Pfeil von (Q')' nach S, dass die Funktion (Q')' wie oben erläutert nur auf S

angewendet werden kann. Der gepunktete, gestrichelte graue Pfeil von (Q')' nach Q soll andeuten, dass die Funktion (Q')' auch die Vorgängerfunktion Q selbst verwenden kann (wenn diese existiert), um auf S arbeiten zu können. Darüber hinaus hat die Verwendung der Funktion (Q')' den Vorteil, dass bei einer Schemaänderung, aber keiner oder nur einer teilweisen anschließenden Transformation der Daten, die alte Funktion Q aber auch die neue Funktion Q' verwendet werden kann, um die Auswertung auf den Daten zu gewährleisten. Dabei kommt es darauf an, ob und wie viele Daten transformatiert wurden und wie diese Daten in den Schemata S und S' verteilt sind. Dies ist wichtig für die Wahl der Funktion, also Q bzw. Q', zur Auswertung der Daten. Diese Entscheidung, welche Funktion auf welchen Teildaten in ihrem jeweiligen Schema arbeitet, muss bei der Implementierung der Funktion (Q')' berücksichtig werden. Die Technik des Umschließens (engl. *to wrap*) und damit das Nutzen von bereits bestehenden Funktionen wird in Abschnitt 4.2.3 erläutert.

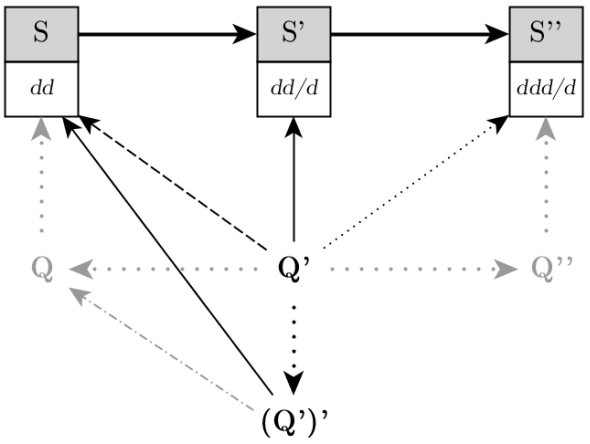


Abbildung 4.5: Erzeugung einer neuen parallelen Version einer Funktion zur Anwendung auf ein veraltetes Schema

Punkt 2 entspricht dem Aspekt aus Abschnitt 4.1, nämlich dass zwei syntaktisch unterschiedliche Funktionen das gleiche Resultat berechnen und parallel existieren. Als Praxisbeispiel sei hier die Berechnung des Salzgehaltes mittels zwei unterschiedlicher physikalischer Größen genannt. Einerseits kann der Salzgehalt mittels der elektrischen Leitfähigkeit, andererseits mittels des optischen Brechungsindexes berechnet werden. Diese beiden Größen entsprechen im Datenbankkontext den Attributen. Wurde im Schema S noch die elektrische Leitfähigkeit abgespeichert und in Funktion F zur Berechnung des Salzgehaltes benutzt, so wird in Schema S' der neu hinzugekommene Brechungsindex in der Funktion Q' zur Berechnung genutzt. Zu erkennen ist, dass die Anwendung von Q' auf Schema S nicht gegeben ist, da der Wegfall des Brechungsindexes nicht kompensierbar ist. Aus dem Kontext heraus ist dem Anwender aber bekannt, dass zu dem Zeitpunkt von Schema S die Funktion F existiert, die wie Funktion Q' dasselbe berechnet. So kann eine kontextbasierte Kompensation stattfinden, nämlich derart, dass anstelle von Q' die parallel existierende Funktion

F auf Schema S aufgerufen wird. Bei der wissenschaftlichen Auswertung kann somit ein Teil mittels F, der andere Teil der Auswertung mittels Q berechnet und später zusammengeführt werden. Dies ist sogar automatisch anhand der Zeitpunkte der Schemaänderung möglich: Funktion F wird bis zum Endzeitpunkt von Schema S bzw. Startzeitpunkt von Schema S' eingesetzt, Funktion Q' ab dem Startzeitpunkt von S'. Konkret kann dies durch eine Vereinigung der Aufrufe von F und Q' erfolgen: F ∪ Q'. Die Funktion F kann darüber hinaus gemäß der Untersuchung des Hinzufügens von Attributen bzw. Relationen aus Abschnitt 4.2.1.1 weiterhin auf S' angewendet werden. Ist aber auch keine kontextbasierte Kompensation möglich, so ist die Funktion Q' auf S nicht anwendbar.

Anwendungsvoraussetzung: Kompensation des Informationsverlusts durch in anderen Attributen/Relationen gespeicherte Informationen oder kontextbasiert möglich.

Aktion & Resultat in Richtung der Versionierung:

a) Wenn Kompensation möglich: Anpassung und Nutzung der Funktion (Versionierung).

b) Wenn keine Kompensation möglich: Löschen durch Kennzeichnung mittels Beenden der Gültigkeitszeit der Funktion. Die Funktion steht danach nicht mehr zur Verfügung.

Aktion & Resultat entgegen der Richtung der Versionierung:

a) Wenn Kompensation möglich: Anpassung und Nutzung der Funktion oder kontextbasierter Aufruf der Vorgängerfunktion.

b) Wenn keine Kompensation möglich: Keine Anwendbarkeit der Funktion gegeben.

4.2.1.3 Umbenennung eines Attributs & Umbenennung einer Relation

Wird ein Attribut oder eine Relation umbenannt, beeinflusst das weder die Informationskapazität noch die Informationserhaltung. Unter Betrachtung in Richtung der Versionierung, also z. B. von S' nach S", kann Funktion Q' nur unter Anpassungen auf S" angewendet werden. Dies entspricht dem Konzept der Versionierung und es entsteht die Nachfolgerfunktion Q". Die Gegenrichtung, also von S' nach S, verhält sich ähnlich: analog zur Abbildung 4.5 in Abschnitt 4.2.1.2 kann eine neue Version, z. B. (Q')' erzeugt werden, die einzig und allein zu dem Zweck entworfen wird, anwendbar auf S zu sein. Diese Funktion wird auch hier parallel zur Ursprungsfunktion Q' als eigenständige Funktion angelegt.

Anwendungsvoraussetzung: Ggf. Kompensation durch mögliche Anwendung der Umkehroperation.

Aktion: Erstellen einer neuen Funktion, die entweder den neuen Namen oder die Umkehroperation, welche in diesem Fall die Umbenennung selbst ist, benutzt.

Resultat: Nutzung der neuen Version der Funktion.

4.2.1.4 Änderung des Datentyps eines Attributs

In diesem Fall ist die Betrachtung der Richtung der Versionierung ohne Relevanz. Als Datentypen werden an dieser Stelle vom DBMS unterstütze Domänen bzw. Wertebereiche der Attribute be-

zeichnet. Dabei kann die Änderung des Datentyps den Wertebereich erweitern oder einschränken. Wird der Wertebereich erweitert, so hat dies keine Auswirkungen auf die Erhaltung der Informationen. Die Informationskapazität wird dagegen erhöht. Wird der Wertebereich stattdessen verkleinert, verringert sich die Informationskapazität und Informationsverluste können auftreten. Als Beispiel sei hier eine Änderung von einem VARCHAR(50) auf einen VARCHAR(25) Datentyp genannt. Die Verringerung der Informationskapazität ist an den Längenangaben der zu speichernden Zeichenkette zu erkennen. Weiterhin kann mit dem Beispiel ein Informationsverlust eingehergehen: alle Werte größer 25 würden auf die Länge von 25 Zeichen gekürzt werden. Sind aber alle bestehenden Werte kleiner oder gleich der Länge von 25 Zeichen, so würde kein Informationsverlust auftreten. Schlussendlich bedeutet die Änderung des Datentyps weder zwingend die Veränderung der Informationskapazität noch einen Informationsverlust. Als Beispiel sei hier die Änderung von CHAR(20) nach VARCHAR(20) genannt.

Es sei hier der Übergang von Schema S' nach S" betrachtet. Die Funktion Q' ist nur dann ohne Einschränkungen auf S" anwendbar, wenn diese nur Datentypen verwendet, die zu dem veränderten Datentyp kompatibel sind. Kompatibel bedeutet an dieser Stelle, dass das DBMS eine implizite Typumwandlung, z. B. von CHAR nach VARCHAR, automatisch vornehmen kann. Andernfalls muss die Typumwandlung explizit stattfinden. Die meisten DBMS unterstützen mittels bereitgestellter Funktionen explizite Typumwandlungen für einige aber nicht alle Basisdatentypen. Basisdatentypen seien hier nummerische Datentypen wie INT oder FLOAT, Zeichenketten wie CHAR oder VARCHAR, Bitfolgen, boolesche oder temporale Datentypen, wie DATE oder TIME [SSH13]. Ist keine Typumwandlung mittels vom DBMS bereitgestellter Funktionen möglich, so kann versucht werden, eine eigene Funktion für die Typumwandlung zu erstellen. Schlägt auch dies fehl, ist die weitere Anwendbarkeit der ursprünglich betrachteten Funktion Q' nicht gegeben. Unter Betrachtung der Gegenrichtung, also von S' nach S, gelten dieselben Kriterien. Analog zur Abbildung 4.5 in Abschnitt 4.2.1.2 ist es auch hier möglich, eine eigene Funktion, z. B. (Q')', zu schreiben, die nur dazu dient, auf Schema S anwendbar zu sein.

Anwendungsvoraussetzung: Kompatible Datentypen oder verlustfreie explizite Typumwandlung zwischen verschiedenen Datentypen möglich.

Aktion: Wenn Kompatible Datentypen, dann implizite Typumwandlung durch DBMS, andernfalls explizite Typumwandlung.

Resultat: Weiternutzung der alten Funktion bei impliziter Typumwandlung, andernfalls Nutzen der neuen Version der Funktion mit expliziter Typumwandlung. Keine Typumwandlung möglich: Keine Anwendbarkeit der Funktion gegeben.

4.2.1.5 Teilen eines Attributs/einer Relation in zwei oder mehrere einzelne Attribute/Relationen

Eine Teilung eines Attributs oder einer Relation verletzt die Annahme aus Abschnitt 4.2, dass nur eine Schemaänderung pro Zeiteinheit betrachtet wird. Um z. B. die Teilung eines Attributs vorzunehmen, müssen zwangsläufig mehrere Änderungsschritte ausgeführt werden. Es können hier neue Attribute hinzukommen, alte Attribute umbenannt oder nach einer erfolgreichen Trans-

formation der Werte die alten Attribute gelöscht werden. Zusätzlich können Änderungen des Datentyps der Attribute auftreten. Trotzdem soll an dieser Stelle der Fall untersucht werden, ob und wann eine Funktion dann noch auf das veränderte Schema angewendet werden kann.

Als erstes wird die Teilung eines Attributs betrachtet. Die Informationskapazität kann einerseits verringert, andererseits vergrößert werden. Dabei kommt es darauf an, ob die neuen Attribute zusammengezählt eine geringere oder höhere Kapazität besitzen als das ursprüngliche Attribut. Weiterhin können durch die Teilung Informationen verloren gehen. Dies kann entweder durch die Änderung des Datentyps und dessen Kapazität (vgl. dazu Abschnitt 4.2.1.4) auftreten oder durch eine fehlerhafte Überführung der alten Daten in die neue Struktur.

Als Beispiel sei hier ein Teil einer Adresse, die im Attribut *plz_ ort* gespeichert wird, betrachtet. Es soll zunächst der Übergang von S' nach S" betrachtet werden. Konkret wird das Attribut *plz_ ort*, welches als Datentyp VARCHAR(100) besitzt, in die Attribute *plz* und *ort* aufgeteilt werden. Dabei kommt es bei dem *plz*-Attribut zu einer Datentypänderung nach INT. Das *ort*-Attribut erhält den Datentyp VARCHAR(50). Es tritt in diesem Fall also eine Verringerung der Informationskapazität auf[3]. Um die Funktion Q' auf das neue Schema S" anwenden zu können, muss diese mittels Versionierung angepasst werden. Dazu muss die Datentypänderung, wie in Abschnitt 4.2.1.4 gezeigt, behandelt werden. Es entsteht eine neue Funktion Q", welche auf das neue Schema ohne Einschränkungen anwendbar ist. Unter Betrachtung der Gegenrichtung, also von S' nach S muss, analog zur Abbildung 4.5 in Abschnitt 4.2.1.2 eine eigenständige Funktion, z. B. (Q')', erstellt werden, die nur auf dem Schema S anwendbar ist.

Bei der Teilung einer Relation erhöht sich die Informationskapazität durch das redundante Speichern des Attributs für den natürlichen Verbund. Entgegen der Teilung eines Attributs wird hier angenommen, dass keine Informationen verloren gehen. Das bedeutet, dass die Datentypen der Attribute aus den einzelnen entstandenen (Basis-)Relationen denjenigen aus der Originalrelation entsprechen. Wie angedeutet muss zur Anwendung einer Funktion nach dieser Schemaänderung ein natürlicher Verbund aus den entstandenen Basisrelationen möglich sein, der die Originalrelation rekonstruiert. Dieses Kriterium heißt Verbundtreue [SSH13]. Unter Betrachtung der Vorwärtsrichtung der Versionierung, also von S' nach S", bedeutet das, dass eine neue Funktion Q" entstehen muss, die auf den entstandenen neuen Basisrelationen arbeiten kann. Für die Gegenrichtung, also von S' nach S, heißt das, dass analog wie bei den Attributen eine neue eigenständige Funktion, z. B. (Q')', erzeugt werden muss, die dann wieder nur auf Schema S anwendbar ist.

Anwendungsvoraussetzung:

a) Teilung von Attributen: Kompatible Datentypen oder verlustfreie explizite Typumwandlung zwischen verschiedenen Datentypen möglich.

[3] Die neuen Datentypen VARCHAR(50) und INT und der alte Datentyp VARCHAR(100) mögen auf den ersten Blick nicht vergleichbar aussehen, aber bei einer angenommenen Speicherung eines Integers mittels 4 Byte, kann dieser maximal 10 Ziffern besitzen: 10 Ziffern + 50 Zeichen = 60 effektive Zeichen gegenüber den ursprünglich 100 Zeichen

b) Teilung von Relationen: Verbundtreue muss gegeben sein.

Aktion: Erstellen einer neuen Funktion, die entweder die neuen Attribute bzw. Relationen oder die Umkehroperation benutzt, welche in diesem Fall die Vereinigung der Attributen bzw. der Relationen ist.

Resultat: Neue Versionen der Funktionen können uneingeschränkt angewendet werden.

4.2.1.6 Vereinigung von zwei oder mehreren Attributen/Relationen zu einem einzigen Attribut/einer einzigen Relation

Dieser Fall entspricht der Umkehrung des vorherigen Falles aus Abschnitt 4.2.1.5 und verletzt ebenso die Annahme der atomaren Schemaänderung pro Zeitpunkt aus Abschnitt 4.2. Unter der Betrachtung der Zusammenführung von Attributen zu einem neuen Attribut kann auch hier eine Verringerung oder Erhöhung der Informationskapazität auftreten. Dies geschieht genau dann, wenn das neue Attribut eine geringere bzw. eine höhere Kapazität als die Summe der beteiligten Attribute besitzt. Ein Informationsverlust kann auftreten, wenn die Informationskapazität verringert wurde und die Werte während der Transformation in das neue Attribut gekürzt werden. Bezogen auf das Beispiel aus dem vorherigen Abschnitt 4.2.1.5 werden die Attribute *plz* (INT) und *ort* (VARCHAR(50) in das neue Attribut *plz_ ort* (VARCHAR(100)) zusammengeführt. Dabei kommt es im Gegensatz zum vorherigen Fall aus Abschnitt 4.2.1.5 zu einer Erhöhung der Informationskapazität und demnach bei einer erfolgreichen Transformation der Daten zu keinem Informationsverlust. Auch hier muss aber gewährleistet werden, dass die Datentypen zueinander kompatibel, d.h. ineinander umwandelbar, sind (siehe Abschnitt 4.2.1.4).

Als Vereinigung von Relationen wird wird an dieser Stelle angenommen, dass zwei oder mehrere Relationen unter der Bedingung der Verbundtreue zu einer neuen Relation verbunden werden. Dazu müssen Attribute für den natürlichen Verbund existieren. Diese Attribute treten in der neuen Zielrelation nur noch einmalig auf, da die Vereinigung anhand dieser Attribute stattgefunden hat. Daraus folgt, dass es in diesem Fall zu einer Verringerung der Informationskapazität kommt, da die zuvor auf mehrere Relationen verteilten Attribute nach der Vereinigung nur noch in der entstandenen Verbundrelation vorkommen. Die Vereinigung von Relationen ohne Verbundattribute in Form von (Fremd-)Schlüsseln wird aus den Untersuchungen ausgeschlossen.

In beiden betrachteten Aspekten, also der Vereinigung von Attributen und der Vereinigung von Relationen, gelten dieselben Anwendbarkeitskriterien. Bezogen auf Abbildung 4.5 wird der Übergang von Schema S' nach S" betrachtet. Die Funktion Q' ist wieder nur durch Anpassungen mittels Versionierung auf S" anwendbar. Dazu sind etwaige auftretende Datentypänderungen bei der Vereinigung von Attributen gemäß Abschnitt 4.2.1.4 zu behandeln. Dasselbe gilt wieder für die Gegenrichtung der Versionierung, also von S' nach S. Es entsteht wieder eine neue eigenständige Funktion (Q')' die nur auf S ohne Einschränkungen anwendbar ist.

Anwendungsvoraussetzung:

a) Vereinigung von Attributen: Kompatible Datentypen oder verlustfreie explizite Typumwandlung zwischen verschiedenen Datentypen möglich.

b) Vereinigung von Relationen: natürlicher Verbund durch entsprechende Verbundattribute.

Aktion: Erstellen einer neuen Funktion, die entweder die neuen Attribute bzw. Relationen oder die Umkehroperation benutzt, welche in diesem Fall die Teilung der Attribute bzw. der Relationen ist.

Resultat: Neue Versionen der Funktionen können uneingeschränkt angewendet werden.

4.2.2 Anwendbarkeit von Funktionen bei Funktionsänderungen

In diesem Abschnitt erfolgt die Untersuchung der Anwendbarkeit von veränderten Funktionen auf Schemata. Zunächst soll beschrieben werden, wann und unter welchen Umständen Funktionsänderungen auftreten. Dabei existieren drei Fälle:

1. Bedingt durch Schemaänderungen
2. Korrekturen von Funktionen
3. Neuanlage von Funktionen

Abbildung 4.6 soll diese verdeutlichen. Der erste Punkt entspricht den in Abschnitt 4.2.1 behandelten Schemaänderungen. Die dort ermittelten Bedingungen beschreiben, wann eine Funktion geändert werden muss, um auf ein verändertes Schema anwendbar zu sein.

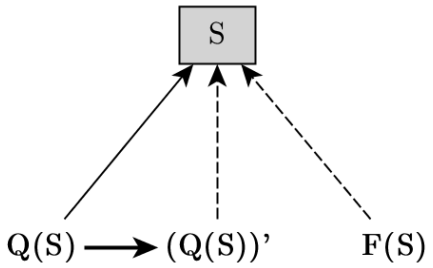


Abbildung 4.6: Visualisierung von Funktionsänderungen bezüglich eines Schemas

Der zweite Punkt entspricht der Anpassung einer Funktion bei einem gleichbleibenden Schema. In Abbildung 4.6 ist dies durch die beiden Funktionen Q(S) und (Q(S))', die beide immer noch auf demselben Schema S arbeiten, dargestellt. Dies entspricht der Versionierung von Funktionen unabhängig von Schemaänderungen. Dabei ist es möglich, dass die versionierte Funktion nicht mehr auf das Schema S anwendbar ist. Dies rührt aus dem Kontext der Anwender, im

speziellen der Wissenschaftler, her. Es kann vorkommen, dass eine Funktion aufgrund neuer Forschungserkenntnisse angepasst wird, diese dann aber nicht mehr auf den ursprünglichen Daten bzw. deren Format, im Datenbankkontext das Schema, anwendbar ist. Das bedeutet wiederum, dass eine Funktionsänderung auch eine Schemaänderung nach sich ziehen kann. Funktionsänderungen in Form von Korrekturen entsprechen Änderungen von Parametern, Änderungen in der Berechnungsvorschrift bzw. im Funktionsrumpf sowie Änderungen bei den Rückgabewerten. Diese Korrekturen werden in den folgenden Fällen untersucht. Die Neuanlage einer Funktion (Punkt 3 in obiger Liste) wird als eigenständige Änderung behandelt und ist als Funktion F in Abbildung 4.6 symbolisiert.

Analog wie bei den Schemaänderungen werden nur lesende Funktionen angenommen, deren Rückgaben oder Ergebnisse nicht dauerhaft im DBMS gespeichert werden. Des Weiteren werden nur Basisdatentypen bei den Änderungen von Parametern und Rückgabewerten betrachtet. Das bedeutet, dass, obwohl Funktionen Tupel und Ergebnisrelationen zurückliefern können, diese Parameter bzw. Rückgabewerte in Form von Mengen nicht untersucht werden. Weiterhin werden keine geschachtelten Funktionen und deren Aufrufe betrachtet. Das heißt, es wird nicht zugelassen, dass eine Funktion eine andere aufruft und anhand deren Rückgabewerte die Ausführungen fortsetzt. Dadurch können Funktionen atomar und unabhängig voneinander betrachtet werden. Die Komposition von Funktionen und deren Änderungen kann Gegenstand zukünftiger Arbeiten sein.

Zu jedem Fall werden wieder die Anwendungsvoraussetzung, die möglichen Aktionen und die daraus entstehenden Resultate angegeben. Daraus entsteht Tabelle 4.7, welche die betrachteten Fälle übersichtlich zusammenfasst.

4.2.2.1 Hinzufügen & Entfernen von Parametern

Unter der Annahme, dass in dieser Arbeit nur Parameter mit Basisdatentypen betrachtet werden, haben das Hinzufügen bzw. das Entfernen von Parametern bei einer Funktion keine Auswirkungen auf die Anwendbarkeit dieser. Parameter werden an dieser Stelle somit manuell oder aus einer (externen) Anwendung heraus übergeben[4]. Erst im Funktionsrumpf kann durch das Hinzufügen bzw. durch das Entfernen eines Parameters eine Veränderung derart stattfinden, dass dort Anpassungen unter bestimmten Bedingungen notwendig sind, um die Anwendbarkeit der Funktion zu gewährleisten. Dies wird in Abschnitt 4.2.2.4 beschrieben. Ferner hat das Hinzufügen eines OUT/INOUT-Parameters die Bedeutung des Hinzufügens bzw. Änderns von Rückgabewerten, was in Abschnitt 4.2.2.5 behandelt wird.

Anwendungsvoraussetzung: keine
Aktion: keine
Resultat: Funktion kann uneingeschränkt weiter benutzt werden.

[4] Es sei hier die XML-Konfigurationsdatei von GODESS aus Abschnitt 1.2.2 erwähnt.

	Funktionsänderungen					
	Hinzufügen & Entfernen von Parametern	**Umbenennung von Parametern oder Funktion**	**Datentypänderung eines Parameters**	**Änderung des Funktionsrumpfs**	**Änderung von Rückgabewerten**	**Neuanlage einer Funktion**
Anwendungs-voraussetzung	keine	keine	Kompatible Datentypen oder verlustfreie Typumwandlung zwischen verschiedenen Datentypen möglich.	Kompensation durch andere Relationen/Attribute, Vorgängerfunktion (kontextbasiert) oder Schemaänderung..	keine	Kompensation möglich durch Anpassung der Funktion bzw. Erzeugung einer Wrapper-Funktion oder durch Schemaänderung.
Aktion	keine	keine; ggf. Anpassung externer Anwendungen oder weiterverarbeitender Funktionen.	Kompatibel, dann implizite Typumwandlung durch DBMS, andernfalls explizite Typumwandlung.	Kompensation möglich: Anpassung der Funktion (Versionierung) bzw. kontextbasierter Aufruf der Vorgängerfunktion oder Änderung des Schemas. Keine Kompensation möglich: Anlegen der Funktion im Voraus als ungültig.	keine; ggf. Anpassung externer Anwendungen oder weiterverarbeitender Funktionen.	Kompensation möglich: Anpassung der Funktion (Versionierung) oder durch Änderung des Schemas.
Resultat	Funktion kann uneingeschränkt weiter benutzt werden.	Funktion kann uneingeschränkt weiter benutzt werden.	Weiternutzung der alten Funktion bei impliziter Typumwandlung, andernfalls Nutzen der neuen Version der Funktion mit expliziter Typumwandlung. Keine Typumwandlung möglich: Keine Anwendbarkeit gegeben.	Kompensation möglich: Anwendung der neuen Funktion (Versionierung) bzw. durch Kontext gegeben; ggf. neues (Nachfolger-)Schema. Keine Kompensation möglich: Keine Anwendbarkeit gegeben.	Funktion kann uneingeschränkt weiter benutzt werden.	Kompensation möglich: Nutzung der neuen Nachfolge- bzw. Wrapper-Funktion oder des geänderten Schemas. Keine Kompensation möglich: Keine Anwendbarkeit gegeben.

Abbildung 4.7: Zusammenfassung der Anwendbarkeit von Funktionen bei Funktionsänderungen

4.2.2.2 Umbenennung eines Parameters & Umbenennung der Funktion

Durch die atomare Betrachtung der Funktionen hat sowohl die Umbenennung von Parametern als auch die Umbenennung einer Funktion keine Auswirkungen auf die weitere Anwendbarkeit dieser. Die Bezeichner der Parameter werden nur intern in der Funktion selbst verwendet und tragen zur Lesbarkeit dieser bei. Obwohl einerseits die Namen von OUT/INOUT-Parameter als Rückgabewerte andererseits der Name der Funktion in der weiteren Verarbeitung benutzt werden können, z. B. durch externe Anwendungen oder durch Komposition von Funktionen, ist die Anwendbarkeit der Funktion unabhängig vom Kontext oder der Umgebung in jedem Fall gegeben. Gegebenenfalls muss aber in beiden Fällen der Umbenennung die nachfolgende Verarbeitungskette, die auf den Bezeichnern der Rückgabewerte einer Funktion mit OUT/INOUT-Parametern oder auf den Funktionsnamen aufbaut, angepasst werden.

Anwendungsvoraussetzung: keine
Aktion: keine; ggf. Anpassung externer Anwendungen oder weiterverarbeitender Funktionen.
Resultat: Funktion kann uneingeschränkt weiter benutzt werden.

4.2.2.3 Datentypänderung eines Parameters

An dieser Stelle werden nur Basisdatentypen und keine mengenwertigen Typen wie Tupel betrachtet. Die Änderung des Datentyps eines Parameters einer Funktion kann ähnlich der Datentypänderungen von Attributen von Relationenschemata in Abschnitt 4.2.1.4 behandelt werden. Eine Kompensation ist nur durch implizite oder explizite Typumwandlung möglich. Wird eine explizite Umwandlung umgesetzt, so entspricht dies der Versionierung der Funktion. Ist dagegen keine Umwandlung möglich, ist die Anwendbarkeit der Funktion nicht mehr gegeben. Daher entsprechen die Anwendungsvoraussetzung, die Aktion und das Resultat denjenigen der Datentypänderungen von Attributen von Relationenschemata aus Abschnitt 4.2.1.4.

Im Kontext von Datenbanken existieren neben den Basisdatentypen noch drei weitere Arten von Typen von Parametern: IN, INOUT, OUT. Diese Typen geben an, ob es sich bei dem Parameter um einen nur-lesenden (IN), lesenden und schreibenden (INOUT) oder nur-schreibenden (OUT) Parameter handelt. Da die beiden schreibenden Varianten (INOUT, OUT) der Semantik von Rückgabewerten entsprechen, werden diese in dem entsprechenden Abschnitt 4.2.2.5 behandelt. OUT-Parameter werden aber nur in STPs und nicht in UDFs unterstützt. Wird ein IN-Parameter zu einem INOUT-Parameter geändert, so hat dies keine Auswirkungen auf die Anwendbarkeit der Funktion. Die Änderungen eines IN-Parameters zu einem OUT Parameter entspricht dagegen dem Entfernen eines Parameters, da OUT-Parameter beim Funktionsaufruf nicht belegt werden dürfen. Der entgegengesetzte Fall, die Änderung eines OUT-Parameters zu einem IN/INOUT-Parameter entspricht dabei dem Hinzufügen eines Parameters. Diese Änderungen wurden bereits in Abschnitt 4.2.2.1 bei der Untersuchung des Hinzufügens und Entfernens von Parametern behandelt.

Anwendungsvoraussetzung: Kompatible Datentypen oder verlustfreie Typumwandlung zwischen

verschiedenen Datentypen möglich.

Aktion: Kompatibel, dann implizite Typumwandlung durch DBMS, andernfalls explizite Typumwandlung.

Resultat: Weiternutzung der alten Funktion bei impliziter Typumwandlung, andernfalls Nutzen der neuen Version der Funktion mit expliziter Typumwandlung. Keine Typumwandlung möglich: Keine Anwendbarkeit gegeben.

4.2.2.4 Änderung des Funktionsrumpfs

Eine Änderung im Funktionsrumpf kann einerseits getrennt von anderen Funktionsänderungen, andererseits bedingt durch andere Funktionsänderungen, z. B. Änderungen der Parameter, stattfinden. Darüber hinaus kann eine Änderung an dieser Stelle die Rückgabewerte (siehe Abschnitt 4.2.2.5) beeinflussen.

Zunächst wird eine isolierte Änderung betrachtet. Das bedeutet, die Änderung ist nicht durch andere stattgefundene Änderungen bedingt und zieht keine weiteren Folgen bzw. Änderungen anderer Komponenten der Funktion, z. B. der Rückgabewerte, nach sich. Gegenüber externen Anwendungen oder anderen Funktionen wirkt die Funktion nach einer Modifikation im Funktionsrumpf unverändert, da sich die Signatur der Funktion, welche aus dem Funktionsnamen, der Reihenfolge und der Anzahl der Parameter sowie den Rückgabewerten zusammen setzt, nicht ändert. Trotzdem kann, wie bereits in Abschnitt 4.2.2 beschrieben, die Änderung dazu führen, dass die Funktion als solche nicht mehr auf das Schema anwendbar ist. Das bedeutet, dass der Aufruf zwar unverändert funktionieren würde, aber das DBMS durch Fehler anzeigen würde, dass die Funktion unbekannte, also im Schema nicht vorhandene Attribute oder Relationen benutzt. Eine Möglichkeit der Kompensation besteht aus der Verwendung einer oder mehrerer Relationen bzw. Attributen, die die fehlenden Strukturen ersetzen können, ohne dass die Semantik der Funktion beeinflusst wird. Unter Betrachtung des Kontextes kann auch der Aufruf der Vorgängerfunktion eine Kompensationsmöglichkeit darstellen.

Als nächstes sollen bedingte Änderungen der Funktion, d.h. vorherigen Änderungen bedingen eine Modifikation des Funktionsrumpfes bzw. der Rückgabewerte, untersucht werden. An dieser Stelle werden nur IN-Parametern sowie Basisdatentypen als Werte betrachtet, welche nach Abschnitt 4.2.2.1 von (externen) Anwendungen oder anderen Funktionen übergeben werden. Wird ein Parameter hinzugefügt, so wird dieser Bestandteil der Verarbeitung im Funktionsrumpf. Einerseits können damit Anfragen auf das Schema, z. B. durch Selektion, beeinflusst werden, andererseits kann der Parameter auch in Kontrollstrukturen, wie *if-else*-Anweisungen, benutzt werden, was wiederrum eine Beeinflussung auf die Anfragen auf das Schema nach sich ziehen kann. Auch hier gilt wieder, dass dadurch die Funktion nicht mehr auf das Schema anwendbar sein kann. Die Kompensation ist an dieser Stelle sehr kontextabhängig und kann wie im vorherigen Absatz beschrieben, durch andere Relationen oder Attribute erfolgen. Als Beispiel für eine kontextabhängige Kompensation durch andere Attribute sei hier die Selektion auf ein Attributwert genannt, der eine andere physikalische Einheit gegenüber dem Parameter besitzt. Durch eine Berechnungsvor-

schrift, die sich ggf. durch andere Attributwerte zusammensetzt, kann eine Umrechnung erfolgen, so dass der Attributwert der Einheit des Parameters oder umgekehrt entspricht. Gegebenenfalls kann diese Kombination durch eine UDF oder Wrapper-Funktion realisiert werden. Darüber hinaus ist es möglich, dass der Parameter nur Bestandteil einer mathematischen Formel und somit keiner Anfrage ist. Das bedeutet, dass der hinzugefügte Parameter keine Auswirkungen auf die Anwendbarkeit der Funktion hat.

Das Entfernen eines IN-Parameters einer Funktion und die dadurch bedingte Anpassung des Funktionsrumpfs haben keine Auswirkungen auf die Anwendbarkeit der Funktion. Es kann zwar zur Beeinflussung der zurückgelieferten Datenmenge bei einer Anfrage durch eine fehlende einschränkende Selektion anhand des fehlenden Parameters stattfinden, aber dies hat dennoch keine Auswirkungen auf die Anwendbarkeit der Funktion auf dem gegebenen Schema.

Die Kompensation entspricht in diesem Abschnitt der Versionierung durch Anpassung und damit Erzeugung einer Nachfolgerfunktion. Ist diese Art von Kompensation nicht gegeben, kann dies eine Schemaänderung nach sich ziehen, was eine weitere Art der Kompensation ist. Diese Fälle sind bereits in Abschnitt 4.2.1 beschrieben. Ist auch keine Schemaänderung möglich, so ist die Anwendbarkeit der Funktion nicht gegeben. Nach Abbildung 4.6 bedeutet das, dass die Nachfolgerfunktion (Q(S))' von Q(S) nur innerhalb des Zeitraums des Anlegens existiert hat. Dieser Zeitraum ist sehr klein gegenüber den normalen Gültigkeitszeiträumen von Funktionen. Er besitzt als Startzeitpunkt den Endzeitpunkt der Vorgängerfunktion Q(S) und als Endzeitpunkt den Startzeitpunkt der Nachfolgerfunktion (Q(S))'. Bei grober Granularität der Zeitdarstellung können diese Zeitpunkte identisch sein. Das bedeutet, dass die Nachfolgerfunktion, deren Anwendbarkeit im Voraus nicht gegeben war, als ungültig angelegt worden ist. Dies gilt aber nur in dem Fall, in dem keine Schemaänderungen möglich sind.

Anwendungsvoraussetzung: Kompensation durch andere Relationen/Attribute, Vorgängerfunktion (kontextbasiert) oder Schemaänderung.

Aktion:

a) Kompensation möglich: Anpassung der Funktion (Versionierung) bzw. kontextbasierter Aufruf der Vorgängerfunktion oder Änderung des Schemas.

b) Keine Kompensation möglich: Anlegen der Funktion im Voraus als ungültig.

Resultat:

a) Kompensation möglich: Anwendung der neuen Funktion (Versionierung) bzw. durch Kontext gegeben; ggf. neues (Nachfolger-)Schema.

b) Keine Kompensation möglich: Keine Anwendbarkeit gegeben.

4.2.2.5 Änderung von Rückgabewerten

Rückgabewerte hängen einerseits von den Parametern, andererseits von der Verarbeitung im Funktionsrumpf ab. Es existieren verschiedene Arten von Rückgaben: mittels OUT/INOUT-

Parameter oder dem klassischen RETURN. Weiterhin können einerseits Basisdatentypen in Form von skalaren Werten, andererseits ganze Ergebnisrelationen zurückgegeben werden.

Rückgabewerte beeinflussen die Anwendbarkeit einer Funktion im Allgemeinen nicht. Im Speziellen sei hier die Komposition von Funktionen genannt, welche aber in dieser Arbeit nicht betrachtet wird. Daneben können externe Anwendungen existieren, die die Rückgabewerte weiterverarbeiten oder zur Darstellung von Diagrammen nutzen. Ändert sich die Anzahl der Rückgabewerte, z. B. durch Parameteränderungen nach OUT/INOUT, müssen diese Anwendungen angepasst werden. Dasselbe gilt für Änderungen des zurückgegebenen Typs der Rückgabewerte.

Da in dieser Arbeit von nur lesenden Funktionen und auf der Ebene der Originaldaten ausgegangen wird, sind Rückgabewerte schemaunabhängig. Das bedeutet, dass keine dauerhafte Speicherung im DBMS in Schema S durch eine bereitgestellte Ergebnisrelation erfolgt.

Darüber hinaus kann es vorkommen, dass durch eine neue Funktion und damit veränderte Rückgabewerte die Einheit des Wertes nicht mehr übereinstimmt. Dies könnte durch eine Wrapper-Funktion kompensiert werden, welche, sofern möglich, die Einheit wieder umrechnet. Dieser hier beschriebene Fall ist aber sehr kontextabhängig und hat keine Auswirkungen auf die eigentliche Anwendbarkeit einer Funktion.

Anwendungsvoraussetzung: keine
Aktion: keine; ggf. Anpassung externer Anwendungen oder weiterverarbeitender Funktionen.
Resultat: Funktion kann uneingeschränkt weiter benutzt werden.

4.2.2.6 Neuanlage einer Funktion

Die Neuanlage einer Funktion kann durch neue Erkenntnisse aus der Forschung bedingt sein. Das bedeutet, dass z. B. ein neues Verfahren oder eine neue Berechnung eines physikalischen Wertes anhand anderer physikalischer Parameter und Werte in der jeweiligen Fachdisziplin eingeführt wurde. Damit eine solche neue externe Funktion auf ein Schema anwendbar ist, muss dieses die entsprechenden Attributwerte zur Berechnung bereitstellen. Im Allgemeinen gilt, dass eine externe neue Funktion nicht auf bestehende Datenstrukturen, das Schema, anwendbar ist. Die Anwendbarkeit kann dabei durch eine Wrapper- bzw. Nachfolgerfunktion oder durch eine Änderung des Schemas selbst kompensiert werden. Liegt keine externe neue Funktion vor, wurde eine neue Funktion anhand des vorliegenden Schemas entworfen. Diese ist somit ohne Einschränkungen auf das Schema anwendbar. Ist keine Kompensation möglich, so ist die Anwendbarkeit der neuen externen Funktion nicht gegeben.

Anwendungsvoraussetzung: Kompensation möglich durch Anpassung der Funktion bzw. Erzeugung einer Wrapper-Funktion oder durch Schemaänderung.
Aktion: Kompensation möglich: Anpassung der Funktion (Versionierung) oder durch Änderung des Schemas.

Resultat:

a) Kompensation möglich: Nutzung der neuen Nachfolge- bzw. Wrapper-Funktion oder des geänderten Schemas.

b) Keine Kompensation möglich: Keine Anwendbarkeit gegeben.

4.2.3 Möglichkeiten zur Kompensation von Funktions- und Schemaänderungen

Bisher wurde in den Untersuchungen der Anwendbarkeitsvoraussetzungen bei den Schema- und Funktionsänderungen über eine mögliche Kompensation der Änderungen durch Versionierung und Einführung neuer Funktionen gesprochen. Dabei wurde nicht weiter auf die Möglichkeiten bzw. die Techniken zur Kompensation eingegangen. Dies soll in diesem Abschnitt anhand von Techniken des Schema-Mappings skizziert werden. Neben den hier dargestellten Möglichkeiten besteht immer die Option, eine neue Version einer Funktion manuell zu erzeugen, so dass diese auf ein verändertes Schema anwendbar ist.

Bei der Untersuchung der Schemaänderungen wurde davon ausgegangen, dass der *Snapshot*-Mechanismus zur Versionierung von Schemata aus Abschnitt 4.1.2 zur Verfügung steht. Das bedeutet, es liegen versionierte „eingefrorene“ Schemata aus der Vergangenheit sowie alle Änderungen, die zu den Nachfolgerschemata führten, im Log vor. Abbildung 4.8 soll in diesem Abschnitt zur Verdeutlichung beitragen. Dazu wurde, wie in Abschnitt 4.2 eingeführt, die Funktion Q’ um das Argument des Schemas S’ erweitert, auf das diese Funktion zum aktuellen betrachteten Zeitpunkt anwendbar ist. Das Δ bezeichnet die aufgezeichneten Veränderungen im Log. Es gelten die Beziehungen S = S’ - Δ sowie S” = S’ + Δ. Die gestrichelten bzw. gepunkteten Linien sollen das Ziel verdeutlichen, dass die Funktion Q’ auf S bzw. S” durch die beschriebenen Beziehungen anwendbar werden soll.

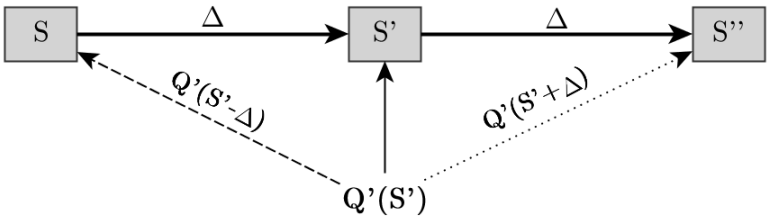


Abbildung 4.8: Veranschaulichung der Kompensationstechniken mit Hilfe der *Snapshot*-Variante

Die Herausforderung besteht nun darin, eine neue Version der Funktion Q’ derart zu erzeugen, dass diese ausgehend von Schema S’ je nach Richtung der Versionierung auf das Vorgänger- bzw. Nachfolgerschema anwendbar ist. Dieser Aufgabe kann durch Techniken des Schema-Mappings

begegnet werden, welche aus dem Bereich der Informationsintegration stammen. Die folgenden Informationen in diesem Abschnitt stammen sofern nicht anders gekennzeichnet aus [BH13]. Ziel des Schema-Mappings ist es, eine Abbildung zwischen zwei Schemata zu finden. Dazu müssen Korrespondenzen zwischen den Schemata gefunden bzw. festgelegt werden. Korrespondenzen stellen eine Zuordnung zwischen den Elementen aber auch zwischen den Werten der beteiligten Schemata her. Als Beispiel sei hier die Teilung eines *name*-Attributs in zwei weitere Attribute *vorname*, *nachname* genannt (vgl. dazu Abschnitt 4.2.1.5). Dies stellt gleichzeitig ein Beispiel für eine 1:n-Abbildung bezüglich des Schema-Matchings dar und wird in Abbildung 4.9 veranschaulicht. Dabei soll jeweils ein Relationenschema *person* im Schema S' bzw. S'' mit den entsprechenden Attributen existieren.

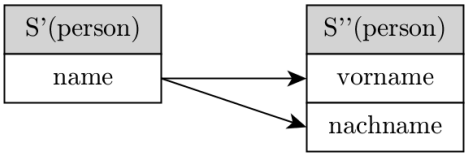


Abbildung 4.9: Beispiel: Teilen eines Attributs

Die Pfeile entsprechen den Korrespondenzen. Zeitgleich stellen diese die Änderung dar, die von Schema S' nach S'' stattgefunden haben. Diese Änderungen sind im Log aufgezeichnet. Das bedeutet, dass die Korrespondenzen bzw. Änderungen automatisch vorliegen und aus dem Log ermittelt werden können. Dies entspricht im Bereich der Informationsintegration der Technik des Schema-*Matchings*. Die Herausforderungen dort sind das Finden und das Bewerten aller möglichen Korrespondenzen sowie die Wahl der optimalen Menge dieser, so dass möglichst viele Schemaelemente aus dem Ausgangsschema den Elementen des Zielschemas zugeordnet werden. Dabei ist das Finden der Korrespondenzen nicht trivial, da dies z. B. von der Art und Größe der Schemata (Denormalisiert, Redundanzen), den Namen der Elemente, z. B. fehlende Namenskonventionen, Synonyme, Homonyme, oder von dem Grad der semantischen und schematischen Heterogenität abhängt [Les16]. Für das Finden der Korrespondenzen existieren daher verschiedene Verfahren, die die Namen der Schemaelemente (Labelbasiert), die eigentlichen Daten (Instanzbasiert) oder die Struktur des Schemas (Strukturbasiert) verwenden [Les16]. Da aber die Korrespondenzen als die Änderungen der Schemata betrachtet werden können und diese im Log vorliegen, entfällt der gesamte Prozess und Aufwand des Schema-Matchings. Der nächste Schritt beim Schema-Mapping ist die Ableitung in ein oder mehrere logische Mappings [BH13], auf das an dieser Stelle nicht weiter eingegangen wird.

Als nächstes werden die Korrespondenzen in Transformationsanfragen übersetzt, welche an dieser Stelle SQL-Anfragen darstellen, um die Daten des Quellschemas in die Struktur des Zielschemas zu überführen [BH13]. Das Übersetzen der Korrespondenzen und damit der aufgezeichneten Änderungen im Log entspricht der Erzeugung einer neuen Version der Funktion Q'. Der Unterschied zu den Transformationsanfragen besteht darin, dass an dieser Stelle keine Daten in das neue Sche-

ma, z. B. S", überführt werden, sondern nur die Anwendbarkeit der veralteten Funktion Q' auf das neue Schema S" hergestellt werden soll. In Abbildung 4.8 soll dies durch die diagonalen Pfeile von Q' auf S bzw. S" und den dort verwendeten Argumenten, nämlich das Schema, sowie die Korrespondenzen als Änderungen Δ, dargestellt werden. Zusätzlich zu den ursprünglichen Transformationsanfragen müssen im Funktionsrumpf von Q' alle dort verwendeten Relationen und Attribute berücksichtigt werden. Im weiteren Verlauf werden Transformationsanfragen und das Übersetzen der Änderungen zur Erzeugung einer neuen Version einer Funktion synonym verwendet, obwohl keine Überführung der Daten stattfindet. Das Erzeugen dieser Transformationsanfragen ist nicht trivial, da die Korrespondenzen häufig als DDL-Anweisungen im Log vorliegen. Diese DDL-Anweisungen müssen in DML-Anweisungen übersetzt werden um in einer neuen Version der Funktion anwendbar zu sein. Wie diese Übersetzung stattfinden kann, ist nicht Bestandteil dieser Arbeit. Als Beispiel sei dasjenige aus Abbildung 4.9 aufgegriffen. Im Log stehen dazu z. B. zwei DDL und eine DML Anweisung (PostgreSQL Syntax):

```
ALTER TABLE person ADD COLUMN nachname VARCHAR(100);
UPDATE person SET
  nachname = SUBSTRING(name FROM STRPOS(name, ' ')+1 FOR LENGTH(name)),
  name = SUBSTRING(name FROM 1 FOR STRPOS(name, ' ')-1)
;
ALTER TABLE person RENAME COLUMN name TO vorname;
```

Anweisung 4.4: Ausschnitt aus dem Log nach dem Teilen eines Attributs

Auf der einen Seite müssen die DDL-Anweisungen übersetzt werden, auf der anderen Seite sind die SET-Klauseln der UPDATE-Anweisung zu extrahieren und zu übersetzen. Darüber hinaus muss die Richtung der Versionierung betrachtet werden. Das Hinzufügen eines Attributs in der einen Richtung entspricht dem Entfernen des Attributs in der Gegenrichtung. Dazu müssen inverse DDL-Anfragen existieren. Das dies in dieser Arbeit gegeben ist, wurde bereits in Abschnitt 4.2 erwähnt und soll an dieser Stelle durch Tabelle 4.4 veranschaulicht werden. Inverse Operationen bzw. DML-Anweisungen sind aufgrund der Komplexität nicht trivial zu ermitteln und nicht Bestandteil dieser Arbeit.

Anweisung	**Inverse**
Hinzufügen Attribut/Relation	Entfernen Attribut/Relation
Entfernen Attribut/Relation	Hinzufügen Attribut/Relation
Umbenennung Attribut/Relation	Umbenennung Attribut/Relation
Datentypänderung	Datentypänderung
Teilen Attribut/Relation	Vereinigung Attribut/Relation
Vereinigung Attribut/Relation	Teilen Attribut/Relation

Tabelle 4.4: In dieser Arbeit behandelte Fälle der Anwendbarkeit von Funktionen bei Schemaänderungen aus Abschnitt 4.2.1 und deren Umkehroperationen

Zur Fortführung des Beispiel sei angenommen, dass die Funktion Q' nur aus der Anweisung 4.5 besteht.

```
SELECT name FROM person
```

Anweisung 4.5: Beispielfunktion Q' vor der Teilung eines Attributs

Da eine Teilung des *name*-Attributs stattgefunden hat, entsteht durch die vorliegenden Informationen aus dem Log 4.4 und der nicht näher spezifizierten Übersetzung die neue Anweisung 4.6, welche die Nachfolgerfunktion Q" bildet.

```
SELECT vorname || ' ' || nachname AS name FROM person
```

Anweisung 4.6: Nachfolgerfunktion Q" nach der Teilung eines Attributs

Zu erkennen ist, dass die inverse Operation der Teilung einer Zeichenketten die Verkettungsoperation der neuen Attribute ist.

Das Beispiel soll nun in Gegenrichtung, also die Zusammenführung von Attributen, betrachtet werden, um eine weitere Möglichkeit aufzuzeigen. Im Kontext des Schema-Matchings stellt diese Operation eine n:1-Abbildung dar und ist in Abbildung 4.10 zu sehen. Dazu existieren die Anweisungen 4.7 (PostgreSQL Syntax) im Log. Die Funktion Q' sei die Anweisung 4.8.

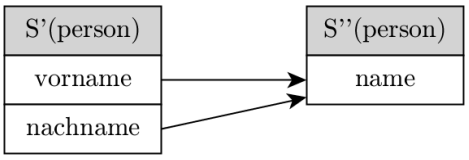


Abbildung 4.10: Beispiel: Zusammenführung von zwei Attributen

```
UPDATE person SET vorname = vorname || ' ' || nachname;
ALTER TABLE person DROP COLUMN nachname;
ALTER TABLE person RENAME vorname TO name;
```

Anweisung 4.7: Ausschnitt aus dem Log nach dem Zusammenführung von zwei Attributen

```
SELECT vorname, nachname FROM person
```

Anweisung 4.8: Beispielfunktion Q' vor der Zusammenführung von zwei Attributen

Aus den vorliegenden Informationen aus dem Log und der Existenz der zur Konkatenation inversen Operation aus dem vorherigen Beispiel, entsteht die neue Anweisung 4.9, welche der Funktion Q" entspricht.

```
SELECT vorname, nachname FROM (
  SELECT
    SUBSTRING(name FROM STRPOS(name, ' ')+1 FOR LENGTH(name)) AS vorname,
    SUBSTRING(name FROM 1 FOR STRPOS(name, ' ')-1) AS nachname
  FROM person
)
```

Anweisung 4.9: Nachfolgerfunktion Q" nach der Zusammenführung von zwei Attributen

Zu erkennen ist, dass im Gegensatz zum vorherigen Beispiel die Transformation nicht in den Attributen (Projektion) stattgefunden hat, sondern in der FROM-Klausel. Es wurde das alte Schema S' mittels der inversen DML-Operationen auf das neue Schema S" unter Verwendung einer Unterabfrage rekonstruiert. Diese Unterabfrage kann einerseits direkt in die neue Abfrage eingesetzt werden, andererseits kann daraus auch eine Sicht (engl. *view*) erzeugt werden. Unter der Annahme, dass dieser VIEW den Namen *person_view* trägt, sieht die neue Anweisung 4.10 bzw. Q" folgendermaßen aus:

```
SELECT vorname, nachname FROM person_view
```

Anweisung 4.10: Nachfolgerfunktion Q" nach der Zusammenführung von zwei Attributen unter Verwendung eines VIEWs

Die in den beiden Beispielen verwendete Technik zur Erzeugung der neuen Anfrage Q" durch die Verwendung der ursprünglichen Anfrage Q' sowie das Einfügen der Änderungen Δ in diese, wird Anfragetransformation (engl. *query rewriting*) genannt.

Abschließend soll eine weitere Möglichkeit zur Kompensation beschrieben werden, die bei einigen Fällen, wie z. B. der Umbenennung einer Funktion, angewendet werden kann. Dazu wird die ursprüngliche Funktion von einer neuen Version der Funktion umschlossen (engl. *to wrap*) bzw. gekapselt und durch Aufruf weiter verwendet. So können Änderungen, welche für eine externe Anwendung eine Rolle spielen, kompensiert werden, ohne dass die Anwendung angepasst werden muss. Als Beispiel sei hier die Umbenennung einer Funktion oder die Umbenennung von INOUT/OUT-Parametern genannt. In diesem konkreten Beispiel besitzt die Wrapper-Funktion die Signatur der alten Funktion und steht der externen Anwendung zur Verfügung. Weiterhin kann aber auch die Kompensation von fehlenden Attributen oder Relationen, sofern möglich, damit realisiert werden. Dabei werden in der Wrapper-Funktion anhand der vorhandenen Informationen im neuen Schema die fehlenden Informationen der weggefallenen Attribute oder Relationen rekonstruiert und der ursprünglichen Funktion zur Verfügung gestellt. Die Wrapper-Funktion kann an dieser Stelle als Nachfolgefunktion im Sinne der Versionierung gesehen werden - unabhängig davon ob die alte Funktion im Sinne der Gültigkeitszeit beendet wird oder ob beide Versionen parallel existieren.

4.3 Provenance und temporalen Daten

Nachdem aufgezeigt wurde, welche Kriterien bei der Anwendung von veränderten Funktionen auf veränderte Schemata einzuhalten bzw. zu beachten sind, soll an dieser Stelle das Zusammenspiel zwischen den versionierten Schemata respektive Funktionen und der Provenance beleuchtet werden.

Es stellt sich die Frage, wie temporale Aspekte die Provenance unterstützen können und was für Provenance-Anfragen dabei entstehen bzw. überhaupt möglich sind. Die Verknüpfung dieser beiden Aspekte ist dabei eine Herausforderung, der in der aktuellen Forschung begegnet wird. So untersuchen Glavic et al im Projekt „Provenance using Temporal Databases“, wie die Berechnung von Provenance von temporalen Datenbanktechniken profitieren kann [Bor16b]. Die in dem Projekt entwickelten Techniken sind dabei in das GProM-System [AGR+14, Bor16a] eingeflossen, welches das Nachfolgersystem von im Abschnitt 3.2.3 erwähnten PERM-System darstellt.

Ziel der Provenance ist die Rekonstruktion beliebiger Auswertungen an beliebigen Zeitpunkten. Das bedeutet, dass eine Auswertung, die an einem vergangenen Zeitpunkt auf den zu dem Zeitpunkt gültigen Daten mit der entsprechenden gültigen Funktion gemacht wurde, zu späteren Zeitpunkten wiederholt und rekonstruiert werden kann. Um dies zu gewährleisten, ist es notwendig, dass die Provenance-Anfragen bzw. Mechanismen einerseits die versionierte Auswertefunktion, andererseits die versionierten Daten zur Verfügung hat. Dabei muss sichergestellt werden, dass die Auswertefunktion auf den Daten im ggf. veränderten Schema anwendbar ist. Dies sind die in Abschnitt 4.2 ermittelten Kriterien. Ist dies z. B. nicht der Fall, kann das Provenance-System diese Informationen dem Anwender bereitstellen, indem es aufzeigt, warum die Funktion nicht auf die Daten angewendet werden kann. Dazu können die Kriterien aufgelistet werden, welche verletzt bzw. nicht automatisch kompensierbar sind. Zusätzlich kann ein Verlauf der Änderungen des Schemas sowie der Funktion angezeigt werden, da diese im Log bzw. in der Versionierungstabelle vorliegen. Die Repräsentation dieser Änderungen ist im Sinne der Provenance selbst eine extensionale Antwort. Die Antwort mit Hilfe der Kriterien bzw. der Kompensierbarkeit hingegen eine intensionale.

In der vorliegenden Arbeit wurde der Fokus auf die Versionierung von SQL-Funktionen, weniger auf die eigentliche Provenance gelegt. Konkrete Anfragen können nur durch theoretische SQL-Konstrukte erzeugt werden, welche z. B. an die Syntax des PERM Systems angelehnt sind. Dazu soll ein konstruiertes Beispiel in Anweisung 4.11 gezeigt werden, welche die SQL-PLE von PERM sowie temporale Klauseln aus dem SQL:2011-Standard nutzt.

```
SELECT PROVENANCE ON CONTRIBUTION (COPY)
  evaluate_temp(temperature)
FROM payload
FOR SYSTEM_TIME BETWEEN DATE '2015-09-13' AND DATE '2015-09-15'
```

Anweisung 4.11: Konstruierte *Where*-Provenance-Anfrage mit PERM SQL-PLE und SQL:2011-Standard

Durch die temporale FOR SYSTEM_TIME-Klausel ist vorgegeben, in welchem Zeitraum die Auswertung über die Temperatur erfolgen soll. Das Provenance-System muss nun in der Lage sein, die entsprechende Auswertefunktion aus diesem Zeitraum sowie das zu dem Zeitpunkt gültige Schema zu ermitteln, die Auswertung zu machen und dann die Provenance-Informationen, in diesem Fall eine *Where*-Provenance-Anfrage dargestellt durch die (COPY)-Klausel, zurückzugeben. Dabei müssen, wie oben erwähnt, die Kriterien für die Anwendbarkeit der Auswertefunktion *evaluate_temp* auf das Schema in dem Zeitraum gegeben sein. Ist die Anwendbarkeit nicht gegeben, muss das System dies dem Anwender extensional oder intensional mitteilen.

Anhand dieses relativ simplen Beispiels, welches nur aus einer Auswertefunktion mit einem Parameter besteht, ist zu erkennen, dass die Herausforderung in der Verknüpfung der temporalen Datenhaltung mit den Provenance-Anfragen besteht. Das Provenance-System muss „wissen“, dass es im Falle einer temporalen Auswertung die entsprechenden Informationen über die Auswertefunktion und das Schema zu den Zeitpunkten, an vordefinierten Stellen (Versionierungstabellen, Logs) zusammen suchen und dabei die Anwendbarkeit überprüfen muss. Dieses Zusammenspiel von Provenance, temporaler Datenhaltung und der Anwendung von (versionierten) Transformationen in Form von Abfragen oder Auswertefunktion kann Bestandteil zukünftiger Arbeiten sein.

5 Prototyp

An dieser Stelle soll exemplarisch das Konzept der Versionierung von UDFs/STPs aus Abschnitt 4.1 an einem konkreten DBMS, PostgreSQL, umgesetzt werden. Dazu werden Funktionalitäten, die zur Ausführung der Schritte benötigt werden, in eigenen SQL-Funktionen gekapselt. Eine erste Hilfsfunktion, auf die nicht weiter eingegangen wird, liefert einen Unix-Zeitstempel und kann in Anhang A.5 eingesehen werden. Ferner werden die beiden Funktionen zur Umrechnung der Rohdaten in SI-Einheiten benötigt. Die alte Funktion kann durch Anweisung 5.1, die neue Funktion mittels Anweisung 5.2 erzeugt werden.

```
CREATE OR REPLACE FUNCTION raw_to_units_press(
  p_deployment_id INTEGER,
  a0 NUMERIC, a1 NUMERIC, a2 NUMERIC
) RETURNS TABLE(p NUMERIC)
AS $$
BEGIN
  RETURN QUERY SELECT ((a2^2)*pressure + a1*pressure + a0)
  FROM payload_raw WHERE deployment_id = p_deployment_id;
END;
$$ LANGUAGE plpgsql
```

Anweisung 5.1: Alte Funktion zur Umrechnung in SI-Einheiten

```
CREATE FUNCTION raw_to_units_press(
  p_deployment_id INTEGER,
  a0 NUMERIC, a1 NUMERIC, a2 NUMERIC, a3 NUMERIC
) RETURNS TABLE(p NUMERIC)
AS $$
BEGIN
  RETURN QUERY SELECT ((a3^3)*pressure + (a2^2)*pressure + a1*pressure + a0)
  FROM payload_raw WHERE deployment_id = p_deployment_id;
END;
$$ LANGUAGE plpgsql
```

Anweisung 5.2: Neue Funktion zur Umrechnung in SI-Einheiten

Zu erkennen ist, dass der Name gleich bleibt, sich dagegen die Anzahl der Parameter sowie die Berechnungsvorschrift geringfügig geändert haben. Die Namen der Koeffizienten sowie die Werte der Polynome entstammen dabei aus der in Abschnitt 1.2.2 erwähnten XML-Konfigurationsdatei.

Des Weiteren wird angenommen, dass eine *payload_raw*-Tabelle ähnlich der *payload*-Tabelle 1.2 existiert, nur dass diese die Rohdaten der Sensoren anstelle der umgerechneten Daten in SI-Einheiten enthält. Die Funktionen können mittels Anweisung 5.3 aufgerufen werden und liefern als Ergebnis eine Tabelle 5.1 mit einem Attribut *pressure*, dessen Werte den umgerechneten Rohdaten entsprechen.

```
SELECT raw_to_units_press(14, -1.038561e+001, 4.23071e-003, 4.946985e-010^2) AS pressure
```

Anweisung 5.3: Aufruf der alten Funktion zur Umrechnung in SI-Einheiten

pressure
201.45873183
201.11604432
201.0652758
...

Tabelle 5.1: Ausschnitt aus dem Ergebnis der Ausführung der alten Funktion zur Umrechnung in SI-Einheiten

Bevor die Tabelle zur Versionierung der SQL-Funktionen angelegt werden kann, muss die PostgreSQL-Erweiterung „*btree_gist*“ geladen werden. Diese ist in aktuellen PostgreSQL-Installationen[1] enthalten und kann durch die Anweisung 5.4 aktiviert werden.

```
CREATE EXTENSION btree_gist
```

Anweisung 5.4: Aktivierung der PostgreSQL-Erweiterung *btree_gist*

Die Erweiterung erlaubt die Nutzung von sogenannten *Exclusion-Constraints* auf skalare Datentypen[2]. Dadurch kann sichergestellt werden, dass mehrere Versionen einer Funktion nur in jeweils einem Zeitraum gültig sein können, ohne dass es zu Überlappungen kommen darf. Es werden also Überschneidungen der Zeiträume unterschiedlicher Versionen einer Funktion ausgeschlossen.

Die Tabelle zur Versionierung der SQL-Funktionen ist der aus dem Konzept nachempfunden und ist in PostgreSQL Syntax in Anweisung 5.5 abgebildet.

```
CREATE TABLE routines_history (
  id SERIAL,
  routine_name VARCHAR(200),
  routine_new_name VARCHAR(200) DEFAULT '',
  routine_arguments VARCHAR(1000),
  routine_declaration TEXT,
```

[1] Benutzt wurde in dieser Arbeit Version 9.5.1
[2] Aus der Dokumentation: „...to define exclusion constraints on plain scalar data types...“ [Pos16d]

```
    routine_result VARCHAR(100),
    vt_begin TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6),
    vt_end TIMESTAMP(6) DEFAULT '9999-12-31 23:59:59',
    tt TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6),
    valid_time TSRANGE DEFAULT
      tsrange(CURRENT_TIMESTAMP(6)::TIMESTAMP, '9999-12-31 23:59:59'::TIMESTAMP, '[)'),
    EXCLUDE USING GIST (id WITH =, valid_time WITH &&), -- no overlapping
    CONSTRAINT routines_history_pk PRIMARY KEY (id, vt_begin, vt_end)
)
```

Anweisung 5.5: PostgreSQL Versionierungstabelle für UDFs/STPs

Zu erkennen ist hier das Attribut *valid_ time*, welche einen *Range*-Datentyp besitzt. Die dort enthaltenen Informationen sind redundant mit den Attributen *vt_ begin* und *vt_ end*. Trotzdem wurde an dieser Stelle der Übersicht halber die Darstellung mit den separaten Attributen gewählt. Die *valid_ time*- sowie *tt*-Attribute werden in folgenden Abbildungen dieser Relation nicht weiter dargestellt. Weiterhin auffällig ist die Zeile „EXCLUDE...“, die einen *Exclusion-Constraint* auf die Attribute *id* und den *Range*-Datentyp *valid_ time* definiert. Dadurch wird sichergestellt, dass es zu keinen sich überschneidenden Zeiträume unterschiedlicher Versionen der Funktionen mit gleicher *id* kommen kann.

Im Folgenden werden wieder die drei Fälle aus dem Konzept betrachtet: Anlegen einer neuen Funktion, Ersetzen einer bestehenden Funktion sowie das Löschen einer bestehenden Funktion.

5.1 Anlegen einer neuen Funktion

Davon ausgehend, dass bisher keine Auswertefunktion existiert, kann diese mit der Anweisung 5.1 angelegt werden. Zur Versionierung ist es erforderlich, diese in die entsprechende Tabelle einzutragen. Dazu existiert die Hilfsfunktion *routine_ create* (Anhang A.6), welche den Namen der soeben erstellten Funktion sowie optional den Startzeitpunkt der Gültigkeitszeit übergeben bekommt. Anhand des Funktionsnamens werden die weiteren Informationen, wie Argumente und deren Typen, die eigentliche Funktionsdefinition und der Rückgabewert mittels spezifischer PostgreSQL-Funktionen herausgesucht und in der Versionierungstabelle abspeichert. Wichtig ist die Verwendung des *Object Identifier*[3]-Aliases *regproc* im Selektionsprädikat. So muss nicht mehr anhand des Schemas selektiert werden, da dies automatisch durch die Implementierung des Aliases gehandhabt wird[4]. Der Aufruf erfolgt durch Anweisung 5.6, das Ergebnis entspricht Tabelle 4.1 aus dem Konzept.

[3] Von PostgreSQL genutzter interner Datentyp (int). Ausgezeichnete Werte besitzen Alias-Bezeichner: *regproc* z. B. bezeichnet den Funktionsnamen [Pos16c]

[4] Aus der Dokumentation: „...handles the table lookup according to the schema path setting, and so it does the 'right thing' automatically.“ [Pos16c]

```
SELECT routine_create('raw_to_units_press', '1960-01-01')
```

Anweisung 5.6: Speicherung der neu angelegten Funktion in der Versionierungstabelle

5.2 Ersetzen einer bestehenden Funktion

Ähnlich wie im Konzept gezeigt, besteht diese Operation aus drei Schritten: Versionierung der alten Funktion, Anlegen der neuen Funktion sowie bekannt machen der neuen Funktion und deren Informationen in der Versionierungstabelle. Dazu werden die Hilfsfunktionen *routine_ rename* (Anhang A.7) und *routine_ update_ after_ rename* (Anhang A.8) benutzt. Folgende drei Schritte müssen zum Ersetzen einer bestehenden Funktion ausgeführt werden:

1. Aufruf der *routine_ rename*-Funktion
2. Anlegen der neuen Funktion im DBMS
3. Aufruf der *routine_ update_ after_ rename*-Funktion

Der Aufruf der *routine_ rename*-Funktion ist in Anweisung 5.7 beispielhaft dargestellt. Als Parameter werden der alte Funktionsname, der neue Funktionsname sowie der Endzeitpunkt der Gültigkeitszeit erwartet.

```
SELECT routine_rename('raw_to_units_press', 'raw_to_units_press', '1978-09-06')
```

Anweisung 5.7: Aufruf der Hilfsfunktion *routine_ rename*

Die *routine_ rename*-Funktion benennt mittels dynamischen SQL die alte Funktion im DBMS unter Verwendung des im Konzept eingeführten Unix-Zeitstempels um. Zusätzlich führt sie zwei SQL-Anweisungen aus, die den SQL:2011-Standard simulieren. Dabei wird die alte Funktion derart versioniert, als dass der neue Funktionsname mit Zeitstempel, welcher dem DBMS bekannt ist, gespeichert und der Endzeitpunkt der Gültigkeitszeit auf den Zeitpunkt des optionalen Parameters *p_ vt_ end* gesetzt wird. Wird dieser Parameter weggelassen, wird der aktuelle Zeitpunkt des Aufrufs der *routine_ rename*-Funktion benutzt. Zusätzlich wird die neue Funktion mit aktuellem Gültigkeitszeitintervall angelegt.

Die weiteren Informationen bezüglich der Funktionsparameter, Berechnungsvorschrift und Rückgabewerte können erst nach dem Anlegen der neuen Funktion im DBMS mittels Anweisung 5.2 durch die *routine_ update_ after_ rename*-Funktion geliefert werden. Durch den Aufruf dieser Funktion mittels Anweisung 5.8 werden die nun vorhandenen Informationen aus dem DBMS ermittelt und der dazugehörigen Datensatz in der Versionierungstabelle aktualisiert. Als Parameter wird der Name der neuen Funktion erwartet. Nach Ausführung entsteht durch die verwendeten Beispieldaten die Tabelle 4.2.

```
SELECT routine_update_after_rename('raw_to_units_press')
```

Anweisung 5.8: Aufruf der Hilfsfunktion *routine_ update_ after_ rename*

5.3 Löschen einer bestehenden Funktion

Das temporale Löschen, d.h. das Setzen des Endzeitpunkts des Gültigkeitszeitintervalls auf einen Zeitpunkt größer als der Startzeitpunkt, wird analog zur Vorgehensweise im Konzept mittels der Funktion „*routine_ delete*“ (Anhang A.9) respektive Anweisung 5.9 durchgeführt. Als Parameter werden der Funktionsname sowie der Endzeitpunkt der Gültigkeitszeit erwartet.

```
SELECT routine_delete('raw_to_units_press', '2016-07-25 15:30:00')
```

Anweisung 5.9: Aufruf der Hilfsfunktion *routine_ delete*

Auch hier gilt, dass die Funktion bisher nicht aus dem DBMS selbst gelöscht wurde und dies ggf. manuell mittels der DROP-Anweisung und des Namens inklusive Unix-Zeitstempels ausgeführt werden kann. Nach der Ausführung der *routine_ delete*-Funktion entsteht Tabelle 4.3.

6 Zusammenfassung und Ausblick

In diesem Kapitel erfolgt die Zusammenfassung der vorliegenden Arbeit. Es wird aufbauend auf den Grundlagen beschrieben, was der Kontext und der Untersuchungsgegenstand dieser Arbeit war. Ferner wird auf die Ergebnisse eingegangen und dargelegt, was diese bedeuten. Im Ausblick werden weitere Themen und Aspekte angesprochen, die in weiterführenden Arbeiten behandelt werden können.

6.1 Zusammenfassung

Durch das (automatische) Erzeugen von einer Vielzahl an Forschungsrohdaten ist es notwendig, dass diese langfristig gespeichert und zur Nachnutzung bereitgestellt werden. Zur Sicherstellung bzw. Überprüfung von Forschungsergebnissen in wissenschaftlichen Publikationen müssen neben den Originaldaten auch die darauf ausgeführten Analyse- und Auswertefunktionen verfügbar sein. Dabei können die Verfahren, die zur Auswertung genutzt wurden, bereits veraltet und durch aktuellere Methoden ersetzt worden sein. Die Frage nach der Herkunft eines (wissenschaftlichen) Ergebnisses führt daher zur Ermittlung der zugrunde liegenden (Forschungs-)Daten und der darauf angewendeten Analysemethoden. Diese Art von Fragen, konkret nach der Herkunft von Daten bzw. dem Entstehen von wissenschaftlichen Ergebnissen durch angewendete Analysefunktionen, sollen durch Provenance-Management beantwortet werden. Dabei können sich die Art, das Format der Daten sowie die Auswertefunktionen über längere Zeiträume hinweg verändern, so dass eine Aufzeichnung dieser stattgefundenen Veränderungen notwendig wird, um eine nachvollziehbare Rekonstruierbarkeit zu gewährleisten.

Im Kontext des Leibniz-Instituts für Ostseeforschung Warnemünde (IOW), konkret der GODESS-Arbeitsgruppe, wurde untersucht, wie die Anwendbarkeit von veränderten Auswerte- und Analysemethoden auf sich verändernde Datenformate, konkret im Datenbankkontext die Schemata, gewährleistet werden kann. Da die Auswertefunktionen als Matlab-Skripte vorliegen und die automatische Transformation von Matlab in SQL nicht trivial zu bewerkstelligen ist, wurde eine Beispiel-SQL-Funktion konstruiert, welche im Konzept Verwendung findet. Anhand dieser Funktion wurde gezeigt, wie eine temporale Versionierung von SQL-Funktionen unter Einsatz des SQL:2011-Standards aussehen kann. Dazu wird neben den im DBMS vorhandenen Informationen eine eigene Versionierungstabelle gepflegt, welche alle Informationen über eine SQL-

Funktion, wie Name, Anzahl und Typ der Parameter sowie Rückgabewerte, beinhaltet. Es wurde aufgezeigt, welche SQL-Anweisungen zum Einfügen, Löschen und Aktualisieren von SQL-Funktionen benötigt werden, um die temporale Datenhaltung umzusetzen. Die Versionierung von SQL-Funktionen wurde im Prototyp anhand des DBMS PostgreSQL umgesetzt. Dabei wurden Hilfs-SQL-Funktionen erarbeitet, welche in einer bestimmten Reihenfolge ausgeführt werden müssen (siehe Kapitel 5). Die Vorgehensweise im Konzept respektive im Prototyp ist dabei als Prozess im Sinne des Forschungsdatenmanagements zu verstehen. Dies bedeutet, dass dieser Prozess im Rahmen vom Forschungsdatenmanagement eingehalten werden muss, damit eine korrekte und zweifelsfreie Versionierung und damit Sicherung der Nachweiskette von Daten und Funktionen gewährleistet werden kann.

Da sich darüber hinaus die Schemata über Zeiträume hinweg ändern können, wird hierfür in gleicher Weise ein Versionierungskonzept benötigt. Dies wurde in Abschnitt 4.1.2 nur skizziert und nicht weiter konkret umgesetzt bzw. untersucht. Für die nachfolgenden Untersuchungen der Arbeit wurde angenommen, dass ein Mechanismus existiert, der alle Änderungen der Schemata erfasst und Versionen von Schemata zu beliebigen Zeitpunkten liefern kann.

Schemaänderungen betreffen die im Schema enthaltenen Relationen und deren Attribute. Dazu gehören das Hinzufügen und Entfernen, das Umbenennen oder das Zusammenführen bzw. Teilen von Relationen und Attributen. Dabei wurde nur eine Schemaänderung pro Zeiteinheit untersucht. Anhand der Betrachtung dieser atomaren Veränderungen von Schemata sind Anwendbarkeitsvoraussetzungen für SQL-Funktionen entstanden. Diese sind in Tabelle 4.4 zusammenfassend dargestellt.

Funktionsänderungen betreffen die Signatur der Funktion, den Funktionsrumpf sowie die Rückgabewerte. Dabei wurden die Zusammenhänge von den Parametern der Signatur der Funktion, deren Rückgabewerte sowie die Verarbeitung im Funktionsrumpf beschrieben. Es wurden die Voraussetzungen, nach denen eine Funktion nach einer Änderung auf ein unverändertes Schema anwendbar ist, erarbeitet und in Tabelle 4.7 zusammengefasst.

Durch welche Techniken die Anwendbarkeit von Funktionen auf die beschriebenen Veränderungen von Schemata und Funktionen gewährleistet werden kann, wurde in Abschnitt 4.2.3 skizziert. Dort wurde beschrieben, wie anhand von Techniken des Schema-Mappings, unter Verwendung der Versionierung von Schemata, die (semi-)automatische Kompensation der Änderungen durchgeführt werden kann, um eine Funktion auf ein verändertes Schema anwenden zu können.

Die Kernaussage über die Anwendbarkeit von sich verändernden Funktionen auf verändernde Schemata ist, dass Funktionen nur unter speziellen Rahmenbedingungen automatisiert angepasst und somit weiterhin angewendet werden können. Ist dies nicht der Fall, kann in der Mehrzahl der Fälle eine neue Version der Funktion durch das Versionierungskonzept erzeugt werden, welche wieder auf das Schema anwendbar ist. Dadurch, dass eine Schemaänderung zu einer Funktions-

änderung und eine Funktionsänderung zu einer Schemaanpassung führen kann, bedingen sich die beiden Arten der Änderungen gegenseitig. Dies kann zu einem Henne-Ei-Problem führen bzw. eine Kette an wechselseitigen Veränderungen nach sich ziehen, bis die Anwendbarkeit der Funktion auf das Schema endgültig gegeben ist. In der Praxis wird wohl derjenige Fall sehr viel häufiger auftreten, als dass zuerst eine Schemaänderung erfolgt und danach gewisse Funktionen angepasst werden müssen, als der umgekehrte Fall. Durch die gegebene Anwendbarkeit von neueren Funktionen auf ältere Daten und umgekehrt ist es möglich, dass z. B. neuere Auswerteverfahren, die durch neue Erkenntnisse in der jeweiligen wissenschaftlichen Disziplin entstanden sind, auf alte Datenbestände angewendet werden können. Dadurch ist eine Vergleichbarkeit von Analyseverfahren und der Daten selbst gegeben, was zur Erhöhung der Qualität der wissenschaftlichen Arbeit beitragen kann. Können die Änderungen nicht durch Neuanlage bzw. Versionierung von Funktionen kompensiert werden, ist die Anwendbarkeit von Funktionen gar nicht gegeben.

Schließlich wurde in Abschnitt 4.3 der Zusammenhang von temporaler Datenhaltung, also der Versionierung von Funktionen und Schemata, sowie dem Aspekt der Provenance beschrieben. Dabei wird verdeutlicht, wie komplex die Sachverhalte sind und welche Herausforderungen dort bestehen.

In dieser Arbeit wurde der Fokus auf die Versionierung von SQL-Funktionen sowie auf die Anwendbarkeitsvoraussetzungen sich verändernder SQL-Funktion auf sich verändernde Schemata gelegt. Diese Untersuchungen können die Grundlage für weitere Arbeiten unter den Aspekten des Forschungsdatenmanagements und des Provenance-Managements in Verbindung mit temporaler Datenhaltung bilden.

6.2 Ausblick

Bei Betrachtung der Provenance unter realen Bedingungen kristallisiert sich heraus, dass extensionale Antworten den Anwender nur bedingt bei der Fragestellung nach der Herkunft seiner Daten unterstützen können. So liefert bereits die Antwort auf die Frage, wie eine Auswertung mittels einer Aggregation entstanden ist, extensional alle Datensätze der beteiligten Relationen, was u. U. einer sehr hohen Anzahl an Datensätzen entsprechen kann. Dem Anwender ist also nicht damit geholfen, dass diesem diese hohe Anzahl an Datensätzen präsentiert wird. Eine Lösung dafür können intensionale Antworten bieten. Die Herausforderung dabei besteht aber darin, wie aus dem Kontext der Anwendung heraus intensionale Antworten abgeleitet werden können. Dazu müssen Erklärungen und Beschreibungen in Form von Annotationen, Ontologien und Metadaten oder ggf. (externe) Dienste an das System gekoppelt werden, um aus diesen Informationen dem Anwender eine übersichtliche, aber korrekte Antwort liefern zu können. Die Herausforderung besteht dabei in der Kopplung bzw. dem Mapping der Informationen zu den einzelnen Relationen und Attributen. Dies macht die Erzeugung intensionaler Antworten komplex und kann und ist

bereits Bestandteil weiterer Arbeiten zu diesem Thema (vgl. dazu z. B. die Konzepthierarchien aus [Sva16]).

Eine weitere interessante Fragestellung bezüglich intensionaler Antworten ist, ob Techniken des Data-Mining aus dem Datawarehouse-Kontext, z. B. Assoziationsregeln oder Clusterung, die Erzeugung von intensionalen Antworten unterstützen können. Ziel kann es dabei sein, dem Nutzer nicht extensional alle beteiligten Daten zu präsentieren, sondern durch ähnliche Muster oder Auffälligkeiten in der großen Anzahl beteiligter Daten dem Anwender die interessanten Zusammenhänge intensional darzustellen.

Ein weiterer neuartiger Ansatz zur Analyse der Data-Provenance beliebiger (lesender) SQL-Anfragen stammt aus [Mü15]: Anfragen werden dabei in eine imperative Programmiersprache übersetzt und anschließend mit bekannten Techniken aus der Programmanalyse, wie *Program Slicing*, Kontrollflussanalyse und abstrakter Interpretation, untersucht. Dadurch soll es möglich sein, *Why*- und *Where*-Provenance auf der Granularitätsebene einzelner Tabellenzellen berechnen zu können. Durch Übersetzung der Anfrage in eine (Turing-vollständige) Programmiersprache soll die theoretische Anwendbarkeit auf beliebige Anfragen gegeben sein. Anhand dieses Ansatzes soll nur aufgezeigt werden, dass auch Arbeiten außerhalb des Datenbankkontextes sich der Provenance-Fragestellungen annehmen und es somit ein aktives Forschungsgebiet darstellt.

Obwohl die temporale Datenhaltung schon gut verstanden und durch den SQL:2011-Standard beschrieben ist, mangelt es an konkreten Umsetzungen gerade im Open-Source Bereich. Das hat zur Folge, dass temporale Klauseln durch mehrere einzelne SQL-Anweisungen - gekapselt in eigenständigen (SQL-)Funktionen - nachgebildet werden, um die temporale Datenhaltung zu simulieren. Das in dieser Arbeit verwendete DBMS PostgreSQL liefert mit den *Range*-Datentypen und der *btree_gist*-Erweiterung einen vielversprechenden Ansatz, dennoch fehlen die temporalen Klauseln. Dies hat die Kapselung der Funktionalität der temporalen Datenhaltung zur Folge. Die prototypische Umsetzung des Konzepts der Versionierung von Funktionen nutzt daher bereits eine solche Kapselung. Dies macht es notwendig, dass die Funktionen in einer bestimmten Reihenfolge aufgerufen werden müssen, um der Funktionalität der temporalen Versionierung nachzukommen. Wie in der Zusammenfassung bereits erwähnt, entspricht dies im Kontext des Forschungsdatenmanagements einem Prozess, der eingehalten werden muss, um die Versionierung zu gewährleisten. In dieser Arbeit wurde dabei auf Ebene des DBMS und von SQL-Funktionen gearbeitet. Durch Abstraktion und Weiterführung der Kapselung kann zukünftig eine externe Anwendung entstehen, die dem Anwender dabei unterstützend zur Verfügung steht und die Fehler bei Ausführung des Prozesses der Versionierung von Funktionen minimiert oder gar ganz ausschließt.

In dieser Arbeit wurde das Festhalten von Schemaänderungen über Zeiträume lediglich skizziert. Dabei ist dies selbst ein interessantes Forschungsgebiet, dem bei den Untersuchungen von Schemaevolution und Schemaabbildungen begegnet wird. Die Herausforderung dabei ist die Pro-

tokollierung und Historisierung der Schemaänderungen sowie die automatische Übersetzung von alten Anfragen auf neue Versionen. Der in dieser Arbeit skizzierte Ansatz basiert rein auf temporalen Datenhaltungstechniken und versucht Techniken zu nutzen, die bereits in der Praxis vorhanden sind. Die zu lösende Aufgabe dabei ist, die Versionierung entweder im DBMS selbst als eigene Versionierungstabellen umzusetzen oder die Technik in das DBMS zu implementieren, so dass eine Versionierung intern stattfinden kann. Weiterhin wäre ein Plugin oder eine Erweiterung denkbar. Darüber hinaus kann die in dieser Arbeit skizzierte *Snapshot*-Variante ggf. automatisiert durch sogenannte *Event-Trigger* umgesetzt werden. Diese Art von Trigger können auf DDL-Anweisungen angewendet werden. So kann z. B. auf eine ALTER TABLE-Anweisung prozedural reagiert und die Informationen in der Versionierungstabelle festgehalten werden.

Weiterhin wurden in dieser Arbeit nur atomare Schemaänderungen behandelt, welche eine Relation oder Attribut betreffen. Dabei sind Änderungen von Integritätsbedingungen nicht betrachtet worden. Die Komposition von Schemaänderungen sowie etwaige Änderungen von Integritätsbedingungen können in weiterführenden Arbeiten behandelt werden. Analog gilt bei den Untersuchungen von der Anwendbarkeit von Funktionen bei Funktionsänderungen: die Komposition von Funktionen und darauf aufbauende Änderungen sowie mengenwertige Datentypen wurden nicht betrachtet und können Bestandteil zukünftiger Arbeiten sein.

Die beschriebenen Kompensationsmöglichkeiten und Techniken gehören zu dem weiter oben bereits erwähnten Forschungsgebiet der Schemaevolution und -abbildung. Für die Automatisierung der Kompensation durch Übersetzung der alten Anfragen zur Anwendung auf ein neues Schema, ist es notwendig, dass die hier benutzten DDL-Anweisungen aus dem Log teilweise in DML-Anweisungen übersetzt werden müssen. Dazu gehören auch der Nachweis und das Erzeugen von inversen Abbildungen für diese Anweisungen. Diese und weitere Fragestellungen werden im Gebiet der Schemaevolution und deren Techniken der Schemaabbildungen behandelt.

Die Aspekte der temporalen Datenhaltung, des Provenance-Management sowie das Forschungsdatenmanagement bilden jeder für sich ein komplexes und eigenständiges Forschungsgebiet. Durch die Verknüpfung der unterschiedlichen Themen werden einerseits neue Lösungsmöglichkeiten aufgezeigt, andererseits neue Fragestellungen aufgeworfen, denen durch weitere, bereits aus anderen Forschungsgebieten bekannte, Techniken begegnet werden kann. Diese Komposition und Verknüpfung der unterschiedlichen Forschungsgebiete ist bereits Gegenstand laufender Forschung und kann zukünftig in weiteren wissenschaftlichen Projekten und Arbeiten untersucht werden.

Literaturverzeichnis

[AGR+14] Arab, Bahareh ; Gawlick, Dieter ; Radhakrishnan, Venkatesh ; Guo, Hao ; Glavic, Boris: A Generic Provenance Middleware for Database Queries, Updates, and Transactions. In: *Proceedings of the 6th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2014

[BH13] Bruder, Ilvio ; Heuer, Andreas: *Informationsintegration*. 2013. – Lehrstuhl Datenbank- und Informationssysteme, Institut für Informatik, Universität Rostock, Vorlesung Wintersemester 2012/2013

[BHM11] Büttner, Stephan ; Hobohm, Hans-Christoph ; Müller, Lars: *Handbuch Forschungsdatenmanagement*. 1. Aufl. Bad Honnef : Bock + Herchen, 2011. – ISBN 978–3–883–47283–6

[BKM+16] Bruder, Ilvio ; Klettke, Meike ; Möller, Mark L. ; Meyer, Frank ; Heuer, Andreas ; Jürgensmann, Susanne ; Feistel, Susanne: Daten wie Sand am Meer - Datenerhebung, -strukturierung, -management und Data Provenance für die Ostseeforschung / Lehrstuhl Datenbank- und Informationssysteme, Institut für Informatik, Universität Rostock. 2016. – Forschungsbericht

[BKT00] *Kapitel* Invited Presentations. In: Buneman, Peter ; Khanna, Sanjeev ; Tan, Wang-Chiew: *Data Provenance: Some Basic Issues*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2000. – ISBN 978–3–540–44450–3, 87–93

[BKWC01] *Kapitel* Why and Where: A Characterization of Data Provenance. In: Buneman, Peter ; Khanna, Sanjeev ; Wang-Chiew, Tan: *Why and Where: A Characterization of Data Provenance*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2001. – ISBN 978–3–540–44503–6, 316–330

[Bor16a] Boris Glavic, IIT DBGroup: *IIT DBGroup / research / gprom*. http://www.cs.iit.edu/~dbgroup/research/gprom.php. Version: September 2016

[Bor16b] Boris Glavic, IIT DBGroup: *IIT DBGroup / research / oracletprov*. http://www.

cs.iit.edu/~dbgroup/research/oracletprov.php. Version: September 2016

[BT07] Buneman, Peter ; Tan, Wang-Chiew: Provenance in Databases. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 2007 (SIGMOD '07). – ISBN 978–1–59593–686–8

[CAB+14] Carata, Lucian ; Akoush, Sherif ; Balakrishnan, Nikilesh ; Bytheway, Thomas ; Sohan, Ripduman ; Seltzer, Margo ; Hopper, Andy: A Primer on Provenance. In: *Commun. ACM* 57 (2014), Mai, Nr. 5, 52–60. http://dx.doi.org/10.1145/2596628. – DOI 10.1145/2596628. – ISSN 0001–0782

[CCT09] Cheney, James ; Chiticariu, Laura ; Tan, Wang-Chiew: Provenance in Databases: Why, How, and Where. In: *Found. Trends databases* 1 (2009), April, Nr. 4, 379–474. http://dx.doi.org/10.1561/1900000006. – DOI 10.1561/1900000006. – ISSN 1931–7883

[CJ09] Chapman, Adriane ; Jagadish, H. V.: Why Not? In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 2009 (SIGMOD '09). – ISBN 978–1–60558–551–2

[CMDZ10] Curino, Carlo A. ; Moon, Hyun J. ; Deutsch, Alin ; Zaniolo, Carlo: Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++. In: *Proc. VLDB Endow.* 4 (2010), November, Nr. 2, 117–128. http://dx.doi.org/10.14778/1921071.1921078. – DOI 10.14778/1921071.1921078. – ISSN 2150–8097

[Dic16a] Dictionaries, Oxford: *provenance. Oxford Dictionaries.* http://www.oxforddictionaries.com/definition/english/provenance. Version: September 2016

[Dic16b] Dictionary, Oxford E.: *provenance, n. : Oxford English Dictionary.* http://www.oed.com/view/Entry/153408. Version: September 2016

[Fre09] Freire, Juliana: Provenance Management: Challenges and Opportunities. In: *GI-Edition - Lecture Notes in Informatics (LNI), Datenbanksysteme in Business, Technologie und Web (BTW)* (2009), März, Nr. P-144, S. 4. ISBN 978–3–88579–238–3

[GA09] Glavic, Boris ; Alonso, Gustavo: The Perm Provenance Management System in Action. In: *Proceedings of the 35th ACM SIGMOD International Conference on Management of Data (SIGMOD) (Demonstration Track)*, 2009

[GKT07] Green, Todd J. ; Karvounarakis, Grigoris ; Tannen, Val: Provenance Semirings. In: *Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA : ACM, 2007 (PODS '07). – ISBN 978–1–59593–685–1

[Gla10] Glavic, Boris: *Perm: Efficient Provenance Support for Relational Databases*, University of Zurich, Diss., 2010. http://cs.iit.edu/%7edbgroup/pdfpubls/G10a.pdf

[Gla14] Glavic, Boris: A Primer on Database Provenance / Illinois Institute of Technology. Version: 2014. http://cs.iit.edu/%7edbgroup/pdfpubls/G14.pdf. 2014. – Forschungsbericht

[GMA13] *Kapitel* Using SQL for Efficient Generation and Querying of Provenance Information. In: Glavic, Boris ; Miller, Renée J. ; Alonso, Gustavo: *Using SQL for Efficient Generation and Querying of Provenance Information*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. – ISBN 978–3–642–41660–6, 291–320

[Her11] Herschel, Melanie: *Datenintegration & Datenherkunft: Varianten der Data Provenance*. 2010/2011. – Lehrstuhl für Datenbanksysteme, Universität Tübingen, Vorlesung Wintersemester 2010/11

[Heu15] Heuer, Andreas: METIS in PArADISE Provenance Management bei der Auswertung von Sensordatenmengen für die Entwicklung von Assistenzsystemen. In: *BTW Workshops*, 2015

[Heu16] Heuer, Andreas: *Neueste Entwicklungen in der Informatik: Datenbanken: Übung zu METIS: Provenance Management*. 2016. – Lehrstuhl Datenbank- und Informationssysteme, Institut für Informatik, Universität Rostock, Vorlesung Sommersemester 2016

[HH10] Herschel, Melanie ; Hernández, Mauricio A.: Explaining Missing Answers to SPJUA Queries. In: *Proc. VLDB Endow.* 3 (2010), September, Nr. 1-2, 185–196. http://dx.doi.org/10.14778/1920841.1920869. – DOI 10.14778/1920841.1920869. – ISSN 2150–8097

[KM12] Kulkarni, Krishna ; Michels, Jan-Eike: Temporal Features in SQL:2011. In: *SIGMOD Rec.* 41 (2012), Oktober, Nr. 3, 34–43. http://dx.doi.org/10.1145/2380776.2380786. – DOI 10.1145/2380776.2380786. – ISSN 0163–5808

[Lei16] Leibniz-Institut für Ostseeforschung Warnemünde: *Das Leibniz-Institut für Ostseeforschung Warnemünde*. http://www.io-warnemuende.de/. Version: September 2016

[Les16] Leser, Ulf: *Informationsintegration*. 2016. – Institut für Informatik, Mathematisch-Naturwissenschaftliche Fakultät, Humboldt-Universität zu Berlin, Vorlesung Sommersemester 2016

[Mü15] Müller, Tobias: Where-und Why-Provenance für syntaktisch reiches SQL durch Kombination von Programmanalysetechniken. In: *Proceedings of the 27th GI-Workshop Grundlagen von Datenbanken, Gommern, Germany, May 26-29, 2015*, 2015

[Mö16] Möller, Mark L.: *Aufbau einer Forschungsdatenverwaltung für chemische und physikalische In-Situ-Daten aus der Ostsee*, Universität Rostock, Bachelorarbeit, 2016

[Mey15] Meyer, Frank: *Aufbau einer Artenlistenverwaltung im Benthos-Projekt*, Universität Rostock, Bachelorarbeit, 2015

[Mic16] Microsoft: *Temporal Tables SQL Server 2016*. dn935015. Redmond, Vereinigte Staaten, September 2016. https://msdn.microsoft.com/en-us/library/dn935015.aspx

[MK16] Murray, Chuck ; Kyte, Tom: *Oracle Database Development Guide, 12c Release 1 (12.1)*. E41452-07. Redwood City, Vereinigte Staaten, September 2016. https://docs.oracle.com/database/121/ADFNS/adfns_design.htm#ADFNS967

[Mot94] Motro, A.: Intensional answers to database queries. In: *IEEE Transactions on Knowledge and Data Engineering* 6 (1994), Jun, Nr. 3, S. 444–454. http://dx.doi.org/10.1109/69.334858. – DOI 10.1109/69.334858. – ISSN 1041–4347

[MyS16] MySQL / Oracle Corporation: *MySQL :: MySQL 5.7 Reference Manual :: 22 INFORMATION_SCHEMA Tables*. http://dev.mysql.com/doc/refman/5.7/en/information-schema.html. Version: September 2016

[NS12] Nicola, Matthias ; Sommerlandt, Martin: *Managing time in DB2 with temporal consistency*. Internet/PDF. http://www.ibm.com/developerworks/data/library/techarticle/dm-1207db2temporalintegrity/. Version: Juli 2012

[NSO+12] Neuroth, Heike ; Strathmann, Stefan ; Osswald, Achim ; Scheffel, Regine ; Klump, Jens ; Ludwig, Jens: *Langzeitarchivierung von Forschungsdaten - eine Bestandsaufnahme*. 1. Aufl. Boizenburg, Glückstadt : Hülsbusch, 2012. – ISBN 978–3–864–88008–7

[PIW11] Park, Hyunjung ; Ikeda, Robert ; Widom, Jennifer: RAMP: A System for Capturing and Tracing Provenance in MapReduce Workflows. (2011), August. http://ilpubs.stanford.edu:8090/995/

[Pos16a] PostgreSQL: *SQL2011Temporal - PostgreSQL wiki*. https://wiki.postgresql.org/wiki/SQL2011Temporal. Version: September 2016

[Pos16b] PostgreSQL Extension Network: *temporal_tables: Temporal Tables Extension / PostgreSQL Extension Network*. http://pgxn.org/dist/temporal_tables/. Version: September 2016

[Pos16c] PostgreSQL Global Development Group: *PostgreSQL: Documentation: 9.5: Object Identifier Types*. https://www.postgresql.org/docs/current/static/datatype-oid.html. Version: September 2016

[Pos16d] PostgreSQL Global Development Group: *PostgreSQL: Documentation: 9.5: Range Types*. https://www.postgresql.org/docs/current/static/rangetypes.html#RANGETYPES-CONSTRAINT. Version: September 2016

[Pos16e] PostgreSQL Global Development Group (Vlad Arkhipov): *Temporal features in PostgreSQL*. http://www.postgresql.org/message-id/50D99278.3030704@dc.baikal.ru. Version: September 2016

[PSB11] Prien, R. D. ; Schulz-Bull, D. E.: The Gotland Deep Environmental Sampling Station in the Baltic Sea. In: *OCEANS 2011 IEEE - Spain*, 2011

[PSB16] Prien, R. D. ; Schulz-Bull, D. E.: Technical note: GODESS – a profiling mooring in the Gotland Basin. In: *Ocean Science* 12 (2016), Nr. 4, 899–907. http://dx.doi.org/10.5194/os-12-899-2016. – DOI 10.5194/os–12–899–2016

[SGM00] Snodgrass, Richard ; Gray, Jim ; Melton, Jim: *Developing Time-oriented Database Applications in SQL*. Pap/Com. Morgan Kaufmann Publishers, 2000. – ISBN 978–1–558–60436–0

[Sir09] *Kapitel* Positive Relational Algebra. In: Sirangelo, Cristina: *Encyclopedia of Database Systems*. Boston, MA : Springer US, 2009. – ISBN 978–0–387–39940–9, 2124–2125

[Sno95] Snodgrass, Richard T. (Hrsg.): *The TSQL2 Temporal Query Language*. Kluwer, 1995. – ISBN 0–7923–9614–6

[SSH13] Saake, G. ; Sattler, K.U. ; Heuer, A.: *Datenbanken: Konzepte und Sprachen*. mitp/bhv, 2013. – ISBN 978–3–8266–9453–0

[Sva16] Svacina, Jan: *Intensional Answers for Provenance Queries in Big Data Analytics*, Universität Rostock, Bachelorarbeit, 2016

[Weu16] Weu, Dennis: *Erkennung, Übersetzung und parallele Auswertung von Operatoren und Funktionen aus R in SQL*, Universität Rostock, Bachelorarbeit, 2016

[Wik16] Wikipedia: *Provenance — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Provenance&oldid=723782124. Version: Juni 2016

[YP99] Yoon, Suk-Chung ; Park, E. K.: An approach to intensional query answering at multiple abstraction levels using data mining approaches. In: *Systems Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on* Bd. Track6, 1999

A Anhang

A.1 Einführung - GODESS

A.1.1 Logger-Dateien

```
[Spectrum]
Version=1
IDData=2015-06-29_17-58-12_608
IDDevice=ProPS_D016
IDDataType=SPECTRUM
IDDataTypeSub1=RAW
IDDataTypeSub2=Dark
DateTime=2015-06-29 17:58:12
Comment=SmartDI v6.13.2010 (2014-1
CommentSub1=2015-06-29 17:57:51.24
CommentSub2=1 spectrum number
CommentSub3=21.3s meastime

[Attributes]
CalFactor=1993
IntegrationTime=512
Unit1=$05 $00 Pixel
Unit2=$03 $05 Intensity Counts
Unit3=$f0 $05 Error counts
Unit4=$f1 $00 Status
[END] of [Attributes]

[DATA]
0 9 0 0
1 1342 0 0
2 1321 0 0
3 1318 0 0
4 1314 0 0
5 1307 0 0
6 1324 0 0
7 1310 0 0
8 1315 0 0
9 1322 0 0
10 1329 0 0
11 1314 0 0
12 1334 0 0
13 1335 0 0
14 1330 0 0
15 1338 0 0
16 1382 0 0
17 1405 0 0
18 1493 0 0
19 1622 0 0
20 1823 0 0
21 2112 0 0
22 2558 0 0
23 3156 0 0
```

Abbildung A.1: Ausschnitt Spektraldatensensor-Datei Deployment 14

```
|----------------------------------------
@FILETITEL ASCII-DATA
@FILENAME  C:\ASCIIDAT\A150913A.TXT
@COPYRIGHT
@DATETIME  U1442113053  "2015-09-13" "02:57:33"
@PROGRAM   SmartDI v6.13.2010 (2014-10-15) C:\SDI\SDI\SMARTDI.EXE /pc\sdi\data\SDIIOW2
@VERSION   DPMI  $FC303DA9
@CONTENTS
|----------------------------------------
@METASTART
@CHANNEL,_,_,_,_,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59
@NAME,DATE,TIME,FRAC,SEC,VBatt,Press,Temp,Cond,RawO2,pH,ChlA,Redox,Ti,Turbidity,Polym,Tur_1,Tur_2,Tur_3,Tur_4,CTD-Ch15,ProPS-Normalized,ProPS-Lamp,Datetime,Heading,Pitch,Roll,Temp.,Press
@UNIT,YY-MM-DD,HH:MM:SS,MS,RUNSEC,V,dBar,øC,mS/cm,,,,,,,,,,,,,,,,,deg,deg,deg,øC,Bar,V,m/s,,V,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
@IFTYPE,_,_,_,_,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,COM,NONE,NONE,
@IFNO,_,_,_,_,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,1,1,2,2,2,2,2,2,2,2,2,2,9,9,9,9,9,9,9,9,9,9,9,0,0,0,0,0,0,0,0,0,0,0,10,10,10,10,10,10,10,10,10,10,10,3,3,3,3,3,3
@ACQMODE,_,_,_,_,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS,MEAS
@ACQSTATE,_,_,_,_,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,STOP,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,RUN,STOP,RUN,STOP,STOP,STOP,STOP,STOP,STOP,STOP,STO
@RATE,_,_,_,_,-200,-200,-200,-200,-200,-200,-200,-200,-200,-200,-200,-200,-200,-200,-200,-200,-500,-500,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
@FORMULA,_,_,_,_,LIN,POL3,POL3,POL3,POL3,POL3,POL3,POL3,POL3,POL3,POL3,POL3,POL3,POL3,POL3,POL3,LIN,LIN,LIN,LIN,LIN,LIN,LIN,LIN,LIN,LIN,LIN,LIN,LIN,RAW,RAW,RAW,RAW,RAW,RAW,RAW,RAW,RAW,RA
@METAEND
|----------------------------------------
@DATASTART "2015-09-13" "02:57:33"
$PSDA1,2015-09-13,02:57:33,669,10,30067,61063.000,50423.000,15573.000,14428.000,39438.000,40307.000,4055.000,26288.000,12729.000,811.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:53,251.0,
$PSDA1,2015-09-13,02:57:34,152,11,30073,61073.000,50426.000,15571.000,14429.000,39442.000,40308.000,4102.000,26288.000,12729.000,835.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:54,250.7,
$PSDA1,2015-09-13,02:57:35,672,11,30075,61075.000,50426.000,15571.000,14428.000,39443.000,40307.000,4030.000,26288.000,12731.000,923.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:55,251.1,
$PSDA1,2015-09-13,02:57:35,094,11,30076,61055.000,50425.000,15572.000,14428.000,39439.000,40308.000,4075.000,26288.000,12730.000,963.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:55,251.1,
$PSDA1,2015-09-13,02:57:36,421,12,30078,61071.000,50427.000,15572.000,14428.000,39438.000,40309.000,4105.000,26289.000,12729.000,923.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:56,251.0,
$PSDA1,2015-09-13,02:57:36,909,12,30080,61059.000,50426.000,15572.000,14428.000,39441.000,40308.000,4159.000,26288.000,12730.000,871.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:56,251.0,
$PSDA1,2015-09-13,02:57:36,243,12,30081,61069.000,50424.000,15570.000,14428.000,39439.000,40307.000,4224.000,26288.000,12730.000,823.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:56,251.0,
$PSDA1,2015-09-13,02:57:36,243,13,30081,61069.000,50424.000,15570.000,14428.000,39439.000,40307.000,4224.000,26288.000,12730.000,823.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:57,250.9,
$PSDA1,2015-09-13,02:57:37,813,13,30083,61077.000,50422.000,15572.000,14428.000,39441.000,40308.000,4226.000,26289.000,12731.000,823.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:57,250.9,
$PSDA1,2015-09-13,02:57:37,813,13,30083,61077.000,50422.000,15572.000,14428.000,39441.000,40308.000,4226.000,26289.000,12731.000,823.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:57,250.9,
$PSDA1,2015-09-13,02:57:37,077,13,30084,61055.000,50424.000,15572.000,14428.000,39441.000,40308.000,4204.000,26289.000,12730.000,819.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:57,250.9,
$PSDA1,2015-09-13,02:57:37,295,14,30085,61071.000,50424.000,15571.000,14428.000,39441.000,40309.000,4241.000,26289.000,12730.000,779.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:58,250.7,
$PSDA1,2015-09-13,02:57:38,750,14,30087,61079.000,50423.000,15573.000,14429.000,39438.000,40308.000,4321.000,26289.000,12730.000,791.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:58,250.7,
$PSDA1,2015-09-13,02:57:38,750,14,30087,61079.000,50423.000,15573.000,14429.000,39438.000,40308.000,4321.000,26289.000,12730.000,791.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:58,250.7,
$PSDA1,2015-09-13,02:57:38,007,14,30088,61055.000,50424.000,15571.000,14428.000,39439.000,40308.000,4307.000,26289.000,12730.000,795.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:58,250.7,
$PSDA1,2015-09-13,02:57:38,231,15,30089,61067.000,50425.000,15571.000,14428.000,39439.000,40308.000,4327.000,26288.000,12731.000,783.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:59,251.0,
$PSDA1,2015-09-13,02:57:39,708,15,30091,61077.000,50426.000,15571.000,14428.000,39439.000,40308.000,4421.000,26288.000,12730.000,791.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:59,251.0,
$PSDA1,2015-09-13,02:57:39,708,15,30091,61077.000,50426.000,15571.000,14428.000,39439.000,40308.000,4421.000,26288.000,12730.000,791.000,0.000,0.000,0.000,0.000,0.000,0,0,02:56:59,251.0,
```

Abbildung A.2: Ausschnitt CTD-Datei Deployment 15 mit Meta- und kommaseparierten Nutzdaten

```
2015-06-02 18:01:46 A521C4002946120702010000000007300A0380E05F5FF15000310DC17B40229FE0EFF00FFAAFEF6FE1AFF8CFE2BFF17FE85FECBFF60FFBAFE46FFEAFEC6FE75FE42FF9EFE85FEF9FE45FF70FECBFE1AFEEFFEF4F
2015-06-02 18:01:48 A521C4002947120702010000000007300A0380C05F7FF150003100A17B402B5FEBAFF6BFF74FEB8FFABFE61FF46FE05FF2EFE94FE09FFC2FF90FF7BFF5FFF05FFC0FEF3FE9DFD32FF52FF59FEAAFEF8FE94FE2CF
2015-06-02 18:01:49 A521C4002948120702010000000007400A0380705F8FF160003101616B402CDFE4000D4FE56FF82FF64FF5EFF20FF9AFF4BFF0CFEF5FEFDFE44FF2FFF6AFEBAFFEBFE35FF79FFB2FF18FF2CFFA4FE53FE1AFE95F
2015-06-02 18:01:50 A521C4002949120702010000000007400A0380305F6FF160003101715B402A7FF92FECCFF43FF5AFE10FFC2FE51FF0AFF90FED5FE95FF7FFF13FF7EFE0DFFCFFFB4FF93FE27FFD4FF36FF79FF6200E3FDB5FE41F
2015-06-02 18:01:51 A521C4002950120702010000000007400A0380005F6FF170003104B14B402C6FEB0FEAFFEE4FE89FE3BFE44FF22FF7AFF90FEBFFEB5FE37FFC7FE2DFED6FEDCFF79FE04FF52FC5DFFC5FE0FFFC4FF79FE74FEE3F
2015-06-02 18:01:52 A521C4002951120702010000000007400A038FA04F7FF170003106B13B402FBFD1CFF16FFDEFED2FE06FFFCFEE6FEE4FED6FE93FEBDFE9AFFE7FE90FE55FEB2FEB2FEEDFE56FFF8FEAFFFE5FE21FF96FFA5FF5DF
2015-06-02 18:01:53 A521C4002952120702010000000007400A038F804F8FF170003107712B4027EFFBDFF14FF30FF8AFE53FE0AFFADFF00FFE2FE14FF70FE40FFAFFEDFFEDAFED1FEB5FF7AFF5DFFF4FEB8FECAFF56FEECFE09FE7B0
2015-06-02 18:01:54 A521C4002953120702010000000007400A038F704F6FF180003109411B40254FF88FEB0FFC9FE34FF67FF2AFFB0FE3EFF6BFF00FFFDFE44FEF3FF7BFEBFFED2FE3DFF4FFF4FFF00FF85FE51FE39FF95FE6CFF2CF
2015-06-02 18:01:55 A521C4002954120702010000000007400A038FE04F6FF18000310AF10B40290FE1BFF5EFE3AFF7BFE90FFD5FE56FF37FF1DFFFBFEC8FF15FF26FFE2FEB5FF5EFFBFFEA5FEF6FE99FD1BFEDDFE1EFF39FFE7FF9BF
2015-06-02 18:01:56 A521C4002955120702010000000007300A0380405F6FF170003310BE0FB4023EFFC8FE05FF92FFF7FE9BFFD3FEE8FEABFF47FF8DFFA9FE39FFE4FE14FFB0FE1DFF67FF70FFB8FE3AFF82FED0FE69FF15FF18FF6AF
2015-06-02 18:01:57 A521C4002956120702010000000007400A0381105F6FF18000310DB0EB4022FFF27FE2BFF7BFE3FFF0DFE13FF2AFF84FEDDFCE9FE13FF09FF52FF09FFA9FF33FF98FE06FF97FFE3FE0EFF2AFF51FF1FFE36FFB9F
2015-06-02 18:01:58 A521C4002957120702010000000007300A0381E05F5FF18000310ED0DB40242FFEEFE7BFE0DFFFEFE67FEFDFE23FFCEFE13FF28FF81FE47FFBDFE3BFF1AFF4CFFFEFEA8FE93FFFDFEF6FD5CFF14FF1CFEC7FE4CF
2015-06-02 18:01:59 A521C4002958120702010000000007400A0383005F6FF170003100F0DB4027DFE3CFF95FF62FE0AFF6AFE87FE2DFF28FF2AFFA5FF28FFDBFEDEFE55FF9DFF76FFCAFE7CFEFAFED0FE47FE9FFFA3FF36FE07FF6CF
2015-06-02 18:02:00 A521C4002959120702010000000007400A0384305F5FF17000310530CB402D3FD63FFE0FE47FE7BFEFBFEDBFF21FEC2FEFBFECDFF51FFA8FF21FEB0FE40FFDAFF8BFF96FE77FEF5FD6AFFFBFE00FFACFE52FF11F
2015-06-02 18:02:01 A521C4003000120702010000000007300A0385805F5FF170003106C0BB40254FEEBFE0EFEF6FE16FFEEFE31FFA1FE35FEFDFE33FFB5FD62FF94FE49FFEFFF22FFC0FED1FE51FF6AFFCAFED1FE4FFF12FF4FFE57F
2015-06-02 18:02:02 A521C4003001120702010000000007300A0386D05F5FF170003106E0AB4023DFF26FF56FFE3FE5EFF4CFE5CFFC7FFD7FE56FE65FEFFFE31FF96FFF4FE94FE34FFE4FEBDFF87FF8FFFE5FE73FE7CFE88FEDCFE0AF
2015-06-02 18:02:03 A521C4003002120702010000000007300A0388605F4FF180003109509B402D2FF44FF54FF72FE85FED6FEB8FF9FFFD7FE73FE3AFF16FFB6FF5BFE6AFE72FF86FD23FFD0FE06001BFF38FE9CFF86FF2B0083FF1C0
2015-06-02 18:02:04 A521C4003003120702010000000007300A038A605F3FF17000310B808B4027FFEBDFED8FE37FF57FF28FFECFF57FF78FFF7FEB1FF82FE14FFDEFF34FEE5FE45FF4EFF83FF20FF19FF9DFEF7FE000050FFB3FF05F
2015-06-02 18:02:05 A521C4003004120702010000000007300A038BD05F3FF18000310DA07B402D7FE76FE38FF80FEE7FE10FF36FF45FF1CFF83FE06FF24FF89FEABFE5EFF02FF62FFE7FE3EFF82FE64FE6AFEB5FE2EFE02FF08FE88F
2015-06-02 18:02:06 A521C4003005120702010000000007300A038DC05F2FF170003100207B402F7FE3BFF24FEE2FE82FF00FFA8FE2300D2FE4BFE9BFEFCFDAEFE98FE89FE53FF0AFFB9FF7EFF34FFAEFF01FFF7FD60FE1BFDAFFEBEF
2015-06-02 18:02:07 A521C4003006120702010000000007300A038F805F2FF170003101506B402E0FEFAFE7EFD33FE54FE57FFCCFE2CFFB5FED7FEBCFE9FFF8FFE81FEBEFEA4FE02FF2AFE11FF23FF5AFEFEFEB2FE41FFE9FDD3FF1EF
2015-06-02 18:02:08 A521C4003007120702010000000007300A0381606F2FF170003102B05B40209FF9EFF47FF0C00B5FEFFFEFDFE6CFFEAFE3FFFC3FEDFFEEBFEADFEFEFE70FF05FFCAFEF6FE3EFE01FF4BFF0CFFC0FFFBFE61FE2DF
2015-06-02 18:02:09 A521C4003008120702010000000007300A0383406F1FF180003105404B402E9FE67FF2FFFDDFE05FFC0FEBBFE8CFE07FE13FFFAFEC4FEBBFF77FEEBFE2CFFE0FEFDFEF6FEEAFED5FE9E000CFFF7FEF8FE88FFB9F
2015-06-02 18:02:10 A521C4003009120702010000000007400A0384B06F1FF180003108303B402E3FE9EFF31FF48FF28FF95FEB8FFCCFEF5FE1CFFEAFE16FF06FF47FE38FF6CFF60FE53FF3BFFD8FE95FEE7FEC5FF64FF05FF38FF4BF
2015-06-02 18:02:11 A521C4003010120702010000000007400A0386606F0FF180003109302B402CDFEAEFEB1FF0CFEE5FE54FFB5FEA7FF2AFF5CFF1EFF34FF8AFFFAFE4DFF54FF1EFFDDFE08FFC4FE93FE39FFA1FF82FFCDFEF8FEFFF
2015-06-02 18:02:12 A521C4003011120702010000000007300A0388306EFFF17000310B601B40239FFEEFEC9FE5AFF9FFE79FF08FF68FF3CFEB7FE09FFF2FD36FEDEFE81FFFDFE0BFF93FEB5FEA9FEBAFE7AFF59FE6BFDEBFE22FF62F
2015-06-02 18:02:13 A521C4003012120702010000000007400A0389B06F0FF18000310DA00B402F4FF7EFF7C0058FFABFE08FF06FF5DFF6CFF5BFF19FF26FFF3FE40FF8BFFE0FF82FFD5FDE9FEDEFE47FF6FFEE7FEC4FECEFE18FFDAF
2015-06-02 18:02:14 A521C4003013120702010000000007400A038B406EFFF170003100200B40233FF83FE30FF69FFCBFE16FF6DFE52FFBDFEF2FEB0FE0BFF12FF2AFE45FFFAFE0CFF7FFE9CFF95FF88FEBBFEEBFE6CFF09FF6BFE4FF
2015-06-02 18:02:15 A521C4003014120702010000000007400A038CD06EFFF1700021025FFB4022AFE24FFB2FFB4FE12FFDCFF12FF8FFFEDFEA2FE92FF8CFFCBFEB6FEB8FE0000F8FEFEFE6EFFACFE5DFD6CFE31FF5AFEACFEE2FECFF
2015-06-02 18:02:16 A521C4003015120702010000000007400A038E706EEFF160002103CFEB40245FFBCFF3AFF3EFF9BFE1FFF41FFE0FDF7FEFBFE41FFBDFFFBFF2AFFFCFE0BFF0CFF24FF81FF2BFFEFFE11FF40FEB0FD12FEB3FDA4F
2015-06-02 18:02:17 A521C4003016120702010000000007300A038F906EEFF1700021064FDB4021BFF83FE89FFAEFE22FFBBFEA5FEE0FEEFFE78FE8AFFAAFF0DFFC4FFA9FE9DFE0E0091FEABFE2FFFB7FF38FFEBFE0E00CDFEBAFEB9F
2015-06-02 18:02:18 A521C4003017120702010000000007300A0381107EEFF1700021084FCB40227FF3AFF79FFDEFEB1FE80FF74FF0FFF23FF71FE16FF02FF24FED0FE7EFF62FFDEFE5FFF43FF13FEA0FE70FE77FFA5FF05FFECFED9F
2015-06-02 18:02:19 A521C4003018120702010000000007300A0382507EDFF170002109DFBB402D6FF8AFF63FFA1FE9DFF7EFF5CFF7EFFB6FFA8FEDDFE94FFECFD65FF0AFE4AFF36FF7BFE7BFEF3FEC8FEB7FF58FF93FF4CFD1AFFF2F
2015-06-02 18:02:20 A521C4003019120702010000000007300A0383B07EDFF16000210B6FAB402C9FEB3FE91FE24FFABFEA8FE6CFEF8FE59FF0FFFACFE55FFF1FE31FFEFFEA6FE60FFCDFFDCFE34FFBFFEE8FE5EFFA1FFD7FEB6FFE7F
2015-06-02 18:02:21 A521C4003020120702010000000007300A0384D07ECFF17000210C1F9B40239FF01FF05FF64FF7BFE6BFED7FEEFFE0DFF68FFA0FF52FF0DFFA9FF3CFF1DFFD4FE72FF40FFA3FF61FFFAFE5800DBFF95FD100015F
2015-06-02 18:02:22 A521C4003021120702010000000007300A0386107ECFF16000210EDF8B40259FF07FF4BFE8AFF0FFFEDFE53FFF5FD7AFF37FEB9FF77FE72FEA6FE12FF45FF52FF80FF82FFE2FE8DFF3BFF8EFEF3FEAEFF2CFEF1F
2015-06-02 18:02:23 A521C4003022120702010000000007300A0387107ECFF1600021024F8B40282FE82FE12FF26FF7EFF64FFE2FEC4FEE6FE21FFC9FE5BFE7BFE84FE00FFD2FE24FF7EFF1EFF4FFE71FEFFFE9DFE00FFCEFEEAFD1EF
2015-06-02 18:02:24 A521C4003023120702010000000007300A0388607ECFF1500021022F7B402BBFFCBFF9FFE2CFF97FEF8FE2CFF60FE05FFF8FE08FFDEFE59FFFFFE9DFE53FEF4FE52FF9CFEA8FEC8FE08FEF1FE25FFDCFD1BFED0F
2015-06-02 18:02:25 A521C4003024120702010000000007300A0389007ECFF1600021046F6B402EEFEFAFE86FFBFFE23FF100056FE42FE21FF79FE4300A4FEC3FE23FFB8FF5FFF02FF10FFB5FE2EFF47FF48FF2DFF07FF1BFF8BFE230
2015-06-02 18:02:26 A521C4003025120702010000000007400A038A207EBFF1600021058F5B402AFFFF4FEBDFF2600CCFEACFFEEFED0FFCFFE66FFE5FE81FE5CFF6CFF0DFF74FF63FF6AFFD4FE2EFEF5FDA5FE16FEE8FEA9FEB7FE6CF
2015-06-02 18:02:27 A521C4003026120702010000000007300A038AD07EBFF1500021079F4B40205FF8DFEFDFEC7FE2BFFCBFFB3FE7AFF3DFF16FEB1FF87FFD2FE54FFF0FE79FE1E0047FEF3FE90FFA1FF71FF81FED8FE38FF26004DF
2015-06-02 18:02:28 A521C4003027120702010000000007400A038BA07ECFF1600021097F3B402DCFE7EFF14FF10FE19FF40FF43FF52FF02FF1FFF7AFF0BFFCAFE79FF62FFD0FE2CFFB4FEF2FE97FF28FF4FFF34FE5DFF8AFC29FEABF
```

Abbildung A.3: Ausschnitt Strömungsmesser-Datei Deployment 14

```
<?xml version="1.0" encoding="UTF-8"?>
<GODESS_deployment>
    <Metadaten>
        <deployment>
            <deployment_number>15</deployment_number>
            <deployment_date>2015-07-26</deployment_date>
            <deployment_time>16:00:00</deployment_time>
            <deployment_position>
                <lat_deg>57</lat_deg>
                <lat_min>19.15</lat_min>
                <lon_deg>20</lon_deg>
                <lon_min>8.12</lon_min>
            </deployment_position>
            <deployment_cruise_ID>EMB100</deployment_cruise_ID>
            <deployment_CTD_file>0001F01</deployment_CTD_file>
            <deployment_nut_file>nase</deployment_nut_file>
            <deployment_schedule_file>nase</deployment_schedule_file>
        </deployment>
        <recovery>
            <recovery_cruise_ID/>
            <recovery_date>2015-09-24</recovery_date>
            <recovery_time>11:00:00</recovery_time>
            <recovery_CTD_file>nase</recovery_CTD_file>
            <recovery_nut_file>nase</recovery_nut_file>
        </recovery>
    </Metadaten>
    <SN_Powerlogger>01</SN_Powerlogger>
    <Sensors>
        <PressureSensor>
            <SerialNumberCTD>468</SerialNumberCTD>
            <ChannelNumber>8</ChannelNumber>
            <SerialNumberSensor>4681</SerialNumberSensor>
            <CalibrationDate>16.07.2014</CalibrationDate>
            <A0>-1.038561e+001</A0>
            <A1>4.23071e-003</A1>
            <A2>4.946985e-010</A2>
            <A3>-3.546359e-015</A3>
        </PressureSensor>
        <TemperatureSensor>
            <SerialNumberCTD>468</SerialNumberCTD>
            <ChannelNumber>9</ChannelNumber>
            <SerialNumberSensor>4681</SerialNumberSensor>
            <CalibrationDate>16.07.2014</CalibrationDate>
            <A0>-2.33476e+000</A0>
            <A1>5.90981e-004</A1>
            <A2>0.e+000</A2>
            <A3>0.e+000</A3>
        </TemperatureSensor>
```

Abbildung A.4: Ausschnitt XML-Konfigurationsdatei Deployment 15

A.1.2 SQL-Anweisung für die Beispiel-Tabellen

```
CREATE TABLE deployment (
    deployment_id INTEGER NOT NULL PRIMARY KEY,
    name VARCHAR(255),
    deployed_ts TIMESTAMP,
    recoverd_ts TIMESTAMP,
    lon NUMERIC(7,5),
    lat NUMERIC(7,5),
    tt_start TIMESTAMP(6) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    tt_stop TIMESTAMP(6) NOT NULL  DEFAULT CURRENT_TIMESTAMP
)
COMMENT ON COLUMN deployment.deployment_id IS
  'An incrementing id handled by IOW Team.';
COMMENT ON COLUMN deployment.deployed_ts IS
  'The date when the plattform was deployed.';
COMMENT ON COLUMN deployment.recoverd_ts IS
  'The date when the plattform was recovered.';
COMMENT ON COLUMN deployment.lon IS
  'The longitude given in decimal degree of
  the location where this deployment has operated.';
COMMENT ON COLUMN deployment.lat IS
  'The latitude given in decimal degree of
  the location where this deployment has operated.';
COMMENT ON COLUMN deployment.tt_start IS
  'The transaction start timestamp with precision
  of 6 handled automaticly by the DBMS while inserting new data.';
COMMENT ON COLUMN deployment.tt_stop IS
  'The transaction stop timestamp with precision
  of 6 handled automaticly by the DBMS while inserting new data';
```

Anweisung A.1: Erzeugung der deployment-Tabelle

```
CREATE TABLE payload (
    id SERIAL PRIMARY KEY,
    deployment_id INTEGER NOT NULL REFERENCES deployment,
    profile_ts TIMESTAMP NOT NULL,
    pressure NUMERIC,
    temperature NUMERIC,
    conductivity NUMERIC,
    tt_start TIMESTAMP(6) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    tt_stop TIMESTAMP(6) NOT NULL  DEFAULT CURRENT_TIMESTAMP
)
COMMENT ON COLUMN payload.deployment_id IS
  'The deployment as foreign key referecnes to which deployment
  the particular row belongs.';
COMMENT ON COLUMN payload.profile_ts IS
  'The timesamp of the particular row generated while executing a profile.';
COMMENT ON COLUMN payload.pressure IS
  'The pressure at given timestamp in dBar.';
COMMENT ON COLUMN payload.temperature IS
  'The temperature at given timestamp in celsius.';
COMMENT ON COLUMN payload.conductivity IS
  'The conductivity at given timestamp in mS/cm.';
COMMENT ON COLUMN payload.tt_start IS
  'The transaction start timestamp with precision of 6 handled automaticly
  by the DBMS while inserting new data.';
COMMENT ON COLUMN payload.tt_stop IS
  'The transaction stop timestamp with precision of 6 handled automaticly
  by the DBMS while inserting new data';
```

Anweisung A.2: Erzeugung der payload-Tabelle

A.2 Grundlagen Provenance

A.2.1 intensionale Antworten

Die Anweisung A.3 beschreibt die Relation 3.2 aus Abschnitt 3.2.2.2. Genutzt wurde PostgreSQL.

```
SELECT
    cols.column_name AS name,
    CASE WHEN cols.column_name = 'lon'
    THEN deployment.lon
    ELSE deployment.lat
    END AS value,
    (
        SELECT pg_catalog.col_description(c.oid, cols.ordinal_position::int)
        FROM pg_catalog.pg_class c
        WHERE c.oid = (SELECT 'deployment'::regclass::oid)
        AND c.relname = cols.table_name
    ) AS info
FROM
    information_schema.columns AS cols,
    deployment
WHERE cols.table_catalog = 'postgres'
AND cols.table_name   = 'deployment'
AND cols.table_schema = 'public'
AND (cols.column_name = 'lon' OR cols.column_name = 'lat')
AND deployment.deployment_id = 15
```

Anweisung A.3: Erzeugung von Meta-Informationen einer Anfrage

A.3 Konzept

A.3.1 Versionierung von UDFs/STPs

```
CREATE TABLE routines_history (
  id INTEGER NOT NULL,
  routine_name VARCHAR(200),
  routine_new_name VARCHAR(200) DEFAULT '',
  routine_arguments VARCHAR(1000),
  routine_declaration CLOB,
  routine_result VARCHAR(100),
  vt_begin TIMESTAMP(6) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  vt_end TIMESTAMP(6) NOT NULL DEFAULT TIMESTAMP '9999-12-31 23:59:59',
  tt TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID,
  PERIOD BUSINESS_TIME (vt_begin, vt_end),
  CONSTRAINT pk_id_vt PRIMARY KEY (id, BUSINESS_TIME WITHOUT OVERLAPS)
)
```

Anweisung A.4: Versionierungstabelle für UDFs/STPs

A.4 Prototyp

A.4.1 Versionierung von UDFs/STPs: Hilfsfunktionen

```
CREATE OR REPLACE FUNCTION unix_ts_as_string()
  RETURNS CHAR(10)
AS $$
BEGIN
  RETURN extract(EPOCH FROM now())::CHAR(10);
END
$$ LANGUAGE plpgsql
```

Anweisung A.5: Hilfsfunktion *unix_ts_as_string* zur Erzeugung eines Unix-Zeitstempels

```
CREATE OR REPLACE FUNCTION routine_create(
  function_name VARCHAR(100),
  p_vt_begin TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6)
)
  RETURNS VOID
  AS $$
BEGIN
  INSERT INTO routines_history
    (routine_name, routine_arguments, routine_declaration, routine_result,
    vt_begin, valid_time)
    SELECT
      proname,
      pg_get_function_identity_arguments(oid),
      pg_get_functiondef(oid),
      pg_get_function_result(oid),
      p_vt_begin,
      tsrange(p_vt_begin::TIMESTAMP, '9999-12-31 23:59:59'::TIMESTAMP, '[)')
    FROM pg_proc
    WHERE proname = function_name::regproc::text;
END
$$ LANGUAGE plpgsql
```

Anweisung A.6: Hilfsfunktion *routine_create* zur Speicherung einer neuen Funktion in der Versionierungstabelle

```
CREATE OR REPLACE FUNCTION routine_rename(
  old_function_name VARCHAR(100),
  new_function_name VARCHAR(100),
  p_vt_end TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6)
)
  RETURNS VOID
AS $$
DECLARE
  old_function_name_temporal VARCHAR(100);
  full_qualified_old_function_name VARCHAR(100);
BEGIN

  old_function_name_temporal := old_function_name || '_' || (SELECT unix_ts_as_string());

  -- get full qualified function signature
  SELECT proname || '(' || pg_get_function_identity_arguments(oid) ||')'
  INTO full_qualified_old_function_name
  FROM pg_proc
  WHERE proname = old_function_name::regproc::text;

  -- dynamic sql: rename old function to old function with ts
  EXECUTE FORMAT('ALTER FUNCTION %s RENAME TO %s',
    full_qualified_old_function_name, old_function_name_temporal);

  -- 1 update old function: new_name and valid time
  UPDATE routines_history
  SET
    routine_new_name = old_function_name_temporal,
    vt_end = p_vt_end,
    valid_time = tsrange(lower(valid_time), p_vt_end::TIMESTAMP, '[)')
  WHERE routine_name = old_function_name
  AND vt_begin <= CURRENT_TIMESTAMP(6)
  AND vt_end = '9999-12-31 23:59:59'
  AND lower(valid_time) <= CURRENT_TIMESTAMP(6)
  AND upper(valid_time) = '9999-12-31 23:59:59';

  -- 2) insert a dummy function to update later
  INSERT INTO routines_history (id, routine_name, vt_begin, valid_time)
    SELECT id, new_function_name, p_vt_end,
      tsrange(p_vt_end::TIMESTAMP, '9999-12-31 23:59:59'::TIMESTAMP, '[)')
    FROM routines_history
    WHERE routine_name = old_function_name
    AND vt_end = p_vt_end
    AND upper(valid_time) = p_vt_end;

END
$$ LANGUAGE plpgsql
```

Anweisung A.7: Hilfsfunktion *routine_rename* für die Umbenenennung einer Funktion in der Versionierungstabelle

```
CREATE OR REPLACE FUNCTION routine_update_after_rename(
  function_name VARCHAR(100)
)
  RETURNS VOID
AS $$
  BEGIN
    UPDATE routines_history
    SET
      routine_arguments = (SELECT pg_get_function_identity_arguments(oid) FROM pg_proc
                           WHERE proname = function_name::regproc::text),
      routine_declaration = (SELECT pg_get_functiondef(oid) FROM pg_proc
                             WHERE proname = function_name::regproc::text),
      routine_result = (SELECT pg_get_function_result(oid) FROM pg_proc
                        WHERE proname = function_name::regproc::text)
    WHERE routine_name = function_name
    AND vt_begin <= CURRENT_TIMESTAMP(6)
    AND vt_end = '9999-12-31 23:59:59';
  END;
$$ LANGUAGE plpgsql
```

Anweisung A.8: Hilfsfunktion *routine_update_after_rename* für die Umbenenennung einer Funktion in der Versionierungstabelle

```
CREATE OR REPLACE FUNCTION routine_delete(
  function_name VARCHAR(100),
   p_vt_end TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6)
)
  RETURNS VOID
AS $$
  BEGIN
    UPDATE routines_history
    SET
      vt_end = p_vt_end,
      valid_time = tsrange(lower(valid_time), p_vt_end::TIMESTAMP, '[)')
    WHERE routine_name = function_name
    AND vt_begin <= CURRENT_TIMESTAMP(6)
    AND vt_end = '9999-12-31 23:59:59';
  END;
$$ LANGUAGE plpgsql
```

Anweisung A.9: Hilfsfunktion *routine_delete* zum Löschen einer Funktion in der Versionierungstabelle

Selbstständigkeitserklärung

Hiermit erkläre ich an Eides statt, dass ich

- die vorliegende Arbeit selbstständig angefertigt,
- keine anderen als die angegebenen Quellen benutzt,
- die wörtlich oder dem Inhalt nach aus fremden Arbeiten entnommenen Stellen, bildlichen Darstellungen und dergleichen als solche genau kenntlich gemacht,
- keine unerlaubte fremde Hilfe in Anspruch genommen habe,
- die Arbeit bisher keiner Prüfungsbehörde vorgelegt und
- die Arbeit auch noch nicht eröffentlicht wurde.

Rostock, d. 13.09.2016 _____

Frank Meyer