Universität Rostock Traditio et Innovatio

Bachelorarbeit

Beschreibung der Veränderungen von Schemata und Daten am IOW mit Schema-Evolutions-Operatoren

eingereicht von: Erik Manthey
Matrikelnummer: 216203889

eingereicht am: 03.03.2020

Gutachter: Prof. Dr. rer. nat. habil. Andreas Heuer
Dr.-Ing. Holger Meyer

Betreuer/-innen: M.Sc. Tanja Auge
Dr. rer. nat. Susanne Feistel
Prof. Dr. rer. nat. habil. Andreas Heuer
Dipl.-Inf. Susanne Jürgensmann

Abstract

Bekannte Provenance-Techniken beziehen sich meist auf feste Schemata, welche sich nicht verändern. Es existieren in der Realität jedoch viele Anwendungsfälle, in denen Schema-Änderungen auftreten, so auch beim Leibniz-Institut für Ostseeforschung Warnemünde. In dieser Bachelorarbeit befasse ich mich mit diesen Änderungen und den Anwendungsmöglichkeiten von Provenance-Techniken. Zu diesem Zwecke analysiere ich zunächst Datensätze verschiedener Jahre und erstelle Entity-Relationship-Modelle für sie. Anschließend systematisiere ich die Schema-Änderungen und beschreibe sie mit Hilfe grundlegender Schema-Evolutions-Operatoren, wobei ich fehlende Operatoren selbst definiere. Nach der Analyse der Schema-Änderungen untersuche ich diese auf ihre Verträglichkeit mit Data Provenance. Dabei kann ich feststellen, dass auf fast allen verwendeten Operatoren Provenance-Techniken sinnvoll angewendet werden können. Außerdem beschreibe ich eine weitere Art der Data-Provenance-Anfrage, um hinterlegte Typen von Spalten feststellen zu können. Abschließend folgt eine Implementierung eines Schemas inklusive der Anwendung von Schema-Evolutions-Operatoren sowie eine Zusammenfassung sowie sich ergebende weitere Fragestellungen meiner Arbeit.

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Bachelorarbeit ergründet Schwierigkeiten im Forschungsdatenmanagement, welche durch Veränderungen von Tabellen und enthaltenen Daten auftreten können. Dazu werden exemplarisch solche Änderungen betrachtet, welche am Leibniz-Institut für Ostseeforschung Warnemünde von 1977 bis 2019 aufkamen. Dabei liegt das Augenmerk besonders auf der Data Provenance sowie der Schema-Evolution. Unter Forschungsdatenmanagement wird die Verwaltung und Bereitstellung von Ergebnissen aus der Forschung verstanden. Dazu zählen unter anderem die Datenarchivierung, -aufbereitung und -strukturierung. Außerdem nehmen auch die Dokumentation sowie die Publikation von Forschungsdaten und zugehörigen Erkenntnissen bzw. Ergebnissen einen nicht weniger wichtigen Teil im Management von Forschungsdaten ein [ZS]. Durch Veränderungen der Datenstrukturen und Tabellen in Forschungsdatenbanken, ausgelöst durch zum Beispiel neue Erkenntnisse oder Vorschriften, ist die Reproduzierbarkeit vieler Daten nicht ohne Komplikationen zu bewältigen. Aus diesem Grund soll eruiert werden, ob die Nachverfolgbarkeit von Daten (**Provenance**) mit grundlegenden Schema-Evolutions-Operatoren im Kontext des IOW verträglich ist und welcher Anteil der Änderungen überhaupt mit diesen beschrieben werden kann.

Leibniz-Institut für Ostseeforschung Warnemünde (IOW) Das Leibniz-Institut für Ostseeforschung Warnemünde (IOW) wurde 1992 gegründet und ging aus dem Institut für Meereskunde der ehemaligen Deutschen Demokratischen Republik hervor. Dieses Institut gehörte der Akademie der Wissenschaften der DDR an und wurde wiederum 1958 aus dem Seehydrographischen Dienst der DDR (1950) heraus gegründet [Fen]. Seit den 1960er Jahren besteht an der Universität Rostock eine Abteilung für Meeresbiologie sowie später auch ein eigener Lehrstuhl unter der Leitung von Inna Sokolova (Stand 24.10.2019), welcher mehrere kooperative Forschungsprojekte mit dem IOW betreibt.

In ihren Untersuchungen der Ostsee erforschen die 122 Wissenschaftlerinnen und Wissenschaftler am Leibniz-Institut für Ostseeforschung (Stand 21.12.2018) unter anderem Lebewesen, welche auf dem oder im Meeresgrund leben (Sektion `Biologische Meereskunde`). Weiterhin wird die Erforschung der Ostsee eingeteilt in die Sektionen `Marine Geologie`, welche sich auf geologische sowie geochemische Komponenten konzentriert, die `Physikalische Ozeanographie`, in welcher die physikalischen Prozesse der Ostsee erforscht und Modelle dafür entworfen werden, und die Sektion `Meereschemie`. In dieser werden unter anderem Stoffflüsse untersucht und die Veränderungen chemischer Größen über lange Zeiträume betrachtet. Erhoben werden diese und weitere Daten sowohl durch stationäre Messplattformen wie der GODESS, beschrieben in [Möl] im Abschnitt 1.1 sowie 2.1, als auch durch regelmäßige Forschungsfahrten.

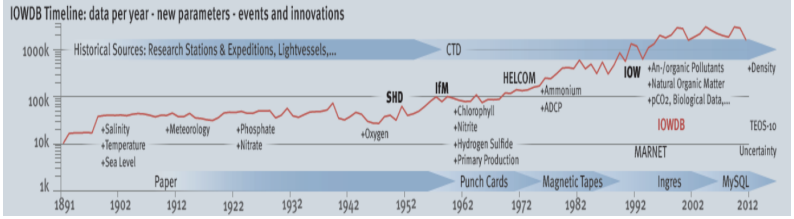

Abbildung 1.1: Logarithmischer Datenzuwachs am IOW pro Jahrzehnt [BFJ]

Wie in Abbildung 1.1 erkennbar, liegen einige Datensätze schon seit 1891 vor. Diese wurden unter anderem in Gedser (Dänemark) erhoben und dem späteren IOW zur Verfügung gestellt. Weiterhin verdeutlicht das Diagramm den enormen Zuwachs an Rohdaten sowie die Evolution der verwendeten Speichermedien. So wurden in den ersten Jahrzehnten Rohdaten von Stationen und Fahrten zunächst handschriftlich, später auf Lochkarten und Magnetbändern archiviert. Erst ab den 1990er Jahren war die Verwendung von Datenbanken die Standardmethode im Forschungsdatenmanagement des IOW, also der Nutzbarmachung und Bereitstellung von Forschungsdaten (sowohl von Roh- als auch aufbereiteten Daten).

Schema-Evolution Innerhalb der letzten rund 130 Jahre änderten sich mehrfach die Messverfahren, wodurch die resultierenden Ergebnisse, wie in Abbildung 1.1 erkennbar, viel mehr Informationen enthielten, als solche, die nur ein paar Jahre zuvor erhoben wurden. So wurden durch Gewinn neuer Erkenntnisse, beispielsweise der Entdeckung neuer Arten, viele biologische Einteilungen überarbeitet, was eine Auswertung über den gesamten Zeitraum und die Rückverfolgbarkeit von Tupeln erheblich erschwert. So variierten beispielsweise wissenschaftliche Taxonomien über den genannten Zeitraum in mehreren Fällen. Diese neuen und alten Taxonomien sowie der Ablauf der Veränderungen werden in der sogenannten BenthosDB gespeichert und können rückverfolgt werden. Diese Datenbank beinhaltet alle am IOW erfassten Lebewesen, welche auf dem oder im Meeresgrund der Ostsee leben. Dazu zählen unter anderem verschiedene Krebs- und Weichtiere. In Kooperation mit der Universität Rostock hat das IOW hierzu eine Datenbank auf Basis von MySQL eingerichtet.

Neben den Veränderungen biologischer Einteilungen werden manche chemischen Größen erst seit den 1960er Jahren erfasst, da frühere Forschungen nicht auf diese ausgelegt waren. Somit unterscheiden sich auch hier frühere von aktuellen Daten und Tabellen maßgeblich in ihrer Struktur. So werden beispielsweise Spalten umbenannt oder gänzlich neu erstellt oder gelöscht. Auch können diese Attribute zusammengefasst oder aufgeteilt werden.

Longitude [°]	Latitude [°]	Depth[mm]	Time	PSAL7IDD	TEMP7IDD	SVEL7PRD
12.553	54.585	1000	18.02.1990 16:25:00	8.755	4.331	1434.4
12.553	54.585	3000	18.02.1990 16:25:00	8.751	4.332	1434.45
12.553	54.585	4000	18.02.1990 16:25:00	8.752	4.336	1434.47
12.553	54.585	5000	18.02.1990 16:25:00	8.754	4.338	1434.51
12.553	54.585	6000	18.02.1990 16:25:00	8.758	4.327	1434.49

Tabelle 1.1: Untersuchung von Salzgehalt (PSAL7IDD), Temperatur (TEMP7IDD) und Schallgeschwindigkeit (SVEL7PRD) 1990 (Ausschnitt) [Leia]

Denkbare Schema-Evolutionen ausgehend von Tabelle 1.1 wären beispielsweise die Umbenennung des Attributes `Time` in `DateTime`, um beide enthaltenen Bestandteile zu nennen. Weiterhin wäre ein Aufsplitten des Attributes `Time` in die beiden Attribute `Time` und `Date` denkbar. Außerdem zählen auch Löschungen und Neuerstellungen von Tabellen sowie Attributen zu den Schema-Evolutionen. Neben diesen sogenannten **S**chema **M**odification **O**peratoren existieren auch **I**ntegrity **C**onstraints **M**odification **O**peratoren, welche Veränderungen in Schlüsselattributen und anderen Integritätsbedingungen beschreiben.

Ein Ziel dieser Bachelorarbeit ist es, diese Änderungen exemplarisch anhand einiger gewählter Beispiele durch sogenannte Schema-Evolutions-Operatoren zu beschreiben. Dabei stütze ich mich auf vorhergegangene Arbeiten ([Möl], [Yan]). Am Ende dieser Beschreibung und Systematisierung steht im Idealfall ein Universalschema, welches alle erkennbaren Veränderungen in den vorliegenden Schemata beschreibt.

Provenance Management Provenance beschreibt die Ursprünge und Historie von Daten im Allgemeinen. Eingeteilt wird die Provenance in Metadata, Workflow und Data Provenance. In dieser Bachelorarbeit werde ich mich vorwiegend mit letzterer auseinandersetzen und die anderen beiden Formen in kurzer Form erläutern.

Die Data Provenance untersucht die Herkunft von bestimmten Datensätzen und Anfrageergebnissen. So behandelt sie grob die Fragestellung, *woher* welche Daten *warum* kommen und *wie* sie ermittelt wurden. So können beispielsweise Anomalien untersucht und deren Ursachen ergründet werden.

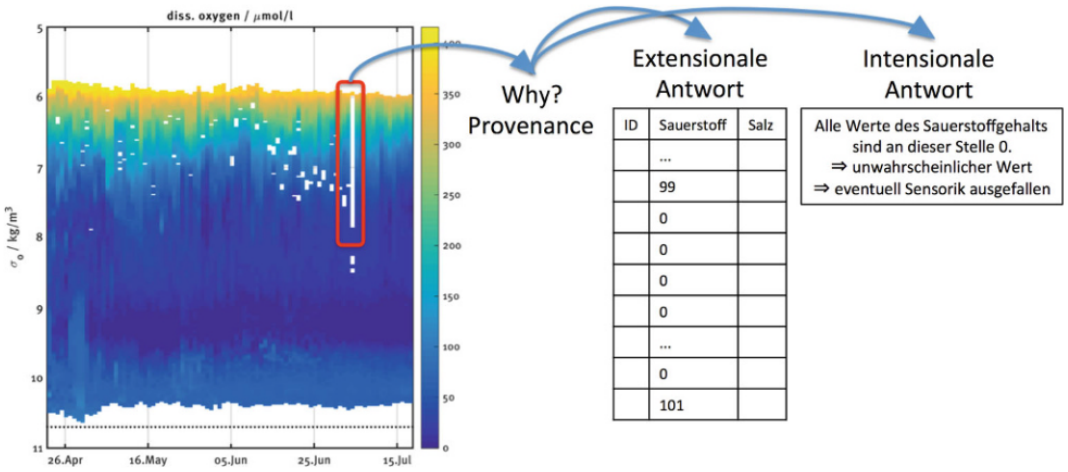

Abbildung 1.2: Anomalie im Sauerstoffgehalt [BKM+17]

Eine solche Anomalie ist beispielsweise in nachfolgender Abbildung 1.2 erkennbar. Infolgedessen wird eine Anfrage im Sinne der sogenannten WHY-Provenance gestellt, woraufhin die extensionale Antwort (Ausgabe verantwortlicher Tupel der Tabelle(n)) zeigt, dass viele Datensätze den gleichen Sauerstoffgehalt von 0 aufweisen. Die intensionale Antwort (Beschreibung der Ursache in Prosa) gibt eine mögliche Begründung der Anomalie an. Die vorliegende Grafik ist aus [BKM+17] Seite 194 übernommen. Die Anfrage- und Antworttypen der Data Provenance werde ich im Abschnitt 3.3.2 tiefergehend beleuchten.

Im nun folgenden Kapitel gebe ich einen geschichtlichen Abriss des Leibniz-Instituts für Ostseeforschung Warnemünde, beschreibe dessen Struktur und Aufgabe(n) und gehe auf die untersuchten Daten und deren Messung ein. Anschließend folgt ein Überblick über die Struktur dieser Arbeit.

2 Das Leibniz-Institut für Ostseeforschung Warnemünde

Das nachfolgende Kapitel bezieht sich maßgeblich auf einen meereswissenschaftlichen Bericht von Wolfgang Fennel aus dem Jahre 2018, in welchem die Geschichte des Leibniz-Instituts für Ostseeforschung Warnemünde (IOW) von Anfang der 1950er Jahre bis 1997 aufgegriffen wird [Fen]. Weitere Quellen sind entsprechend ihrer Verwendung markiert und zitiert.

Geschichte des IOW Die Geschichte des IOW beginnt in der Deutschen Demokratischen Republik der 1950er Jahre. Zu dieser Zeit, Anfang 1950, wurde der **Seehydrographische Dienst der DDR** gegründet, um losgelöst vom gesamtdeutschen Äquivalent agieren zu können. Er beinhaltete unter anderem eine eigene Abteilung für Meereskunde, welche 1960 als „Institut für Meereskunde Warnemünde"(IfM-W) in die Akademie der Wissenschaften aufgenommen wurde. Bemerkenswert ist, dass ein Großteil der damalig angestellten Wissenschaftler/-innen keine ozeanographische Vorbildung aufweisen konnten, sondern vermehrt Meteorolog/-innen sowie Naturwissenschaftler/-innen angeworben wurden. Trotz der globalen politischen Umstände dieser Zeit (beispielsweise die Nicht-Anerkennung der Deutschen Demokratischen Republik), gelangte das IfM-W durch eigene Forschungsfahrten zu internationaler Anerkennung und konnte sich als feste Größe in der weltweiten Meeresforschung etablieren.

Die Erforschung der Ostsee, eine der Hauptaufgaben des Instituts, begann 1964 mit der „Internationalen synoptischen Aufnahme der Ostsee", wobei mehrere Länder aus Ost und West ihre Erkenntnisse über die Ostsee miteinander teilten und verglichen. In den folgenden Jahrzehnten wurden die regelmäßigen Terminfahrten, aus welchen auch die Datensätze für diese Bachelorarbeit stammen, zu einem wichtigen Bestandteil der Ostseeforschung innerhalb des IfM-W.

Nach dem Beitritt der ostdeutschen Gebiete zum Geltungsbereich des bundesdeutschen Grundgesetzes 1990 mussten auch Forschung und Wissenschaft im gesamtdeutschen Staat neu geregelt und geordnet werden. Dazu wurde 1991 ein Gründungskomitee berufen, welches bis Februar 1992 fünfmal tagte. Am Ende seiner Arbeit stand die Entscheidung, ein neues Institut zu gründen. Das neu entstehende Institut für Ostseeforschung Warnemünde feierte am 28.02.1992 seine Eröffnung.

Struktur des IOW Die heutige Struktur des IOW lässt sich grob in drei Teile gliedern:

- Verwaltung & Administration (in Abbildung 2.1 blau hinterlegt)
- Sektionen (in Abbildung 2.1 dunkelgrün hinterlegt)
- zentrale Einheiten (in Abbildung 2.1 hellgrün hinterlegt)

Die vorliegende Bachelorarbeit entstand in Zusammenarbeit mit der zentralen Einheit „EDV", welche in Abbildung 2.1 an der roten Markierung erkennbar ist. Die zentralen Einheiten sind nicht auf spezielle Sektionen beschränkt. So wird beispielsweise die EDV für alle Sektionen oder auch für administrative Zwecke benötigt.

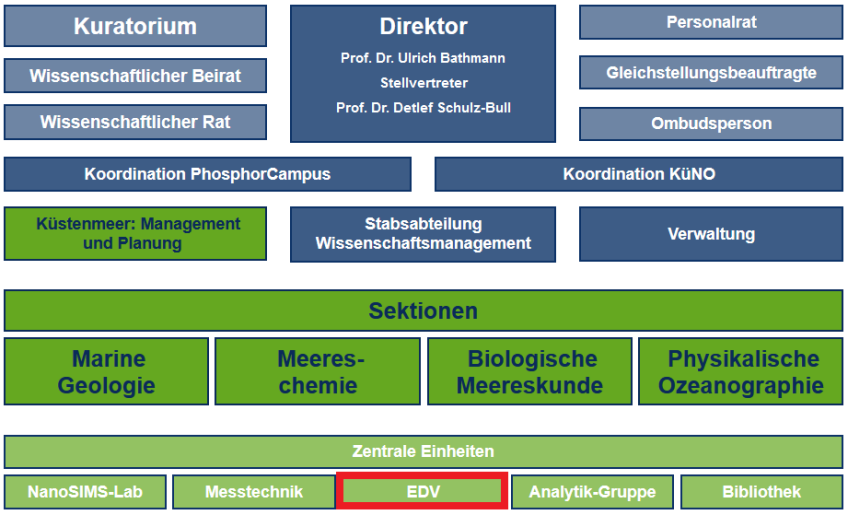

Abbildung 2.1: Organigramm des IOW [Leic]

Aufgabe des IOW Wie zuvor erwähnt, besteht die Hauptaufgabe des IOW in der Erforschung der Ostsee. Dabei werden verschiedene Aspekte untersucht. Verantwortlich sind dafür die unterschiedlichen Sektionen (siehe Abbildung 2.1). Die Sektion `Biologische Meereskunde` untersucht beispielswiese Lebewesen, welche auf dem oder im Meeresgrund leben. Weiterhin werden die Sektionen eingeteilt in die `Marine Geologie`, welche sich auf geologische sowie geochemische Komponenten konzentriert, die `Physikalische Ozeanographie`, in welcher die physikalischen Prozesse der Ostsee erforscht und Modelle dafür entworfen werden, und die Sektion `Meereschemie`. In dieser werden unter anderem Stoffflüsse untersucht und die Veränderungen chemischer Größen über lange Zeiträume betrachtet.

Daten & Datenbanken am IOW Durch die konsequent erfolgte Archivierung von Forschungsdaten können viele Aufzeichnungen ab dem Jahre 1951, also aus der Anfangsphase des Seehydrographischen Dienstes der Deutschen Demokratischen Republik, nachvollzogen werden. Auch die Öffentlichkeit hat mehrere Möglichkeiten, an Informationen über Forschungsfahrten und stationäre Forschungsanlagen zu gelangen. So bietet beispielsweise die Internetseite `http://iowmeta.io-warnemuende.de` vielfältige Möglichkeiten, Metadaten von Fahrten oder Projekten herunterzuladen, mit welchen Daten aus der Datenbank IOWDB verknüpft sind. In dieser Datenbank finden sich unter anderem Fahrtdaten sowie Informationen über Stationen und gemessene Parameter. Einen umfangreicheren Zugriff auf Daten aus der IOWDB können Interessierte auch über `https://odin2.io-warnemuende.de` erhalten (siehe Anhang, Abbildung 8.2). Eine gute Unterstützung bieten hierbei die Visualisierung der angeforderten Daten sowie die interaktive Navigation durch die angefahrenen Stationen von Fahrten.

Eine weitere Datenbank ist die BenthosDB, welche in Kooperation mit der Universität Rostock auf theoretischer Grundlage dort geschriebener Abschlussarbeiten entstanden ist. Zu nennen sind hierbei unter anderem Jessica Zierke [Zie] und Frank Meyer [Mey], auf welchen diese Bachelorarbeit in Teilen aufbaut. Die BenthosDB wurde für die „Arbeitsgruppe Ökologie benthischer Organismen“ entwickelt, welche sich mit den im und auf dem Meeresboden der Ostsee lebenden Organismen und deren Taxonomien, der biologischen Einteilung der Arten, beschäftigt.

CTD-Messung Die betrachteten Daten sind vornehmlich CTD-Daten. Das bedeutet, von Interesse sind vor allem die Parameter **C**onductivity (elektrische Leitfähigkeit), worüber der Salzgehalt bestimmt wird, **T**emperature (Wassertemperatur) und **D**epth (Tiefe), welche über den vorherrschenden Druck ermittelt wird. Erhoben werden diese Daten über eine bestimmte Sonde, die sogenannte **CTD-Sonde**

(vgl. Abbildung 2.2), welche von mehreren Wasserschöpfern umgeben ist, wodurch zusätzlich Wasserproben entnommen werden können. Diese werden dann für weitere Analysen genutzt. Außerdem besitzt die CTD-Sonde Sensoren für die Messung des Druckes, der Leitfähigkeit und der Temperatur.

Abbildung 2.2: CTD-Sonde [Leib]

Struktur dieser Arbeit Nach den ersten beiden einleitenden Kapiteln folgen nun die theoretischen Grundlagen, welche in dieser Bachelorarbeit angewendet beziehungsweise vorausgesetzt werden. Dabei führe ich nacheinander das sogenannte Entity-Relationship-Modell, das relationale Datenbankmodell, Provenance Management und die Schema Evolution Operators ein. Nachdem die grundlegende Theorie eingeführt wurde, gehe ich auf den „State of the Art“ ein, den aktuellen Entwicklungsstand auf den Gebieten Provenance, Schema-Evolution und der Kombination aus beiden Gebieten. Danach untersuche ich einige selbst entwickelte Schemata des IOW auf Veränderungen und beschreibe sie mit Hilfe der Schema Evolution Operators. Dann folgt die zweite Aufgabenstellung meiner Bachelorarbeit, die Kombination von Schema-Evolution und Provenance.

Zuletzt folgen einige Erkenntnisse aus der praktischen Implementierung meiner Überlegungen sowie eine Zusammenfassung inklusive eines Fazits und einem Ausblick für nachfolgende Arbeiten.

3 Theoretische Grundlagen

Im folgenden Kapitel werden die, der vorliegenden Bachelorarbeit zugrunde liegenden, theoretischen Grundlagen des relationalen Datenbankmodells (siehe Abschnitt 3.2), des Provenance Managements (siehe Abschnitt 3.3) sowie der Schema-Evolutions-Operatoren (siehe Abschnitt 3.4) näher beleuchtet. Dabei soll, durch Anwendung der Kapitelinhalte auf die Tabelle 3.1, ein praktischer Bezug zum IOW-Umfeld hergestellt werden. Diese enthält Fahrtdaten, dessen Werte zur übersichtlicheren Auswertung angepasst wurden.

3.1 Entity-Relationship-Modell

Das **Entity-Relationship-Modell** (kurz: ER-Modell) ist ein Konzeptionsmodell, bei dem Objekte (Entities) und deren Beziehungen (Relationships) untereinander beschrieben werden. Es kann prinzipiell in vielen verschiedenen Bereichen Anwendung finden, oftmals ist es jedoch ein probates Mittel zur grundlegenden Beschreibung von relationalen Datenbanken. Die Überführung eines ER-Modells in ein relationales Datenbankmodell (vgl. Abschnitt 3.2) soll in dieser Bachelorarbeit keine übergeordnete Rolle spielen und wird aufgrund der Kenntnisse des Informatik-Grundstudiums vorausgesetzt. Vielmehr dient dieser Abschnitt der Einführung verwendeter Notationen, die vor allem in den späteren Abschnitten (5.2 und folgende) eingesetzt werden.

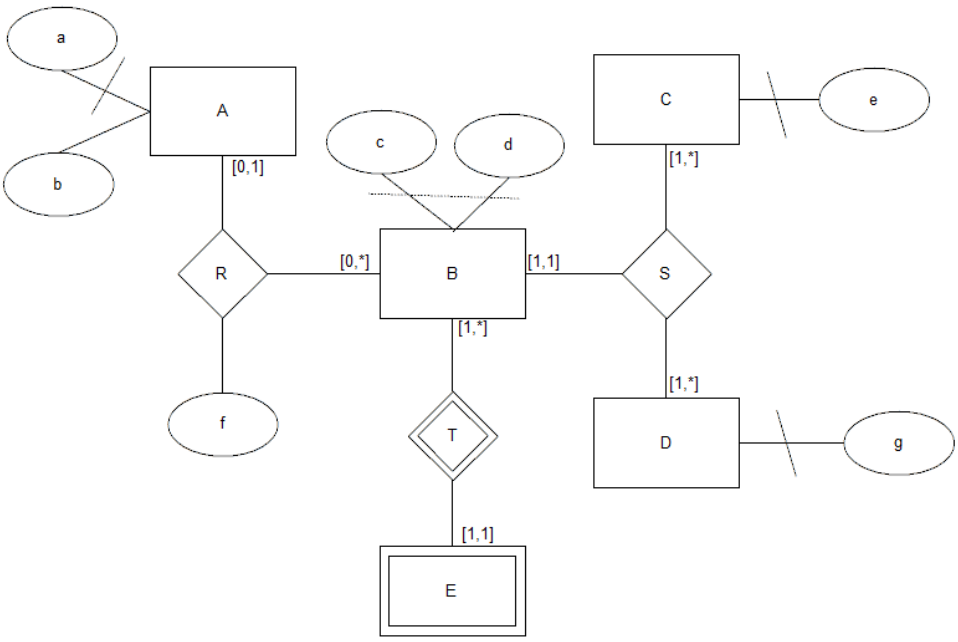

Abbildung 3.1: Beispiel für ein Entity-Relationship-Modell

Grundbegriffe Anhand von Abbildung 3.1 lässt sich diese grundlegende Notation schnell erklären. Objekte (Entity-Typen) werden durch Rechtecke dargestellt, Beziehungen (Beziehungstypen) als Rauten und Eigenschaften (Attribute) als Ovale. Entity-Typen bezeichnen hierbei die generelle Art von Objekten, deren Instanzen Entities genannt werden. So ist beispielsweise die Stadt *Rostock* ein Entity des Entity-Typen *Stadt*. Analog verhält es sich mit Beziehungstypen und Beziehungen. Diese Entity- und Beziehungstypen sind jeweils durch Striche miteinander verbunden. Attribute können sowohl an Entity-Typen (siehe Entity `C` und Attribut `e`) als auch an Beziehungstypen (siehe Beziehungstyp `R` und Attribut `f`) gebunden sein.

Beziehungstypen und abhängige Entities Der Beziehungstyp R ist eine sogenannte zweiwertige Beziehung zwischen `A` und `B`. Dies ist der einfachste Beziehungstyp, da nur zwei Entity-Typen miteinander verbunden werden. Im Gegensatz dazu ist `S` ein dreiwertiger Beziehungstyp, welcher die Entity-Typen `B`, `C` und `D` verknüpft. Dies bedeutet, dass wenn zwei der beteiligten Entity-Typen an dem Beziehungstyp `S` teilnehmen, muss auch zwangsläufig der dritte Entity-Typ daran beteiligt sein. Abschließend ist noch das Konzept eines abhängigen Entity-Typen zu erklären. Es wird durch einen doppelten Rand gekennzeichnet und gibt an, dass Entities von diesem Entity-Typ nur in Verbindung mit Entities von dem ihm übergeordneten Entity-Typ existieren können. In Abbildung 3.1 ist beispielsweise `E` ein abhängiger Entity-Typ von `B` über den Beziehungstyp `T`. Dabei geht es die Beziehung B genau einmal ein. Zur besseren bildhaften Darstellung wird auch der Beziehungstyp mit einem doppelten Rand versehen.

Schlüsselarten Ein **Primärschlüssel** besteht aus einem oder mehreren Attributen und dient der eindeutigen Identifizierung eines Entity-Typen oder eines Beziehungstypen. Gekennzeichnet werden diese Schlüssel durch Querstriche in den Verbindungsstrichen zwischen Entity-Typ beziehungsweise Beziehungstyp und Attribut. So ist beispielsweise das Attribut `a` der Primärschlüssel des Entity-Typen `A`.

Der Primärschlüssel des Entity-Typen `B` ist ein **zusammengesetzter Schlüssel** aus den Attributen `c` und `d`. Nur diese beiden Schlüsselattribute zusammen können eine Instanz von `B` eindeutig identifizieren.

Unter anderem bei abhängigen Entity-Typen, wie beispielsweise dem Entity-Typen `E`, ist eine weitere Art von Schlüsseln von Bedeutung, welche in ER-Modellen jedoch nicht explizit markiert werden. Diese sogenannten **Fremdschlüssel** spielen im späteren Verlauf jedoch durchaus eine Rolle und sollen hier kurz erklärt werden. Ist ein Entity-Typ abhängig von einem weiteren, so übernimmt es dessen Primärschlüssel als Fremdschlüssel. Im Beispiel von Abbildung 3.1 erhält also Entity-Typ `E` den zusammengesetzten Schlüssel `(c,d)` von dem ihm übergeordneten Entity-Typen `B`.

Kardinalitäten Sind zwei oder mehr Entity-Typen über einen Beziehungstyp miteinander verbunden, stellt sich die Frage, wie oft sie diese Beziehung eingehen. Eine mögliche Notationsart hierfür sind die sogenannten Kardinalitäten. Sie geben an, wie oft Entities von einem Entity-Typ `minimal` und `maximal` an Beziehungen des Beziehungstyps teilnehmen. Wie in Abbildung 3.1 erkennbar, werden diese Angaben in eckige Klammern an den jeweiligen Entity-Typen geschrieben. Betrachten wir folgendes Beispiel. Entities vom Typ `A` stehen über den Relationship-Typ `R` in Beziehung zu Entities vom Typ `B`. Dabei besitzt `A` die Kardinalität `[0,1]`, was bedeutet, dass `A`-Entities die Beziehung `R` mindestens kein Mal (0) und höchstens ein Mal (1) eingehen. Umgangssprachlich kann ein `A`-Entity also maximal einmal an der Beziehung `R` teilnehmen, es besteht jedoch kein Zwang. Entities vom Typ `B` hingegen besitzen die Kardinalität `[0,*]`. Das heißt, dass `B`-Entities wiederum mindestens kein Mal (0) an der Beziehung `R` teilnehmen. Das Sternchen

bedeutet, dass B-Entities maximal unendlich oft an R teilnehmen können. Umgangssprachlich bedeutet dies also, dass B-Entities beliebig oft (auch gar nicht) an R teilnehmen.

3.2 Relationales Datenbankmodell

Das relationale Datenbankmodell ist eine der am weitesten verbreiteten Techniken zur Entwicklung strukturierter Datenbanken. 1970 wurde es von Ted Codd (IBM) als konzeptionelle Grundlage zu diesem Zweck etabliert.

In diesem Modell sind die zu speichernden Daten in Tabellen, sogenannten **Relationen** organisiert. Eine solche Relation Data ist beispielhaft in Tabelle 3.1 abgebildet. Innerhalb einer Datenbank hat jede Relation einen eindeutigen Namen, anhand dessen sie identifiziert werden kann. Im zuvor genannten Entity-Relationship-Modell sind Entity-Typen mit den Relationen des relationalen Modells vergleichbar. Das exakte Vorgehen beim Abbilden des ER-Modells auf das relationale Modell soll hier nicht betrachtet werden, da es durch das Grundstudium als vorausgesetzt angesehen wird.

Das **Relationenschema** enthält den Relationennamen sowie die Namen der zugehörigen Attribute. **Attribute** sind, graphisch ausgedrückt, die Spaltennamen der Tabelle. Sie haben einen bestimmten Wertebereich. So sind beispielsweise die Attribute „Longitude“ und „Latitude“ vom Typ *Double*, beinhalten also Gleitkommazahlen. Das gesamte Relationenschema der Tabelle 3.1 lautet dementsprechend {Longitude, Latitude, Depth, Time, PSAL7IDD, TEMP7IDD, SVEL7PRD}. Ein **Tupel** ist eine Zeile einer Tabelle, also ein Eintrag einer Relation mit bestimmten Attributwerten. So ist beispielsweise (12.553333333333, 54.585, 4000, 18.02.1990 16:25:00, 8.752, 4.336, 1434.47) ein Tupel der Relation aus Tabelle 3.1.

Data

Longitude	Latitude	Depth	Time	PSAL7IDD	TEMP7IDD	SVEL7PRD
12.553	54.585	1000	18.02.1990 16:25:00	8.755	4.331	1434.4
12.553	54.585	3000	18.02.1990 16:25:00	8.751	4.332	1434.45
12.553	54.585	4000	18.02.1990 16:25:00	8.752	4.336	1434.47
12.553	54.585	5000	18.02.1990 16:25:00	8.754	4.338	1434.51
12.553	54.585	6000	18.02.1990 16:25:00	8.758	4.327	1434.49

Tabelle 3.1: Untersuchung von Salzgehalt (PSAL7IDD), Temperatur (TEMP7IDD) und Schallgeschwindigkeit (SVEL7PRD) im Jahre 1990; Ausschnitt aus [Leia]

Zusätzlich zu den genannten Bestandteilen gibt es sogenannte Schlüssel- und Integritätsbedingungen, welche hier kurz erläutert werden sollen. Dazu gehören zunächst die Primärschlüssel(-mengen). Diese sind ein oder mehrere Attribute, welche ein Tupel eindeutig identifizieren. Zu den Integritätsbedingungen gehören zudem die sogenannten UNIQUE-Constraints (Restriktionen), welche alternative Schlüssel (neben dem Primärschlüssel) darstellen. Attribute mit einer solchen Beschränkung dürfen alle Attributwerte nur einmal besitzen. Des Weiteren besitzen manche Tabellen sogenannte Fremdschlüssel. Das sind Attribute, welche einer anderen Tabelle entstammen und auf diese referenzieren.

3.3 Provenance Management

Dieser Abschnitt basiert unter anderem auf den Arbeiten von Cheney et al. [CCT09] und Herschel et al. [HDB17]. Außerdem enthält vor allem der Abschnitt 3.3.2 Informationen aus einem Lehrvideo von Majd Eddin Sila („Navigator.mp4“, siehe digitaler Anhang), welches mir freundlicherweise zur Verfügung gestellt wurde.

Wie eingangs erwähnt, beschäftigt sich ein Teil des Forschungsdatenmanagements nicht vorwiegend mit den Daten an sich, sondern beleuchtet die Hintergründe dieser, die sogenannten **Metadaten**. Dabei können diese ganz unterschiedliche Formen haben. So könnte es im Kontext des IOW beispielsweise möglich werden, über Provenance Management herauszufinden, *welche* Sensoren auf *welcher* Fahrt von *wem* überprüft wurden, im Zuge derer ein bestimmter Datensatz erhoben wurde.

Dabei geht es grundsätzlich um Fragen der Authentizität und Integrität der Daten sowie ihrer Reproduzierbarkeit, also der Vertrauenswürdigkeit und Korrektheit von Datensätzen. Außerdem können durch Provenance Management im Fehlerfall die verantwortlichen Quellen gefunden und die Auswirkungen auf das System besser eingeschätzt werden. All das trägt zur Verbesserung der Verständlichkeit und Qualität der Daten bei [HDB17].

3.3.1 Provenance-Arten

Betrachtet man die Grafik in Abbildung 3.2, so ergeben sich nach [HDB17] vier Formen der Provenance, welche hierarchisch angeordnet werden können. So unterscheiden sich diese unter anderem im Grad ihrer Spezifität, wobei die sogenannte **Data Provenance** dabei den höchsten Grad innehat (linke Seite der Pyramide). Auf der rechten Seite der Pyramide ist das Level der Instrumentierung/Automatisierung erkennbar, also die Möglichkeit, die Provenance-Untersuchung zu automatisieren.

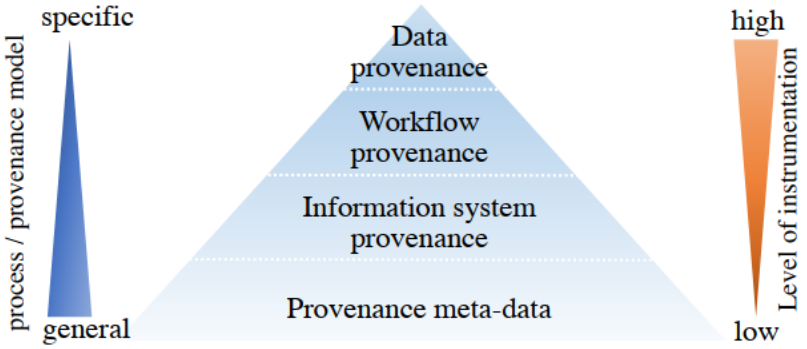

Abbildung 3.2: Provenance-Typen-Hierarchie nach [HDB17], Abschnitt 1.2

Provenance Meta-Data Aufgrund der geringen Spezifität und des niedrigen Automatisierungslevels der **Provenance Meta-Data** unterliegt diese Form zum Einen nur wenigen Einschränkungen. Zum Anderen stehen aber auch verhältnismäßig wenig Techniken zum Sammeln der Provenance-Informationen dieser Art zur Verfügung. In Bezug auf das fortlaufende Beispiel aus Tabelle 3.1 gehören zur Provenance Meta-Data beispielsweise die Messplattform, in diesem Falle das Forschungsschiff Alexander von Humboldt, Personendaten von verantwortlichen Wissenschaftler/-innen, hier von Prof. Dr. Dietwart Nehring, oder das zugehörige Institut, hier das Institut für Meereskunde der Akademie der Wissenschaften der DDR. Diese und weitere Metadaten sind in Abbildung 3.3 zu sehen.

Cruise > Platform	Forschungsschiff Alexander von Humboldt
Cruise > Platform > Helcom-Code	44
Cruise > Platform > ICES-Code	07AL
Cruise > Chief Scientist > Name	Nehring
Cruise > Chief Scientist > First Name	Dietwart
Cruise > Chief Scientist > Title	Prof. Dr. habil.
Cruise > Chief Scientist > Phone	D 0381 5197 300
Cruise > Chief Scientist > Fax	D 0381 5197 440
Cruise > Responsible Institution	Institut fuer Meereskunde
Cruise > Responsible Institution > Description	Akademie der Wissenschaften der DDR
Cruise > Responsible Institution > Street	Seestrasse 15
Cruise > Responsible Institution > City	Warnemuende
Cruise > Responsible Institution > Country	Deutsche Demokratische Republik

Abbildung 3.3: Ausschnitt von Metadaten aus [Leia]

Information System Provenance Eine Stufe präziser ist die sogenannte Information System Provenance, welche sich als erste ausschließlich mit der Produktion und dem Entstehungsprozess digitaler Daten beschäftigt. So werden hier Metadaten gesammelt, welche Fragen beantworten wie:

- Welche(s) System(e) wurden genutzt, um die vorliegenden Daten zu erhalten?
- Welcher Input, zum Beispiel in Form von Primärdaten, führte zu den vorliegenden Datensätzen?
- Welche Parameter waren an der Erfassung beteiligt?

Workflow Provenance Die Workflow Provenance beschreibt alle Provenance-Metadaten, welche den Entstehungsprozess von Daten und Datensätzen beschreiben. So beantwortet diese Form beispielsweise Fragen wie:

- Wer hat diese Daten wann erstellt?
- Im Zuge welchen Prozesses entstanden diese Daten. Welche Primärdaten liegen diesen Zugrunde?
- Wurden diese Daten verändert? Wenn ja, wann und von wem? [DF08]

Ein konkretes Beispiel für das IOW ist die Validierung von Daten. In diesem Kontext erörtert die Workflow Provenance, wer die Daten wann erhoben und wer diese wann mit welchen Mitteln validiert hat.

Data Provenance An der Spitze der Hierarchiepyramide steht die Data Provenance. Sie besitzt die höchste Spezifität und enthält Provenance-Informationen auf Datenebene, also zu einzelnen Anfragen und Relationen. Durch diese Form können Fragen geklärt werden, wie beispielsweise:

- *Woher*, aus welchen Ursprungsrelationen, stammen die Daten, welche zu einem bestimmten Anfrageergebnis geführt haben?
- *Wie* wurde das Ergebnis berechnet? Wie sieht die Anfrageformel aus?
- *Warum* sieht das Ergebnis so aus? Welche Tupel sind dafür verantwortlich?
- *Warum* sieht das Ergebnis *nicht* anders aus?

Diese Fragestellungen decken sich mit den vier verschiedenen Anfragetypen, welche im folgenden Abschnitt 3.3.2 tiefergehend beleuchtet werden.

3.3.2 Data Provenance am Beispiel des IOW

Anfragetypen In der Data Provenance unterscheidet man üblicherweise zwischen vier Anfragetypen [AH19]. Diese lauten *How*-, *Where*-, *Why*- und *Why not*- Provenance. Im Rahmen dieser Bachelorarbeit werden jedoch vorwiegend die ersten drei Arten behandelt.

Die *Where*-Provenance zeigt die Ursprungsrelationen auf, aus welchen bestimmte Tupel der Ergebnisrelation einer Anfrage stammen und wie diese verknüpft wurden. Die *Why*-Provenance klärt allgemein die Frage, *warum* ein bestimmtes Tupel in der Ergebnisrelation einer spezifischen Anfrage enthalten ist, wohingegen die *Why not*-Provenance das Gegenteil beleuchtet: *Warum* sind ein oder mehrere Tupel im Ergebnis *nicht* enthalten? Dadurch kann die Sinnhaftigkeit einer Anfrage und einer Ergebnisrelation überprüft werden. Dabei werden konkrete Tupel herangezogen, die bei der Entstehung eines Ergebnistupels beteiligt waren. Diese Angabe gibt gleichzeitig Auskunft über die zugehörigen Tabellen, da Tupel immer einer Tabelle zugeordnet sind. Dementsprechend beinhaltet die *Why*-Provenance auch Informationen der *Where*-Provenance.

Die umfangreichste Art der Data Provenance stellt die *How*-Provenance dar. Sie besteht aus einem Polynom, dem sogenannten Provenance-Polynom, und beinhaltet dementsprechend Informationen aus *Where*- und *Why*-Provenance. Sie beschreibt, wie Daten bis zum Anfrageergebnis manipuliert wurden, beispielsweise durch Aggregationen und Selektionen.

Aufgrund der Tatsache, dass Provenance-Informationen teilweise ineinander enthalten sind, kann folgende Reduktion vorgenommen werden:

$$\textbf{Where} \preccurlyeq \textbf{Why} \preccurlyeq \textbf{How}$$

What-Provenance – Eigene Erweiterung Im Zuge der Bearbeitung dieser Bachelorarbeit fiel mir an mehreren Stellen auf, dass die bekannten Anfragetypen der Data Provenance nicht unbedingt immer ausreichen. So geben diese bisher keinerlei Auskunft über die verschiedenen Attributtypen, welche ein Tupel beinhaltet. Auch diese Typen können sich durch Schema-Evolutionen verändern und es können Fehler auftauchen, welche beispielsweise während der Konvertierung von Werten auftreten. So ist es durchaus denkbar, dass bei der Konkatenation von `StartYear`, `StartMonth` und `StartDay` der Serie zu einem einzigen Attribut `BegDatum` Konvertierungen stattfanden, welche nicht eindeutig nachvollziehbar sind. Es ist möglich, dass hier drei Ganzzahlen (Integer) oder Zeichenketten (VARCHAR) zu einem Datumsformat (Date) oder einem weiteren VARCHAR umgewandelt wurden. Eine tiefergehende Betrachtung erfolgt im Kontext meines Konzepts im Kapitel 5.

Antworttypen Auf die Anfragetypen kann verschieden „geantwortet“ werden. So unterscheidet man zwischen *extensionalen*, *intensionalen*, *anfragebasierten* und *modifikationsbasierten* Antworten.
Die *extensionale* Antwort besteht aus konkreten Tupeln der Originalrelationen und wird vor allem für die Anfragetypen verwendet, welche in dieser Arbeit vorrangig behandelt werden, also *Where, Why* und *How*. Aufgrunddessen findet auch dieser Antworttyp hier höhere Beachtung.

Die weiteren Antworttypen beantworten Provenance-Anfragen auf andere Weise. So bietet die *intensionale* Antwort eine sprachliche Beschreibung der Daten und der angeforderten Informationen an. Die *intensionale* Antwort kann auch bei „Why not?“-Anfragen angewendet werden, so wie auch die *modifikationsbasierte* Antwort, welche Änderungen an der gestellten Anfrage vorschlägt, durch welche andere Originaltupel im Ergebnis (nicht) erscheinen würden. In Abbildung 3.4 wird beispielsweise auf eine *Why*-Anfrage ein möglicher Grund für die auftretende Anomalie und eben diese selbst formuliert.

Des Weiteren gibt die *anfragebasierte* Antwort Aufschluss über die verwendeten Selektionsprädikate und ist demnach bei Where-Anfragen nicht besonders hilfreich, da hierbei nur die Ursprungsrelationen von Interesse sind. Dagegen kann sie genutzt werden, um *How*-Anfragen zu beantworten, da hier die konkrete Berechnung des Ergebnisses von Interesse ist. Bei der Beantwortung von Provenance-Anfragen beschränken wir uns im Rahmen dieser Arbeit nur auf die extensionale Antwort.

Beispieltabellen Zur Veranschaulichung der Provenance-Anfragen wird eine Datenbank D angenommen, welche aus den beiden Tabellen **DataCTD** und **Stations** besteht. Diese sind im Folgenden, mit einigen Beispieldaten gefüllt, abgebildet. Dabei entstand erstere (Tabelle 3.2)aus der Vereinfachung und Modifikation von Tabelle 1.1. Die Relation Stations (Tabelle 3.3)wurde frei erdacht.

DataCTD

Longitude	Latitude	Depth	PSAL7IDD	TEMP7IDD	SVEL7PRD	
13.553	54.385	1000	8.755	4.331	1434.4	d_1
12.553	54.585	3000	8.751	4.332	1434.45	d_2
11.553	54.285	4000	8.752	4.336	1434.47	d_3
12.753	54.185	5000	8.754	4.338	1434.51	d_4
12.953	54.085	6000	8.758	4.327	1434.49	d_5

Tabelle 3.2: Modifizierte Tabelle **DataCTD**

Stations

Name	Longitude	Latitude	
Station 1	13.00	54.585	s_1
Station 2	12.00	55.550	s_2
Station 3	12.553	54.585	s_3

Tabelle 3.3: Fiktive Tabelle **Stations**

Die nun folgenden Abschnitte zu den behandelten Provenance-Anfragearten basieren zum großen Teil auf der Arbeit von Cheney et al. Im ersten Kapitel von [CCT09] werden die folgenden drei Anfragearten überblicksartig erläutert.

Where-Provenance Die **Where**-Provenance beschreibt die Beziehung zwischen den Eingabelokationen und den Ausgabelokationen. Sie gibt also Antwort auf die Frage, aus welchen Tabellen einzelne Tupel der Ergebnisrelation stammen. In anderen Definitionen bezieht sich eine Lokation auf ein konkretes Attribut, also eine konkrete Spalte, eines Tupels einer Relation (vgl. [CCT09]). Dementsprechend kann sie graphisch mit einer Zelle einer Tabelle verglichen werden. Dieses Verständnis von **Where**-Provenance findet in dieser Bachelorarbeit jedoch keine Anwendung und dient rein dem Informationszweck.

Zur Beschreibung der extensionalen Antwort sowie zur besseren Übersichtlichkeit wird allen Tupeln ein Label, eine Tupelnummer zugewiesen. Diese trägt den Anfangsbuchstaben der sie umfassenden Relation und eine fortlaufende Nummer.

Sei folgende Anfrage (Query) Q1 gegeben:

```
Q1:   SELECT   d.Longitude AS Longitude,
               s.Latitude AS Latitude,
               s.Name AS Name
      FROM     DataCTD d, Stations s
      WHERE    d.Longitude = s.Longitude
      AND      d.Latitude = s.Latitude
```

Nach der Ausführung von Q1 auf die Beispieldatenbank D ergibt sich folgende Ergebnisrelation **R**:

R	d.Longitude	s.Latitude	s.Name	
	12.553	54.585	Station 3	r_1

Die **Where**-Provenance gibt nun den ursprünglichen Speicherort (Lokation) der entstandenen Tupel aus dem Ergebnis an, also aus welchen Tabellen deren Werte kopiert wurden. Dabei ist darauf zu achten, welche Relation(en) in der SELECT-Klausel referenziert wird/werden.

Die *Where*-Provenance des Tupels r_1 aus der Ergebnisrelation **R** lautet somit {DataCTD, Stations}. Auch wenn auf dem Weg der Berechnung von den Originaldaten hin zum Ergebnis viele Relationen verwendet werden sollten, müssen diese nicht zwingend in der **Where**-Provenance auftauchen. Es sind hingegen nur die Relationen von Interesse und damit in der **Where**-Provenance enthalten, aus denen tatsächlich Werte oder ganze Tupel kopiert wurden. Die enthaltenen Informationen sind also recht beschränkt und oftmals werden weitere benötigt. Diese werden in den folgenden Paragraphen näher beleuchtet.

Why-Provenance Im Gegensatz zur **Where**-Provenance, welche die bloßen Lokationen von Input- und Output-Attributwerten in Beziehung zueinander setzt, beschreibt die **Why**-Provenance, *warum* ein Tupel aufgrund einer Query in der Ergebnisrelation erscheint. Es werden also Eingangstupel gesucht, welche die Existenz eines Ergebnistupels „bezeugen“. Dies kann auf mehrere Arten geschehen. Zum Einen können Werte aus Input- in Outputtupeln vorkommen. Dies beschreibt die **Where**-Provenance. Zum Anderen können Tupel auch zur Erzeugung eines Ergebnisses genutzt werden, ohne dass ihr Wert in diesem vorkommt. Dies ist beispielsweise der Fall bei Aggregatfunktionen wie der Summenbildung oder bei Zwischentupeln, welche das Ergebnis von Joins auf Relationen oder von anderen Operationen sein können. Hierbei kann zusätzlich zur extensionalen auch durchaus die intensionale Antwort verwendet werden. Wir beschränken uns allerdings hauptsächlich auf erstere.

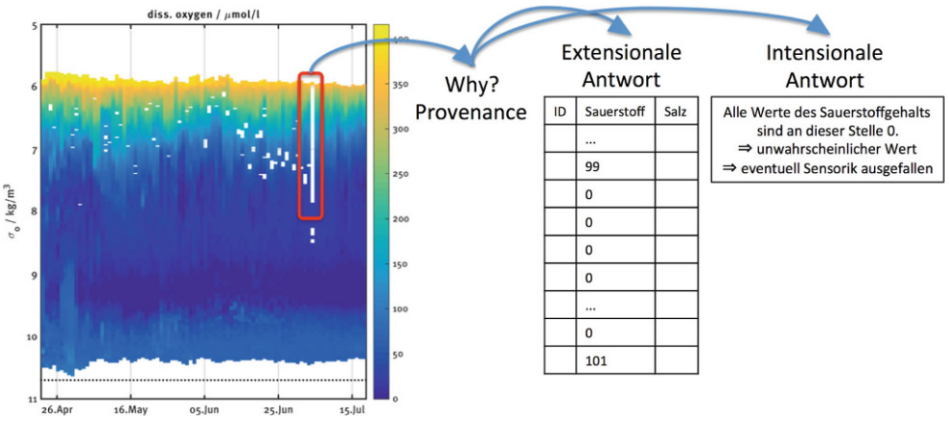

Abbildung 3.4: Anomalie im Sauerstoffgehalt [BKM+17]

Ein erstes Beispiel für einen Anwendungsfall der **Why**-Provenance liefert die Abbildung 3.4. Darin wurde eine Anfrage in Form eines Diagramms veranschaulicht. Diese zielt sinngemäß auf die Menge gelösten Sauerstoffs über einen bestimmten Zeitraum ab. Dabei fällt der lange weiße Strich auf, welcher sich deutlich von den anderen punktuellen Messpunkten abhebt. Die extensionale Antwort lässt erkennen, dass ab der dritten Zeile die Sauerstoffspalte vermehrt den Wert 0 annimmt. Diese Erkenntnis nimmt die intensionale Antwort auf und beschreibt sie sprachlich. Dazu kann außerdem gemutmaßt werden, dass diese Werte womöglich aufgrund eines Sensorikproblems falsch gemessen wurden.

Für die Untersuchung der Why-Provenance auf den zuvor genannten Beispieltabellen aus Abschnitt 3.3.2 habe ich diese in Hinblick auf die enthaltenen Attributwerte wie folgt abgeändert (Änderungen in Tabelle **DataCTD** sind rot markiert):

DataCTD

Longitude	Latitude	Depth	PSAL7IDD	TEMP7IDD	SVEL7PRD	
13.00	54.585	1000	8.755	4.331	1434.4	d_1
12.553	54.585	1000	8.751	4.332	1434.45	d_2
12.553	54.585	1000	8.752	4.336	1434.47	d_3
12.753	54.185	1000	8.754	4.338	1434.51	d_4
12.953	54.085	1000	8.758	4.327	1434.49	d_5

Stations

Name	Longitude	Latitude	
Station 1	13.00	54.585	s_1
Station 2	12.00	55.550	s_2
Station 3	12.553	54.585	s_3

Nun wird an die Datenbank D, welche die veränderten Tabellen **DataCTD** und **Stations** enthält, folgende Anfrage Q2 gestellt:

```
Q2:   SELECT   s.Name AS Name,
               s.Longitude AS Longitude,
               s.Latitude AS Latitude
      FROM     DataCTD d, Stations s
      WHERE    d.Longitude = s.Longitude
      AND      d.Latitude = s.Latitude
      AND      d.TEMP7IDD < 4.338
```

Die Ergebnisrelation dieser Anfrage ist in nachfolgender Tabelle ersichtlich. Zu beachten ist, dass das Tupel (Station 3, 12.553, 54.585) nur einmal in **R** auftaucht, obwohl jeweils d_2 und d_3 sein Vorkommen erzwingen. Genau für solch einen Fall ist die Why-Provenance sehr gut geeignet. Durch sie können alle Tupel gefunden werden, aufgrund derer ein Ergebnistupel existiert. Dabei werden einzelne Tupel oder ihre Kombination als *Zeuge* bezeichnet, wenn sie die Existenz eines Ergebnistupels begründen/bezeugen.

R

Name	Longitude	Latitude	
Station 1	13.00	54.585	r_1
Station 3	12.553	54.585	r_2

Tabelle 3.4: Ergebnisrelation von Q2(D)

Betrachten wir nun die *Why*-Provenance der Ergebnisrelation von Q2. Der „einfache“ Fall ist beim Tupel r_1 zu sehen. Seine Existenz wird, schaut man sich die Ursprungstabellen DataCTD und Stations sowie Q2 an, nur durch die Tupel d_1 in Verbindung mit s_1 bedingt. Daher lautet die *Why*-Provenance dazu **{{DataCTD(d_1), Stations(s_1)}}**.

Umfangreicher wird dies bei r_2 aus R. Beim erneuten Betrachten von Tabellen und Anfrage fällt auf, dass seine Existenz auf mehrere Arten „gerechtfertigt“ wird: Zum Einen durch d_2 und s_3, zum Anderen aufgrund der Verbindung von d_3 mit s_3. Die dazugehörigen Zwischentupel sehen vor der Projektion auf die Attribute Name, Longitude und Latitude folgendermaßen aus:

```
(12.553, 54.585, 1000, 8.751, 4.332, 1434.45, Station 3, 12.553, 54.585)
        Zeuge:  {DataCTD(d2), Stations(s3)}
```

Auf diese beiden wird nun wiederum eine Anfrage im Sinne der Why-Provenance gestellt und die Ergebnisse (jeweils unter den Tupeln) anschließend vereinigt. Dies kann bei nur zwei intermediären Tupeln

(12.553, 54.585, 1000, 8.752, 4.336, 1434.47, Station 3, 12.553, 54.585)
Zeuge: {DataCTD(d_3), Stations(s_3)}

durchaus per Hand vollzogen werden. Bei wachsender Anzahl wird diese Vorgehensweise jedoch schnell unübersichtlich und langwierig. Das Ergebnis der Vereinigung lautet wie folgt. Dadurch können bei even-

$$\{\{\text{DataCTD}(d_2), \text{Stations}(s_3)\}, \{\text{DataCTD}(d_3), \text{Stations}(s_3)\}\}$$

tuellen Unstimmigkeiten in Ergebnissen von Anfragen im Allgemeinen
Störungsursachen in den Originaltupeln erkannt werden. Weiterhin ist es möglich, die Glaubhaftigkeit von Ergebnissen durch mehrere Tupel zu festigen.

How-Provenance Anhand der vorherigen Provenance-Arten konnte festgestellt werden, aus welchen ursprünglichen Tabellen welche Tupel der Ergebnisrelation stammen (**Where**). Des Weiteren kann durch die **Why**-Provenance die Existenz dieser Tupel anhand der verwendeten Originaltupel begründet werden. Die Informationen werden also erweitert. Statt nur einer Tabelle sind zusätzlich die spezifischen Tupel gekennzeichnet, welche verwendet wurden. Dabei kann beobachtet werden, dass Provenance-Informationen der **Where**- in denen der **Why**-Provenance enthalten sind. Auch wenn diese Informationen hilfreich sein können, geben sie bisher keine Auskunft darüber, *wie* ein Ergebnis berechnet wurde und zustande kam. Genau zu diesem Zwecke kann eine weitere Provenance-Art herangezogen werden: die **How**-Provenance.

Diese Art der Data Provenance beschreibt über Polynome, wie ein konkretes Ergebnis oder ein Teil davon zustande gekommen ist. Dabei geht sie bis auf die Tupel-Ebene und enthält somit alle Informationen, aus welchen die zugehörige Why- und damit auch die Where-Provenance abgeleitet werden kann. Es besteht also folgende Verbindung zwischen diesen dreien: **Where** $\preccurlyeq$ **Why** $\preccurlyeq$ **How**, was bedeutet, dass von links nach rechts jede Data Provenance-Art auf die weiter rechts stehenden reduziert werden kann. Zur Einführung der Polynome wird folgende Query Q3 angenommen:

```
Q3:  SELECT  avg(PSAL7IDD)
     FROM    DataCTD d, Stations s
     WHERE   d.Longitude = s.Longitude
     AND     d.Latitude = s.Latitude
     AND     s.name = 'Station 3'
```

Zur besseren Nachvollziehbarkeit der Konstruktion des Provenance-Polynoms seien hier erneut die Tabellen **DataCTD** und **Stations** angegeben.

DataCTD

Longitude	Latitude	Depth	PSAL7IDD	TEMP7IDD	SVEL7PRD	
13.00	54.585	1000	8.755	4.331	1434.4	d_1
12.553	54.585	1000	8.751	4.332	1434.45	d_2
12.553	54.585	1000	8.752	4.336	1434.47	d_3
12.753	54.185	1000	8.754	4.338	1434.51	d_4
12.953	54.085	1000	8.758	4.327	1434.49	d_5

Stations

Name	Longitude	Latitude	
Station 1	13.00	54.585	s_1
Station 2	12.00	55.550	s_2
Station 3	12.553	54.585	s_3

Nach Ausführung der Anfrage ergibt sich die Ergebnisrelation **R**. Aus der **Why**-Provenance wissen wir, dass für dieses Ergebnis nur die Tupel d_2 und d_3 jeweils im Verbund mit s_3 herangezogen wurden.

R	avg(PSAL7IDD)
r_1:	8.7515

Das Polynom für die **How**-Provenance erklärt nun, wie genau das Tupel r_1 berechnet wurde: Dabei steht $*$ für einen natürlichen Verbund zweier Tupel, also die Verbindung über die in der Where-Klausel angegebenen Bedingungen (d.Longitude = s.Longitude AND d.Latitude = s.Latitude).

Listing 3.1: Provenance-Polynom der **How**-Provenance:

$$\frac{(d_2 * s_3) \odot 8.751 \oplus (d_3 * s_3) \odot 8.752}{(d_2 * s_3) \oplus (d_3 * s_3)}$$

Des Weiteren bedeutet $\odot$ die Multiplikation eines Wertes mit einem Teilpolynom und $\oplus$ steht für die „Addition“/Verknüpfung von Teilpolynomen. In diesem Beispiel werden also die Zwischentupel mit ihrem zugehörigen Wert aus PSAL7IDD multipliziert.

3.4 Schema-Evolutions-Operatoren

Schema-Evolution beschreibt die Veränderungen der Schemata einer Datenbank. Das beinhaltet sowohl die Änderungen der Relationenschemata selbst als auch Änderungen in den Schlüssel- und Integritätsbedingungen. Damit, ausgehend vom aktuellen Datenbankzustand, Aussagen über frühere Zustände gemacht und Anfragen an andere Schema-Versionen gestellt werden können, existieren die sogenannten **Schema-Evolutions-Operatoren** (englisch: **Schema Evolution Operators**). Sie beschreiben die Veränderungen von Schemata und Integritätsbedingungen mit Hilfe weniger Operatoren. Diese Operatoren wurden erstmals von Zaniolo et al. ([CMZ08]) eingeführt. Dafür untersuchten die Forscher etliche Schemata in der Praxis auf ihre Veränderungen. Die, welche am häufigsten auftraten, bekamen einen sogenannten Schema Evolution Operator zugeteilt. Dieser Abschnitt bezieht sich somit auf diese Arbeit sowie ihre Folgearbeiten.

Diese Schema-Evolutions-Operatoren werden unterteilt in Schema Modification Operators (SMOs, siehe Abschnitt 3.4.1) und Integrity Constraint Modification Operators (ICMOs, siehe Abschnitt 3.4.2). Während erstere die direkten Veränderungen von Relationen beschreiben können, wie zum Beispiel das Erstellen oder Löschen einzelner Attribute oder ganzer Relationen, werden die ICMOs eingesetzt, sobald sich Schlüssel- oder Integritätsbedingungen in Relationen ändern. So können diese Bedingungen allgemein verschärft oder abgeschwächt werden, indem Restriktionen hinzukommen oder verworfen werden.

3.4.1 Schema Modification Operators (SMOs)

Schema Modification Operators (SMOs) sind grundlegende Befehle, um ein Datenbankschema zu verändern. Dabei können sowohl neue Tabellen erstellt und alte gelöscht, als auch innerhalb von Relationen Veränderungen in Form von beispielsweise neuen Attributen herbeigeführt werden.

SMOs
CREATE TABLE R(a,b,c)
DROP TABLE R
RENAME TABLE R INTO T
COPY TABLE R INTO T
MERGE TABLE R,S INTO T
PARTITION TABLE R INTO S WITH cond, T
DECOMPOSE TABLE R INTO S(a, b), T(a, c)
JOIN TABLE R, S INTO T WHERE cond
ADD COLUMN d [AS const | func(a, b, c)] INTO R
DROP COLUMN c FROM R
RENAME COLUMN b IN R TO d

Tabelle 3.5: Grundlegende Schema Modification Operators nach [CMZ08]; rot markiert sind die Operatoren, welche im späteren Verlauf Verwendung finden

Die grundlegenden Schema Modification Operators (SMOs) nach Curino et al. [CMZ08] sind in Tabelle 3.5 dargestellt. Dabei sind die SMOs, welche im späteren Verlauf tatsächliche Verwendung finden, rot markiert. Im Folgenden werde ich nun die einzelnen Operatoren einführend beschreiben. Die Operatoren, welche in meinen späteren Untersuchungen Anwendung finden, versehe ich zudem mit konkreten Beispielen.

CREATE TABLE R(a,b,c) Dieser SMO erzeugt im Allgemeinen eine neue Tabelle mit Relationennamen **R** und Attributen `a`, `b` und `c`. Um eine brauchbare Tabelle zu erhalten, müssen jedoch weitere Operatoren folgen. So ist es beispielsweise Usus, dass eine Tabelle einen Primärschlüssel besitzt, welcher folglich angelegt werden muss. Diese und weitere Schlüssel- und Integritätsbedingungen werden jedoch nicht durch SMOs, sondern durch die ICMOs beschrieben (siehe Abschnitt 3.4.2).

Im nachfolgenden Beispiel wird eine neue Tabelle `Liegezeit` (siehe Abbildung 3.5) erstellt. Dabei bekommt sie die Attribute `Grund`, `Datum`, `HafenBez` und `Dauer`. Im späteren Verlauf wird diese Tabelle einen zusammengesetzten Schlüssel erhalten, welcher sich aus den Attributen `Datum`, `HafenBez` und `Dauer` ergibt. Da es sich hier außerdem um einen von `Fahrt` abhängigen Entity-Typen handelt, muss es außerdem dessen Primärschlüssel (`FahrtNr`) als Fremdschlüssel annehmen.

DROP TABLE R Dieser SMO löscht eine bestehende Tabelle mit Relationennamen **R**. Tabelle **R** wird mitsamt aller enthaltenen Tupel gelöscht und selbst durch ein erneutes Erstellen einer gleichnamigen Tabelle können diese nicht wiederhergestellt werden.

RENAME TABLE R INTO T Dieser SMO benennt eine bestehende Tabelle **R** in **T** um, wobei die enthaltenen Daten nicht verändert werden.

COPY TABLE R INTO T Dieser SMO kopiert eine bestehende Tabelle **R** samt aller enthaltener Daten in eine Tabelle **T**.

MERGE TABLE R,S INTO T Dieser SMO kopiert alle Spalten und deren Inhalte der beiden Tabellen **R** und **S** in eine neue Tabelle **T**, ohne dies an Bedingungen zu knüpfen.

Ich verstehe die Wirkung dieses Operatoren eher als **UNION**, da hier schlicht eine Vereinigung von Tabellen ohne zu erfüllende Bedingung(en) vollzogen wird. Im späteren Verlauf dieser Arbeit werde ich `MERGE` für das Zusammenführen verschiedener Attribute einführen. Für diese Operation existiert nach [CMZ08] bisher noch kein eigener SMO.

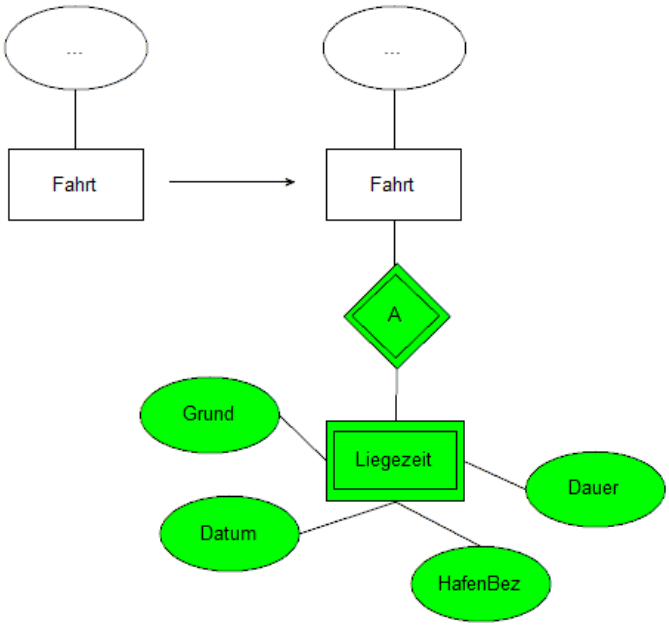

Abbildung 3.5: CREATE TABLE Liegezeit(Dauer, HafenBez, Datum, Grund)

PARTITION TABLE R INTO S WITH cond, T Dieser SMO teilt eine Tabelle R auf zwei andere Tabellen **S** und **T** auf. Dabei gehen Datensätze in **S** ein, wenn sie die Bedingung `cond` erfüllen. Trifft dies nicht zu, werden sie in **T** verschoben.

DECOMPOSE TABLE R INTO S(a, b), T(a, c) Dieser SMO teilt ebenfalls eine Tabelle **R** in zwei weitere auf (**S** und **T**). Hierbei ist jedoch keine Bedingung zu erfüllen. Es werden dagegen die Attribute/Spalten angegeben, welche mit Hilfe der Projektion in die jeweiligen Teiltabellen überführt werden sollen.

JOIN TABLE R, S INTO T WHERE cond Dieser SMO verknüpft zwei Tabellen **R** und **S** zu einer Relation **T**, wobei der natürliche Verbund angewendet über die Bedingung `cond`.

ADD COLUMN d [AS const | func(a, b, c)] INTO R Dieser SMO fügt in eine Tabelle **R** eine neue Spalte **d** ein. Dabei kann dieser optional ein Wert zugewiesen werden, welcher entweder konstant (`const`) oder das Ergebnis einer Funktion (`func`) über anderen Spalten (**a**,**b**,**c**) ist.

Im nachfolgenden Beispiel (siehe Abbildung 3.6) wird in eine Tabelle **Serie** ein neues Attribut **Kommentar** eingefügt, wobei alle vorherigen Attribute erhalten bleiben.

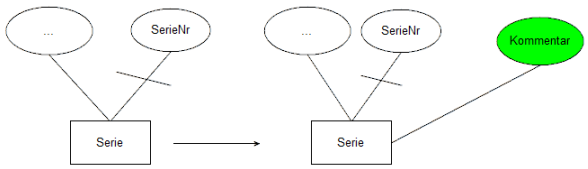

Abbildung 3.6: **ADD COLUMN Kommentar INTO Serie**

SerieNr	...	→	SerieNr	...	Kommentar

DROP COLUMN c FROM R Dieser SMO löscht eine Spalte **c** aus einer Tabelle **R**. Dabei ist darauf zu achten, dass keine Schlüssel- und Integritätsbedingungen verletzt werden. So könnten beispielsweise Spalten, auf denen ein `Value Constraint` oder ein `Primärschlüssel` definiert ist, gelöscht werden und dadurch die Tabelle für viele Fälle unbrauchbar werden, da Tupel nicht mehr so effektiv durchsucht werden können.

RENAME COLUMN b IN R TO d Dieser SMO benennt eine Spalte **b** einer Tabelle **R** in **d** um. Dabei ist darauf zu achten, dass ohne Speicherung des alten Spaltennamens Auswertungen, die für das ältere Schema entworfen wurden, nicht mehr ohne Weiteres funktionieren. Das bedeutet, dass dieser zunächst unscheinbare Befehl weitreichende Folgen mit sich bringen kann. Im nachfolgenden Beispiel (siehe Abbildung 3.7) aus dem Mapping später beschriebener Schemata wird das Attribut `WS` in `WS-ID` umbenannt.

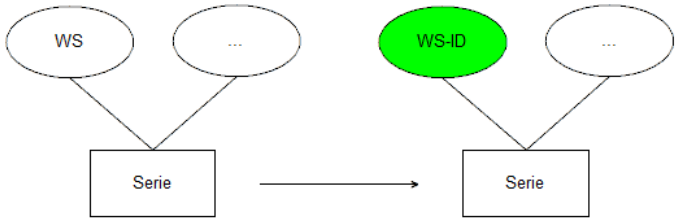

Abbildung 3.7: Attribut WS wird in WS-ID umbenannt

WS	...	→	WS-ID	...

3.4.2 Integrity Constraints Modification Operators (ICMOs)

Neben den zuvor vorgestellten **Schema Modification Operators** existiert noch eine weitere Klasse von **Schema Evolution Operators**. Diese **Integrity Constraints Modification Operators** (ICMOs) wurden erstmals in [CMDZ10] vorgestellt und beschreiben Änderungen in den Integritätsbedingungen von Tabellen. Genauer gesagt behandeln sie die drei Restriktionen `PRIMARY KEY`, `FOREIGN KEY` und `VALUE CONSTRAINT`.

ICMOs
ALTER TABLE R ADD PRIMARY KEY pk1(a, b) <policy>
ALTER TABLE R ADD FOREIGN KEY fk1(c, d) REFERENCES T(a, b) <policy>
ALTER TABLE R ADD VALUE CONSTRAINT vc1 AS R.e = '0' <policy>
ALTER TABLE R DROP PRIMARY KEY pk1
ALTER TABLE R DROP FOREIGN KEY fk1
ALTER TABLE R DROP VALUE CONSTRAINT vc1

Tabelle 3.6: Grundlegende **Integrity Constraints Modification Operators** nach [CMZ08]; rot markiert sind die Operatoren, welche im späteren Verlauf Verwendung finden

ALTER TABLE R ADD PRIMARY KEY pk1(a, b) <policy> Dieser ICMO fügt zu Tabelle **R** einen neuen Primärschlüssel **pk1** auf die Attribute **a** und **b** hinzu. Dabei ist auf die *Enforcement Policy* **<policy>** zu achten. Diese kennt zwei Ausprägungen:

- **CHECK**: Es wird überprüft, ob die aktuelle Tabelle den neuen Primärschlüssel erfüllt; andernfalls wird kein Primärschlüssel hinzugefügt
- **ENFORCE**: Der Primärschlüssel wird in jedem Fall hinzugefügt; Tupel, welche gegen den neuen Schlüssel verstoßen, werden gelöscht

Diese *Enforcement Policy* findet auch beim Hinzufügen eines Fremdschlüssels sowie eines Value Constraints Anwendung.

ALTER TABLE R ADD FOREIGN KEY fk1(c, d) REFERENCES T(a, b) <policy> Dieser ICMO fügt zu Tabelle **R** einen Fremdschlüssel **fk1** auf die Attribute **c** und **d** hinzu, welcher auf die Attribute **a** und **b** aus Tabelle **T** referenziert.

ALTER TABLE R ADD VALUE CONSTRAINT vc1 AS R.e = '0' <policy> Dieser ICMO fügt zu Tabelle **R** einen Value Constraint (Wert-Restriktion für ein Attribut) **vc1** auf das Attribut **a** hinzu. Dieses darf somit nur den Wert 0 enthalten.

ALTER TABLE R DROP PRIMARY KEY pk1 Dieser ICMO löscht den Primärschlüssel **pk1** aus Tabelle **R**.

ALTER TABLE R DROP FOREIGN KEY fk1 Dieser ICMO löscht den Fremdschlüssel **fk1** aus Tabelle **R**.

ALTER TABLE R DROP VALUE CONSTRAINT vc1 Dieser ICMO löscht den Value Constraint **vc1** aus Tabelle **R**.

Nachdem ich nun alle notwendigen Grundlagen eingeführt habe, gebe ich im folgenden Kapitel einen Überblick zum aktuellen Stand von Forschung und Technik (State of the Art) auf den Gebieten Schema-Evolution, Provenance sowie der Kombination der beiden Themen.

4 State of the art

Schema-Evolution Ein Ziel dieser Bachelorarbeit ist es, auftretende Änderungen in den Schemata des IOW mit Hilfe von sogenannten **Schema-Evolutions-Operatoren** zu beschreiben. Diese werden eingeteilt in **Schema Modification Operators** und **Integrity Constraints Modification Operators**. Dabei stütze ich mich auf die Definitionen von Curino et al., welche in mehreren Artikeln veröffentlicht wurden ([CMZ08], [CMDZ10], [CMTZ08]).

Die Auswahl der **Schema-Evolutions-Operatoren** beruht auf den in ihren Untersuchungen am häufigsten auftretenden Änderungen. Gegenstand ihrer Untersuchungen war dabei die Datenbank der Online-Enzyklopädie *Wikipedia*. Eine große Herausforderung bestand darin, dass jede Gruppe von Menschen und auch Einzelpersonen Artikel verfassen können, welche dann nach Prüfung des Inhaltes in die Datenbank aufgenommen werden. Folglich waren und sind dort Schema-Änderungen an der Tagesordnung. Genauer gesagt, existierten innerhalb von vier Jahren und sieben Monaten laut [CMTZ08] 171 verschiedene Schema-Versionen von *MediaWiki*, der Datenbank hinter *Wikipedia*. Dabei bedeutet jede Schema-Änderung auch eine Ungewissheit, ob alte Anfragen weiterhin auf dem neuen Schema funktionieren können.

Die Ergebnisse der Untersuchung zeigen, dass einige SMOs deutlich häufiger als andere auftraten. So wurden Tabellen an sich viel seltener verändert (erstellt, gelöscht etc.; 36mal) als darin enthaltene Spalten (233mal). Dabei traten vor allem die Befehle `ADD COLUMN` (104mal), `DROP COLUMN` (71mal) und `RENAME COLUMN` (43mal) in den Vordergrund. Dementsprechend wurden Operatoren, welche selten bis gar nicht verwendet wurden, nicht in die Sammlung der **Schema Modification Operators** aufgenommen (beispielsweise `DISTRIBUTE TABLE` oder `MOVE COLUMN`). Im späteren Verlauf dieser Bachelorarbeit werden wir jedoch sehen, dass auch die definierten **Schema Modification Operators** nicht für alle Schema-Änderungen ausreichen.

Diese **Schema Modification Operators** finden Anwendung im sogenannten *PRISM* bzw. *PRISM++*, welches zusätzlich ICMOs beinhaltet. Diese beiden Systeme unterstützen Datenbankadministrator/-innen bei der Schema-Evolution insbesondere beim Schema Mapping und Query Rewriting, d.h. dem Umschreiben von Datenbankanfragen. Dabei handelt es sich bei den Systemen um tatsächliche Umgebungen, welche die theoretischen Fortschritte in der Praxis durchführbar machen.

Neben den eigentlichen **Schema-Evolutions-Operatoren** und deren Anwendung auf ein Schema gibt es noch eine Reihe von Punkten, die dabei beachtet werden müssen. So können beispielsweise **Schema Modification Operators** einen Einfluss auf Integritätsbedingungen haben und umgekehrt.

Ein weiterer wichtiger Aspekt bei der Schema-Evolution ist die Frage nach der Invertierbarkeit von SMOs und ICMOs. Konkret heißt das:

- Können die Auswirkungen eines Operators durch Anwendung eines anderen wieder rückgängig gemacht werden?
- Wenn ein solcher inverser Operator existiert, können dann zusätzlich zum vorherigen Schema auch die Daten wiederhergestellt werden?

In der nachfolgenden Tabelle (Abbildung 4.1) sind alle **Schema Modification Operators** inklusive ihrer Inversen, sofern eine existiert, dargestellt. Dabei bedeutet „unique“, dass genau eine Inverse zu dem jeweiligen SMO existiert. Können zusätzlich die Daten, welche im Original vorhanden waren, wiederhergestellt werden, spricht man von einer „perfekten“ Inversen.

SMO	unique	perfect	Inverse(s)
CREATE TABLE	yes	yes	DROP TABLE
DROP TABLE	no	no	CREATE TABLE COPY TABLE NOP
RENAME TABLE	yes	yes	RENAME TABLE
COPY TABLE	no	no	DROP TABLE MERGE TABLE JOIN TABLE
MERGE TABLE	no	no	PARTITION TABLE COPY TABLE RENAME TABLE
PARTITION TABLE	yes	yes	MERGE TABLE
JOIN TABLE	yes	yes/no	DECOMPOSE TABLE
DECOMPOSE TABLE	yes	yes/no	JOIN TABLE
ADD COLUMN	yes	yes	DROP COLUMN
DROP COLUMN	no	no	ADD COLUMN, NOP
RENAME COLUMN	yes	yes	RENAME COLUMN
NOP	yes	yes	NOP

Abbildung 4.1: **Schema Modification Operators** inklusive ihrer Inversen [CMZ08]

PRISM++ [CMDZ10] erweitert das System PRISM [CMZ08] um die Betrachtung von Integritätsbedingungen. Genauer gesagt werden die drei grundlegenden Restriktionen *Primärschlüssel*, *Fremdschlüssel* und einfache *Value Constraints*, also Einschränkungen in Attributwerten, unterstützt. Dazu werden die sogenannten **Integrity Constraints Modification Operators** eingeführt, welche in nachfolgender Tabelle 4.1 abgebildet sind.

ALTER TABLE R ADD PRIMARY KEY pk1(a, b) <policy>
ALTER TABLE R ADD FOREIGN KEY fk1(c, d) REFERENCES T(a, b) <policy>
ALTER TABLE R ADD VALUE CONSTRAINT vc1 AS R.e = ’0’ <policy>
ALTER TABLE R DROP PRIMARY KEY pk1
ALTER TABLE R DROP FOREIGN KEY fk1
ALTER TABLE R DROP VALUE CONSTRAINT vc1

Tabelle 4.1: Übersicht über die Integrity Constraints Modification Operators

Die ersten drei ICMOs der Tabelle 4.1 fügen eine neue Integritätsbedingung hinzu (Primärschlüssel, Fremdschlüssel, Value Constraint), die letzten drei löschen diese in der gleichen Reihenfolge. Außerdem enthalten die ersten drei ICMOs der Tabelle eine sogenannte „<policy>“, welche die *Enforcement Policy* darstellt, also die Art der Durchsetzung neuer Integritätsbedingungen. Dabei wird unterschieden zwischen:

- **CHECK**: PRISM++ prüft, ob die aktuelle Tabelle die Integritätsbedingung erfüllt; andernfalls wird der ICMO nicht wirksam
- **ENFORCE**: Der ICMO wird in jedem Fall wirksam; Tupel, welche gegen die neue Integritätsbedingung verstoßen, werden gelöscht

Dabei geht es allgemein um folgende Fragestellung:

- **Gegeben**: Menge von Constraints IC1 auf Schema S1
- **Evolution** von S1 über Sequenz M (SMOs, ICMOs) nach Schema S2
- **Frage**: Welche Constraints IC2 muss S2 garantieren?

Bei Änderungen der Integritätsbedingungen ist außerdem zu beachten, dass diese nicht nur direkt durch *ICMOs* verändert werden können, sondern auch indirekt durch **Schema Modification Operators** (SMOs). Beispielsweise könnte durch den Operator `DROP COLUMN b FROM R` eine Integritätsbedingung verloren gehen, sofern diese über das Attribut `b` definiert ist.

Provenance Der Begriff **Provenance** kommt aus dem Englischen und bedeutet so viel wie „Herkunft“. In nachfolgender Abbildung ist eine Hierarchie von Provenance-Arten abgebildet, wie sie in [HDB17] beschrieben wurde. Angefangen mit der untersten Stufe, der „Provenance meta-data“, welche im Grunde jegliche Art von Informationen beschreiben kann, welche zur Erhebung von (digitalen/analogen) Daten eine Rolle spielen.

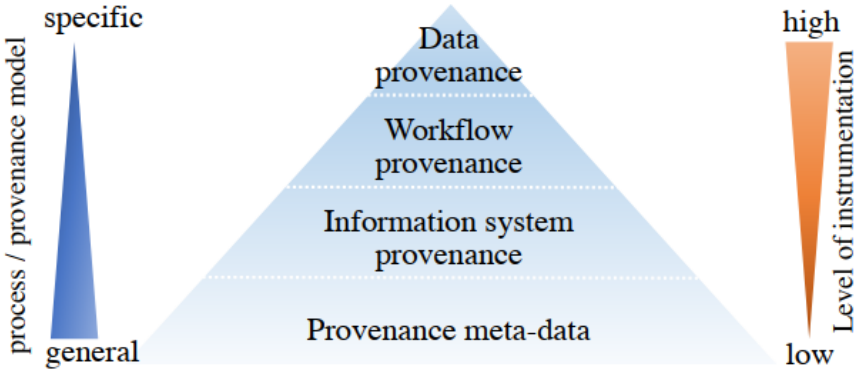

Abbildung 4.2: Provenance-Typen-Hierarchie nach [HDB17], Abschnitt 1.2

Die sogenannte „Information system provenance“ ist da schon etwas präziser. Mit ihrer Hilfe werden Metadaten gesammelt, welche Aufschluss über den Entstehungsprozess digitaler Daten geben. Dabei werden beispielsweise Fragen über benutzte Systeme sowie über In- und Output von Produktionsketten digitaler Daten beantwortet.

Die sogenannte „Workflow provenance“ beschreibt genauer als die vorherigen, was während des Entstehungsprozesses von Daten und Datensätzen passiert ist. Dabei geht es beispielsweise um Fragen über verantwortliche Personen sowie über Prozesse und deren Primärdaten, aufgrund welcher die vorliegenden Ergebnisdaten entstanden sind.

Die für diese Bachelorarbeit wichtigste Provenance-Art stellt die „Data provenance“ dar. Die ihr angehörigen Provenance-Informationen geben Antworten zu den Fragen:

- **Woher**, d.h. aus welchen Ursprungsrelationen, stammen die Daten, welche zu einem bestimmten Anfrageergebnis geführt haben?
- **Wie** wurde das Ergebnis berechnet? Wie sieht die Anfrageformel aus?
- **Warum** sieht das Ergebnis so aus? Welche Tupel sind dafür verantwortlich?
- **Warum** sieht das Ergebnis **nicht** anders aus?

Aus diesen Fragestellungen ergeben sich auch die vier Anfragetypen, welche im Umfeld der *Data Provenance* zu finden sind: **where**, **why**, **how** und **why not**, wobei wir uns allerdings auf die ersten drei konzentrieren werden. Außerdem beschränken wir uns auf die *extensionale* Antwort als Antworttyp. Dabei besteht das Ergebnis der Provenance-Anfrage aus konkreten Tupeln der Originaldaten. Andere Antworttypen wie beispielsweise die *intensionale* Antwort, bei welcher Daten beschrieben werden, werden nicht betrachtet. Sowohl die Anfrage- als auch die Antworttypen wurden im Kapitel 3.3.2 näher beschrieben, weshalb hier diese kurze Erklärung genügen soll.

Schema-Evolution und Provenance Somit kommen wir zum zweiten Ziel meiner Bachelorarbeit. Dabei soll überprüft werden, ob die Techniken der *Data Provenance* in Verbindung mit den Schema-Änderungen, welche durch die `Schema Evolution Operators` (SMOs & ICMOs) beschrieben wurden, ganz „normal“ angewendet werden können oder ob bei bestimmten Änderungen Probleme auftreten.

Einen Ansatz für die Verbindung von Schema-Evolution mit Provenance bieten Zaniolo et al. mit dem sogenannten „Archived Metadata and Provenance Manager (AM & PM)“[GZ12]. Dieser unterstützt unter anderem

- **Provenance Tracing**: Speichern von Provenance-Informationen zu Daten und zu Metadaten, welche den Prozess der Schema-Evolution beschreiben
- **Provenance Verifying**: Kombination der Provenance von Daten und Metadaten, ermöglicht die Betrachtung von Updates auf Daten und Änderungen auf Schemata
- **Source Data Recovery**: Wiederherstellung der Quelldatenbank aus Daten und archivierten Versionen der Datenbank

Dazu wird das Informationsschema (auch Data Dictionary, System Catalog; speichert Metadaten über Tabellen, Schemata, ...) dazu befähigt, auf Basis von Zeitstempeln, die Provenance-Informationen von Daten und Metadaten zu speichern. Dabei machen sie sich ein sogenanntes „Temporal Data Model“ zu Nutze, wobei sie sich auf die „Transaction Time“ beziehen. Das bedeutet, dass Schema-Änderungen und Updates auf Daten mit dem aktuellen Zeitstempel der Systemzeit ausgestattet werden. Dementsprechend ist es der Zeitpunkt, zu dem die Einträge gültig im Sinne des Datenbanksystems sind. Dabei sind Tupelwerte von dem Zeitpunkt ihres Zeitstempels bis zum aktuellen Zeitpunkt gültig.

Neben den Zeitstempeln werden ebenfalls die zuvor beschriebenen `Schema Modification Operators` und die `Integrity Constraints Modification Operators` eingeführt, um Schema- und Integritätsänderungen adäquat beschreiben zu können.

Durch die kombinierte Archivierung von Informationen über erfolgte Schema-Evolutionen und generelle Provenance-Informationen können mit Hilfe des *AM & PM Managers* drei Arten von Provenance-Anfragen an eine Datenbank gestellt werden:

- **Data Provenance-Anfragen**: Zusätzlich zu „Standard“- Anfragen wie **where**, **why** und **how** können auch **when**-Anfragen gestellt werden (zum Beispiel: **Wann** wurde Tupel x eingefügt oder geändert?)
- **Schema Provenance-Anfragen**: Ermöglichen Wiederherstellung früherer Datenbankzustände mit Hilfe von SMOs und ICMOs
- **Statistische Anfragen**: Statistik der Datenbank; beispielsweise Anzahl von Primär- und Fremdschlüsseln

Die Provenance von Daten und Metadaten werden dabei in der *Provenance Database* des Managers gespeichert, welche folgende Tabellen enthält.

- **Transaction** und **Transaction_Text**: enthalten Informationen zu Updates auf Daten mit Zeitstempeln;
- **SMO**: enthält SMOs mit Zeitstempeln, Quell- und Zielschema und Beschreibung des SMOs;
- **ICMO**: enthält ICMOs mit Zeitstempeln, Quell- und Zielschema und Beschreibung des ICMOs;
- **Table_Timestamp**: enthält Zeitstempel-Informationen zu einzelnen Tupeln einer Tabelle `Table`; jede Tabelle hat eine eigene Zeitstempel-Tabelle

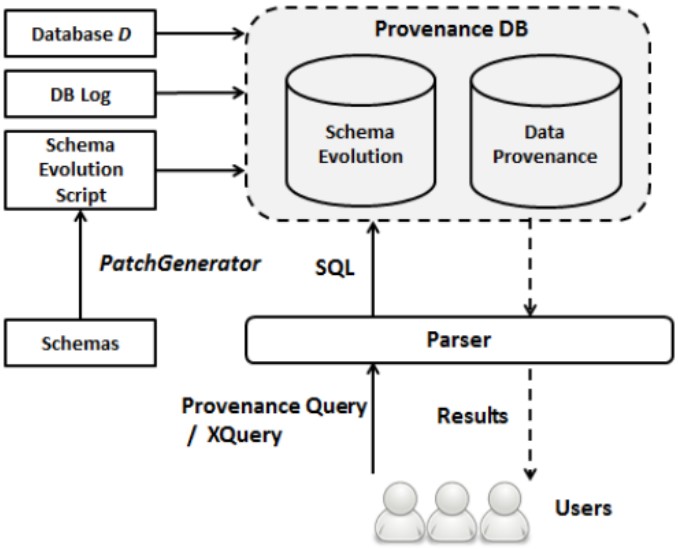

Abbildung 4.3: Architektur des AM & PM Managers [GZ12]

Die grundlegende Architektur des AM & PM Managers ist in Abbildung 4.3 dargestellt. Diese Architektur ist aufgeteilt in zwei Komponenten: der *Provenance Database*, welche die Tabellen mit Informationen zu Schema-Evolutionen und Data Provenance enthält, und dem *Parser*, welcher die Syntax eingehender Anfragen überprüft.

Die Schema-Evolutions-Tabellen werden mit früheren Schemata und den Skripten von Schema-Evolutionen gefüllt, welche die SMOs und ICMOs enthalten. Der Inhalt des Data Provenance-Teils wird bestimmt durch den aktuellen Datenbankinhalt und des Datenbank-Logs. Der in [GZ12] gewählte Ansatz sieht die Verwendung einer temporalen Datenbank vor, welche Zwischenzustände der gesamten Datenbank speichert. Wir hingegen wollen aus aktuellen Zuständen und Anfragen frühere Zustände rückberechnen können.

Im nun folgenden Kapitel stelle ich mein Konzept vor, in dem ich dieses gesetzte Ziel versuche zu erreichen. Darin werde ich zunächst ER-Modelle für die IOW-Schemata ausgewählter Jahre erstellen. Danach gehe ich auf die auftretenden Schema-Änderungen ein und überprüfe diese auf ihre Verträglichkeit mit Data Provenance.

5 Eigenes Konzept

In diesem Kapitel setzen wir uns zunächst mit den erhaltenen Daten des IOW auseinander und erstellen auf dieser Grundlage ER-Modelle für ausgewählte Jahre. Bei der Auswahl dieser Jahre für die Schema-Erstellung beziehe ich mich auf den Zeitraum von 1977 bis 2019. Da ich es aber nicht für sinnvoll halte, für jede Schema-Änderung ein eigenes Modell zu entwerfen, beschränke ich mich auf die drei Jahre, welche die meisten bzw. größten Änderungen enthalten.

Nach der Feststellung der Unterschiede zwischen den Schemata werden diese durch Anwendung von **Schema-Evolutions-Operatoren** beschrieben und die resultierenden Änderungen mit Data Provenance in Verbindung gesetzt. Dadurch soll überprüft werden, ob die auftretenden Schema-Änderungen sowohl durch SMOs beschreibbar und mit Data Provenance verträglich sind.

5.1 Vorliegende Daten - Ein Überblick

Die verwendeten Daten beziehen sich auf die sogenannten Terminfahrten. Diese werden in der Regel fünfmal im Jahr durchgeführt. Dabei sind stets gleiche Positionen anzufahren, um dortige Veränderungen der untersuchten Parameter (unter anderem Leitfähigkeit, Temperatur und Tiefe) aufzeichnen zu können. Die Ordner enthalten jedoch nicht nur die Daten der Terminfahrten, sondern auch andere Untersuchungsergebnisse. Um die richtigen Daten finden und sichten zu können, war der erste Anlaufpunkt der Datenuntersuchung die Datei „00_FAHRTEN_INFO“. Diese enthält grundlegende Informationen zu den Archiven, welche sich in dem Ordner befinden. Eine solche Datei ist exemplarisch in Abbildung 5.1 zu sehen. Der enthaltene Datensatz entstammt dem Jahre 1977. Aus diesen ersten Informationen lassen sich bereits einige Attribute für das spätere ER-Modell, insbesondere für den Entity-Typen `Fahrt`, herausfiltern.

Archnr	Fahrtbezeichnung	Fahrtkurzbez.	Datum	Expeditionsleiter
7796001	Erprobungsfahrt Jan/77	EF Jan/77	11. 1.-26. 1.	F.Moeckel
7796003	TF Feb/77	TF Feb/77	31. 1.-13. 2.	E.Francke
7796004	Dauerstation Darsser Schwel	DS Darßer Schw.	14. 2.-16. 2.	E.Francke
7796005	TF Maerz/77	TF Maerz/77	23. 3.- 6. 4.	D.Nehring
7796006	SCHALL April/77	SCHALL 04/05 77	20. 4.- 3. 5.	F.Moeckel
7796007	TF Mai/77	TF Mai/77	6. 5.- 1. 6.	E.Bengelsdorff
7796008	OPTIK Juni-Juli/77	OPTIK 06/07 77	9. 6.- 1. 7.	L.Gohs
7796009	OEKEX Juli/77	OEKEX 07/77	7. 7.-27. 7.	S.Schulz
7796010	TF Aug/77	TF Aug/77	2. 8.-19. 8.	E.Francke
7796011	TF Okt/77	TF Okt-Nov/77	19.10.-18.11.	D.Nehring
7796012	Erprobungsfahrt Nov-Dez/77	EF 11/12 77	27.11.- 1.12.	E.Bengelsdorff
7796013	Advektion VII	Advektion VII	2. 3.-16. 3.	R.Helm
7796014	BOSEX Sept/77	BOSEX 09/77	4. 9.-26. 9.	H.-J.Brosin

Abbildung 5.1: 00_FAHRTEN_INFO 1977

Exkurs – Daten vor dem Betrachtungszeitraum: 1951 Auch 1951 existierte bereits die Datei „00_FAHRTEN_INFO“, welche erste Daten über die Fahrten enthielt und als erster Anhaltspunkt für meine weiteren Untersuchungen diente. Nach Finden der richtigen Archivnummer gelangte ich über den Pfad `formate/archiv` zu einer Datei mit dem Namen „hc51006.kor“, welche auf den ersten Blick in gewissem Maße konfus auf mich wirkte. Nach kurzem Sortieren der gefundenen Parameter zeigte sich mir das in Abbildung 5.2 dargestellte Bild.

```
1TF
96DDR                ADW DER DDR - IFM     900201899600182091
1101SCHIFF      J.L.KRUEGER
520101520106 E.HEIN
215240000152400016 OSTSEE-TERMINFAHRT JANUAR 1952
    7   31(I1,I4,I5,I7,2I2,I4,I3,I2,I4,3I6,I7,2(I1,I4),I3,I3,I4,I1,2I2,I4,A1,A2,I4,2A
        412,A1,I5,I4,5A1,I4,2I3,2I4,A1,A2,6A1,A4,2A1,A2,A1,2A2,2A1,A2,A1,I1,
        5118(4(I5),3(I4)))
        61(I1,I4,I5,I7,18(4(I5),3(I4)))
    7
2TIEFE          001METER     0.1     0.0     1097DRAHTMESSUNG
12TEMPERATUR     003GRAD C    0.01    0.0     1097KIPPTHERMOMETER
22SALZGEHALT     0040/00      0.01    0.0     1093TITRATION-BERECHNET
32SIG-T          005          0.01    0.0     1099BERECHNET
42CHLORIDGEHALT  115G/DM**3   0.01    0.0     1092TITRATION N. MOHR/KNUDSEN
52SAUERSTOFFGEH. 101ML/L      0.01    0.0     1092TITRATION N. WINCKLER
62SAUERSTOFFSAET.1020/0       0.1     0.0     1099BERECHN.N.GREEN/CARRITT
```

Abbildung 5.2: Inhalt der Datei hc51006.kor – 1951

Da in dieser und anderen Dateien, welche sich auf die Jahre vor meinem eigentlichen Betrachtungszeitraum beziehen, Parameternamen nur zu erahnen, aber nicht festgehalten sind, verzichte ich hierbei darauf, ein eigenes Schema für 1951 zu entwickeln.

Um weitere Informationen über die verwendeten Schemata zu erhalten, sucht man nun das entsprechende Verzeichnis zu der Terminfahrt, in diesem Fall (siehe Abbildung 5.3) also beispielsweise die Archivnummer *7796003*. Darin befinden sich wiederum mehrere Ordner (vgl. Abbildung 5.3), wobei vor allem „validierte_daten“ von Interesse ist.

Abbildung 5.3: Inhalt des Ordners *7796003*

Validiert heißt in diesem Kontext, dass gemessene Daten an Sprungstellen interpoliert wurden. Dadurch erhalten die Wissenschaftler/-innen geschätzte, aber dennoch präzise Werte für Zwischentiefen. Das bedeutet unter anderem eine Zeitersparnis bei Messungen.

Innerhalb der *validierten Daten* folgt man nun den Unterordnern *ctd* und zuletzt *stt*. Darin befinden sich die gemessenen Parameter und weitere Informationen wie beispielsweise Steuerparameter in zwei getrennten Dateien pro Seriennummer, exemplarisch darstellt in Abbildung 5.4.

Abbildung 5.4: Ordnerinhalt der validierten CTD-Daten; je zwei Dateien pro Seriennummer

Dabei enthalten die *DCPD*-Dateien Steuerparameter und dergleichen, die *DUDB*-Dateien hingegen die gemessenen Werte. Ausschnitte ihrer Inhalte sind in den Abbildungen 5.5 und 5.6 zu sehen. Diese Zwei-Dateien-Struktur erstreckt sich bis zum Jahre 1997.

```
001 S:OSTSEE-BLOCK   ------------------------------------
002
010
100 Name des Validations-Bearbeiters:  Biesel              Datum: 1996-07-08
101 Steuerparameter zur Verarbeitungsart
102  GRENZ FIL  STKO DYN  WIRB ZEIK PAKO SAL  STIE FREI
103    0    0    1    0    0    0    0    0    2    0
104
120 Param.code  deutsch   T/F
121 PRES      ,DRUCK/dbar,T
122 DEPH      ,TIEFE     ,F
123 DOXY      ,SAUERSTOFF,F
124 CNDC      ,LEITFAEHIG,F
125 PSAL      ,SALZGEHALT,T
126 TEMP      ,TEMPERATUR,T
127 VARI      ,VARIOSENS ,F
128 SVEL      ,SCHALL    ,F
```

Abbildung 5.5: Ausschnitt aus einer DCPD-Datei von 1977

```
F          4PRES       PSAL       TEMP       ZLNR
S       7740          24          17          1          1          20
41         17          31          43          42         17          31
T  .6000000E+00  .1171000E+02  .1180000E+01  .1000000E+01
T  .1000000E+01  .1147705E+02  .1188046E+01  .2000000E+01
T  .2000000E+01  .1172999E+02  .1214093E+01  .3000000E+01
T  .3000000E+01  .1199057E+02  .1348565E+01  .4000000E+01
T  .4000000E+01  .1216000E+02  .1440000E+01  .5000000E+01
T  .5000000E+01  .1208462E+02  .1411894E+01  .6000000E+01
T  .6000000E+01  .1177000E+02  .1400000E+01  .7000000E+01
T  .7000000E+01  .1246000E+02  .1470000E+01  .8000000E+01
T  .8000000E+01  .1498452E+02  .1515618E+01  .9000000E+01
T  .9000000E+01  .1502410E+02  .1515092E+01  .1000000E+02
T  .1000000E+02  .1589277E+02  .1434802E+01  .1100000E+02
```

Abbildung 5.6: Ausschnitt aus einer DUDB-Datei von 1977

Zu beachten ist in Abbildung 5.5 die Jahreszahl am rechten Rand. An diesem Datum wurden diese Daten an das zu dem Zeitpunkt aktuelle Schema (1996) angepasst und digitalisiert. Aufgrund dieser Tatsache existieren im Grunde zwei Schemata für das Jahr 1977, zum Einen das tatsächlich verwendete und zum Anderen das erneuerte. In dieser Bachelorarbeit werden wir uns allerdings auf das ursprüngliche beschränken.

Die Struktur der *DCPD*- und *DUDB*-Dateien wurde 1997 abgelöst durch eine neue Struktur. Das Konzept der zwei Dateien wurde aufgelöst und an deren Stelle traten eine oder mehrere Dateien, die alle notwendigen Parameter und gemessenen Daten enthalten. Existieren pro Seriennummer mehrere Dateien, so beschreiben diese einzelne Messvorgänge. Die neue Ordnerstruktur ist exemplarisch in Abbildung 5.7 abgebildet. Die „Einstiegs“ - Datei *00_FAHRTEN_INFO* ist hingegen weiterhin vorhanden.

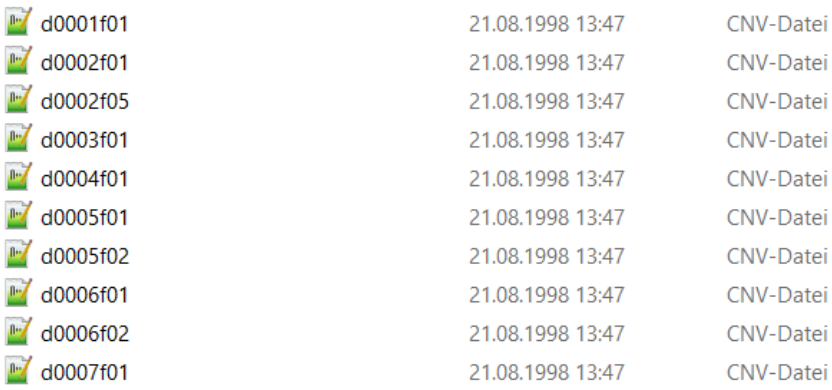

Abbildung 5.7: Struktur der validierten CTD-Daten ab 1997

Eine solche Datei, beispielsweise *d0001f01.cnv*, enthält konkret

- Steuerparameter,
- Fahrt- und Serienparameter,
- die gemessenen Daten,
- die zugehörigen Parameter inklusive der minimalen und maximalen Attributwerte und
- die verwendeten Sensoren beziehungsweise Messwerkzeuge.

Ein Ausschnitt der zuvor genannten Datei ist in Abbildung 5.8 zu sehen. Die gesamte Datei wird im Anhang 8.1 bereitgestellt.

Aus diesen Dateien und ihren Äquivalenten der anderen untersuchten Jahre werde ich im folgenden Abschnitt die jeweiligen ER-Modelle entwerfen. Danach untersuche ich die auftretenden Schema-Änderungen und beschreibe sie mit Hilfe von **Schema-Evolutions-Operatoren**.

```
* Sea-Bird SBE 9 Raw Data File:
* FileName = C:\SBE\DAT\0001F01.DAT
* Software Version 4.206
* Temperature SN = 1589
* Conductivity SN = 1349
* Number of Bytes Per Scan = 30
* Number of Voltage Words = 4
* System UpLoad Time = Feb 06 1997 10:36:04
* ReiseNr  = 44/97/02 Ostseemonitoring
* StationNr= 0001
* EinsatzNr= 0001
* SerieNr  = 01
* StatBez  = TFO5 aus < MONISTAT >
* Startzeit= 10:32:36 utc 06-FEB-97
* GPS_Posn = 54 13.9940N  012 04.5280E
* Echolote = XXXXXXX m 0011.92 m
* Luftdruck= 1027.1 hPa
* ThermoSal=  .666 grdC  12.166 ppth
* WS_Anzahl=  4
* Operator = Ruickoldt
# nquan = 15
# nvalues = 12
# units = metric
# name 0 = pr: pressure [db]
# name 1 = t090: temperature, ITS-90 [deg C]
# name 2 = iowSv: IOW sound velocity [m/s]
# name 3 = haardtP: Dr Haardt, phycoerythrin
# name 4 = haardtC: Dr Haardt, chlorophyll a
# name 5 = haardtT: Dr Haardt, turbidity
# name 6 = timeS: time [s]
# name 7 = scan: scan number
# name 8 = sal00: salinity, PSS-78 [PSU]
# name 9 = sigma-t00: density, sigma-t [kg/m^3]
# name 10 = potemp090: potential temperature, ITS-90 [deg C]
# name 11 = dm: dynamic meters
# name 12 = dz/dt: descent rate [m/s]
# name 13 = flag:  0.000e+00
# name 14 = nbin: number of scans per bin
# span 0 = 1.306, 11.747
# span 1 = 0.6169, 0.6205
# span 2 = 9.02, 9.10
# span 3 = 0.4221, 0.4298
# span 4 = 0.7317, 0.9027
# span 5 = 0.0609, 0.0669
```

Abbildung 5.8: Ausschnitt aus einer Seriendatei von 1997

5.2 Verwendete Schemata

In den vorliegenden Daten des IOW bestehen mehrere Unterschiede. So weisen oftmals die Primärdaten einer Fahrt weniger Parameterwerte auf als die der zugehörigen validierten Dateien. Um dies in den Tabellen zu realisieren, wird ein identifizierendes Attribut *validiert (y/n)* eingeführt und fungiert zusammen mit der Zeilennummer (ZLNR) sowie der Seriennummer (SerieNr) als zusammengesetzter Schlüssel für Tupel in der Datentabelle.

Ein weiterer Unterschied besteht zwischen den tatsächlich auf der jeweiligen Fahrt gemessenen Parametern und denen der Metadaten. Dies hat unter anderem seinen Ursprung im Zeitpunkt der Validierung, welcher mitunter 19 Jahre später liegt, so etwa bei den Daten von 1977. Die folgenden Schemata der einzelnen Jahre beziehen sich ausschließlich auf die ersteren, also tatsächlich gemessenen Daten. Schema-Änderungen zwischen diesen beiden Datensätzen werden nicht betrachtet.

Die Erstellung und der Vergleich der Schemata beginnen in dieser Bachelorarbeit mit dem Jahre 1977. Diese Jahreszahl ist zum Einen so gewählt, um einen Vergleich zu der Masterarbeit von Jiawei Yan [Yan] zu erleichtern. Der Hauptgrund besteht jedoch darin, dass in den Jahren vor 1977 für mich kein Schema eindeutig zu identifizieren war. Vielmehr waren lediglich Daten gespeichert, welche aber keinen klaren Rückschluss auf die ihnen zugrunde liegenden Parameter zuließen (vgl. Abbildung 5.9). Informationen, welche sich aus der Datei in Abbildung 5.9 herauslesen lassen, sind beispielsweise das verantwortliche

Institut („ADW DER DDR - IFM“) inklusive des zugehörigen Landes („DDR“) oder die Parameternamen „Tiefe, Temperatur, Salzgehalt“ inklusive der Messmethoden „Drahtmessung, Kippthermometer, Titration-berechnet“.

```
1TF
96DDR                ADW DER DDR - IFM     900201899600182091
1101SCHIFF      J.L.KRUEGER
520101520106 E.HEIN
215240000152400016 OSTSEE-TERMINFAHRT JANUAR 1952
    7   31(I1,I4,I5,I7,2I2,I4,I3,I2,I4,3I6,I7,2(I1,I4),I3,I3,I4,I1,2I2,I4,A1,A2,I4,2A
        412,A1,I5,I4,5A1,I4,2I3,2I4,A1,A2,6A1,A4,2A1,A2,A1,2A2,2A1,A2,A1,I1,
        5118(4(I5),3(I4)))
        61(I1,I4,I5,I7,18(4(I5),3(I4)))
    7
2TIEFE          001METER     0.1     0.0     1097DRAHTMESSUNG
12TEMPERATUR     003GRAD C    0.01    0.0      1097KIPPTHERMOMETER
22SALZGEHALT     0040/00      0.01    0.0      1093TITRATION-BERECHNET
32SIG-T          005          0.01    0.0      1099BERECHNET
42CHLORIDGEHALT  115G/DM**3   0.01    0.0      1092TITRATION N. MOHR/KNUDSEN
52SAUERSTOFFGEH. 101ML/L      0.01    0.0      1092TITRATION N. WINCKLER
62SAUERSTOFFSAET.1020/0       0.1     0.0      1099BERECHN.N.GREEN/CARRITT
```

Abbildung 5.9: Datensatz von 1951 (Ausschnitt)

Untersucht habe ich die Schemata aus den Jahren 1977, 1989, 1992, 1997, 2000, 2005, 2010, 2015, 2017, 2018 und 2019. Da diese sich jedoch häufig nicht oder nur geringfügig geändert haben, habe ich nur jene aus den Jahren 1977 als Anfangsschema, 1997 als umfassendste Änderung und 2019 als aktuelles Schema in diesen Abschnitt aufgenommen. Die restlichen sind im Anhang (Kapitel 8) ersichtlich. Die dort rot markierten Entity-Typen, Beziehungstypen und Attribute zeigen die Unterschiede zum jeweils vorhergegangenen Schema auf.

Da die zwei früheren Schemata, 1977 (vgl. Abbildung 5.10) und 1997 (vgl. Abbildung 5.11), auf das Schema von 2019 (vgl. Abbildung 5.12) als Universalschema übertragen (gemappt) werden, sind im 2019er-Schema die Veränderungen gekennzeichnet, in grüner Farbe die von 1977 auf 2019 und in roter Farbe von 1997 auf 2019.

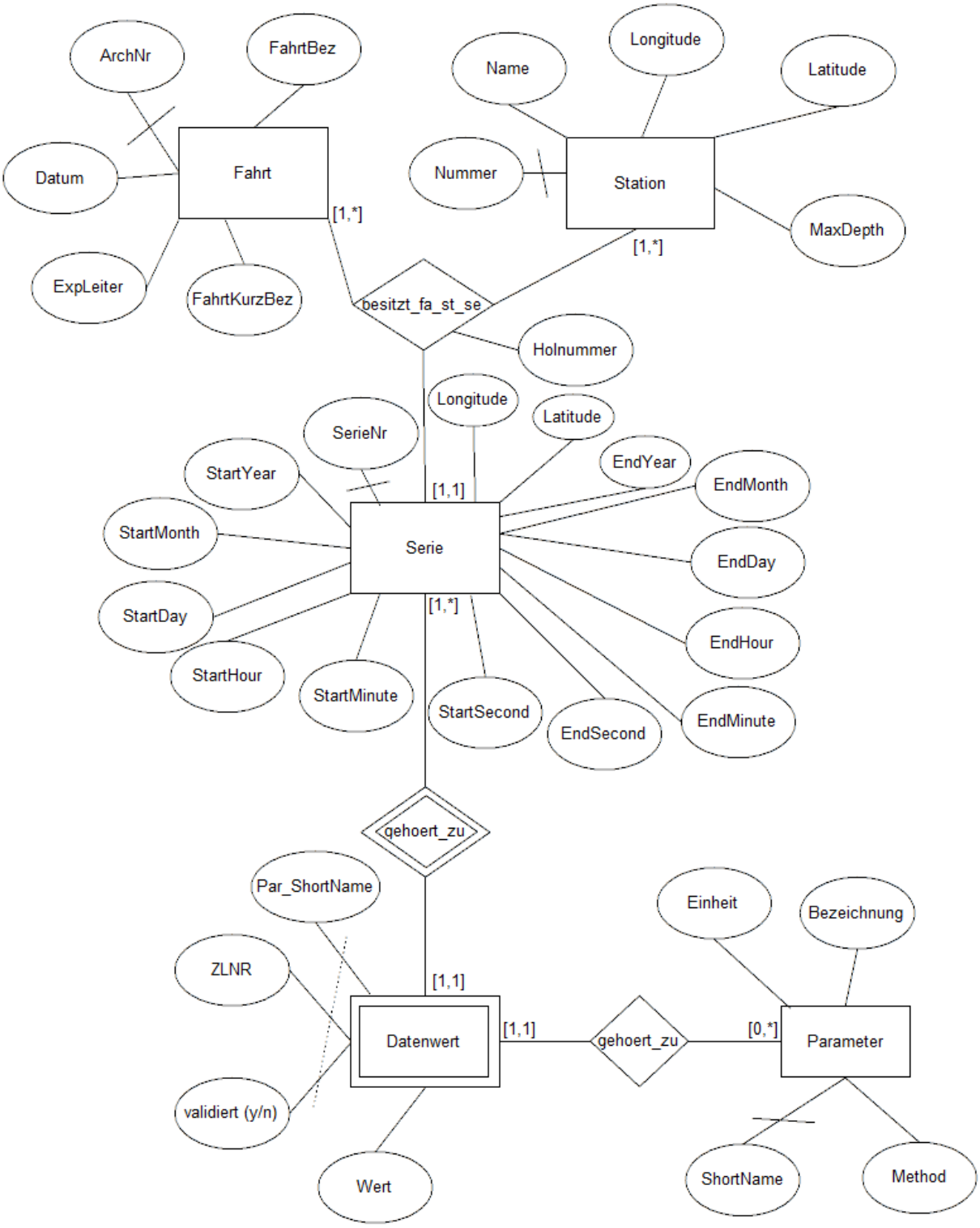

Abbildung 5.10: Entity-Relationship-Modell für das Schema von 1977

5.2.1 Das Schema von 1977

Das Schema von 1977 (vgl. Abbildung 5.10) ist im Vergleich zu den folgenden Schemata am übersichtlichsten. Es besteht aus den fünf Entity-Typen `Fahrt`, `Station`, `Serie`, `Datenwert` und `Parameter`, welche über insgesamt drei Beziehungstypen miteinander verbunden sind. Dabei stehen die ersten drei über den dreiwertigen Beziehungstyp `besitzt_fa_st_se` in Beziehung zueinander. Die Kürzel im Beziehungstypnamen stehen für die drei anliegenden Entity-Typen. Dazu sei gesagt, dass alle in den ER-Modellen vorkommenden Beziehungsnamen frei ausgedacht worden sind. Die Holnummer beschreibt, zum wievielten Male während einer `Fahrt` eine bestimmte `Station` angefahren und dabei eine `Serie` erstellt wurde. Die Begründung für die Wahl einer dreiwertigen Beziehung ist, dass

1. während einer `Fahrt` immer `Stationen` angefahren werden, wobei immer mindestens eine `Serie` aufgenommen wird;
2. eine `Serie` nur während einer `Fahrt` aufgenommen wird; wenn eine `Station` angefahren wird
3. eine `Station` einen geographischen Punkt beschreibt, welcher auch unabhängig von `Fahrten` und `Serien` existiert.

Bei der Erstellung des Relationenschemas wird aus dem Beziehungstyp `besitzt_fa_st_se` eine gleichnamige Tabelle, welche neben der `Holnummer` auch die Primärschlüssel der angrenzenden Entity-Typen `Fahrt` und `Serie` als Attribute enthält. Statt des Primärschlüssels des Entity-Typen `Station` verweist die Beziehung per Fremdschlüssel auf den Stationsnamen `Name`, welcher im IOW-Umfeld einzigartig und noch dazu leichter zuzuordnen ist.

Sowohl die `Nummer` des Entity-Typen `Station` als auch die `SerieNr` des Entity-Typen `Serie` sind sogenannte fortlaufende Nummern. Das heißt, dass neue Einträge in den resultierenden Tabellen eine neue Identifikationsnummer, quasi einen künstlich eingefügten Primärschlüssel, bekommen. Deshalb ist es für die Datenpflege leichter, den Namen einer Station einzupflegen, da dieser ja in den Daten vorhanden ist. Künstliche Schlüssel setzen oftmals eine vorherige Suche nach diesem voraus.

Der Entity-Typ `Datenwert` ist von `Serie` abhängig, da ohne eine Serie keine Daten aufgenommen werden. Dabei gehört ein `Datenwert` immer zu genau einer `Serie`. Der Primärschlüssel eines `Datenwerts` setzt sich zusammen aus den Attributen `ZLNR` (Zeilennummer in der Ursprungsdatei), `validiert (y/n)` (Kennzeichen, ob Ursprungsdatei validiert ist), `Par_ShortName` (`ShortName` als Primärschlüssel des zugehörigen `Parameters`) und der `SerieNr` (Primärschlüssel der übergeordneten `Serie`).

Der Entity-Typ `Parameter` steht über den Beziehungstyp `gehoert_zu` mit `Datenwert` in Beziehung. Dabei wird jedem `Datenwert` ein `Parameter` über das Attribut `ShortName` zugeordnet. Das hat den einfachen Grund, dass keine Werte gemessen werden, ohne einen entsprechenden Parameter festzulegen.

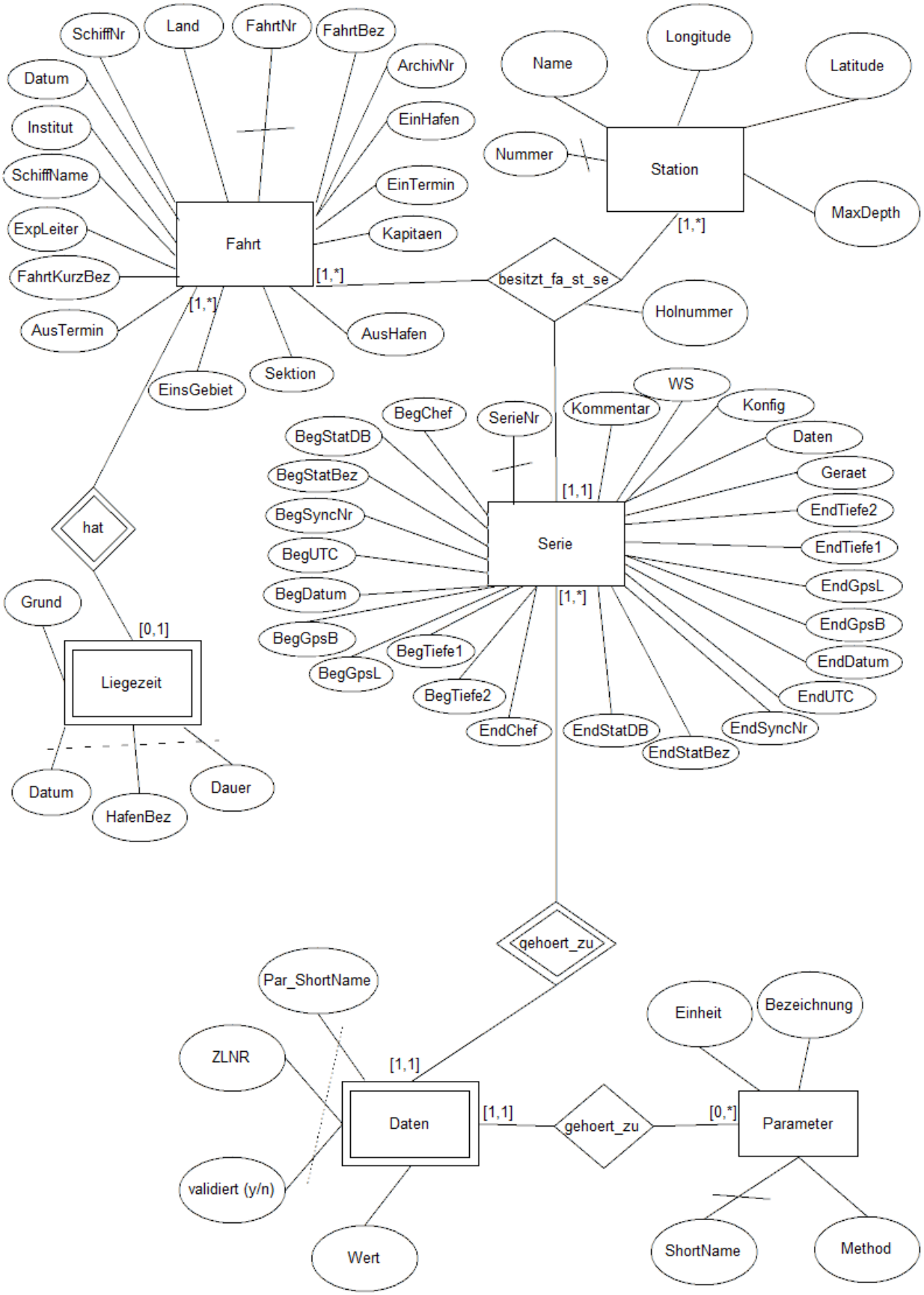

Abbildung 5.11: Entity-Relationship-Modell für das Schema von 1997

5.2.2 Das Schema von 1997

Im Vergleich zum vorhergehenden Schema von 1977 ist das Schema von 1997 (vgl. Abbildung 5.11) gewachsen und komplexer geworden. Die zuvor vorhandenen Entity-Typen wurden übernommen, wobei sich einige ihrer Attribute geändert haben. Zusätzlich wurde ein neuer Entity-Typ `Liegezeit` hinzugefügt, welcher anfallende Liegezeiten auf `Fahrten` beschreibt. Dementsprechend ist die `Liegezeit` von diesem Entity-Typen abhängig. Bei der Umformung in das Relationenschema besteht der Primärschlüssel der `Liegezeit` dementsprechend aus dem Teilschlüssel `(Datum, HafenBez, Grund)` und dem Primärschlüssel des übergeordneten Entity-Typen `Fahrt`. Der gesamte Primärschlüssel der Liegezeit lautet also `(FahrtNr, Datum, HafenBez, Grund)`.

Durch den neuen Primärschlüssel, welcher wiederum aus dem neu eingeführten Attribut `FahrtNr` besteht, könnten Probleme auftreten. So gibt es für den Standard- oder Default-Wert bei neu hinzugefügten Attributen nur die folgenden Möglichkeiten, wobei nur die dritte und vierte von diesen Möglichkeiten gültige Werte für einen Primärschlüssel erzeugt. Dabei würde ich der Einfachheit halber die letztere Möglichkeit bevorzugen.

- Kein Wert angegeben → `null`-Werte
- Default-Wert angegeben → gleiche Werte in allen Tupeln
- Wert dynamisch über Funktion angeben
- Wert für vorhandene Tupel entspricht Wert der ArchNr

Neben der neuen Tabelle haben sich bei den Entity-Typen `Fahrt` und `Serie` die zugehörigen Attribute stark geändert. So wurden viele Attribute hinzugefügt und beim genaueren Hinsehen auch zusammengefasst beziehungsweise aufgeteilt. Die Spezifikation dieser und weiterer Änderungen folgt im späteren Verlauf (siehe Abschnitt 5.3) dieses Kapitels.

Auffällig an diesem Schema ist, dass bei einer Serie viele Attributwerte zweimal festgestellt werden. So wird einmal zum Anfang der Messung ein Wert für das Attribut `BegXXX` festgestellt und zum Ende der Messung ein Wert für das Attribut `EndXXX`. Dabei ist „XXX“ als Platzhalter für verschiedene Attribute zu sehen wie beispielsweise `GpsL` oder `GpsB`. Nach meinen Analysen wurde hier offensichtlich das ursprüngliche Attribut `Longitude` aufgeteilt in die zwei Attribute `BegGpsL` und `EndGpsL`. Da 1977 jedoch nur ein Wert gemessen wurde, wird dieser höchstwahrscheinlich beim Mapping auf das Schema von 1997 (sowie auch 2017) in beide Attribute `Beg-` und `EndGpsL` eingefügt.

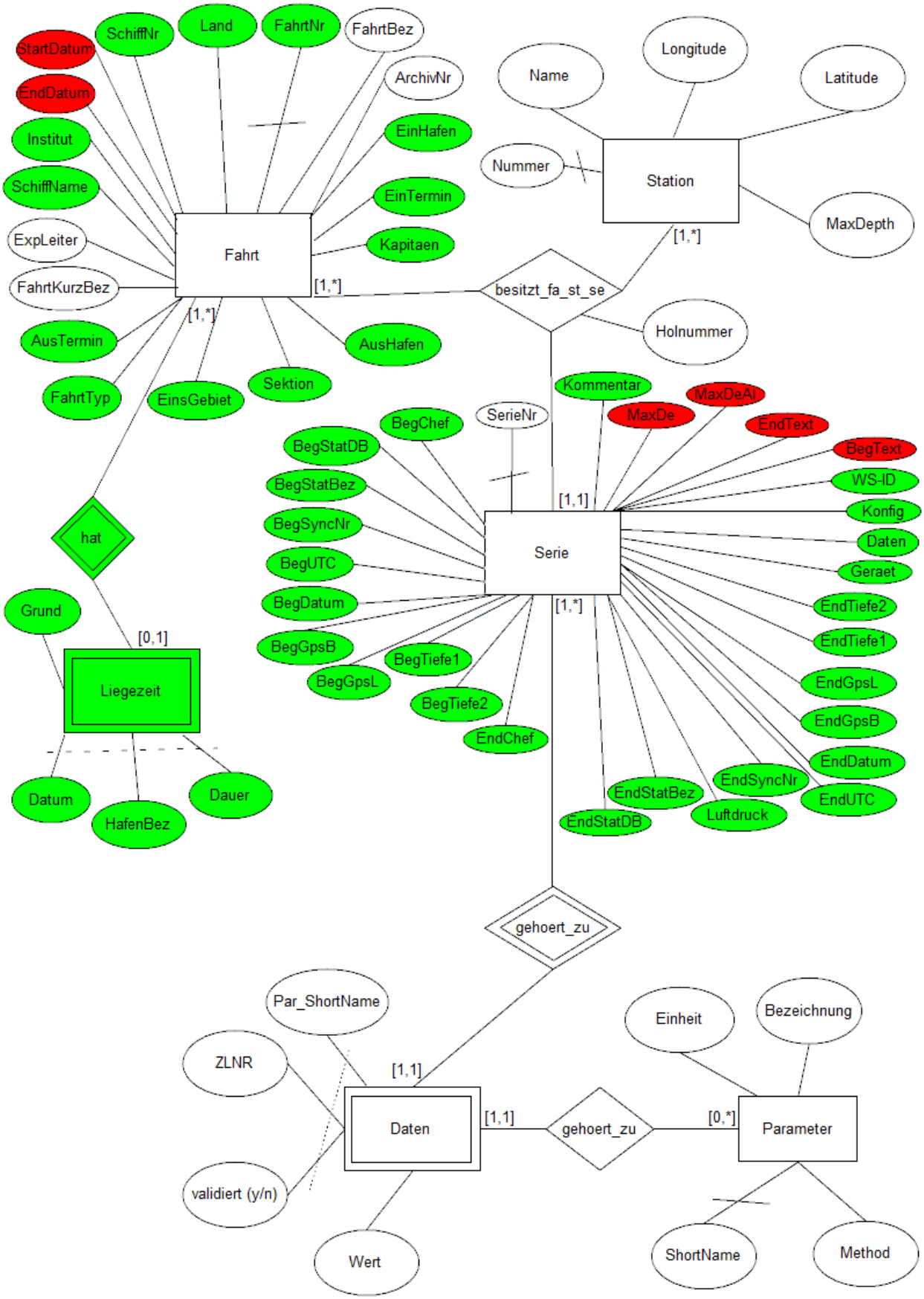

Abbildung 5.12: Entity-Relationship-Modell zu den Schemata von 2017, 2018 und 2019;
Änderungen von 1977 bis 2019 in grüner Farbe;
Änderungen von 1997 bist 2019 in roter Farbe

5.2.3 Das Schema ab 2017

Das Schema der Jahre ab 2017 (vgl. Abbildung 5.12) dient in dieser Bachelorarbeit zusätzlich als Universalschema, da es im Vergleich zu den anderen die meisten Informationen enthält und das Schema ist, auf welchem die IOWDB zum Zeitpunkt der Erstellung dieser Bachelorarbeit arbeitet.

Zu sehen sind die gleichen Entity-Typen und Beziehungstypen wie beim Schema von 1997, hier sind jedoch Änderungen farblich hervorgehoben. Alle grünen Markierungen symbolisieren die Schema-Änderungen vom Schema von 1977 bis 2017. Dagegen stellen die roten Markierungen alle Schema-Änderungen vom Schema von 1997 bis 2017 dar.

Die Grundstruktur des Schemas von 2017 hat sich im Vergleich zu dem von 1997 nicht wesentlich verändert. So sind die Entity-Typen und Beziehungstypen gleich geblieben, lediglich einige Attribute haben sich verändert. So fällt bei der Farbbetrachtung als erstes auf, dass im Vergleich zu der Anzahl der Änderungen von 1977 auf 2017 viel weniger Änderungen von 1997 auf 2017 erfolgten. So beschränken sich diese auf lediglich fünf Änderungen, alle auf Attributebene. Dagegen bleibt die Verhältnismäßigkeit jedoch erhalten. So stellt bei beiden Mappings das bloße Hinzufügen von Attributen die mit Abstand größte Anzahl an Schema-Änderungen.

Im nun folgenden Abschnitt arbeite ich die auftretenden Änderungen heraus und beschreibe sie mit Hilfe der **Schema-Evolutions-Operatoren**. Dabei betrachte ich die Änderungen des Schemas von 1977 auf das Schema von 2017 zusammen mit den Änderungen des Schemas von 1997 auf das Schema von 2017. Nach diesen Untersuchungen überprüfe ich die quantitativen Zusammenhänge zu den Untersuchungen von Zaniolo et al. in Artikel [CMTZ08].

5.3 Spezifikation der Änderungen auf Schemata und Daten

In diesem Kapitel werden die Schema-Änderungen von 1977 und 1997 auf 2019 gemeinsam betrachtet, da beim Übergang von 1997 auf 2019 keine Schema-Änderungen auftreten, welche nicht zuvor durch 1977 abgedeckt werden. Zu diesem Zwecke sollen zunächst alle Typen von Veränderungen genannt, erläutert und jeweils an einem konkreten Beispiel aus den Schemata belegt werden. Dabei ist unter anderem die Frage zu klären, ob alle diese Änderungen auf grundlegende Schema-Evolutions-Operatoren (vgl. Abschnitt 3.4) zurückgeführt und wo Grenzen dieser aufgezeigt werden können.

Hinzufügen eines Attributes Diese Art der Änderung ist eine der am leichtesten zu beschreibenden Schema-Änderungen und tritt vergleichsweise häufig auf, am deutlichsten jedoch beim Übergang von 1977 auf 2019. Betrachtet man beispielsweise den Entity-Typ `Serie` beider Schemata, sieht man den nahezu vollständigen Austausch der zugehörigen Attribute, auch wenn diese Änderungen nicht ausschließlich auf `ADD COLUMN` zurückzuführen sind.

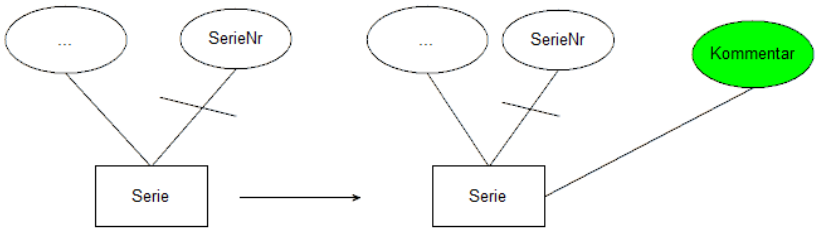

Abbildung 5.13: ADD COLUMN Kommentar INTO Serie

Diese Art der Schema-Änderung lässt sich durch den Schema-Modification-Operator (SMO) **ADD COLUMN** d **INTO** R ausdrücken, wobei d ein neues Attribut (entspricht im relationalen Datenbankentwurf einer neuen Spalte) und R den zugehörigen Entity-Typ (entspricht im relationalen Datenbankentwurf einer Tabelle) darstellt (vgl. Abbildung 5.13). Beispielsweise kommt das Attribut `Kommentar` zum Entity-Typ `Serie` hinzu. Dies wird ausgedrückt durch den SMO

```
ADD COLUMN Kommentar INTO Serie;
```

Diese Schema-Änderung taucht beim Vergleich der beiden Schemata von 1977 und 1997 mit dem von 2019 insgesamt mehr als fünfzigmal auf und ist so mit großem Abstand die häufigste Art. Diese Beobachtung deckt sich auch mit den Erkenntnissen aus [CMTZ08]. Dabei waren 38.7% aller Schema-Änderungen von diesem Typ.

Löschen eines Attributes Entgegen anfänglicher Beobachtungen meinerseits stellte sich bei näherer Betrachtung heraus, dass während des Mappings der früheren Schemata von 1977 und 1997 auf das aktuelle Schema von 2019 kein Attribut wirklich gelöscht wird. Vielmehr werden Attribute, welche in früheren Schemata zu finden sind und in neueren nicht, nicht direkt gelöscht, sondern zusammengefasst beziehungsweise auf mehrere Attribute aufgeteilt. Diese Arten der Schema-Änderungen werden am Ende dieses Abschnitts ausführlicher beschrieben.

Der Umstand, dass augenscheinlich keine Attribute komplett gelöscht werden, ist auf der einen Seite natürlich positiv zu sehen. So kommt es dadurch nicht zu Informationsverlusten, weshalb die Herkunft (*Provenance*) der Daten auch bis weit in die Vergangenheit nachgewiesen werden kann. Betrachtet man jedoch die Zahlen und Prozente in [CMTZ08], wo diese Art der Schema-Änderung mit 26.4% verhältnismäßig oft auftrat, sieht man einen großen Unterschied. Dies könnte daran liegen, dass bei der

Datenbank von Wikipedia laut [CMTZ08] einige temporäre Tabellen und Spalten existierten, welche des Öfteren gelöscht und wieder neu erstellt worden sind, nachdem die Informationen in andere Tabellen kopiert wurden.

Erstellen einer Tabelle Diese Form der Schema-Änderung taucht in den gewählten Beispielen nur einmal beim Vergleich von 1977 und 2019 auf. Erkennbar ist sie am neuen abhängigen Entity-Typ `Liegezeit`, welcher ab 1997 Bestandteil des Schemas ist.

Diese Art der Schema-Evolution kann ausgedrückt werden durch den nachfolgenden Schema Modification Operator, veranschaulicht in Abbildung 5.14.

```
CREATE TABLE Liegezeit(Dauer, HafenBez, Datum, Grund, FahrtNr);
```

Dabei ist zu beachten, dass neben dem SMO `CREATE TABLE` auch Integritätsbedingungen angegeben werden müssen. So ist es zum Einen notwendig, einen Primärschlüssel zu erstellen. Dieser wird sich aus den Attributen `Datum`, `HafenBez` und `Dauer` zusammensetzen. Zum Anderen muss aufgrund der Abhängigkeit der `Liegezeit` von `Fahrt` ein darauf referenzierender Fremdschlüssel angegeben werden. Dieser wird aus dessen Primärschlüssel `FahrtNr` bestehen.

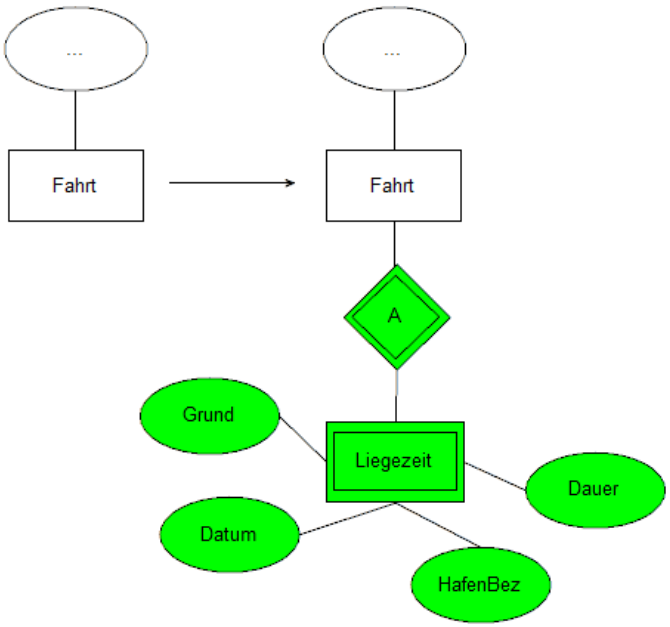

Abbildung 5.14: **CREATE TABLE** Liegezeit(Dauer, HafenBez, Datum, Grund)

Hinzufügen eines Primärschlüssels Mit dem Hinzufügen der neuen Relation `Liegezeit` wird laut der Theorie des relationalen Datenbankmodells auch ein neuer Primärschlüssel benötigt, um Tupel eindeutig identifizieren zu können. Des Weiteren wird beim Mapping des Schemas von 1977 auf das Schema von 2019 der Primärschlüssel des Entity-Typen `Fahrt` geändert, was einem Hinzufügen des neuen und Löschen des alten Primärschlüssels gleichkommt. Für diesen Abschnitt ziehen wir jedoch das Beispiel der neuen Relation `Liegezeit` heran.

Der bisher erreichte Stand der im vorherigen Abschnitt erstellten neuen Relation `Liegezeit` ist in Abbildung 5.14 zu sehen. Offensichtlich fehlt ein Primärschlüssel. Dieser kann nun, bevor etwaige Daten in die Relation eingefügt werden, durch den folgenden ICMO erstellt werden:

```
ALTER TABLE Liegezeit
ADD PRIMARY KEY pkLZ(FahrtNr, Datum, HafenBez, Dauer) CHECK;
```

Das Attribut `FahrtNr` wurde vom Entity-Typ `Fahrt` übernommen, da die `Liegezeit` davon abhängig ist und damit der Primärschlüssel als Fremdschlüssel übernommen werden muss.

Dabei ist auf die *Enforcement Policy* „CHECK“ zu achten. Durch diese Angabe wird zunächst die Tabelle `Liegezeit` dahingehend überprüft, ob einzelne Tupel gegen diese Integritätsbedingung verstoßen. Ist dies auch nur bei einem enthaltenen Tupel der Fall, wird der ICMO `ADD PRIMARY KEY` nicht angewendet. Dementsprechend sollte der Primärschlüssel direkt nach oder während der Erstellung der neuen Tabelle angelegt werden.

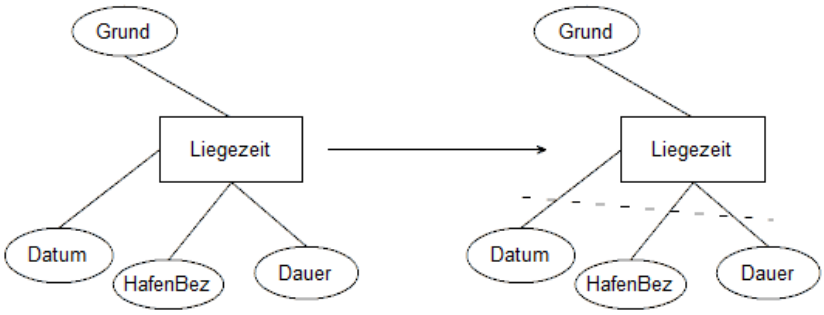

Abbildung 5.15: ALTER TABLE Liegezeit ADD PRIMARY KEY pkDHD(Datum, HafenBez, Dauer);

Löschen eines Primärschlüssels Diese Art der Schema-Änderung tritt in den Übergängen der untersuchten Schemata einmal auf. So wird der Primärschlüssel des Entity-Typen `Fahrt` mit Hinzukommen des neuen Attributes `FahrtNr` durch eben dieses ersetzt (vgl. Abbildung 5.16). Dies kommt einem Hinzufügen eines neuen Primärschlüssels mit anschließendem Löschen des alten Primärschlüssels gleich. Da die erste Art der Schema-Änderung bereits zuvor beschrieben wurde, beschränken wir uns in diesem Abschnitt auf das Löschen des alten Primärschlüssels `ArchivNr`. Dies kann einfach vollzogen werden durch den folgenden ICMO:

```
ALTER TABLE Fahrt DROP PRIMARY KEY ArchivNr;
```

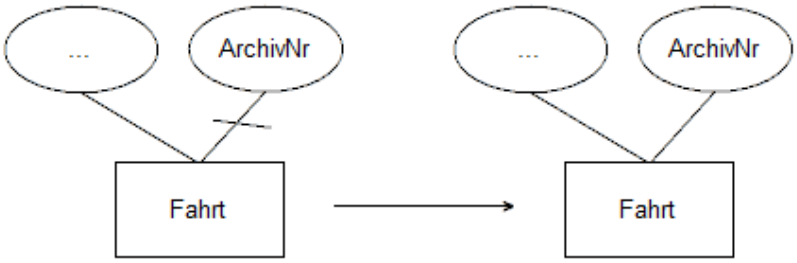

Abbildung 5.16: ALTER TABLE Fahrt DROP PRIMARY KEY ArchivNr

Hinzufügen eines Fremdschlüssels Bei der Erstellung der neuen Tabelle `Liegezeit` muss neben dem Hinzufügen eines Primärschlüssels auch ein Fremdschlüssel angegeben werden, da es sich hierbei um einen

von `Fahrt` abhängigen Entity-Typen handelt. Der Fremdschlüssel ergibt sich dabei aus dem Primärschlüssel des übergeordneten Entity-Typen und besteht dementsprechend aus dem Attribut `FahrtNr`. Der zugehörige ICMO lautet daher

```
ALTER TABLE Liegezeit
ADD FOREIGN KEY fkFahrtNr(FahrtNr)
REFERENCES Fahrt(FahrtNr) CHECK;
```

Umbenennung eines Attributes Beim Mapping des Schemas von 1997 auf 2019 kommt es einmal zur Umbenennung eines Attributes. Dabei wird `WS` in `WS-ID` umbenannt, veranschaulicht in Abbildung 5.17. Diese Schema-Änderung kann durch den folgenden SMO beschrieben werden:

```
RENAME COLUMN WS IN Serie TO WS-ID;
```

Dieser Schema Modification Operator sieht auf den ersten Blick eher harmlos aus, da ja „nur“ eine Umbenennung stattfindet. Allerdings sollte man dabei in jedem Fall beachten, dass alte Auswertungen, welche das Attribut `WS` erwarten, nicht mehr ohne Anpassungen funktionieren werden.

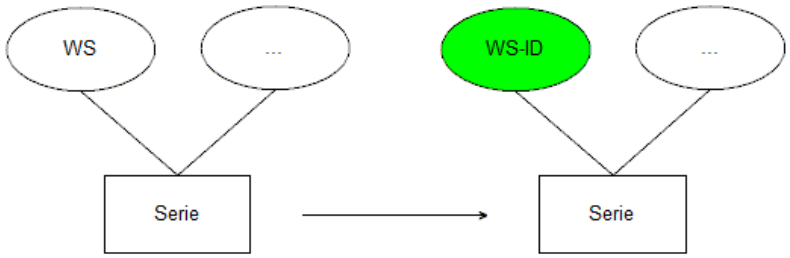

Abbildung 5.17: Das Attribut WS wird in WS-ID umbenannt

Zusammenfassen mehrerer Attribute zu einem Diese Form der Schema-Änderung tritt vor allem beim Mapping des Entity-Typen Serie auf. Im Laufe der Jahre beziehungsweise Jahrzehnte werden hier mehrere Attribute zusammengefasst wie beispielsweise `StartYear`, `StartMonth` und `StartDay` von 1977 zu `BegDatum` in 2019 (vgl. Abbildung 5.18). Für diese Änderungsart existiert zum Zeitpunkt der Erstellung dieser Bachelorarbeit kein eigenständiger Schema Modification Operator. Durch die Hintereinanderausführung anderer SMOs inklusive einer Funktion (beispielsweise in SQL geschrieben) ist es aber durchaus möglich, diese zu beschreiben. Dazu könnte man wie folgt vorgehen. Zunächst erstellt man eine neue Spalte BegDatum über den folgenden Operator:

```
ADD COLUMN BegDatum AS func(StartYear, StartMonth, StartDay) INTO Serie
```

Dabei muss die Funktion **func** noch weiter definiert werden, sodass das neue Attribut den gewünschten konkatenierten Wert enthält. Das könnte, je nach hinterlegtem Typ oder vorgegebenen Format, sowohl beispielsweise eine einfache Zeichenkette (String), eine Ganzzahl (Integer) oder aber ein Datumstyp (Date) sein.

Nach der Ausführung ist nun die neue gewünschte Spalte entstanden und anschließend werden die drei nun nicht mehr benötigten Attribute `StartYear`, `StartMonth` und `StartDay` aus dem Schema wie folgt entfernt:

```
DROP COLUMN StartYear FROM Serie;
DROP COLUMN StartMonth FROM Serie;
DROP COLUMN StartDay FROM Serie;
```

Die Gesamtausführung greift jedoch nicht einzig und allein auf das Spektrum der Schema Modification Operators zurück, sondern muss sich zwangsläufig einer externen Funktion **func** bedienen, welche beispielsweise in SQL geschrieben wurde. Somit kann durch dieses Beispiel die erste Grenze der SMOs aufgezeigt werden, da eine solche Änderung von Schemata in der Praxis durchaus häufig vorkommen kann. Die gesamte Hintereinanderausführung der Befehle sowie ihre Auswirkung sind im Folgenden zu betrachten:

```
ADD COLUMN BegDatum AS func(StartYear, StartMonth, StartDay) INTO Serie;
DROP COLUMN StartYear FROM Serie;
DROP COLUMN StartMonth FROM Serie;
DROP COLUMN StartDay FROM Serie;
```

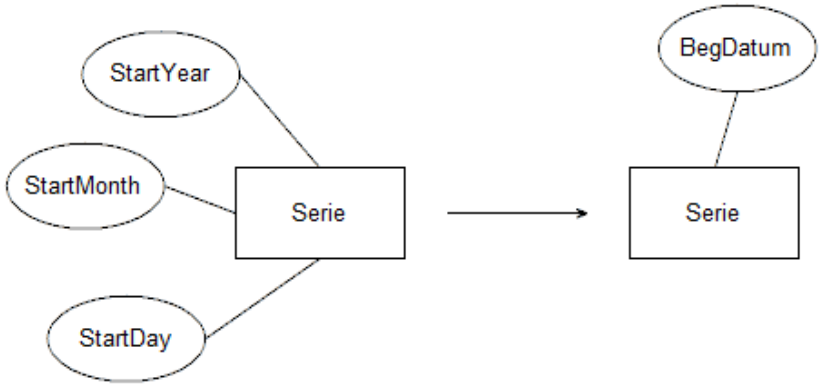

Abbildung 5.18: Die drei Attribute auf der linken Seite werden zu BegDatum zusammengefasst

Gehen wir davon aus, dass der resultierende Typ des Attributs `BegDatum` eine Zeichenkette (VARCHAR) im Format „TT.MM.YYYY“ ist, könnte ein möglicher Ansatz für die Funktion **func** wie folgt aussehen:

```
func := CONCAT(StartDay, '.', StartMonth, '.', StartYear)
```

Damit kann der Ausdruck im SMO `ADD COLUMN` ersetzt werden, womit man die folgende Gesamtbefehlsfolge erhält.

```
ADD COLUMN BegDatum AS
        (CONCAT(StartDay, '.', StartMonth, '.', StartYear)) INTO Serie;
DROP COLUMN StartYear FROM Serie;
DROP COLUMN StartMonth FROM Serie;
DROP COLUMN StartDay FROM Serie;
```

Dabei muss jedoch in jedem Fall beachtet werden, dass diese Implementierung von **func** eben nur ein Beispiel ist, welches viele Bedingungen voraussetzt. So funktioniert diese Funktion nur, wenn vor Anwendung der Schema Modification Operators die Attribute `StartDay`, `StartMonth` und `StartYear` alle vom Typ `VARCHAR` sind und in einem bestimmten Format vorliegen. In diesem Falle müssen `StartDay` und `StartMonth` jeweils zwei Zeichen lang sein, die führende glqq 0“ muss dementsprechend vorhanden sein. Zusätzlich muss `StartYear` alle vier Ziffern des jeweiligen Jahres enthalten. Außerdem muss auch das resultierende `BegDatum` dementsprechend in dem Format *dd.mm.yyyy* vorliegen und ebenfalls vom Typ String (beziehungsweise VARCHAR2) sein.

Nachdem ich das Mergen von Attributen nun exemplarisch erläutert und auf eventuelle Probleme aufmerksam gemacht habe, folgt nun ein Ansatz, um allgemein Mergen von Attributen als eine Art der Schema Modification Operators formaler zu beschreiben. Dabei orientiere ich mich am Ansatz der Funktion `func`.

MERGE COLUMNS a, b, c IN R TO d AS func(a,b,c)
ADD COLUMN d AS func(a,b,c) INTO R DROP COLUMN a FROM R DROP COLUMN b FROM R DROP COLUMN c FROM R

Tabelle 5.1: Ansatz für `MERGE ATTRIBUTES`; Spalten a,b,c werden zu d zusammengefasst

Der in Tabelle 5.1 dargestellte Ansatz sieht vor, die drei Attribute `a`, `b` und `c` zu einem Attribut `d` zusammenzufassen. Die Anzahl ist dabei variabel. Der `MERGE`-Befehl setzt sich dabei aus den einzelnen Befehlen `ADD COLUMN` und `DROP COLUMN` zusammen.

Aufteilen eines Attributes in mehrere Attribute Ähnlich wie beim vorherigen Beispiel verhält es sich mit dem Aufsplitten eines Attributes in mehrere Attribute. Zu beobachten ist dies unter anderem im Entity-Typen `Fahrt`. Sowohl 1977 als auch 1997 existierte das Attribut `Datum`, welches Start- und Enddatum der jeweiligen Fahrt enthielt. 2019 wurde dies aufgeteilt in eben diese beiden Bestandteile: `StartDatum` und `EndDatum` (vgl. Abbildung 5.19). Wie auch beim Zusammenfassen sind hierfür bis dato noch keine eigenen Schema Modification Operators bekannt, die dieses bewerkstelligen könnten. Allerdings kann auch hier unter Berücksichtigung mehrerer externer Funktionen diese Art der Änderung durch die Hintereinanderausführung anderer SMOs beschrieben werden. Dabei kann man das Zusammenfassen „invertiert“ ausführen. Das bedeutet, man erstellt zunächst die beiden neuen Spalten/Attribute `StartDatum` und `EndDatum` über die folgenden Operatoren.

```
ADD COLUMN StartDatum AS func1(Datum) INTO Fahrt;
ADD COLUMN EndDatum AS func2(Datum) INTO Fahrt;
```

Auch hier müssen die Funktionen func1 und func2 durch beispielsweise SQL beschrieben werden. Zu beachten ist, dass diese nicht unbedingt gleich sein müssen beziehungsweise können und dass sowohl Format und Typ der vorherigen Spalte als auch die der aufgeteilten Spalten zwingend berücksichtigt werden müssen. Nach der Ausführung existieren nun zwei neue Spalten mit aufgesplitteten Werten und die „alte“ Spalte `Datum` kann gelöscht werden. Hierfür nutzen wir den folgenden Befehl.

```
DROP COLUMN Datum FROM Fahrt;
```

Da auch hier durch die zwangsläufige Verwendung von beispielsweise SQL-Funktionen eine alleinige Anwendung der Schema Modification Operators nicht möglich erscheint, liegt hier eine zweite Grenze derer Möglichkeiten. Die Gesamtausführung sowie ihr Ergebnis ist im Folgenden zu sehen.

```
ADD COLUMN StartDatum AS func1(Datum) INTO Fahrt;
ADD COLUMN EndDatum AS func2(Datum) INTO Fahrt;
DROP COLUMN Datum FROM Fahrt;
```

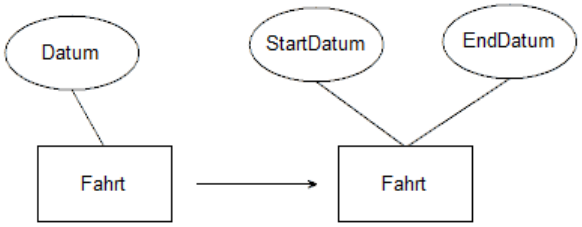

Abbildung 5.19: Die Spalte Datum wird in seine Bestandteile Start- und EndDatum zerlegt

Wir betrachten nun eine mögliche Implementierung der Funktionen `func1` und `func2`. Auch hierbei wird von bestimmten Formaten und Typen bei den Spalten `Datum`, `StartDatum` sowie `EndDatum` ausgegangen. Bei anders gearteten Typen und Formaten müssen die Funktionen dementsprechend angepasst werden. Um einen ersten Ansatz zu präsentieren, seien folgende Annahmen als gegeben zu betrachten:

- Spalte `Datum`:
  - Typ: Zeichenkette (VARCHAR)
  - Format: StartDatum-EndDatum
- Spalten `StartDatum` & `EndDatum`
  - Typ: Zeichenkette (VARCHAR)
  - Format: TT.MM.YYYY

Unter diesen Annahmen ist eine Arbeit mit der *SUBSTRING*-Funktion möglich. So würden die Funktionen `func1` und `func2` wie folgt aussehen.

```
func1 := SUBSTRING(Datum, 1, 10)
func2 := SUBSTRING(Datum, 12, 10)
```

Dieser statische Ansatz ist nur als vereinfachtes Beispiel anzusehen, da er relativ fehleranfällig ist. So könnten Datumswerte falsch (z.B. Tippfehler) oder andersartig (z.B. amerikanische Datumsformat) gespeichert worden sein. Komplexere Funktionen, welche diese Umstände betrachten, würden beispielsweise bestimmte Muster aus den Zeichenketten herausfiltern, um das entsprechende Format zu bestimmen. Deshalb ist mein Ansatz nur als einfachste Möglichkeit zu betrachten.

Damit sieht die Gesamtimplementierung des Splits der Spalte `Datum` inklusive der Funktionen wie folgt aus.

```
ADD COLUMN StartDatum AS SUBSTRING(Datum, 1, 10) INTO Fahrt;
ADD COLUMN EndDatum AS SUBSTRING(Datum, 12, 10) INTO Fahrt;
DROP COLUMN Datum FROM Fahrt;
```

Auch für das Aufsplitten einer Spalte soll hier nach der exemplarischen Erläuterung ein allgemeiner Ansatz vorgestellt werden, welcher dem Ansatz für `MERGE COLUMNS` folgt.

SPLIT COLUMN a IN R TO b USING func1(a), c USING func2(a)
ADD COLUMN b AS func1(a) INTO R ADD COLUMN c AS func2(a) INTO R DROP COLUMN a FROM R

Tabelle 5.2: Ansatz für `SPLIT COLUMN`; Spalte a wird in die Spalten b und c aufgeteilt

Der Ansatz, welcher in Tabelle 5.2 vorgestellt wurde, sieht vor, die Spalte `a` in die beiden Spalten `b` und `c` aufzuteilen. Die hinter `USING` stehenden Funktionen bestimmen dabei, wie der Wert der originalen Spalte aufgeteilt und der neuen Spalte zugeteilt wird.

Statistischer Vergleich – Schema-Evolution in Wikipedia [CMTZ08] Die Schema Modification Operators wurden auf Grundlage der Untersuchung von Schema-Evolutionen bei der Wikipedia-Datenbank [CMTZ08] getroffen. Dabei wurden die häufigsten Änderungsarten ermittelt und ihnen ein solcher Schema Modification Operator zugeordnet. Nach meiner Betrachtung der Schema-Änderungen in den Schemata des IOW von 1977 und 1997 auf 2019 stellte sich mir nun die Frage, ob die gegebenen statistischen Informationen auch annähernd auf meinen Untersuchungsgegenstand zutreffen. Dabei betrachte ich nur die Schema Modification Operators und nicht die Integrity Constraints Modification Operators, da diese in der zugrundeliegenden Betrachtung [CMTZ08] ebenfalls nicht betrachtet beziehungsweise noch nicht definiert wurden. In nachfolgender Tabelle habe ich zeilenweise die in [CMTZ08] betrachteten SMOs notiert. Daneben deren tatsächliche Anzahl und prozentualen Anteil an den gesamten Schema-Änderungen.

SMO	**Anzahl**		**proz. Anteil**	
	Wikipedia	*IOW*	*Wikipedia*	*IOW*
CREATE TABLE	24	1	8.9%	1.5%
DROP TABLE	9	0	3.3%	0%
RENAME TABLE	3	0	1.1%	0%
DISTRIBUTE TABLE	0	0	0%	0%
MERGE TABLE	4	0	1.5%	0%
COPY TABLE	6	0	2.2%	0%
ADD COLUMN	104	44	38.7%	75%
DROP COLUMN	71	13	26.4%	22%
RENAME COLUMN	43	1	16%	1.5%
MOVE COLUMN	1	0	0.4%	0%
COPY COLUMN	4	0	1.5%	0%

Tabelle 5.3: Vergleich der Anzahl von SMOs bei Schema-Änderungen in der Wikipedia-Datenbank mit denen am IOW

Ganz abgesehen von den Unterschieden in der Häufigkeit des Vorkommens der einzelnen Schema Modification Operators fällt auf, dass zum Zeitpunkt der Erstellung des Papers [CMTZ08] noch andere SMOs in der Planung waren. So wurden beispielsweise `DISTRIBUTE TABLE` oder auch `MOVE COLUMN` nicht in den späteren Katalog aufgenommen. Das könnte an ihrem geringen Auftreten im untersuchten Beispiel liegen.

Die Verteilung meiner Beobachtungen weist im Vergleich zu den Ergebnissen von Zaniolo et al. [CMTZ08] deutlichere Extrema auf. Am auffälligsten ist dieser Unterschied bei dem SMO `ADD COLUMN` zu erkennen. Das erkläre ich mir damit, dass ich lediglich die Schema-Änderungen von zwei Jahren auf ein anderes betrachtet habe und nicht, wie im Beispiel der Wikipedia-Datenbank, 171 verschiedene Schemata.

Durch die Untersuchung einer größeren Zahl von Schemata werden die Extrema offensichtlich herabgesetzt. Was ich hingegen bestätigen kann, ist, dass die Schema Modification Operators, welche nicht in den späteren Katalog übernommen wurden, auch bei mir keine Anwendung fanden. Weiterhin habe ich am Ende ein annähernd gleiches Auftreten des SMOs `DROP COLUMN`, jedoch nur, weil ich das Zusammenfassen (`MERGE COLUMNS`) sowie das Aufteilen (`SPLIT COLUMN`), beziehungsweise deren Einzelschritte hinzuzähle.

Nachdem ich nun die Schema-Änderungen dargelegt und mit Hilfe der Schema-Evolutions-Operatoren beschrieben habe, folgt nun der nächste Teil meiner Arbeit. Dieser befasst sich mit der Kombination von Schema-Evolutions-Operatoren und Provenance. Anschließend folgt ein Kapitel zur Implementierung.

5.4 Kombination von Schema-Evolutions-Operatoren und Provenance

Im vorhergehenden Kapitel wurden die Schema-Änderungen, welche bei den Mappings der Schemata von 1977 und 1997 auf das von 2017 auftraten, durch Schema Modification Operators sowie Integrity Constraints Modification Operators beschrieben, sofern dies möglich war. Nun soll es um die Frage gehen, ob und wie Anfragen, die für andere Schema-Versionen entworfen wurden, auch auf anderen Versionen ausführbar gemacht werden können. Dabei geht es im Einzelnen darum, welche Informationen verfügbar sein müssen, um diese Anfragen umschreiben zu können. Dabei spielt die Herkunft der Daten eine wichtige Rolle.

Diese Herkunft (engl. *Provenance*) kann durch die sogenannte `Data Provenance` verschieden beschrieben werden (siehe Abschnitt 3.3.2). Dabei werde ich mich auf die `Why`-Provenance beschränken. Dies resultiert daraus, dass zum einen die Informationen der `Where`-Provenance in dieser enthalten beziehungsweise daraus ableitbar sind. Zum Anderen müssen keine Berechnungsformeln im Sinne der `How`-Provenance angegeben werden, da die einzigen notwendigen Berechnungen bei den Schema-Änderungen *MERGE COLUMNS* und *SPLIT COLUMN* vorkommen und diese durch die entsprechenden Funktionen beschrieben werden.

Im Folgenden werde ich die einzelnen Schema Modification Operators, welche in Kapitel 5.3 verwendet wurden, auf die Verträglichkeit mit Provenance-Techniken überprüfen. Es geht also darum, ob und mit welchen zusätzlichen Informationen nachvollzogen werden kann, woher bestimmte Daten stammen. Gelingt dies, können beispielsweise Anfragen an andere Schema-Versionen entsprechend angepasst werden, um andere Versionen zu bedienen.

ADD COLUMN Dieser SMO trat im Vergleich zu den anderen am häufigsten auf. Betrachten wir das vorherige Beispiel `ADD COLUMN Kommentar INTO Serie`. Bei der Betrachtung von SQL-Auswertungen müssen die Schema-Versionen beachtet werden. So besitzen einerseits frühere Versionen (vor Anwendung des SMOs) diese neue Spalte noch nicht. Andererseits können auch SQL-Abfragen, welche für frühere Schema-Versionen entworfen wurden, zu Problemen führen, wenn eine neue Spalte eingefügt wird. Beispielsweise könnten dann einige Vereinigungen von Tabellen nicht mehr funktionieren, wenn sie eine bestimmte Anzahl von Spalten erwarten.

Dementsprechend ist es wichtig zu wissen, was die Schema-Änderung konkret beinhaltete und wie man sie rückgängig machen kann, sofern dies möglich ist. Da bei `ADD COLUMN` keine Informationen verloren gegangen sind, sondern lediglich eine zusätzliche Spalte hinzugefügt wurde, ist eine Invertierung nicht allzu kompliziert.

Die folgenden Informationen müssen gespeichert werden, um den Originalzustand vor der Schema-Änderung wiederherzustellen:

- Name des SMOs → ADD COLUMN
- Name der Spalte → Kommentar
- Name der Tabelle → Serie

Zusätzlich sollte der Zeitpunkt der Schema-Änderung bekannt sein. So kann bestimmt werden, welche Schema-Auswertungen von der Änderung betroffen sind. Die eindeutige Inverse lautet also wie folgt.

```
DROP COLUMN Kommentar FROM Serie;
```

Provenance-Anfragen bringen in diesem Fall nicht das gewünschte Ergebnis, da bei neueren Daten nicht festgestellt werden kann, wie sie in früheren Schema-Versionen (vor der Schema-Änderung) aussahen. So kann die Herkunft der Daten nur bis zu dem Punkt zurückverfolgt werden, an dem die neue Spalte hinzugefügt wurde.

Serie(1977)

SerieNr	...	
1	...	s_1, 1977
2	...	s_2, 1977
3	...	s_3, 1977

→ **Serie(2017)**

SerieNr	...	`Kommentar`	
1	...	A	s_1, 2017
2	...	B	s_2, 2017
3	...	C	s_3, 2017

Tabelle 5.4: Spalte `Kommentar` wird hinzugefügt

In Tabelle 5.4 ist die Schema-Änderung durch das Hinzufügen von `Kommentar` veranschaulicht. Nun stellen wir Provenance-Anfragen an die Tupel des Schemas von 2017 im Sinne der **Where**-, **Why**- und **How**-Provenance. Das Ergebnis ist in nachfolgender Tabelle (Tabelle 5.5) dargestellt.

`Tupel-ID`	**Where**	**Why**	**How**
(s_1, 2017)	Serie	{{(s_1, 1977)}}	A ⊙ (s_1, 1977)
(s_2, 2017)	Serie	{{(s_2, 1977)}}	B ⊙ (s_2, 1977)
(s_3, 2017)	Serie	{{(s_3, 1977)}}	C ⊙ (s_3, 1977)

Tabelle 5.5: Provenance-Informationen von Tupeln der Tabelle **Serie (2017)**

Aus der **Why**-Provenance erkennen wir, dass die originale Herkunft der Tupel die Tupel mit der gleichen ID haben. Durch die **How**-Provenance wissen wir, dass die über `ADD COLUMN` hinzugefügte Spalte mit den Werten A, B und C gefüllt wurden.

CREATE TABLE Dieser SMO trat in den untersuchten Schema-Mappings genau einmal auf und zwar bei der Neuerstellung der Tabelle `Liegezeit`.
Der exakte SMO dafür lautet `CREATE TABLE Liegezeit(Datum, HafenBez, Dauer, Grund, FahrtNr)`. Da hier keine vorhandenen Daten verändert werden, lässt sich diese Schema-Änderung gut invertieren.

Laut Abbildung 4.1 ist die zugehörige Inverse das Löschen der Tabelle über den SMO `DROP TABLE`. Dazu muss lediglich der Name der Tabelle bekannt sein. Da keine andere Tabelle von `Liegezeit` abhängig ist, müssen Integritätsbedingungen nicht überprüft werden. Anders sähe dies beispielsweise bei der Tabelle `Fahrt` aus. Würde sie gelöscht werden, müssten die Integritätsbedingungen der Tabellen `Liegezeit` und `Serie` geändert werden, was mitunter einen Informationsverlust aufgrund unvollständiger Schlüssel bedingen würde.

Um nun den Datenbankzustand vor der Schema-Änderung zu erhalten, wird der folgende inverse SMO angewendet, welcher restlos alle Daten der Tabelle löscht.

```
DROP TABLE Liegezeit;
```

Wie schon bei `ADD COLUMN` können auch hier Provenance-Techniken nur bedingt angewendet werden. So kann die Herkunft von Daten der Tabelle nur so weit zurückverfolgt werden, wie die Tabelle tatsächlich existiert. Für den Zeitraum vor der Erstellung der Tabelle können keine Provenance-Informationen angegeben werden.

RENAME COLUMN Dieser SMO trat in meinen Untersuchungen genau einmal auf. Dabei wurde die Spalte `WS` der Tabelle `Serie` umbenannt in `WS-ID`. Diese Änderung mag auf den ersten Blick nicht sonderlich bedeutsam zu sein, da „wir“ ja erkennen können, dass es sich um die gleiche Spalte handelt und es nur einen anderen Namen bekommen hat. Für SQL-Auswertungen ist dies jedoch eine vollkommen andere Spalte. So versagen alle Auswertungen, die zuvor den Spaltennamen `WS` referenziert haben. Beispielsweise würde die folgende SQL-Anfrage auf neueren Schemata in einen Fehler laufen, da nach der Umbenennung der alte Spaltenname nicht mehr vorhanden ist.

```
SELECT SerieNr, WS FROM Serie;
```

Dementsprechend ist es wichtig, Informationen zu speichern, um diese Änderung rückgängig machen zu können oder die SQL-Abfragen umzuschreiben. Dafür ist zunächst zu klären, ob durch die Anwendung des SMOs `RENAME COLUMN WS IN Serie TO WS-ID` Informationen verloren gehen, da dies nicht rückgängig gemacht werden könnte. Da die Änderung eines Spaltennamens jedoch nicht die enthaltenen Daten in irgendeiner Weise verändert, kann die Wirkung des SMOs einfach rückgängig gemacht werden.

Die Informationen, die für eine Inverse verfügbar sein müssen, sind:

- Name des SMOs → RENAME COLUMN
- Name der Spalte → WS
- neuer Name der Spalte → WS-ID
- Name der Tabelle → Serie

Auch hier ist es sinnvoll, den Zeitpunkt der Schema-Änderung zu speichern, um unterscheiden zu können, welche Auswertungen umgeschrieben werden müssen. Laut Abbildung 4.1 in Kapitel 4 existiert auch für den SMO `RENAME COLUMN` eine eindeutige Inverse, die ebenfalls `RENAME COLUMN` ist. Die Inverse, um die oben genannte Schema-Änderung rückgängig zu machen beziehungsweise Auswertungen umzuschreiben, lautet demnach wie folgt.

```
RENAME COLUMN WS–ID IN Serie TO WS;
```

Geht man davon aus, dass die Umbenennung der Spalte `WS` keine enthaltenen Daten verändert hat, kann auch über Provenance-Anfragen herausgefunden werden, woher die Daten in der „neuen“ Spalte `WS-ID` stammen.

In Tabelle 5.6 ist die Schema-Änderung durch das Umbenennen von `WS` in `WS-ID` beim Mapping des Schemas von 1997 auf das Schema von 2017 veranschaulicht. Nun stellen wir wiederum Provenance-Anfragen an die Tupel des Schemas von 2017 im Sinne der **where**-, **why**- und **how**-Provenance. Das Ergebnis ist in nachfolgender Tabelle (Tabelle 5.7) dargestellt.

Serie (1997)

SerieNr	...	WS	
1	...	10	s_1, 1997
2	...	20	s_2, 1997
3	...	23	s_3, 1997

$\rightarrow$ **Serie (2017)**

SerieNr	...	WS-ID	
1	...	10	s_1, 2017
2	...	20	s_2, 2017
3	...	23	s_3, 2017

Tabelle 5.6: Spalte `WS` wird umbenannt in `WS-ID`

Tupel-ID	Where	Why	How
(s_1, 2017)	Serie	{{(s_1, 1977)}}	(s_1, 1977)
(s_2, 2017)	Serie	{{(s_2, 1977)}}	(s_2, 1977)
(s_3, 2017)	Serie	{{(s_3, 1977)}}	(s_3, 1977)

Tabelle 5.7: Provenance-Informationen von Tupeln der Tabelle **Serie (2017)**

Aus diesen drei Provenance-Informationen geht hervor, dass die Tupel des Schemas von 2017 auf die Tupel von 1977 zurückgeführt werden können. Dabei verändert sich einzig die Jahreszahl (Schema-Version) der Tupel, die ID ist ansonsten identisch. Da hier keine Informationen über die Schema-Änderung durch das Umbenennen von `WS` in `WS-ID` gegeben werden, muss diese Information gesondert gespeichert werden.

MERGE COLUMNS Während der Untersuchung der Schema-Änderungen fiel mir an einigen Stellen auf, dass nicht alle Änderungen durch einen einzelnen Schema Modification Operator beschrieben werden konnten. Dazu zählte unter anderem das Zusammenfassen von Spalten zu einer einzelnen Spalte. Am Ende dieser Betrachtung habe ich einen Ansatz vorgestellt, welcher `MERGE COLUMNS` beschreibt. Dieser Ansatz sieht vor, dass zunächst die neue Spalte mit Hilfe einer Funktion hinzugefügt wird und danach die „alten“ Spalten gelöscht werden. So wurden die drei Spalten `StartYear`, `StartMonth` und `StartDay` zu einer Spalte `BegDatum` zusammengefasst. Hierbei gingen zwar keine Informationen verloren, eine Invertierung gestaltet sich trotzdem umfangreicher als bei den zuvor beschriebenen Schema-Änderungen.

So müssen für eine Invertierung mehr Informationen gespeichert werden, um die entstandene Spalte wieder so aufzuteilen, dass der alte Datenbankzustand wieder erreicht wird. Diese Informationen sind:

- Name des SMOs $\rightarrow$ MERGE COLUMNS
- Namen der zusammengefassten Spalten $\rightarrow$ StartYear, StartMonth, StartDay
- Name der neuen Spalte $\rightarrow$ BegDatum
- MERGE-Funktion(en)
- Zeitpunkt der Schema-Änderung

Sind diese Informationen vorhanden, müssen zunächst aus der MERGE-Funktion inverse Funktionen gebildet werden, um die Werte der drei alten Spalten wiederherstellen zu können, sofern dies möglich ist. Beinhaltet die MERGE-Funktion beispielsweise nur eine Konkatenation, wie sie im Folgenden dargestellt ist, kann durch die Arbeit mit Substrings, also Teiltexten des gesamten `BegDatum`s, eine Inverse erzeugt werden.

```
MERGE COLUMNS StartYear, StartMonth, StartDay IN Serie
TO BegDatum AS func(StartYear, StartMonth, StartDay);

func := CONCAT(StartDay, '.', StartMonth, '.', StartYear);
```

Die Inverse zu `MERGE COLUMNS` ist gerade mein vorgestellter Ansatz für `SPLIT COLUMN`. Dementsprechend lautet der inverse SMO wie folgt.

```
SPLIT COLUMN BegDatum IN Serie
TO StartDay   USING func1(BegDatum),
   StartMonth USING func2(BegDatum),
   StartDay   USING func3(BegDatum);

func1 := SUBSTRING(BegDatum, 1, 2);
func2 := SUBSTRING(BegDatum, 4, 2);
func3 := SUBSTRING(BegDatum, 7, 4);
```

Um nicht für jede Anfrage eine Schema-Änderung rückgängig machen zu müssen, können die Originalspalten und damit auch die ursprünglichen Werte ermittelt werden. Dazu müssen Provenance-Informationen versioniert gespeichert werden. Das bedeutet, dass jedes Mal, wenn die entsprechende Tabelle eine Schema-Änderung erfährt, neue Provenance-Informationen gespeichert werden. So kann zurückverfolgt werden, woher die Daten stammen.

Serie (1977)

SerieNr	`StartDay`	`StartMonth`	`StartYear`	...	
1	23	02	1977	...	s_1, 1977
2	24	02	1977	...	s_2, 1977
3	25	02	1977	...	s_3, 1977

Tabelle 5.8: Tabelle **Serie** 1977, vor dem Zusammenfassen von `StartDay`, `StartMonth` und `StartYear` zu `BegDatum`

→ **Serie (2017)**

SerieNr	`BegDatum`	...	
1	23.02.1977	...	s_1, 2017
2	24.02.1977	...	s_2, 2017
3	25.02.1977	...	s_3, 2017

Tabelle 5.9: Tabelle **Serie** 2017, nach dem Zusammenfassen von `StartDay`, `StartMonth` und `StartYear` zu `BegDatum`

In den beiden oben stehenden Tabellen (Tabellen 5.8 und 5.9) ist die Schema-Änderung veranschaulicht, welche durch den `MERGE` von `StartDay`, `StartMonth` und `StartYear` zu `BegDatum` beschrieben wird. Nun stellen wir erneut Provenance-Anfragen an die Tupel des Schemas von 2017 im Sinne der **where**-, **why**- und **how**-Provenance. Das Ergebnis ist in nachfolgender Tabelle (Tabelle 5.10) dargestellt.

`Tupel-ID`	`Where`	`Why`	`How`
(s_1, 2017)	Serie	{{(s_1, 1977)}}	(23, 02, 1977) $\odot$ (s_1, 1977)
(s_2, 2017)	Serie	{{(s_2, 1977)}}	(24, 02, 1977) $\odot$ (s_2, 1977)
(s_3, 2017)	Serie	{{(s_3, 1977)}}	(25, 02, 1977) $\odot$ (s_3, 1977)

Tabelle 5.10: Provenance-Informationen von Tupeln der Tabelle **Serie (2017)** (Tabelle 5.9)

Anhand dieser Provenance-Informationen sehen wir, dass zur Berechnung der aktuellen Tupel (Schema 2017) die gleichen Tupel der vorherigen Version (Schema vor der Schema-Änderung, beispielsweise Schema 1977) genutzt werden. Dabei ändert sich nur die Jahreszahl (Schema-Version). Die Informationen der **How**-Provenance gibt an, wie die Berechnung konkreter funktioniert. Dabei werden die Tupel von 1977 (z.B. (s_1, 1977)) mit einem Tupel bestehend aus drei Werten über $\odot$ verknüpft. Diese Angabe bezieht sich auf die `MERGE`-Funktion.

Diese erwartet als Parameter die drei Werte `StartDay`, `StartMonth` und `StartYear`, welche durch $\odot$ übergeben werden. Dementsprechend muss diese Funktion zur Berechnung gespeichert werden.

SPLIT COLUMN Eine weitere Schema-Änderung, für welche noch kein eigenständiger Schema Modification Operator in [CMZ08] definiert wurde, ist das Aufteilen (`SPLIT`) einer Spalte in mehrere Spalten. Am Ende der Betrachtung eines konkreten Beispiels aus der Schema-Evolution am IOW habe ich einen allgemeinen Ansatz vorgestellt, welcher `SPLIT COLUMN` beschreibt. Dieser Ansatz sieht zunächst vor, dass die neuen Spalten mit Hilfe von verschiedenen Funktionen erzeugt werden und danach die „alte“ Spalte gelöscht wird.

Am konkreten Beispiel bedeutet das, dass ich zunächst die Spalten `StartDatum` und `EndDatum` der Tabelle `Fahrt` über die beiden Funktionen `func1` und `func2` erzeugt werden. Danach kann ich die Spalte `Datum` löschen. Durch die restlose Übernahme aller Informationen der Spalte `Datum` in die beiden neuen Spalten, gehen keine Informationen verloren, was eine Invertierung ermöglicht.

Um den vorherigen Datenbankzustand wiederherstellen zu können, müssen einige Informationen zur Schema-Änderung gespeichert werden:

- Name des SMOs →`SPLIT COLUMN`
- Name der aufgeteilten Spalte → `Datum`
- Namen der neuen Spalten → `StartDatum`, `EndDatum`
- `SPLIT`-Funktionen
- Zeitpunkt der Schema-Änderung

Sind diese Informationen vorhanden, muss zunächst aus den `SPLIT`-Funktionen eine inverse Funktion gebildet werden, welche den alten Zustand wiederherstellt. Dies ist nicht immer möglich. Enthält eine Funktion beispielsweise eine Aggregatfunktion, wie die Berechnung einer Summe oder eines Durchschnitts, können nicht in jedem Fall alle ursprünglichen Werte wiederhergestellt werden. Im konkreten Beispiel, welches im Folgenden beschrieben ist, kann jedoch eine inverse Funktion gefunden werden.

```
SPLIT COLUMN Datum IN Fahrt
TO     StartDatum USING func1(Datum),
       EndDatum   USING func2(Datum);

func1 := SUBSTRING(Datum, 1, 10);
func2 := SUBSTRING(Datum, 12, 10);
```

Die Inverse zu `SPLIT COLUMN` ist gerade mein vorgestellter Ansatz für `MERGE COLUMNS`. Dementsprechend lautet der inverse SMO wie folgt.

```
MERGE COLUMNS StartDatum, EndDatum IN Fahrt
TO Datum AS func(StartDatum, EndDatum);

func := CONCAT(StartDatum, '-', EndDatum);
```

Die Anwendung von Provenance-Techniken verhält sich bei diesem `SPLIT` ähnlich wie beim `MERGE`. Wenn die oben genannten Informationen zur Schema-Änderung gespeichert werden, lassen sich ehemalige Tupel gut rekonstruieren bzw. die Herkunft von Datensätzen gut nachvollziehen. Um dies zu veranschaulichen, folgen nun zwei Tabellen, welche die Tabelle `Fahrt` einmal im Jahre 1977 (Tabelle 5.11) und einmal

im Jahre 2017 (Tabelle 5.12) zeigen. Die Werte in der ersten Tabelle **Fahrt** (Tabelle 5.11) sind frei erdacht.

Fahrt (1977)

ArchNr	Datum	...	
1	01.01.1977-14.01.1977	...	f_1, 1977
2	01.03.1977-14.03.1977	...	f_2, 1977
3	01.05.1977-14.05.1977	...	f_3, 1977

Tabelle 5.11: Tabelle **Fahrt** 1977, vor dem Aufsplitten von `Datum` in `StartDatum` und `EndDatum`

→ **Fahrt** (2017)

FahrtNr	StartDatum	EndDatum	...	
1	01.01.1977	14.01.1977	...	f_1, 2017
1	01.03.1977	14.03.1977	...	f_2, 2017
1	01.05.1977	14.05.1977	...	f_3, 2017

Tabelle 5.12: Tabelle **Fahrt** 2017, nach dem Aufsplitten von `Datum` in `StartDatum` und `EndDatum`

In den beiden oben stehenden Tabellen (Tabellen 5.11 und 5.12) ist die Schema-Änderung veranschaulicht, welche durch den `SPLIT` von `Datum` zu `StartDatum` und `EndDatum` beschrieben wird. Nun stellen wir erneut Provenance-Anfragen an die Tupel des Schemas von 2017 im Sinne der **where**-, **why**- und **how**-Provenance. Das Ergebnis ist in nachfolgender Tabelle (Tabelle 5.13) dargestellt.

Tupel-ID	Where	Why	How
(f_1, 2017)	Fahrt	{{(f_1, 1977)}}	01.01.1977-14.01.1977 $\odot$ (f_1, 1977)
(f_2, 2017)	Fahrt	{{(f_2, 1977)}}	01.03.1977-14.03.1977 $\odot$ (f_2, 1977)
(f_3, 2017)	Fahrt	{{(f_3, 1977)}}	01.05.1977-14.05.1977 $\odot$ (f_3, 1977)

Tabelle 5.13: Provenance-Informationen von Tupeln der Tabelle **Fahrt** (2017) (Tabelle 5.12)

Diese Provenance-Informationen sind ähnlich wie beim vorherigen SMO `MERGE ATTRIBUTES` zu betrachten. Dies erscheint mir auch logisch, da `MERGE` und `SPLIT` ja zueinander invers sind. So werden auch hier zur Berechnung der aktuellen Tupel (Schema von 2017) die gleichen Tupel der vorherigen Version (Schema 1977) herangezogen. Die Informationen der **How**-Provenance erläutern die konkrete Berechnung der neuen Tupel noch etwas genauer. So werden die jeweiligen originalen Tupel mit einem Wert über den Operator $\odot$ verknüpft (z.B. 01.01.1977-14.01.1977). Dieser Wert bezieht sich wiederum auf die Definition der `SPLIT`-Funktion, welche diesen Wert als Parameter erwartet. Zur Berechnung der Werte wird dementsprechend diese Funktion ebenfalls benötigt.

5.5 What-Provenance

Die geläufige Definition von Provenance, im Speziellen die der Data Provenance, ist für viele Anwendungsbeispiele durchaus ausreichend, jedoch stieß ich vor allem bei den Implementierungsansätzen für das Splitten und Mergen von Spalten auf Schwierigkeiten. Bisher klärt die Definition nämlich nicht, wie die Werte von Spalten definiert sind. Konkreter fehlen Informationen über die erlaubten Datentypen und deren Formate. Dies fällt vor allem auf, wenn die Frage aufkommt, ob und wie Datentypen konvertiert werden müssen, um Fehler bei beispielsweise Rundungsverfahren von Gleitkommazahlen zu vermeiden. Weiterhin sind einige vordefinierte SQL-Funktionen nur mit bestimmten Spaltentypen kompatibel, so etwa die Konkatenation, wie sie mein Ansatz für das Mergen der Spalten `StartYear`, `StartMonth` und `StartDay` im vorhergehenden Abschnitt vorsieht. Diese Operation würde auf beispielsweise Typen wie Integer oder Date in dieser Form nicht funktionieren.

Um solche Fehlerquellen auszuschließen und die Entwicklung des genutzten Typsystems untersuchen zu können, sehe ich Provenance-Informationen über verwendete Typen als sehr sinnvoll an. Diese würde ich der Data Provenance zuordnen und sie, sprachlich angelehnt an die übrigen Fragewörter, **What**-Provenance nennen. Zur Veranschaulichung seien im Folgenden noch einmal die beiden Tabellen aus vorherigem Kapitel abgebildet, welche das eben erwähnte `MERGE` beschreiben.

Serie (1977)

SerieNr	`StartDay`	`StartMonth`	`StartYear`	...	
1	23	02	1977	...	s_1, 1977
2	24	02	1977	...	s_2, 1977
3	25	02	1977	...	s_3, 1977

Tabelle 5.14: Tabelle **Serie** 1977, vor dem Zusammenfassen von `StartDay`, `StartMonth` und `StartYear` zu `BegDatum`

→ **Serie (2017)**

SerieNr	`BegDatum`	...	
1	23.02.1977	...	s_1, 2017
2	24.02.1977	...	s_2, 2017
3	25.02.1977	...	s_3, 2017

Tabelle 5.15: Tabelle **Serie** 2017, nach dem Zusammenfassen von `StartDay`, `StartMonth` und `StartYear` zu `BegDatum`

Für das gewählte Beispiel (veranschaulicht in den Tabellen 5.14 und 5.15) gehe ich bei Tabelle 5.14 von den Voraussetzungen aus, dass die `SerieNr` vom Typ Integer und die Spalten `StartDay`, `StartMonth` und `StartYear` vom Typ VARCHAR sind. Stelle ich nun eine Anfrage im Sinne der **What**-Provenance, erhalte ich folgendes Ergebnis (siehe Tabelle 5.16).

`Tupel-ID`	`What`
(s_1, 2017)	{(SerieNr, Integer), (Datum, {VARCHAR(2), VARCHAR(2), VARCHAR(4)}), (..., ...)}
(s_2, 2017)	{(SerieNr, Integer), (Datum, {VARCHAR(2), VARCHAR(2), VARCHAR(4)}), (..., ...)}
(s_3, 2017)	{(SerieNr, Integer), (Datum, {VARCHAR(2), VARCHAR(2), VARCHAR(4)}), (..., ...)}

Tabelle 5.16: **What**-Provenance von Tupeln der Tabelle **Serie** (2017)

Dabei besteht die **What**-Provenance eines Tupels aus einer Menge von geordneten Paaren. Diese Paare bestehen aus dem Spaltennamen und dem dazugehörigen Spaltentyp. Ich betrachte dabei die Spaltentypen der ursprünglichen Spalten, aus welchen die aktuelle Spalte durch die Schema-Änderung entstanden ist.

6 Implementierung

Dieses Kapitel befasst sich mit der Praxistauglichkeit meiner erstellten ER-Modelle. Dazu werde ich zunächst das Schema von 1977 erstellen und mit entsprechenden Testdaten aus den mir vorliegenden originalen Datensätzen befüllen. Anschließend werde ich die im Kapitel 5.3 herausgearbeiteten Veränderungen der Schemata versuchen, auf diesem Schema nachzuvollziehen. Entgegen meines ursprünglichen Planes verzichte ich darauf, ein eigenes Schema für 1997 zu erstellen, da beim Mapping von diesem auf das Schema von 2017 keine signifikanten Änderungen auftauchen, die nicht schon beim Mapping von 1977 auf 2017 vorhanden sind, ausgenommen das Umbenennen von `WS` in `WS-ID`. Für die Implementierung dieser Änderung werde ich zunächst das Attribut `WS` zur Tabelle `Serie` hinzufügen und anschließend umbenennen.

6.1 Benutzte Software

Als Datenbanksystem nutze ich für die Implementierung **PostgreSQL** in der Version 12.1 auf einem Windows 10-Betriebssystem (64-bit). Zur Entwicklung des Schemas, Datenpflege und Anwendung von Schema Modification Operators und Integrity Constraints Modification Operators nutze ich das Tool **pgAdmin 4**. Zur Visualisierung (beispielsweise ER-Modell)des fertigen Schemas sowie der Auswirkungen von dessen Änderungen nutze ich das Open Source-Programm **DBeaver**.

6.2 Anwendung SMOs / ICMOs

Im Folgenden werde ich die in Kapitel 5.3 vorgestellten Schema-Änderungen auf das zuvor erstellte Schema von 1977 anwenden. Dabei trenne ich die Schema Modification Operators von den Integrity Constraints Modification Operators, um einen besseren Überblick zu schaffen. Nach dieser Betrachtung gehe ich gesondert auf die Spezialfälle `MERGE COLUMNS` und `SPLIT COLUMN` ein. Beginnen wir nun mit den Schema-Änderungen, welche ich auf die grundlegenden SMOs zurückführen konnte.

6.2.1 SMOs

ADD COLUMN Beim Mapping des Schemas von 1977 auf das Schema von 2017 wurde dem Entity `Serie` unter anderem das Attribut `Kommentar` hinzugefügt. Nachdem ich das erstere Schema implementiert und alle Tabellen mit Testdaten befüllt hatte, war das Hinzufügen von `Kommentar` die erste Operation, die ich durchgeführt habe. Nachfolgend ist der SQL-Befehl zu sehen, welche der Tabelle `Serie` das Attribut beziehungsweise die Spalte `Kommentar` hinzufügt. Dabei besitzen anfangs alle vorhandenen Tupel dort den initialen Wert `null`.

```
ALTER TABLE Serie ADD COLUMN Kommentar VARCHAR(50);
```

Der zugehörige Schema Modification Operator lautet wie folgt, wobei jedoch kein Attributtyp angegeben wird.

```
ADD COLUMN Kommentar INTO Serie;
```

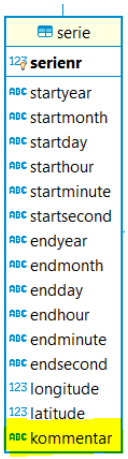

Abbildung 6.1: Ausschnitt des Entity-Relationship-Modells des Schemas von 1977, `Serie` erweitert um `Kommentar` (gelb markiert)

CREATE TABLE Beim Mapping des Schemas von 1977 auf das Schema von 2017 wurde außerdem ein neues Entity `Liegezeit` hinzugefügt. Daraus resultierte in meiner Implementierung eine eigene gleichnamige Tabelle. Der zugehörige SQL-Befehl ist nachfolgend abgebildet.

```
CREATE TABLE Liegezeit(
        FahrtNr VARCHAR(20),
        Datum Date,
        HafenBez varchar(50),
        Dauer Integer,
        Grund VARCHAR(100)
);
```

Dabei resultiert das Attribut `FahrtNr` aus der Abhängigkeit der `Liegezeit` von der jeweiligen `Fahrt`. Deshalb muss in der neuen Tabelle der Primärschlüssel der Tabelle `Fahrt` als Fremdschlüssel vorhanden sein. Außerdem ist die zuvor getroffene Einteilung zu beachten, welche ich für dieses Kapitel vorgesehen habe. So betrachte ich etwaige Integritätsbedingungen erst im nachfolgenden Unterkapitel.

RENAME COLUMN Beim Mapping des Schemas von 1997 auf das Schema von 2017 wurde, neben anderen Änderungen, das Attribut `WS` des Entities `Serie` in `WS-ID` umbenannt. Da dieses Attribut jedoch im implementierten Schema von 1977 noch nicht vorkommt, erstelle ich es zunächst über den nachfolgenden SQL-Befehl.

```
ALTER TABLE Serie ADD COLUMN WS INTEGER;
```

Nachdem dies geschehen ist, kann die neue Spalte/das neue Attribut `WS` umbenannt werden in `WS-ID`. Dies wird durch den folgenden SQL-Befehl vollzogen.

```
ALTER TABLE Serie RENAME COLUMN WS TO WS-ID;
```

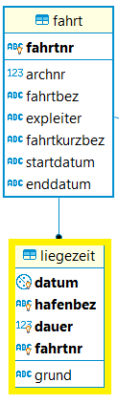

Abbildung 6.2: Ausschnitt des Entity-Relationship-Modells des Schemas von 1977, erweitert um neue Tabelle `Liegezeit` (gelb markiert)

Der zugehörige Schema Modification Operator kann hier also verschieden ausgedrückt werden, abhängig vom zu modifizierenden Schema:

- Schema 1977:

```
ADD COLUMN WS INTO Serie;
RENAME COLUMN WS IN Serie TO WS_ID;
```

- Schema 1997:

```
RENAME COLUMN WS IN Serie TO WS_ID;
```

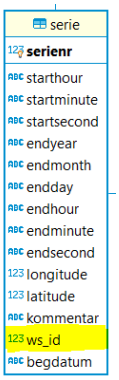

Abbildung 6.3: Ausschnitt des Entity-Relationship-Modells des Schemas von 1977, umbenannte Spalte `WS_ID` gelb markiert

6.2.2 ICMOs

Die zuvor ausgearbeiteten Änderungen in den Integritätsbedingungen drehen sich am Ende um die Entities `Fahrt` und `Liegezeit`. So wird zunächst durch den „Primärschlüsseltausch“ im Entity `Fahrt` der ICMO `DROP PRIMARY KEY` angewendet. Danach wird sowohl in dieser Tabelle als auch in der Tabelle `Liegezeit` ein neuer Primärschlüssel angelegt, was den ICMO `ADD PRIMARY KEY` fordert. Außerdem muss eine Fremdschlüssel-Referenzierung in der Tabelle `Liegezeit` vorgenommen werden, da diese abhängig von `Fahrt` ist.

Somit behandle ich diese drei ICMOs in einem komplexeren Beispiel und verzichte hierbei auf die zuvor klare Abgrenzung. Beim Auftauchen eines konkreten ICMOs werde ich darauf aufmerksam machen.

Das eigentliche Ziel dieses Abschnittes ist es, der Tabelle `Liegezeit` einen Primärschlüssel zu geben, welcher wegen der Abhängigkeit von `Liegezeit` einen Fremdschlüssel der Tabelle `Fahrt` enthält. Durch die Abhängigkeit der Beziehung `besitzt_fa_st_se` gestaltete sich dies jedoch komplizierter, als ich es zuvor im ER-Modell erahnt habe.

Zunächst habe ich eine neue Spalte `FahrtNr` über die beiden folgenden SQL-Befehle in der Tabelle `Fahrt` erstellt und ihr der Einfachheit halber den Wert der Spalte `ArchNr` der jeweiligen Tupel zugewiesen.

```
ALTER TABLE Fahrt ADD COLUMN FahrtNr VARCHAR(20);
UPDATE Fahrt SET FahrtNr = ArchNr WHERE FahrtNr IS NULL;
```

Danach habe ich die Fremdschlüssel-Referenzierung auf das alte Attribut `ArchNr` in der Tabelle `besitzt_fa_st_se` aufgehoben, um dann den alten Primärschlüssel in `Fahrt` löschen zu können und einen neuen Primärschlüssel darin anzulegen.

```
ALTER TABLE besitzt_fa_st_se DROP CONSTRAINT besitzt_fa_st_se_archnr_fkey;
ALTER TABLE Fahrt DROP CONSTRAINT fahrt_pkey;
ALTER TABLE Fahrt ADD PRIMARY KEY(FahrtNr);
```

Anschließend habe die Spalte `ArchNr` in `besitzt_fa_st_se` so angepasst, dass der neue Name `FahrtNr` und derin `Fahrt` verwendete Attributtyp VARCHAR(20) verwendet wird. Danach habe ich eine neue Fremdschlüssel-Referenzierung auf `FahrtNr` vorgenommen.

```
ALTER TABLE besitzt_fa_st_se RENAME COLUMN ArchNr TO FahrtNr;
ALTER TABLE besitzt_fa_st_se ALTER COLUMN FahrtNr TYPE VARCHAR(20);
ALTER TABLE besitzt_fa_st_se
  ADD FOREIGN KEY(FahrtNr) REFERENCES Fahrt(FahrtNr);
```

Nachdem die alten Fremdschlüssel-Beziehungen aktualisiert wurden, konnte ich nun zu guter Letzt zunächst den Fremdschlüssel auf dem Attribut `FahrtNr` in der Tabelle `Liegezeit` anlegen und anschließend den Primärschlüssel, bestehend aus den Spalten `FahrtNr`, `Datum`, `HafenBez` und `Dauer` anlegen.

```
ALTER TABLE Liegezeit ADD FOREIGN KEY(FahrtNr) REFERENCES Fahrt(FahrtNr);
ALTER TABLE Liegezeit ADD PRIMARY KEY(FahrtNr, Datum, HafenBez, Dauer);
```

6.2.3 MERGE COLUMNS & SPLIT COLUMN

Im nun folgenden Abschnitt beschäftige ich mich mit den Schema-Änderungen, welche bisher mit keinem eigene Schema Modification Operator bedacht wurden: das Zusammenfassen von Spalten (MERGE) und dem Aufteilen einer Spalte in mehrere (SPLIT). Hierbei war es notwendig, eigene Funktionen zu schreiben. Ich habe mich für eine Implementierung der Funktionen in SQL entschieden, weil hierbei vordefinierte Funktionen existieren, die meinen Anforderungen genügen.

MERGE COLUMNS Beginnen wir nun mit dem Mergen von Spalten. Dies geschah am häufigsten beim Mapping des Schemas von 1977 auf das Schema von 2017. So wurden unter anderem die drei Spalten `StartYear`, `StartMonth` und `StartDay` zu einem Attribut `BegDatum` zusammengefasst.

Dazu habe ich, meinem Ansatz aus dem Kapitel 5.3 folgend, zunächst eine neue Spalte `BegDatum` über den folgenden SQl-Befehl erstellt.

```
ALTER TABLE Serie ADD COLUMN BegDatum VARCHAR(10);
```

Hierbei ist wiederum zu beachten, dass ich die Typen der Spalten so gewählt habe, dass alle vom Typ VARCHAR sind, um eine einfache Implementierung über Textmanipulation zu ermöglichen.

Als nächstes folgte die Implementierung der Funktion, welche im Grunde einfach die drei vorhandenen Werte konkateniert und zwischen ihnen ein Trennzeichen (ein Punkt, wie bei Daten üblich) setzt. Mit dieser Funktion habe ich dann die Spalte `BegDatum` gefüllt, nachfolgend dargestellt durch den SQL-Befehl `UPDATE`.

```
UPDATE Serie
SET BegDatum = CONCAT(StartDay, '.', StartMonth, '.', StartYear)
WHERE BegDatum IS NULL;
```

Danach musste ich nur noch die alten Spalten `StartYear`, `StartMonth` und `StartDay` löschen. Somit ist auch das MERGEN von Spalten implementiert.

```
ALTER TABLE Serie DROP COLUMN StartYear;
ALTER TABLE Serie DROP COLUMN StartMonth;
ALTER TABLE Serie DROP COLUMN StartDay;
```

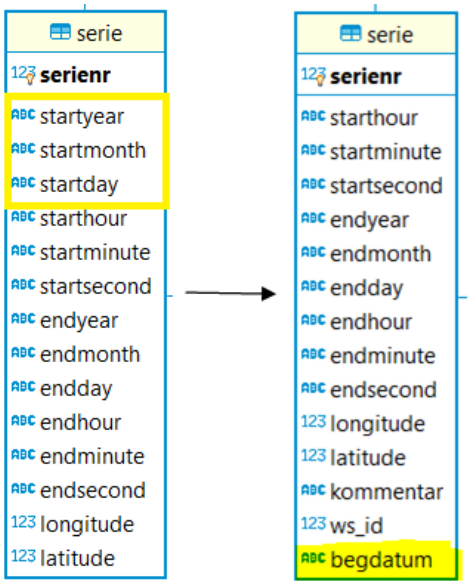

Abbildung 6.4: Die Spalten `StartYear`, `StartMonth` und `StartDay` (links, gelb markiert) werden zu `BegDatum` (rechts, gelb) zusammengefasst

SPLIT COLUMN Die letzte Implementierung, die ich vorgenommen habe, war das SPLITTEN der Spalte `Datum`. Dies bestand im originalen Schema von 1977 aus einem längeren VARCHAR, welcher sowohl das Start- als auch das Enddatum enthielt. Beim Mapping auf das Schema von 2017 wurden diese beiden Daten aufgetrennt in die beiden Spalten `StartDatum` und `EndDatum`. Auch bei dieser Implementierung halte ich mich an meinen Ansatz aus dem Kapitel 5.3 und treffe die Annahmen, dass alle verwendeten Spalten vom Typ VARCHAR sind und das Attribut `Datum` im Format „TT.MM.YYYY-TT.MM.YYYY“ vorliegt. Dementsprechend habe ich also auch die Werte in die Tabelle eingefügt.

Zunächst erstelle ich die beiden neuen Spalten `StartDatum` und `EndDatum` über die beiden folgenden SQl-Befehle.

```
ALTER TABLE Fahrt ADD COLUMN StartDatum VARCHAR(10);
ALTER TABLE Fahrt ADD COLUMN EndDatum VARCHAR(10);
```

Danach befülle ich diese beiden neuen Spalten mit dem Funktionswert der folgenden `SUBSTRING`-Funktionen. Diese zielen mit den Angaben von Index und Länge des entstehenden VARCHARs auf eben genanntes vorgegebenes Format ab. Eine Fehlerbehandlung habe ich nicht implementiert. So müsste beispielsweise in der Realität das Format überprüft und falsche/irrationale Werte ausgeschlossen werden.

```
UPDATE Fahrt
SET StartDatum = SUBSTRING(Datum, 1, 10)
WHERE StartDatum IS NULL;

UPDATE Fahrt
SET EndDatum = SUBSTRING(Datum, 12, 10)
WHERE EndDatum IS NULL;
```

Nachdem die neuen Spalten erstellt und entsprechend befüllt wurden, lösche ich abschließend noch die alte Spalte `Datum` und habe damit auch den SPLIT problemfrei implementieren können.

```
ALTER TABLE Fahrt DROP COLUMN Datum;
```

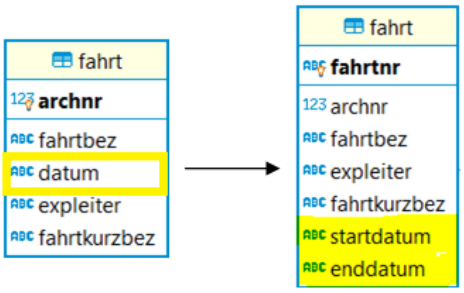

Abbildung 6.5: Die Spalte `Datum` (links, gelb markiert) wird in die Spalten `StartDatum` und `EndDatum` (rechts, gelb markiert) aufgeteilt

7 Zusammenfassung und Ausblick

In diesem Kapitel fasse ich noch einmal überblicksartig zusammen, welche Erkenntnisse ich während der Bearbeitung dieser Arbeit gemacht habe. Dabei gehe ich auf die Arbeitsschritte und Vorgehensweisen ein, welche zur Erstellung meines Konzepts notwendig waren. Abschließend werfe ich im Ausblick einige Fragestellungen auf, welche sich im Zuge dieser Arbeit auftaten und zu diesem Zeitpunkt noch unbeantwortet sind. Diese können als Ausgangspunkt für weiterführende Arbeiten genutzt werden.

7.1 Zusammenfassung

Das Leibniz-Institut für Ostseeforschung Warnemünde erhebt seit vielen Jahrzehnten Messdaten über die Ostsee in verschiedenen Bereichen. Ich habe mich mit Messdaten der CTD-Messung, also Leitfähigkeit, Temperatur und Druck, beschäftigt. Seit Beginn der Messungen veränderten sich die Methoden und Parameter, nach denen gemessen wird. Dementsprechend veränderten sich auch sowohl die Struktur der Daten als auch die damit verbundenen Schemata.

Ein Ziel meiner Arbeit war es, diese Schema-Änderungen zu bestimmen und mit Hilfe sogenannter Schema-Evolutions-Operatoren zu beschreiben. Dafür habe ich zunächst Datensätze vieler Jahre analysiert und für die mir am wichtigsten erscheinenden Schemata Entity-Relationship-Modelle (ER-Modelle) erstellt. Die Wichtigkeit richtete sich nach der Anzahl und Art der Schema-Änderungen. Nachdem ich die ER-Modelle für die Jahre 1977, 1997 und 2017 angefertigt hatte, beschrieb ich die auftretenden Änderungen mit eben jenen Schema-Evolutions-Operatoren. Ein Problem dabei bestand darin, dass die Tragweite einiger Änderungen nicht unmittelbar aus den ER-Modellen heraus sichtbar war. So bedingte der Tausch des Primärschlüssels im Entity-Typ `Fahrt` eine Kette von Integritätsänderungen in anderen Entity-Typen (siehe Abschnitt 6.2.2).

Des Weiteren fiel mir auf, dass für einige Änderungen zum Zeitpunkt der Erstellung dieser Arbeit noch keine eigenen Operatoren existieren. Für diese Operatoren (`MERGE ATTRIBUTES` und `SPLIT ATTRIBUTE`) habe ich eigene Ansätze formuliert und in der späteren Implementierung getestet. Aufgrund des Umstandes, dass diese Ansätze jedoch SQl-Funktionen benötigen, konnte ich feststellen, dass nicht alle Schema-Änderungen durch die grundlegenden Schema-Evolutions-Operatoren, wie sie in den Artikeln [CMZ08] und [CMDZ10] eingeführt wurden, beschrieben werden konnten. Mit der Implementierung des Schemas von 1977 und der Anwendung der herausgearbeiteten Schema-Änderungen konnte ich zeigen, dass auch meine Ansätze für `MERGE ATTRIBUTES` und `SPLIT ATTRIBUTE` funktionieren.

Im zweiten Teil meiner Arbeit beschäftigte ich mich mit der Frage, ob die herausgearbeiteten Schema-Änderungen mit dem Ziel der Data Provenance, also der Herkunft von Daten, verträglich sind. Die Kombination dieser beiden Gebiete fiel mir verhältnismäßig schwer, unter anderem weil mir bisher nur ein Artikel bekannt ist, welcher sich mit dieser Thematik auseinandersetzt [GZ12]. Die meisten veröffentlichten Artikel behandeln die Provenance in festen, sich nicht verändernden Schemata.

Bei der Untersuchung der Provenance-Verträglichkeit habe ich mich auf Schema-Änderungen durch Schema Modification Operators konzentriert. Dafür habe ich zunächst für alle auftretenden Arten die Inversen bestimmt und konnte in der anschließenden Untersuchung feststellen, dass die meisten Änderungen (bis auf `CREATE TABLE`) mit Provenance-Techniken verträglich sind. In einigen Fällen (`MERGE ATTRIBUTES` und `SPLIT ATTRIBUTE`) müssen jedoch weitere Informationen über die Schema-Änderung und genutzte Funktionen vorliegen, um die Herkunft der Daten nachvollziehen zu können. Gerade bei Funktionen, die mit konkreten Werten arbeiten, fiel mir auf, dass die bisher definierten Anfragetypen der Data Provenance nicht immer ausreichen. Das Problem besteht darin, dass die hinterlegten Typen der Spalten nicht beachtet werden. Dafür habe ich einen Ansatz formuliert (**WHAT**-Provenance, Abschnitt 5.5), der dieses Problem aufgreift.

7.2 Ausblick

Nach Beendigung dieser Arbeit bleiben meiner Meinung nach einige Punkte für die weitere Bearbeitung offen. So stellte sich mir bei meinen Ansätzen für die neuen Schema-Evolutions-Operatoren die Frage, ob es noch weitere Schema-Änderungen gibt, die noch nicht durch die bisher definierten Operatoren beschrieben werden können. Dafür müssten mehr Schemata verschiedener Art auf Änderungen untersucht werden.

Des Weiteren habe ich in dieser Arbeit lediglich die Verträglichkeit von solchen Schema-Evolutions-Operatoren mit Provenance-Techniken untersucht, welche keine Integritätsbedingungen verändern. Eine Untersuchung der Kombination von Provenance mit integritätsverändernden Operatoren steht also noch aus. Dazu könnten auch die Beispiele genutzt werden, welche ich im Abschnitt 5.3 herausgearbeitet habe.

8 Anhang

```
* Sea-Bird SBE 9 Raw Data File:
* FileName = C:\SBE\DAT\0001F01.DAT
* Software Version 4.206
* Temperature SN = 1589
* Conductivity SN = 1349
* Number of Bytes Per Scan = 30
* Number of Voltage Words = 4
* System UpLoad Time = Feb 06 1997 10:36:04
* ReiseNr  = 44/97/02 Ostseemonitoring
* StationNr= 0001
* EinsatzNr= 0001
* SerieNr  = 01
* StatBez  = TFO5 aus < MONISTAT >
* Startzeit= 10:32:36 utc 06-FEB-97
* GPS_Posn = 54 13.9940N  012 04.5280E
* Echolote = XXXXXXX m 0011.92 m
* Luftdruck= 1027.1 hPa
* ThermoSal=  .666 grdC  12.166 ppth
* WS_Anzahl=  4
* Operator = Ruickoldt
# nquan = 15
# nvalues = 12
# units = metric
# name 0 = pr: pressure [db]
# name 1 = t090: temperature, ITS-90 [deg C]
# name 2 = iowSv: IOW sound velocity [m/s]
# name 3 = haardtP: Dr Haardt, phycoerythrin
# name 4 = haardtC: Dr Haardt, chlorophyll a
# name 5 = haardtT: Dr Haardt, turbidity
# name 6 = timeS: time [s]
# name 7 = scan: scan number
# name 8 = sal00: salinity, PSS-78 [PSU]
# name 9 = sigma-t00: density, sigma-t [kg/m^3]
# name 10 = potemp090: potential temperature, ITS-90 [deg C]
# name 11 = dm: dynamic meters
# name 12 = dz/dt: descent rate [m/s]
# name 13 = flag:  0.000e+00
# name 14 = nbin: number of scans per bin
# span 0 = 1.306, 11.747
# span 1 = 0.6169, 0.6205
# span 2 = 9.02, 9.10
# span 3 = 0.4221, 0.4298
# span 4 = 0.7317, 0.9027
# span 5 = 0.0609, 0.0669
# span 6 = 31.012, 63.394
# span 7 = 745, 1522
# span 8 = 12.1906, 12.3007
# span 9 = 9.7325, 9.8211
# span 10 = 0.6169, 0.6205
# span 11 = 0.0127, 0.1974
# span 12 = 0.077, 0.388
# span 13 = 0.000e+00, 0.000e+00
# span 14 = 15.0000, 108.0000
# interval = decibars: 1
# start_time = Feb 06 1997 10:36:04
# bad_flag = -9.990e-29
# sensor 0 = Frequency  0  temperature, 1589, 23-01-97
# sensor 1 = Frequency  1  conductivity, 1349, 23-01-97, cpcor = -9.5700e-08
# sensor 2 = Frequency  2  pressure, 57440, 02-03-1994
# sensor 3 = Frequency  3  IOW, sound velocity (frequency)
# sensor 4 = Frequency  4  IOW, hydro-bios bottles closed (frequency)
# sensor 5 = Extrnl Volt  2  altimeter
# sensor 6 = Extrnl Volt  5  Dr Haardt, phycoerythrin, 40812, nicht vorhanden
# sensor 7 = Extrnl Volt  6  Dr Haardt, chlorophyll a, 4081, nicht vorhanden
# sensor 8 = Extrnl Volt  7  Dr Haardt, turbidity, 4081, nicht bekannt
# datcnv_date = Mar 20 1997 12:35:57, 4.221
# datcnv_in = 0001F01.DAT 449702_A.CON
# datcnv_skipover = 0
# presscor_date = Thu Mar 20 13:43:03 1997, v3.3 24.1.1997
# presscor_pre_correction = 989.91 hPa
# presscor_luftdruck = 1027.1 hPa
# presscor_value_added = -0.3719 decibars (-37.19 hPa)
# do_om_date = Thu Mar 20 14:12:50 1997, v3.3 24.1.1997
```

```
# do_om_B1_B2_B3 = -0.033096, 0.014259, -0.0017
# do_om_K = 1
# split_date = Mar 20 1997 14:27:35, 4.221
# split_in = 0001F01.CNV
# split_excl_bad_scans = yes
# strip_date = Mar 20 1997 14:39:54, 4.221
# strip_in = D0001F01.CNV
# wildedit_date = Mar 20 1997 14:52:54, 4.221
# wildedit_in = D0001F01.CNV
# wildedit_pass1_nstd = 2.0
# wildedit_pass2_nstd = 20.0
# wildedit_npoint = 100
# wildedit_vars = pr t090 c0mS/cm iowSv haardtP haardtC haardtT
# wildedit_excl_bad_scans = yes
# wfilter_date = Mar 20 1997 22:24:50, 4.221
# wfilter_in = D0001F01.CNV
# wfilter_excl_bad_scans = yes
# wfilter_action pr = median, 231
# wfilter_action t090 = median, 231
# wfilter_action c0mS/cm = median, 231
# wfilter_action iowSv = median, 231
# celltm_date = Mar 21 1997 07:48:11, 4.221
# celltm_in = D0001F01.CNV
# celltm_alpha = 0.0300, 0.0000
# celltm_tau = 9.0000, 0.0000
# filter_date = Mar 21 1997 08:09:04, 4.221
# filter_in = D0001F01.CNV
# filter_low_pass_tc_A = 0.100
# filter_low_pass_tc_B = 0.150
# filter_low_pass_A_vars =
# filter_low_pass_B_vars = pr
# loopedit_date = Mar 21 1997 08:28:15, 4.221
# loopedit_in = D0001F01.CNV
# loopedit_minVelocity = 0.270
# loopedit_excl_bad_scans = yes
# derive_date = Mar 21 1997 09:02:41, 4.221
# derive_in = D0001F01.CNV 449702_A.CON
# derive_time_window_dzdt = seconds: 2
# depth_date = Fri Mar 21 09:05:55 1997, v3.3 24.1.1997
# depth_error = 0076
# strip_date = Mar 21 1997 09:21:08, 4.221
# strip_in = D0001F01.CNV
# binavg_date = Mar 21 1997 09:32:55, 4.221
# binavg_in = D0001F01.CNV
# binavg_bintype = Pressure Bins (No Interpolation)
# binavg_binsize = 1.00
# binavg_excl_bad_scans = yes
# binavg_downcast_only = no
# binavg_skipover = 0
# binavg_surface_bin = no, min = 0.000, max = 0.000, value = 0.000
# file_type = ascii
*END*
      1.306     0.6205       9.04     0.4244     0.8244     0.0609     31.012
      745    12.1906     9.7325     0.6205     0.0127      0.077  0.000e+00   108.0000
      2.024     0.6182       9.03     0.4287     0.8361     0.0611     35.729
      859    12.1919     9.7336     0.6182     0.0256      0.306  0.000e+00    74.0000
      2.969     0.6172       9.03     0.4296     0.8803     0.0611     38.542
      926    12.1956     9.7365     0.6172     0.0423      0.388  0.000e+00    61.0000
      3.972     0.6170       9.03     0.4221     0.8926     0.0616     41.271
      992    12.2004     9.7404     0.6170     0.0601      0.355  0.000e+00    70.0000
      5.048     0.6169       9.05     0.4225     0.8874     0.0611     44.104
      1060    12.2031     9.7426     0.6169     0.0791      0.364  0.000e+00
      66.0000
      5.979     0.6170       9.06     0.4268     0.9027     0.0610     46.875
      1126    12.2055     9.7445     0.6170     0.0956      0.353  0.000e+00
      67.0000
      7.012     0.6176       9.08     0.4227     0.8855     0.0614     49.688
      1194    12.2275     9.7622     0.6176     0.1138      0.357  0.000e+00
      68.0000
      7.983     0.6184       9.09     0.4268     0.8798     0.0618     52.417
      1259    12.2503     9.7805     0.6184     0.1310      0.375  0.000e+00
      63.0000
      8.987     0.6185       9.10     0.4247     0.8596     0.0631     55.167
```

```
          1325    12.2581     9.7868     0.6185     0.1487      0.349  0.000e+00
          69.0000
135       9.973     0.6183       9.10     0.4225     0.8385     0.0641     57.932
          1391    12.2636     9.7912     0.6183     0.1661      0.352  0.000e+00
          63.0000
136       11.031     0.6184       9.06     0.4221     0.8270     0.0639     61.202
          1470    12.2746     9.8001     0.6185     0.1847      0.323  0.000e+00    56.0000
137       11.747     0.6190       9.02     0.4298     0.7317     0.0669     63.394
          1522    12.3007     9.8211     0.6190     0.1974      0.119  0.000e+00    15.0000
138
```

Abbildung 8.1: 1997 – Datei „d0001f01.cnv“

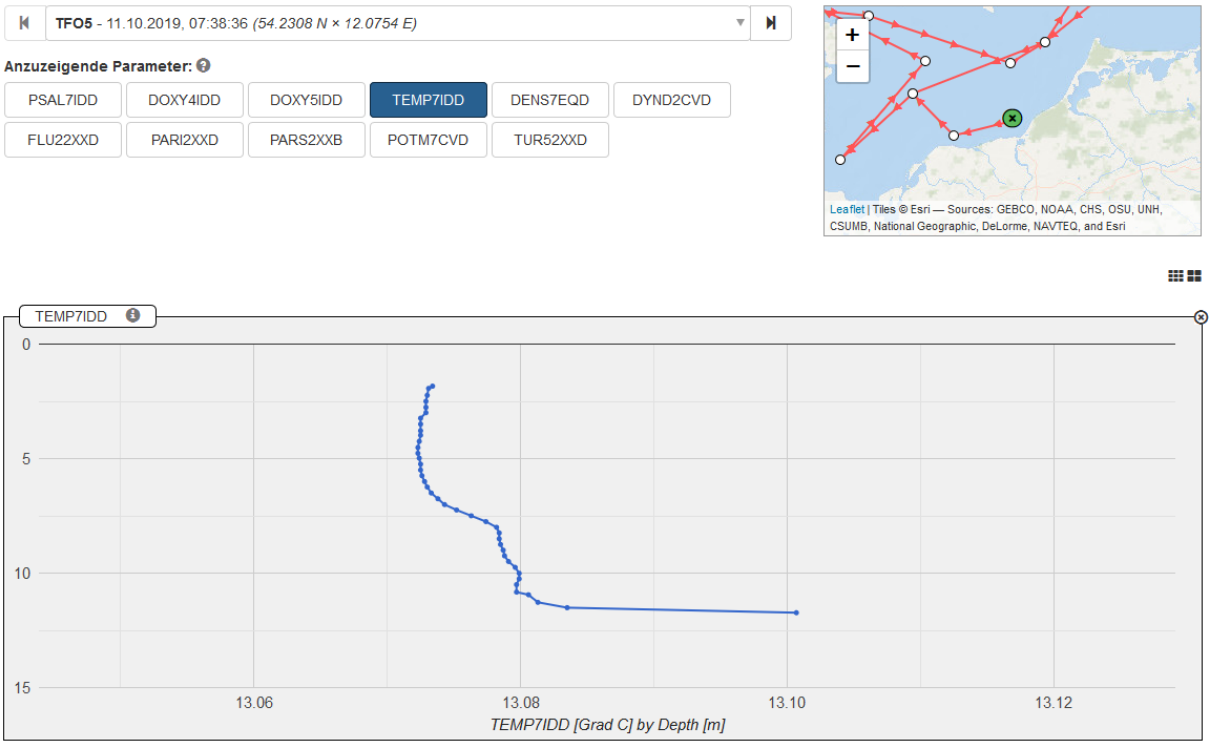

Abbildung 8.2: Visualisierung von Daten der Terminfahrt 2019/10 (24.02.2020 16:35Uhr)

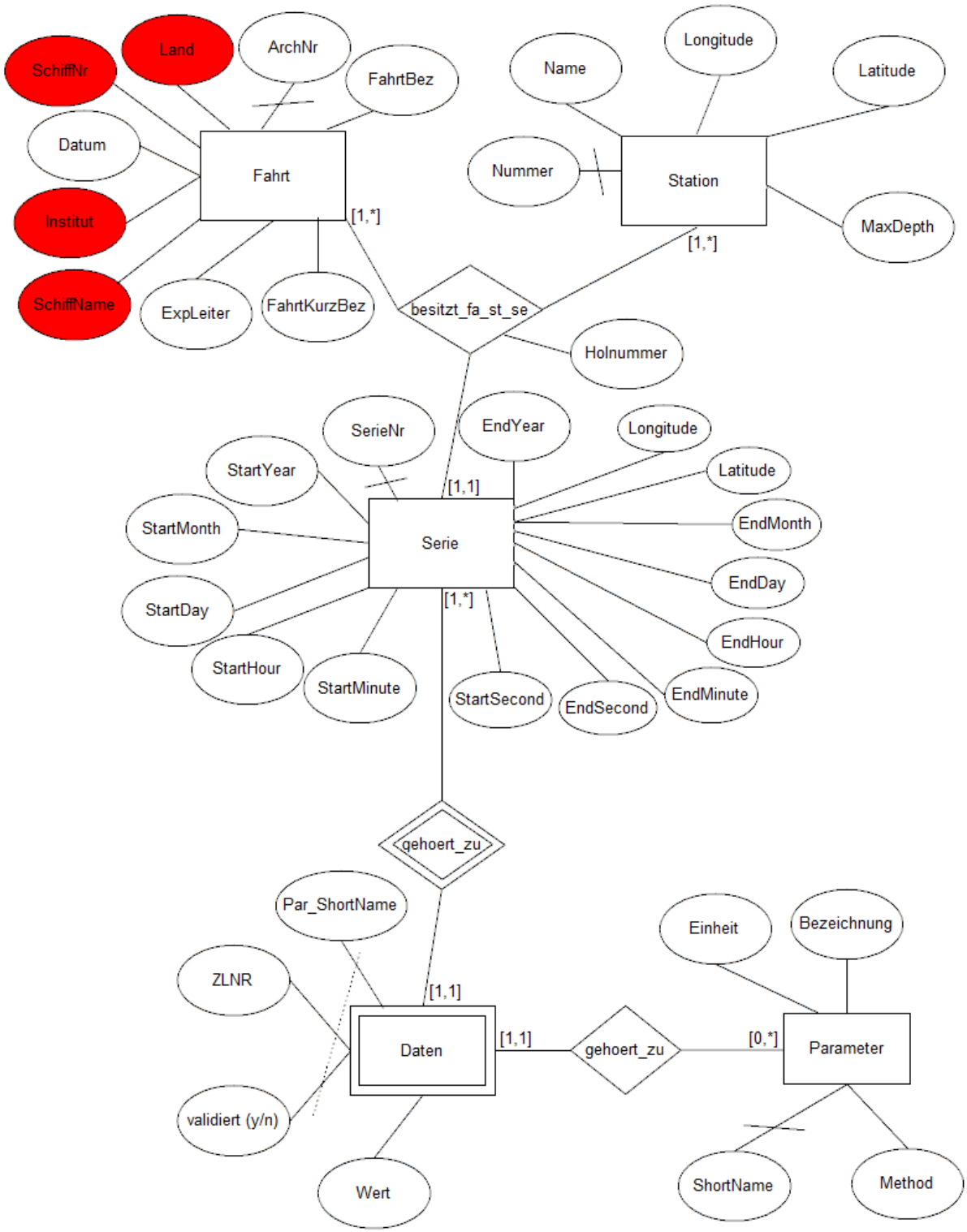

Abbildung 8.3: Entity Relationship Modell 1989; in rot hervorgehoben sind die Änderungen zum Schema von 1977

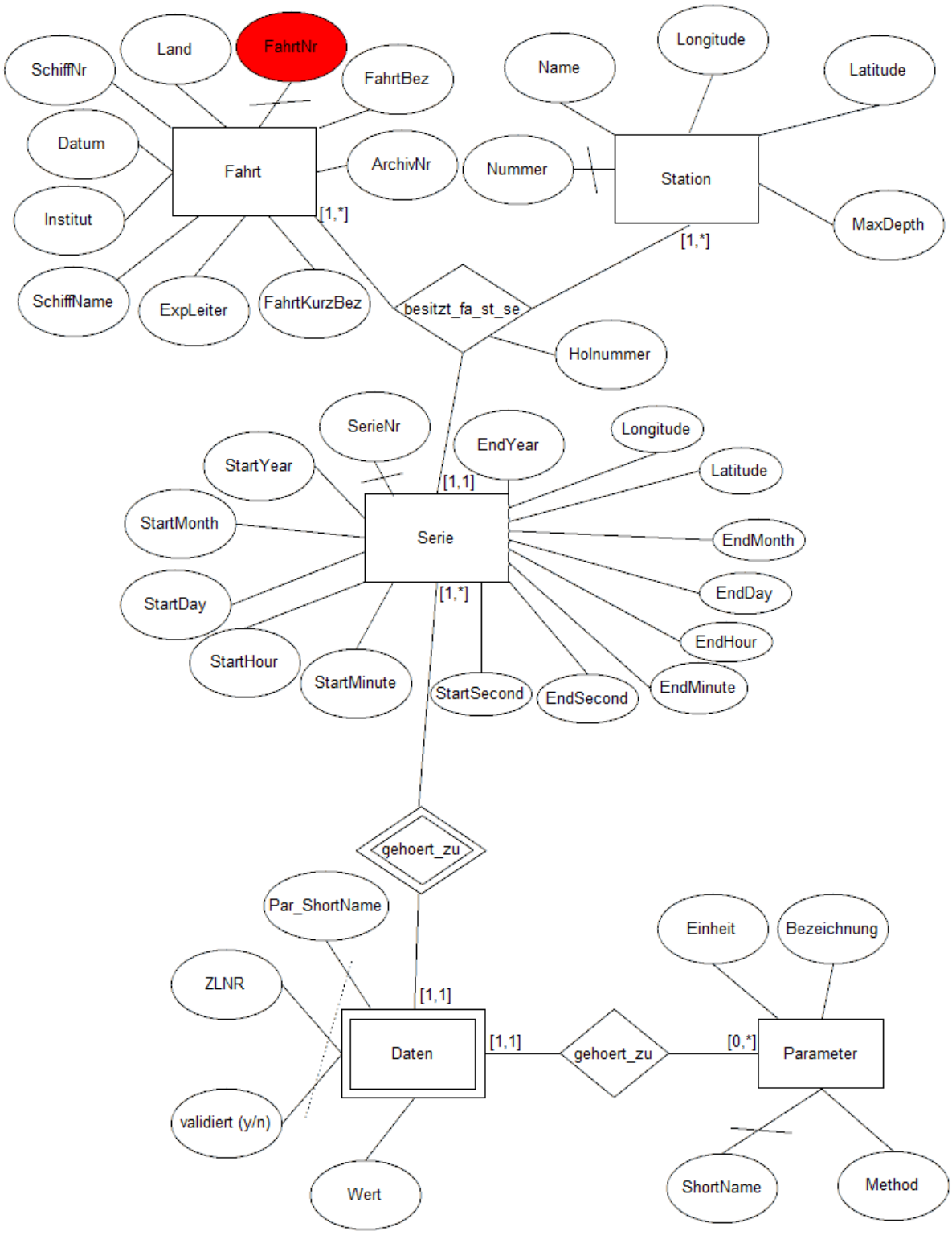

Abbildung 8.4: Entity Relationship Modell 1992; in rot hervorgehoben sind die Änderungen zum Schema von 1989

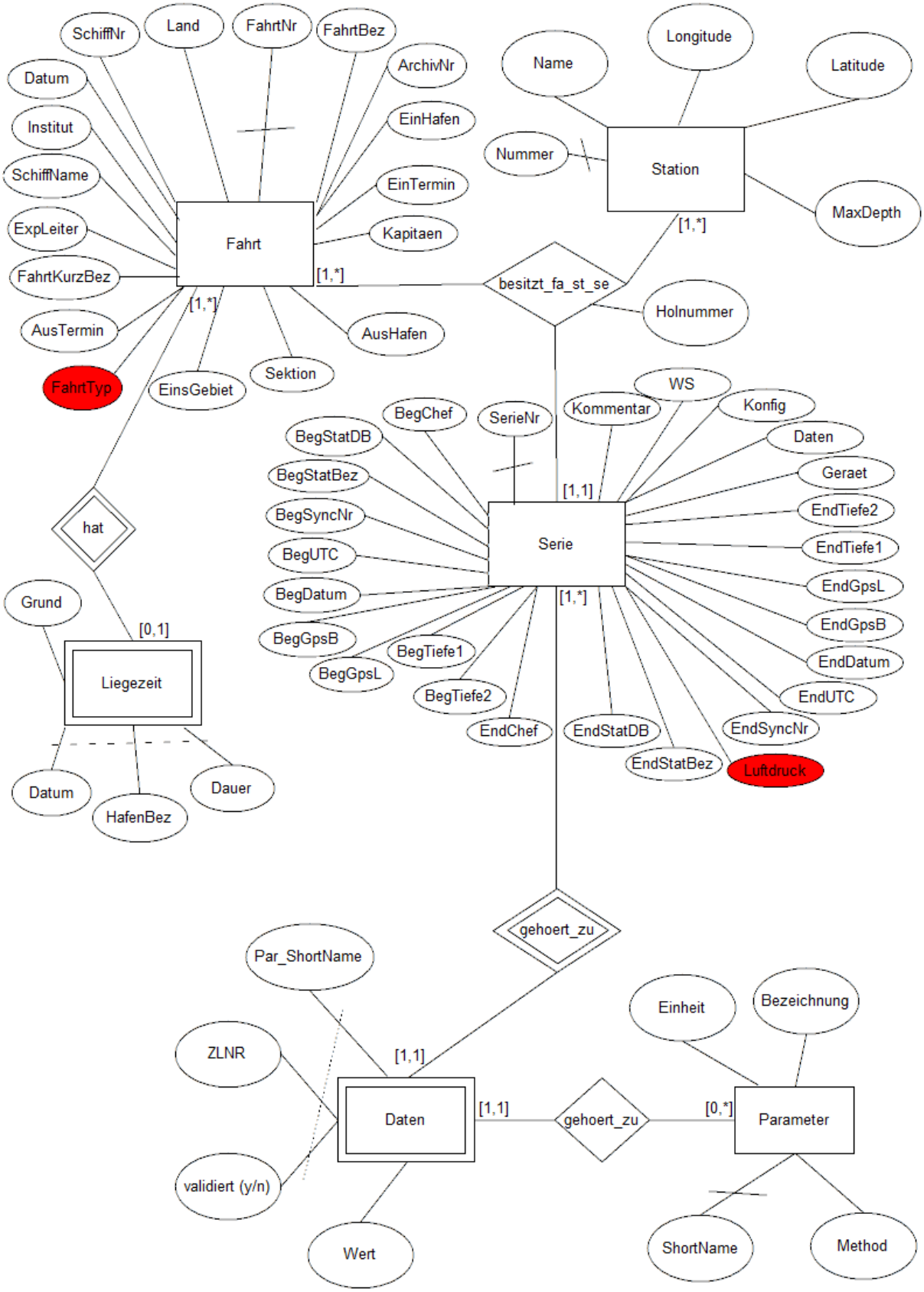

Abbildung 8.5: Entity Relationship Modell 2000 und 2005; in rot hervorgehoben sind die Änderungen zum Schema von 1992

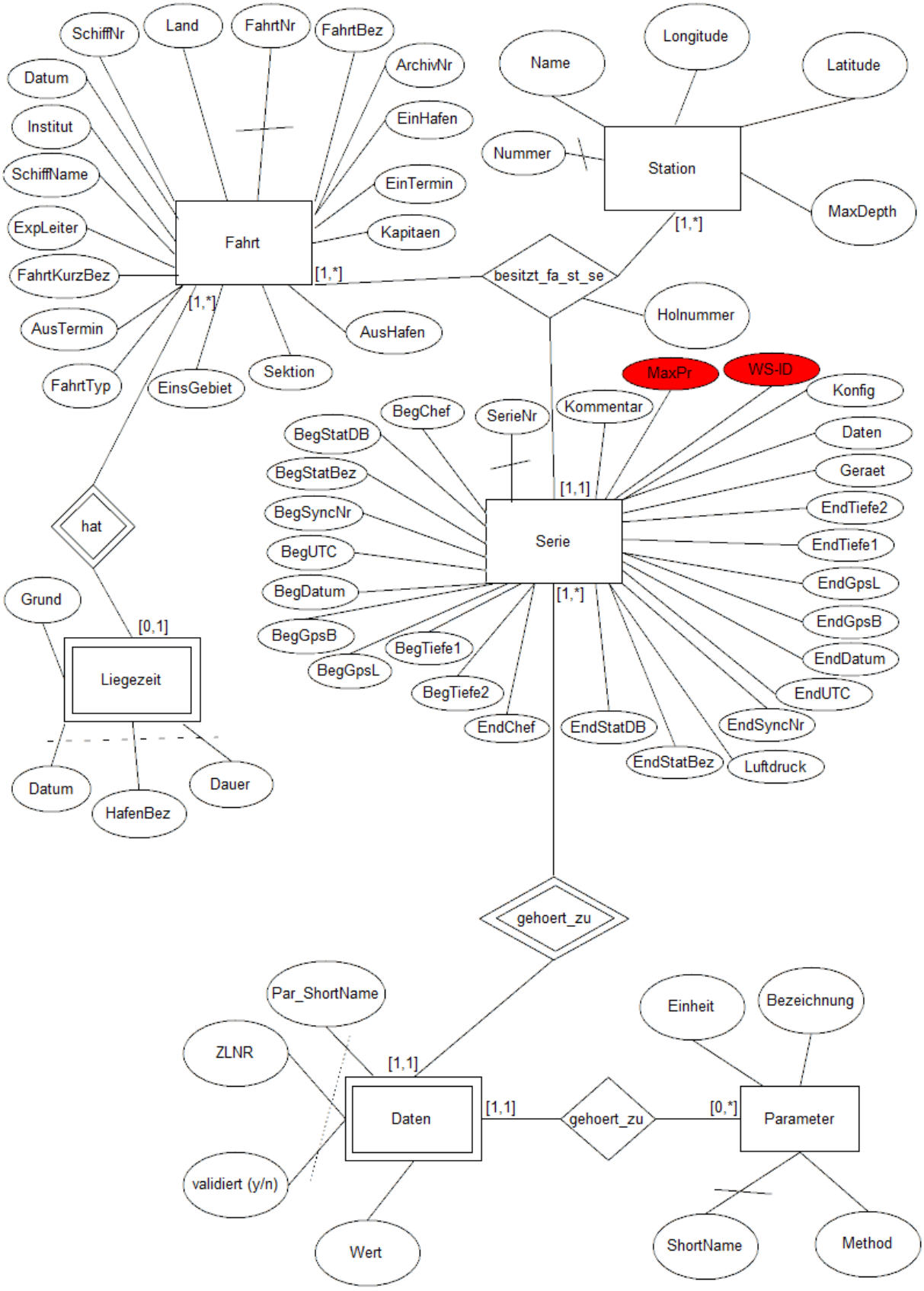

Abbildung 8.6: Entity Relationship Modell 2010 und 2015; in rot hervorgehoben sind die Änderungen zum Schema von 2000 bzw. 2005

Leitfaden für die digitale Version

In diesem Kapitel gebe ich einen Überblick über die beiliegende digitale Version meiner Bachelorarbeit sowie die enthaltenen Ordner und Dateien. Die Ordnerstruktur beim Öffnen des Datenträgers sieht wie folgt aus:

- Bachelorarbeit_TeX
- besuchte_Internetseiten
- Fahrtdaten
- genutzte_Literatur
- Implementierung

Die folgenden Überschriften entsprechen den enthaltenen Ordnern.

Bachelorarbeit_TeX Dieser Ordner enthält die TeX-Dateien meiner Bachelorarbeit, die digitale Version meiner Bachelorarbeit („BA_Manthey.pdf“) sowie im Unterordner „Bilder“alle enthaltenen Abbildungen.

besuchte_Internetseiten Dieser Ordner enthält Screenshots aller besuchten Internetseiten inklusive Zeitpunkt des letzten Besuches im Dateinamen. Dabei bezieht sich die Datei
„Screenshot_Leia_29_02_2020_23Uhr04.png“auf [Leia]
und die Datei „Screenshot_Leic_01_03_2020_11Uhr13.png“auf [Leic].

Fahrtdaten Dieser Ordner enthält Unterordner, welche die verwendeten Fahrtdaten enthalten, die mir vom IOW zur Verfügung gestellt wurden. Die Ordnernamen stehen dabei für die entsprechenden Jahre, aus denen die Daten stammen. Außerdem enthält der Ordner **Fahrtdaten** den Unterordner „Leia“, welcher die Datei zur Quelle [Leia] enthält.

genutzte_Literatur Dieser Ordner enthält die genutzte Literatur als .pdf-Dateien. Diese tragen den Namen ihrer jeweiligen Quellenangabe.

Implementierung Dieser Ordner enthält die Definition des implementierten Schemas von 1977 inklusive aller zugehöriger Tabellen in den Dateien „Schema_DDL.sql“ und „Schema_DDL.txt“. Außerdem enthält der Unterordner „PSQL_12“ meine aktuelle Installation von Postgresql inklusive des modifizierten Schemas. Leider habe ich es nach vielen Versuchen nicht geschafft, auf dem Datenträger eine lauffähige virtuelle Maschine zu erstellen.

Literaturverzeichnis

[AH19] Auge, Tanja und Andreas Heuer: *ProSA - Using the CHASE for Provenance Management.* In: *Advances in Databases and Information Systems - 23rd European Conference, ADBIS 2019, Bled, Slovenia, September 8-11, 2019, Proceedings*, Band 11695 der Reihe *Lecture Notes in Computer Science*, Seiten 357–372. Springer, 2019.

[BFJ] Bock, Steffen, Susanne Feistel und Susanne Jürgensmann: *Poster: Data Management at IOW, 2014.*

[BKM+17] Bruder, Ilvio, Meike Klettke, Mark Lukas Möller, Frank Meyer, Andreas Heuer, Susanne Jürgensmann und Susanne Feistel: *Daten wie Sand am Meer - Datenerhebung, -strukturierung, -management und Data Provenance für die Ostseeforschung.* Datenbank-Spektrum, 17(2):183–196, 2017.

[CCT09] Cheney, James, Laura Chiticariu und Wang Chiew Tan: *Provenance in Databases: Why, How, and Where.* Foundations and Trends in Databases, 1(4):379–474, 2009.

[CMDZ10] Curino, Carlo, Hyun Jin Moon, Alin Deutsch und Carlo Zaniolo: *Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++.* PVLDB, 4(2):117–128, 2010.

[CMTZ08] Curino, Carlo, Hyun Jin Moon, Letizia Tanca und Carlo Zaniolo: *Schema Evolution in Wikipedia - Toward a Web Information System Benchmark.* In: *ICEIS 2008 - Proceedings of the Tenth International Conference on Enterprise Information Systems, Volume DISI, Barcelona, Spain, June 12-16, 2008*, Seiten 323–332, 2008.

[CMZ08] Curino, Carlo, Hyun Jin Moon und Carlo Zaniolo: *Graceful database schema evolution: the PRISM workbench.* PVLDB, 1(1):761–772, 2008.

[DF08] Davidson, Susan B. und Juliana Freire: *Provenance and scientific workflows: challenges and opportunities.* In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, Seiten 1345–1350. ACM, 2008.

[Fen] Fennel, Wolfgang: *Meereskunde in Warnemünde - Ausgangslage, Abwicklung und Neugründung in der Zeit von 1990 bis 1994.* In: *Meereswissenschaftliche Berichte.*

[GZ12] Gao, Shi und Carlo Zaniolo: *Provenance Management in Databases Under Schema Evolution.* In: *4th Workshop on the Theory and Practice of Provenance, TaPP'12, Boston, MA, USA, June 14-15, 2012.* USENIX Association, 2012.

[HDB17] Herschel, Melanie, Ralf Diestelkämper und Houssem Ben Lahmar: *A survey on provenance: What for? What form? What from?* VLDB J., 26(6):881–906, 2017.

[Leia] Leibniz-Institut für Ostseeforschung Warnemünde: *BMP: Monitoring Cruise 90/02 (TF0003, 54.5850 N, 12.5533 E) (IOWDB), https://iowmeta.io-warnemuende.de/geonetwork/srv/eng/catalog.search#/metadata/IOW-IOWDB-STID-24571, zuletzt aufgerufen am 29.02.2020 um 23:04Uhr.*

[Leib] Leibniz-Institut für Ostseeforschung Warnemünde: *CTD im Sondenraum, Terminfahrt März 2012.*

[Leic] Leibniz-Institut für Ostseeforschung Warnemünde: *Organigramm des IOW, https://www.io-warnemuende.de/kurzvorstellung-lageplan.html, zuletzt aufgerufen am 01.03.2020 um 11:13 Uhr.*

[Mey] Meyer, Frank: *Aufbau einer Artenlistenverwaltung im Benthos-Projekt, Bachelorarbeit, Universität Rostock, Institut für Informatik, 2015.*

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

[Yan] Yan, Jiawei: *Conception and Implementation of a Schema Extraction of Long-term in situ Research Data from the Baltic Sea, Masterarbeit, Universität Rostock, Institut für Informatik, 2018.*

[Zie] Zierke, Jessica: *Konzeption der Datenintegration für eine zu entwickelnde Benthos-Datenbank, Masterarbeit, Universität Rostock, Institut für Informatik, 2014.*

[ZS] Ziedorn, Frauke und Volker Sossna: *Forschungsdaten managen - Anforderungen, Methoden, Hilfsmittel.*

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Rostock, den 03.03.2020 _____