Universität Rostock

Traditio et Innovatio

Bachelorarbeit

Aufbau einer Artenlistenverwaltung im Benthos-Projekt

Frank Meyer
Matrikelnummer: 209204737
Abgabedatum: 27.04.2015

Gutachter
Prof. Dr. Andreas Heuer
Dr. Michael Zettler

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

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

Abstract

Zur Artbestimmung von in Bodennähe lebenden Meereslebewesen am Leibniz-Institut für Ostseeforschung Warnemünde muss eine gültige Artenliste bzw. Taxonomie vorliegen. Diese Arbeit beschäftigt sich mit der Untersuchung eines Versionierungskonzeptes zur Verwaltung und Pflege biologischer Taxonomien. Dazu werden mehrere Modelle zur Abbildung von Hierarchien in relationalen Datenbanken vorgestellt und verglichen. Unter Begründung wird ein hybrides Modell gewählt, welches das Modell der Adjazenz-Liste und das Path-Enumeration-Modell verbindet. Der temporale Aspekt der Versionierung wird mittels dem SQL:2011-Standard bitemporal abgebildet. Es wird anhand eines durchgängigen Beispiels gezeigt, dass das Konzept den Anforderungen der Operationen auf Taxonomien stand hält. Das Erstellen einer gültigen Artenliste als Ausschnitt aus der Gesamttaxonomie wird in zwei Varianten gezeigt. Dabei wird auf die Einschränkungen dieser Varianten und des entworfenen Datenbank-Schemas eingegangen. Das Konzept wird prototypisch unter Verwendung des MySQL Datenbanksystems implementiert. Abschließend erfolgt die kritische Bewertung des Konzeptes, der Umsetzung und des verwendeten SQL:2011-Standards.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Das Vorhandensein von Stammbäumen in der Biologie, sei es lokal in Form von Artenlisten in Excel-Protokollen oder global in Form eines Katalogs des Lebens, macht es notwendig diese Baumstrukturen zu verwalten und zu pflegen. Dieses Kapitel erläutert, um was es sich bei dem „Benthos-Projekt“ handelt, warum dort eine derartige Verwaltung gebraucht wird und wie die vorliegende Arbeit aufgebaut ist.

1.1 Benthos-Projekt

Das „Benthos-Projekt“ bezeichnet die Zusammenarbeit zwischen dem Lehrstuhl „Datenbank- und Informationssysteme“ der Universität Rostock und der Arbeitsgruppe „Ökologie benthischer Organismen“ am Leibniz-Institut für Ostseeforschung Warnemünde (IOW) im Zuge der Datenintegration von Excel-Protokollen in eine MySQL Datenbank.

Das IOW beschäftigt sich mit interdisziplinärer Meeresforschung in Küsten und Randmeeren. Der Schwerpunkt der Arbeiten am IOW liegt in der Erforschung des Ökosystems der Ostsee [Lei15b]. Die Arbeitsgruppe „Ökologie benthischer Organismen“, welche der Sektion „Biologische Meereskunde“ angehört, erhebt hierbei Daten über Benthos-Arten im Rahmen des Baltic-Monitoring Programms der HELCOM[1] [ZGDS15]. Der Begriff des Benthos selbst bezeichnet die Gesamtheit aller Lebewesen auf dem Meeresboden und in allen Tiefenzonen [GDS15]. Benthos beinhaltet sowohl Krebstiere, Weichtiere als auch Vielborster (z.B. Wattwurm) sowie verschiedene weitere Arten.

Die Datenerhebung erfolgt durch Sammeln von Benthosproben an verschiedenen Ostseestationen [Lei15a]. Dabei werden weitere Metadaten, wie zum Beispiel der Stationsname, das Datum, die Wassertiefe und -temperatur sowie der Salzgehalt, in Excel-Eingabeprotokollen gespeichert. Die Proben werden danach gesäubert, fixiert und im Labor weiter untersucht. Es erfolgt eine Sortierung, indem die Individuen nach ihren Taxa auf Grundlage zum Zeitpunkt der Bestimmung gültigen Artenliste bestimmt werden. Anschließend werden die Biomassen der Proben in verschiedenen Einheiten bestimmt [Zie14].

[1] Baltic Marine Environment Protection Commission - Helsinki Commission http://helcom.fi/

Diese Daten und Ergebnisse helfen, den ökologischen Gesundheitszustand der Biotope und Lebensräume des Meeresbodens im Rahmen der marinen Umweltüberwachung zu bewerten und einzuschätzen sowie zur Verbesserung des Zustandes der Küstengewässer beizutragen [ZGDS15].

1.2 Motivation

Die Biologie als Teilgebiet der Naturwissenschaften befasst sich unter Anderem mit der Klassifizierung der natürlichen Artenvielfalt. Die so entstehenden Stammbäume oder Hierarchien müssen verwaltet und gepflegt werden. Aufgrund neuer Erkenntnisse in Form von Forschungsergebnissen treten Änderungen in solchen biologischen Hierarchien auf. Das führt zu einer kontinuierlichen Anpassung des Stammbaums. Dieser Wandel von alten zu neuen Daten erfordert das Festhalten der Zustände über die Zeit.

Um aus den gesammelten Benthosproben die Arten bestimmen zu können, muss eine gültige Artenliste vorliegen. Eine solche Artenliste ist immer für ein Jahr gültig. Änderungen werden im Laufe des Jahres notiert aber noch nicht übernommen. Erst zu einem vereinbarten Termin (z.B. Jahresende) wird eine neue aktualisierte Artenliste als neue gültige Version für das kommende Jahr freigegeben.

Zurzeit wird durch Mitarbeiter der Arbeitsgruppe „Ökologie benthischer Organismen“ eine Artenliste in den Excel-Protokollen gepflegt. Es erfolgen Änderungen, indem neue Arten eingetragen und um fehlende Informationen aus frei zugänglichen Datenbanken, wie WoRMS[1] oder BioLib[2], per Hand ergänzt werden. Aus den Eingabe-Protokollen werden Prüfberichte generiert, welche sich zum Zeitpunkt des Erstellens auf die aktuell gültige Artenliste beziehen.

Die manuelle Pflege dieser Artenlisten ist aufwändig. Der Abgleich mit den erwähnten Datenbanken erfolgt per Hand, eine Automatisierung ist zurzeit nicht vorhanden. Da Excel hauptsächlich ein Tabellenkalkulationsprogramm ist, ist es nicht geeignet, große Datenmengen langfristig sicher zu speichern und auswerten zu können [Zie14]. Es bietet keinen Mehrbenutzerbetrieb und keine Rechteverwaltung. Darüber hinaus liegen Excel-Dateien meist lokal auf einem Rechner und es existiert keine zentral geregelte Zugriffsmöglichkeit. Durch die Verwendung eines Netzlaufwerkes kann dieser Umstand zwar kompensiert werden, es greift in dem Fall aber wieder der fehlende Mehrbenutzerbetrieb. Das bewirkt, dass andere Benutzer diese Datei nur im Lesemodus öffnen können und Änderungen gegebenenfalls wieder in einer lokalen Kopie durchführen und später in das Original zusammen führen müssen. Die Benutzung von Excel als Datenverwaltungswerkzeug wird somit mit zunehmender Nutzerzahl und steigenden Datenmengen immer unflexibler [Zie14].

[1] World Register of Marine Species http://www.marinespecies.org
[2] Biological Library http://www.biolib.cz/en/

Für die langfristige Speicherung großer Datenmengen und effiziente Abfragen darauf, haben sich Datenbanksysteme bewährt. Ziel ist es deshalb, die vorhandene Artenliste in eine Datenbank zu überführen und geeignete Werkzeuge zur Verwaltung bereitzustellen.

Parallel zu dieser Arbeit erfolgt die Datenintegration der Excel-Protokolle in eine MySQL-Datenbank durch die Universität Rostock in Zusammenarbeit mit dem Hanse Softwarehaus[3]. Im Zuge dieser Migration soll diese Arbeit untersuchen, wie ein Verwaltungskonzept für Artenlisten umgesetzt kann.

1.3 Aufbau der Arbeit

An dieser Stelle wird auf den Aufbau der Arbeit eingegangen. Kapitel 1 dient der Einführung und erklärt, worum es sich bei dem „Benthos-Projekt“ handelt und warum eine Artenlistenverwaltung gebraucht wird. In Kapitel 2 wird kurz in die biologische Systematik eingeführt. Infolgedessen wird der Begriff der Taxonomie definiert und die darauf ausführbaren logischen Operationen, wie zum Beispiel Namensänderungen verschiedener Rangstufen, beschrieben. Es folgt die Beleuchtung der aktuellen Vorgehensweise der Arbeitsgruppe „Ökologie benthischer Organismen“ sowie eine Beschreibung der Aufgabenstellung und Anforderungen. Kapitel 3 legt die benötigten Grundlagen für das spätere Konzept. Es werden verschiedene Modelle vorgestellt, um Hierarchien in relationalen Datenbanken abzubilden. Danach folgt das temporale Konzept gemäß dem SQL:2011-Standard. Abschließend wird ein Überblick über den Stand der Technik gegeben. In Kapitel 4 wird das formale Konzept zur Lösung der Aufgabenstellung erläutert. Es erfolgt die Wahl eines Hierarchie-Modells sowie die Anwendung der temporalen Aspekte darauf. Das zugehörige Datenbankschema wird vorgestellt. Danach wird die Durchführung der Operationen auf Taxonomien anhand eines durchgängigen Beispiels gezeigt. Schließlich wird auf die eigentliche Erzeugung einer Artenliste eingegangen. Kapitel 5 setzt sich mit der prototypischen Umsetzung des Konzepts in MySQL auseinander. Es wird knapp auf mehreren Implementierungsdetails eingegangen. In Kapitel 6 wird Kritik am Konzept und der Umsetzung geübt. Es zeigt Probleme auf und geht auf alternative Lösungsmöglichkeiten ein. Kapitel 7 fasst die Arbeit zusammen und gibt einen Ausblick auf mögliche Fragestellungen und Entwicklungen.

Die beigelegte CD umfasst alle Bilder und Grafiken dieser Arbeit. Weiterhin sind alle Dateien, die während der Umsetzung erzeugt wurden im Verzeichnis „Umsetzung“ vorhanden. Alle erwähnten Webseiten, sowohl im Text als auch im Literaturverzeichnis, befinden sich als Kopie im PDF Format und als HTML-Dateien im Verzeichnis „Literatur/Webseiten“.

[3] http://www.hanse-softwarehaus.de/

2 Problemstellung

Dieses Kapitel soll als kurze Einführung in die biologische Systematik dienen und definiert in diesem Rahmen den Begriff der Taxonomie. Anschließend werden die Operationen beleuchtet, die auf einer Taxonomie ausgeführt werden können. Danach wird auf die aktuelle Vorgehensweise der Arbeitsgruppe „Ökologie benthischer Organismen“ bei der Erstellung und Pflege einer Artenliste eingegangen. Abschließend werden die Anforderungen dargelegt und die Aufgabenstellung beschrieben.

2.1 Artenliste: Definition, Aufbau und Operationen

In den folgenden Abschnitten wird erläutert, um was es sich bei der Artenliste handelt, welchen Aufbau diese besitzt und welche Operationen auf dieser ausgeführt werden können.

2.1.1 Definition und Aufbau

Die Bestimmung von Arten erfolgt nach der biologischen Systematik und Nomenklatur [Lin59]. Die Theorie und Praxis der biologischen Klassifikation, welche Lebewesen in ein biologisches System einordnet, wird als Taxonomie bezeichnet [Fre04]. Durch diese Tätigkeit entsteht eine Hierarchie bzw. ein Baum. Jeder Knoten der Hierarchie kann einer bestimmten Rangstufe zugeordnet werden und wird als Taxon (Pl. Taxa) bezeichnet. Rangstufen sind z.B. „Klasse“, „Ordnung“, „Familie“ oder „Gattung“. Arten, als unterste biologische Einheit dieser Taxonomie, entsprechen den Blättern des Baumes. Eine Artenliste besteht folglich aus Einträgen der Blätter einer Taxonomie. Es kann aber vorkommen, dass eine Art nicht bestimmt werden kann (siehe 2.2). Dann werden höherrangige Taxa verwendet. Die Artenliste ist demnach eine Taxa-Liste und folglich auch eine Taxonomie im Sinne einer Hierarchie. Aus diesem Grund ist der Begriff der Artenliste nicht mehr genau genug und wird in dieser Arbeit von nun an synonym mit Taxonomie, Taxa-Liste oder Hierarchie verwendet. Jede Ebene des Baums entspricht einer Rangstufe in der biologischen Systematik. Abbildung 2.1 veranschaulicht den allgemeinen Aufbau einer Taxonomie. In der Beispiel-Hierarchie A.1.1 ist die Rangstufe in Klammern im jeweils ersten Taxon einer neuen Ebene angegeben. Die dort verwendete Taxonomie ist sehr vereinfacht und enthält nur die Hauptrangstufen, es wurden Über-/Unter- und Teilgruppen weggelassen.

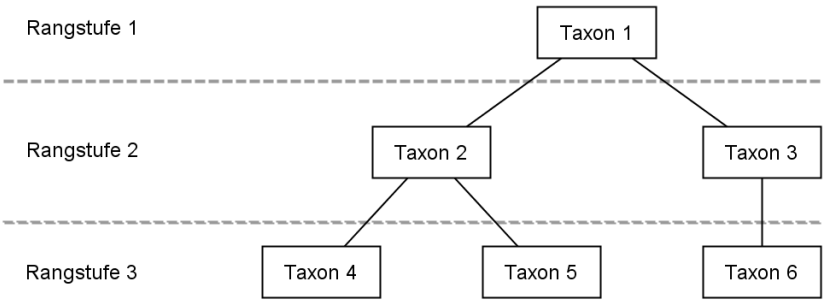

Abbildung 2.1: Allgemeiner Aufbau einer Taxonomie

2.1.2 Operationen

Im Folgenden wird sich immer auf die Beispiel-Taxonomie A.1.1 aus der Literaturarbeit [Mey15] bezogen. Operationen auf Taxonomien können unproblematisch und unkompliziert, aber auch sehr komplex sein. Eine vermeintlich einfache Namensänderung einer Art besteht aus Aktualisierung der alten Art und Einfügen einer neuen Art mit neuem Namen. Zusätzlich wird ein Synonym-Verweis angelegt, welcher diese beiden Arten miteinander in Beziehung setzt. Grafik 2.2 zeigt, wie so eine Namensänderung durchgeführt wird und welche Änderungen auf der Taxonomie erfolgen. Durch das Ändern von Taxa entstehen eine Vielzahl an Synonymen, sogenannte Synonym-Listen, welche immer zum gerade gültigen Taxon führen. Ein Taxon ist genau dann gültig, wenn es den „Internationalen Regeln für die zoologische Nomenklatur“ oder dem „Internationen Code der Nomenklatur für Algen, Pilze und Pflanzen“ entspricht [KZN00]. In WoRMS z.B. weist der Status „accepted“ das Taxon als gültig aus. Durch Anlegen von Synonymen werden keine Daten oder Taxa gelöscht. So kann die Integrität von Arbeiten und Protokollen aus der Vergangenheit, welche sich auf veraltete Taxa und Taxonomien beziehen bzw. diese referenzieren, gewährleistet werden.

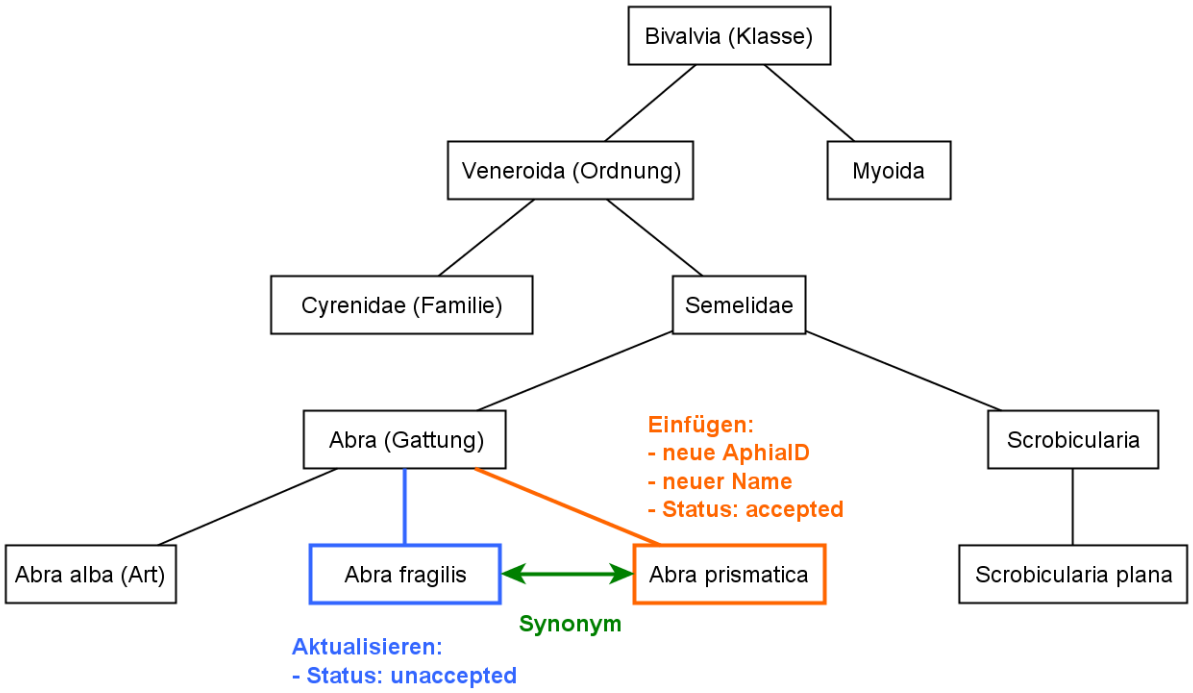

Abbildung 2.2: Operation auf einer Taxonomie - Namensänderung einer Art

Die verschiedenen Farben symbolisieren die verschiedenen Operationen, wie Aktualisieren der alten Daten (**blau**), Einfügen neuer Daten (**orange**) und Anlegen von Synonymen (**grün**). Die Farben werden von nun an auch bei der Erklärung der anderen Beispieloperationen benutzt und die Beschriftung weggelassen.

Anstelle einer Namensänderung einer Art kann auch nur der Status dieser Art auf „unaccepted“ gesetzt werden. Beispiel 2.3 veranschaulicht diese Änderung an einem Ausschnitt der Hierarchie. Zu erkennen ist, dass eine Aktualisierung des veralteten Taxons stattfindet und ein Synonym-Verweis angelegt wird.

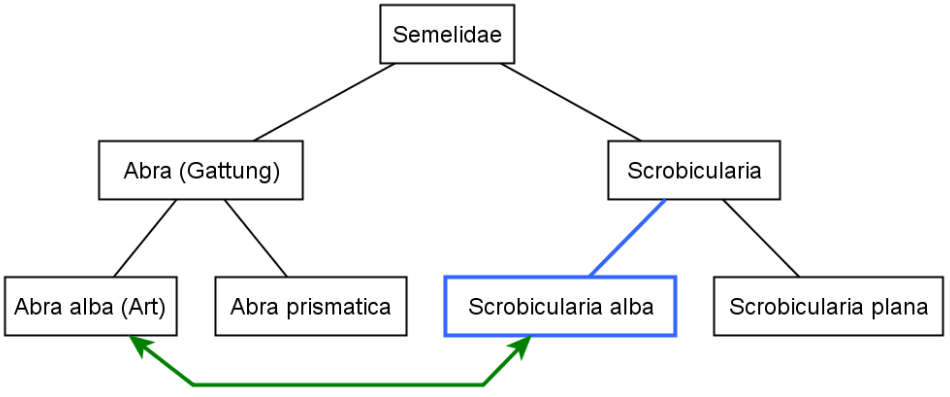

Abbildung 2.3: Operation auf einer Taxonomie - Statusänderung einer Art

Ein weiteres Beispiel in Abbildung 2.4 verdeutlicht die erhöhte Komplexität der Namensänderung einer Gattung, da hier auch alle Kind-Knoten geändert werden müssen, weil sich der Artname selbst aus der Gattung und der Art zusammen setzt. Die Änderung besteht somit aus mehreren Operationen:

1. Extraktion des Teilbaums. Die Wurzel des Teilbaums ist durch die Abfrage bekannt (gelb hinterlegt).
2. Einfügen eines neuen Teilbaums als Kopie des alten Teilbaums (**orange**):
   a) Die Wurzel des neuen Teilbaums bekommt den neuen Vater-Knoten und ggf. einen neuen Namen.
   b) Einfügen der neuen Kinder mit ggf. auch neuem Namen unter der neuen Wurzel.
   c) Anlegen von Synonymen zur alten Wurzel und den alten Kindern (**grün**).
3. Aktualisieren aller Taxa des alten Teilbaums: Status auf „unaccepted“ (**blau**).

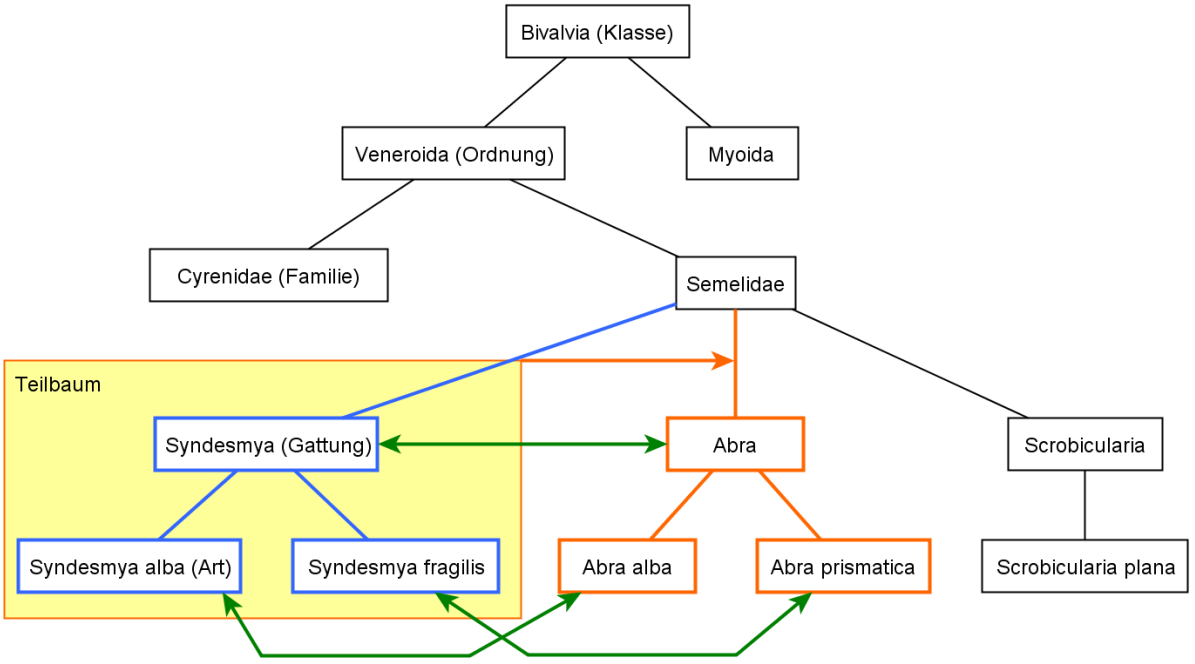

Abbildung 2.4: Operation auf einer Taxonomie - Namensänderung einer Gattung

Gleiches gilt für das Zusammenlegen zweier Gattungen zu einer Gattung, nur ohne das neue Einfügen einer neuen Gattung. Die Arten einer Gattung werden der anderen Gattung zugeordnet und erhalten einen neuen Namen. Die veraltete Gattung und deren veraltete Arten werden Synonyme für die neue Gattung und deren Arten. Das Umhängen einer Gattung ohne Namensänderung zieht dagegen auch keine Namensänderung der Arten nach sich. Die komplette Gattung inklusive ihrer Arten wird dabei unter einer anderen Familie gehängt.

Wird die Namensänderung einer Familie an einem weiteren Beispiel 2.5 untersucht, ist festzustellen, dass wieder die drei Grundoperationen in abgewandelter Form stattfinden. Es wird hier nicht der gesamte Teilbaum benötigt, sondern nur die direkten Kinder der Familie, also alle

Taxa der Rangstufe Gattung. Diese werden aktualisiert, nachdem ein Familien-Taxon mit neuem Namen eingefügt, der Status der alten Familie angepasst und die Synonyme angelegt worden sind. Die Aktualisierung der Kinder bewirkt, dass diese nun auf die neue Familie zeigen. Die schwarzen gestrichelten Verbindungen sollen verdeutlichen, an welcher Stelle in der Hierarchie die direkten Nachfolger vor der Aktualisierung hingen.

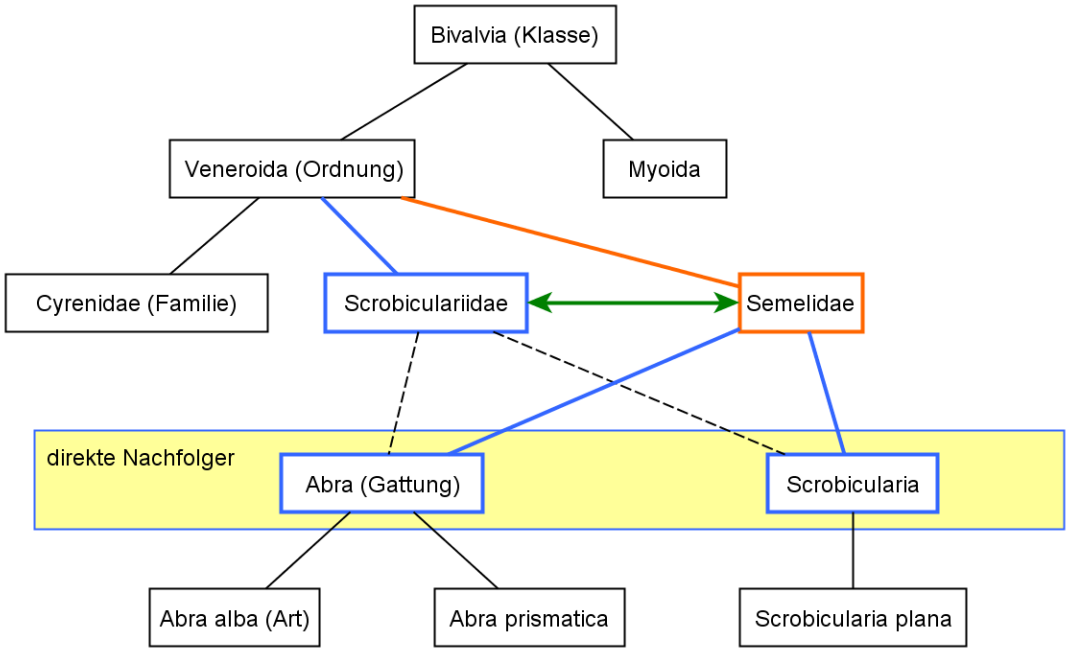

Abbildung 2.5: Operation auf einer Taxonomie - Namensänderung einer Familie

Werden Taxa der Rangstufe Familie und höherer Ebenen betrachtet, so ist festzustellen, dass immer wieder dieselben Operationen stattfinden: Finden der direkten Kinder, Einfügen neuer Daten und Aktualisierung der alten Daten. Je höher aber die Rangstufe ist, desto seltener und damit unwahrscheinlicher sind die Operationen auf den entsprechenden Taxa. Es kommt häufig vor, dass sich eine Art in eine andere Gattung verschiebt, aber es ist sehr selten, dass Änderungen an einem Taxon auf und über der Rangstufe der Familie gemacht werden. Es ist zum Beispiel wahrscheinlicher, dass eine komplett neue Familie entsteht und zu dieser bereits bestehende Gattungen und Arten hinzugefügt werden.

Als Ergebnis der Untersuchung der Operationen auf Taxonomien zeigt sich, dass sowohl das Finden eines Teilbaums oder der direkten Nachfolger als auch das Einfügen der neuen Daten und die Aktualisierung des Teilbaums bzw. der direkten Nachfolger häufige und immer wiederkehrende Operationen sind.

2.2 Aktuelle Vorgehensweise

Wie in der Einleitung 1.2 bereits erwähnt, wird die Artenliste in Excel-Protokollen als eigene Mappe gepflegt. Mit Filter- und Sortier-Techniken ist es möglich, nach Rangstufen zu sortieren und die entsprechenden Arten zu finden. Diese Liste besitzt allerdings nur unzureichende Informationen über die Hierarchie-Beziehungen der Taxa untereinander. So existieren nur die Spalten „Gattung“, „Art“ und der etwas vage Begriff der „Gruppe“, welche meist die Rangstufe der „Familie“ repräsentiert, aber auch dazu verwendet wird, nicht eindeutig identifizierbare Organismen anhand ihrer übergeordneten Rangstufe einzuordnen. Die Bestimmung der Art wird also immer so genau wie möglich durchgeführt, ist aber nicht immer realisierbar. Als Beispiel sei hier ein Stück Material vom Meeresboden genannt, auf welchem die darauf lebenden Organismen derart miteinander verbunden sind, dass eine Bestimmung der einzelnen Art nicht mehr im Bereich des Möglichen liegt. In diesem Fall weiß man zwar, dass diese Organismen zum Beispiel einer bestimmten Gattung angehören, aber nicht welcher Art. Es erfolgt dann die Aufnahme der übergeordneten Rangstufe, hier also der Gattung, in das Protokoll.

Eine Artenliste ist immer für einen bestimmten Zeitraum, meist ein Jahr, gültig. Änderungen werden notiert bzw. in einer bereits neuen parallel dazu geführten Artenliste eingepflegt. Diese neue Artenliste wird mit anderen frei verfügbaren Datenbanken abgeglichen. Dies geschieht zurzeit manuell und ist ein aufwändiger Prozess. An erster Stelle steht dabei der Abgleich mit WoRMS. Wurden dort die entsprechenden Informationen nicht gefunden, wird in BioLib und danach in ITIS[1] nachgeschlagen. In seltenen Fällen muss eine Einzelfallprüfung durchgeführt werden, falls die obigen Quellen keine Informationen liefern können. Von WoRMS wird eine sogenannte AphiaID geliefert, welche ein Taxon eineindeutig identifiziert und dort als Primärschlüssel verwendet wird. Diese ist auch in der Artenliste in den Excel-Protokollen vorhanden. Es besteht aber zurzeit keine Möglichkeit, anhand dieser ID automatisch den WoRMS Eintrag zu ermitteln.

Sind die neuen Änderungen und Aktualisierungen abgeschlossen, so wird die neue Artenliste als neue Version freigegeben und zukünftig in allen Excel-Protokollen verwendet. Derzeit sind die eigentlichen Versionsnummern der Artenliste in den Excel-Protokollen an dem Suffix „_Version#“ zu erkennen. Damit wird von vorherigen Artenlisten-Versionen in zurückliegenden Excel-Protokollen unterschieden.

[1] Integrated Taxonomic Information System http://www.itis.gov/

2.3 Aufgabenstellung und Anforderungen

Aus der Betrachtung in Abschnitt 2.2 ergeben sich die folgenden Anforderungen.

Verwaltung der Artenliste an einer zentralen Stelle.
Verwalten bedeutet, das Hinzufügen, Ändern und Löschen von Taxa. Einerseits soll dies weiterhin per Hand möglich sein, andererseits soll der Abgleich mit WoRMS automatisiert werden, um den manuellen Aufwand zu reduzieren. Dazu bietet sich der von WoRMS bereitgestellte Webservice an. Als zentrale Stelle ist ein Webinterface vorgesehen, welches von Mitarbeitern der AG „Ökologie benthischer Organismen“ über das Intranet im IOW aufgerufen werden kann.

Integrität und Korrektheit von Prüfberichten, die sich auf eine Taxonomie beziehen.
Es ist sehr wichtig, dass sich Prüfberichte aus einem bestimmten Zeitraum immer auf diejenige Taxonomie beziehen, die in diesem Zeitraum gültig war. Wird die Taxonomie durch die Tätigkeit des Verwaltens geändert, muss die Beziehung zwischen Prüfbericht und der ursprünglichen Taxonomie gewährleistet sein. Dies bedeutet, dass sichergestellt werden muss, dass zu jedem Prüfbericht die jeweils zum Zeitpunkt der Erstellung des Prüfberichts gültige Taxonomie auch nachträglich korrekt generiert und zugeordnet werden kann.

Temporale Versionierung und Änderungsverfolgung.
Durch die Verwaltung von Taxa entstehen verschiedene Versionen, die zu verschiedenen Zeitpunkten gültig waren bzw. noch sind. Dies macht ein Versionierungskonzept zur Nachverfolgung der Änderungen über Zeiträume erforderlich.

Aufnahme höherer Rangstufen.
Ober-/Unter- und Teilgruppen, wie sie in den Protokollen aufgenommen wurden, werden weggelassen. Der unscharfe Begriff „Gruppe“ fällt komplett weg. Dafür finden jetzt auch höhere Rangstufen bis zum Rang „Stamm“ in der neuen Datenbank Berücksichtigung. Daraus ergeben sich folgende zu erfassenden Rangstufen (von der Wurzel bis zum Blatt): Stamm $\rightarrow$ Klasse $\rightarrow$ Ordnung $\rightarrow$ Familie $\rightarrow$ Gattung $\rightarrow$ Art.

Es gilt einerseits ein geeignetes Speichermodell zu wählen, mit dem eine Hierarchie (Taxonomie) in einer Datenbank abgebildet werden kann und andererseits ein Versionierungskonzept zu finden, so dass jedem Prüfbericht zu jedem Zeitpunkt die zu diesem Zeitpunkt jeweils gültige Artenliste, sowohl aktuell als auch in der Vergangenheit, zugeordnet werden kann. Anhand von Prototypen wird aufgezeigt, wie der automatisierte Abgleich mit WoRMS stattfinden kann.

3 Grundlagen

Dieses Kapitel behandelt die nötigen Grundlagen für die Problemlösung. Zunächst erfolgt eine Definition verschiedener Begriffe, die in diesem Kapitel benutzt werden. Abschnitt 3.2 stellt verschiedene Modelle für die Speicherung von Hierarchien in relationalen Datenbanken vor. Wie zeitbezogene Daten in relationalen Datenbanken gespeichert werden, ist in Abschnitt 3.3 beschrieben. Abschließend erfolgt in Abschnitt 3.4 eine knappe Skizzierung des Stands der Technik.

3.1 Begriffsdefinitionen

Relation

Mathematischer Formalismus zur Beschreibung von Tabellen. Dabei entspricht die Relation streng genommen den Daten in der Tabelle. Wird oft als Synonym für Tabelle verwendet.

Attribut

Synonym für Spalte bzw. Spaltenüberschriften.

Tupel

Zeile einer Relation, Datensatz.

Primärschlüssel

Ausgezeichnetes Attribut oder Menge von Attributen einer Relation, welches ein Datensatz eindeutig in dieser Relation identifiziert.

Fremdschlüssel

Verweis auf einen Primärschlüssel von einer anderen Relation heraus.

Referentielle Integrität

Bedingungen zur Sicherstellung, dass ein, durch einen Fremdschlüssel, referenzierter Primärschlüssel auch in der referenzierten Relation vorhanden ist.

Rekursion

Aufruf der Funktion durch sich selbst mit bereits vorberechneten Werten derselben Funktion.

Stored Procedure

Hinterlegte Prozeduren im Datenbanksystem, die vom Benutzer aufgerufen werden können.

Trigger

Mechanismus zur automatischen Reaktion einer Datenbank auf benutzerdefinierte Ereignisse [TS06].

SFW-Block

Kurzform für SELECT-FROM-WHERE-Block. Standard SQL-Anweisung zum Ermitteln von Datensätzen.

DDL

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

DML

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

Angelehnt an die Abbildung 1.7. aus [SSH13] verdeutlicht Abbildung 3.1 einige behandelte Grundbegriffe.

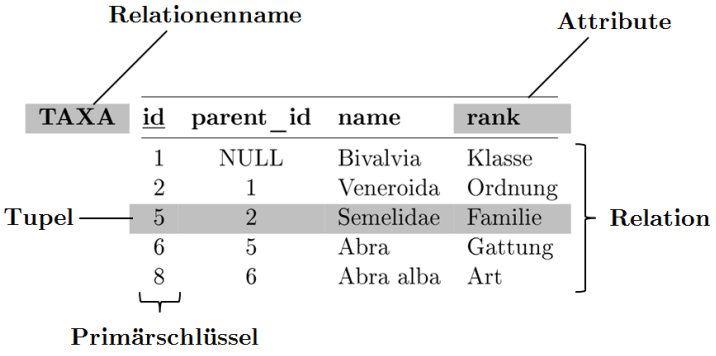

Abbildung 3.1: Veranschaulichung einiger Grundbegriffe

3.2 Speicherung von Hierarchien in Datenbanken

Es existieren verschiedene Modelle, um Hierarchien in relationalen Datenbanken abzubilden. Als Vorleistung zu dieser Arbeit wurden einige Modelle in der Literaturarbeit [Mey15] behandelt und können dort detaillierter nachgelesen werden. An dieser Stelle soll nur eine Kurzfassung aller Modelle mit einem Blick auf die gängigsten Operationen auf Hierarchien, wie das Suchen/Finden, Einfügen und Löschen erfolgen.

3.2.1 Adjazenz-Liste

Dies ist die typische Speicherung mittels Selbstreferenz. Es erfolgt ein Verweis auf den Vorgänger (*id*) durch eine ausgezeichnete Spalte (*parent_id*) in derselben Relation. Die referentielle Integrität ist in diesem Modell gegeben.

id	parent_id	name	rank
1	NULL	Bivalvia	Klasse
2	1	Veneroida	Ordnung
5	2	Semelidae	Familie
6	5	Abra	Gattung
8	6	Abra alba	Art

Tabelle 3.1: Ausschnitt: Speicherung mittels Adjazenz-Liste

Das Finden aller Nachfolger oder Vorgänger bzw. eines Teilbaums oder das Löschen von inneren Knoten sind komplexere Operationen im Vergleich zu den im Folgenden beschriebenen Modellen. Da nicht alle Datenbank-Produkte rekursive Anfragen beherrschen, muss auf die prozedurale Programmierung zurückgegriffen werden. Dies kann entweder mittels SQL, z.B. durch Stored Procedures, oder in der Geschäftsanwendung selbst, realisiert werden. Wird hingegen die Rekursion im Datenbank-Produkt unterstützt, kann durch Verwendung dieser Technik, die Komplexität der Operationen verringert werden.
Operationen wie das Finden von direkten Nachfolgern oder Vorgängern oder das Einfügen von inneren- und Blatt-Knoten können in diesem Modell effizient umgesetzt werden.

3.2.2 Pfad-Aufzählung / Path-Enumeration

Eine ausgezeichnete Spalte (*path*) speichert den Pfad von der Wurzel bis zum jeweiligen Knoten als String-Repräsentation. Der Pfad setzt sich aus den Primärschlüsseln und Trennzeichen zusammen. Die referentielle Integrität ist in diesem Modell nicht gegeben.

id	path	name	rank
1	1/	Bivalvia	Klasse
2	1/2/	Veneroida	Ordnung
5	1/2/5/	Semelidae	Familie
6	1/2/5/6/	Abra	Gattung
8	1/2/5/6/8/	Abra alba	Art

Tabelle 3.2: Ausschnitt: Speicherung mittels Path-Enumeration

Da alle Operationen auf der Manipulation von Zeichenketten basieren, sind diese effektiv und besitzen eine geringere Komplexität gegenüber den anderen vorgestellten Modellen. Zu beachten ist aber, dass LIKE-Klauseln nicht mit einem Wildcard beginnen sollten, da ein sonst vorhandener Index nicht ausgenutzt werden kann. Weiterhin muss die referentielle Integrität durch komplexe CHECK-Constraints sichergestellt oder in der Geschäftsanwendung selbst gepflegt werden [Cel04].

3.2.3 Verschachtelte Mengen / Nested-Sets

Dieses Modell verdankt seinen Namen der Tatsache, dass jede Hierarchie durch ein Mengendiagramm dargestellt werden kann, welches aus verschachtelten Mengen besteht. Abbildung 3.2 verdeutlicht dies anhand der Beispiel-Hierarchie A.1.1 aus der Literaturarbeit [Mey15].

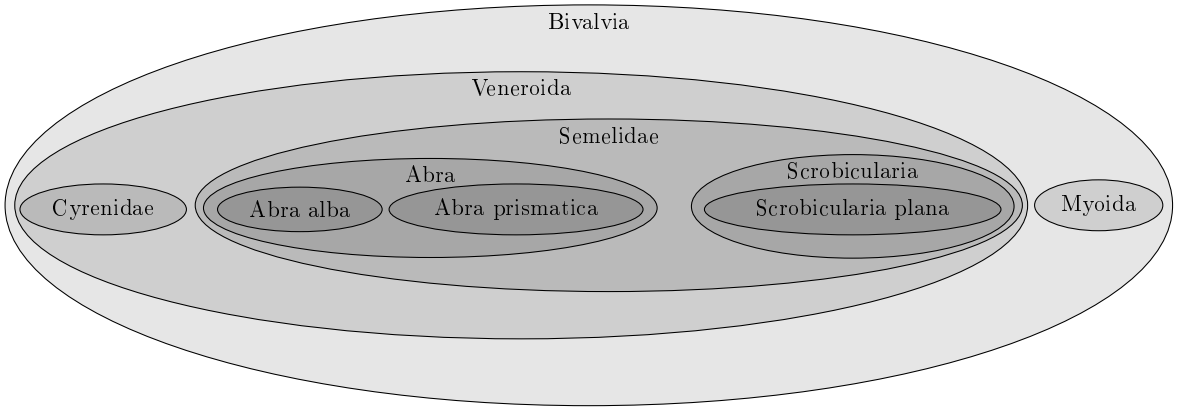

Abbildung 3.2: Darstellung als verschachtelte Mengen

In der Relation 3.3 gibt es zwei ausgezeichnete Spalten (*lft* und *rgt*), welche in Form von Nummern die Informationen über die Menge der Nachfolger für jeden Knoten speichern [Kar10b]. Jeder Knoten ist somit eine Teilmenge mit den Grenzen *lft* und *rgt*. Diese beiden Nummern werden durch eine modifizierte Tiefensuche ermittelt, welche im Anhang A.1.2 anhand der Hierarchie

veranschaulicht ist.

id	lft	rgt	name	rank
1	1	20	Bivalvia	Klasse
2	2	17	Veneroida	Ordnung
3	18	19	Myoida	Ordnung
4	3	4	Cyrenidae	Familie
5	5	16	Semelidae	Familie
6	6	11	Abra	Gattung
7	12	15	Scrobicularia	Gattung
8	7	8	Abra alba	Art
9	9	10	Abra prismatica	Art
10	13	14	Scrobicularia plana	Art

Tabelle 3.3: Speicherung im Nested-Sets-Modell

Einige besondere Eigenschaften [Cel11] erleichtern die Operationen auf die Hierarchie. So gilt, dass die linken und rechten Werte eines Eltern-Knotens immer die Nummern der Kind-Knoten überdecken. Weiterhin gilt für Blatt-Knoten, dass der rechte Wert immer um eins größer als der linke Wert ist. Das Suchen von Nachfolgern und Vorgängern, sowohl die Direkten als auch als Menge eines Teilbaums, ist dadurch sehr effizient. Erkauft wird die günstige Navigation durch teure Einfüge- und Lösch-Operationen, da die Werte der *lft*- und *rgt*-Attribute neuberechnet werden müssen.

3.2.4 Closure-Table

Bei dieser Technik werden die Hierarchie-Informationen in einer separaten Relation gespeichert. Dabei werden alle Pfade von einem Knoten zu all seinen Nachfolgern erfasst. Die Relation in Tabelle 3.5 enthält die Attribute *ancestor* und *descendant*, welche die Knoten in der Ausgangstabelle 3.2.4 referenzieren. Optional, aber sehr von Vorteil, ist das Attribut *path_length*, welches ausgehend von einem Knoten, die Tiefe seiner Kind-Knoten speichert. Die referentielle Integrität ist in diesem Modell gegeben.

Alle gängigen Operationen sind mit diesem Modell sehr effizient. Die verringerte Berechnungskomplexität wird hier durch erhöhten Speicherverbrauch erkauft.

id	name	rank
1	Bivalvia	Klasse
2	Veneroida	Ordnung
5	Semelidae	Familie
6	Abra	Gattung
8	Abra alba	Art

Tabelle 3.4: Speicherung mittels Closure-Table: Taxon Basis-Tabelle

ancestor	descendant	path_length
1	1	0
1	2	1
1	5	2
1	6	3
1	8	4
2	2	0
2	5	1
2	6	2
2	8	3
5	5	0
5	6	1
5	8	2
6	6	0
6	8	1
8	8	0

Tabelle 3.5: Speicherung mittels Closure-Table: Taxon Pfad-Tabelle

3.2.5 Zusammenfassung

Tabelle 3.6 aus [Mey15] soll einen Überblick über die Stärken und Schwächen der jeweiligen Operationen der Modelle liefern. Klassifiziert wurde anhand der Art, Anzahl und der Verwendung von Funktionen in den Operationen.

	Adjazenz-Liste	Rekursion[1]	Path-Enumeration	Nested-Sets	Closure-Table
Nachfolger finden (Teilbaum)	--	+	++	++	++
Direkte Nachfolger finden	++	++	++/+	+	++
Vorgänger finden	--	+	++	++	++
Direkten Vorgänger finden	++	++	++/+	+	++
Blatt einfügen	++	++	++	--	++
Inneren Knoten einfügen	++	++	++	--	++/+
Blatt löschen	++	++	++	-	++
Inneren Knoten löschen	--/-	+	++	--/-	++/+
Referentielle Integrität	Ja	Ja	Nein	Nein	Ja
Tabellen	1	1	1	1	2

Tabelle 3.6: Vergleich der Operationen der Modelle

[1] als Besonderheit der Adjazenz-Liste

Legende:

++ sehr effizient + effizient - ineffizient -- sehr ineffizient

Die Entscheidung für ein Modell muss immer unter dem Gesichtspunkt der Aufgabenstellung, der Anforderungen sowie anhand der Häufigkeit der erwarteten Operationen fallen. Darüber hinaus können einige Modelle miteinander kombiniert werden und es entstehen hybride Modelle, wie die Adjazenz-Liste und das Nested-Sets-Modell oder die Adjazenz-Liste und das Path-Enumeration-Modell.

3.3 Temporale Daten in Datenbanken

Dieses Kapitel beschäftigt sich mit der Speicherung von zeitbezogenen Daten in relationalen Datenbanken. In Abschnitt 3.3.1 werden zunächst einige Begriffe erklärt. Anhand von Beispielen werden die aus den Begriffen entstehenden Tabellen behandelt und die zugehörigen SQL-Anweisungen erklärt. Darauf aufbauend wird bei jedem Beispiel auf die Schlüssel und die referentielle Integrität eingegangen. Es wird versucht, sich am SQL:2011-Standard zu orientieren, aber auch kurz darauf eingegangen, wenn dieser nicht zur Verfügung steht.

Soll ein Verlauf sowie Änderungen von Daten über die Zeit erfasst werden, so sind zwei neue Attribute, welche einen temporalen Datentyp, wie z.B. TIME, DATE oder TIMESTAMP, besitzen, in der Relation mitzuführen. Diese Attribute erzeugen ein Zeitintervall mittels einem Start- und einem Endzeitpunkt. Solch eine *Historisierung*, auch temporale Datenhaltung genannt, muss sicherstellen, dass keine Primärschlüssel-Verletzungen auftreten sowie, falls gefordert, es keine Überlappung verschiedener Zeiträume des zu historisierenden Objekts gibt. Alle DML- und DDL-Anweisungen müssen diese temporalen Abhängigkeiten berücksichtigen. Es werden zum Beispiel keine Datensätze in dem Sinne gelöscht, sondern es erfolgt eine Markierung mittels der temporalen Attribute derart, dass ein „gelöschter“ Datensatz nicht mehr in aktuell gültigen Daten auftaucht. Wie aktuell gültige Daten ermittelt werden, ist in Abschnitt 3.3.2 beschrieben.

Das Abbilden von solchen zeitlichen Zusammenhängen zwischen Datensätzen ist ein aktuell beachtetes Forschungsgebiet. Temporale Datenbanken, und damit die Einführung neuer Datentypen und Prädikate zur Unterstützung von Zeitdimensionen, sowie die Entwicklung von Anfragesprachen für temporale relationale Daten, werden dabei schon seit einigen Jahrzehnten erforscht [SSH13].

Mit SQL:2011 werden einige grundlegenden Aspekte der Speicherung zeitbezogener Daten eingeführt, die über die einfache Speicherung von Zeitpunkten und Zeitdimensionen hinausgehen. So werden Zeitdimensionen automatisiert verwaltet, Integritätsbedingungen berücksichtigt und die Abfragesprache SQL um temporale Konzepte erweitert [SSH13]. Dabei implementiert SQL:2011 nur einen rudimentären Teil der Forschungsergebnisse aus [Sno95] und den temporalen Datenbanken, da die Einführung, z.B. eines komplett neuen Datentyps (Period Data Type), einen tiefen Eingriff in alle bestehenden Datenbanksysteme bedeutet hätte [SSH13].

3.3.1 Grundbegriffe

Zeit-Intervall

Die hier verwendeten Zeit-Intervalle sind immer *closed-open*, d.h. der Start-Zeitpunkt liegt immer im Intervall, der End-Zeitpunkt dagegen nicht mehr [SGM00, SSH13]. Dies erspart das „mühselige“ mitführen der „+1“-Operation oder des „<=“-Operators bei DML-Operationen [SGM00]. Deshalb ist dies die bevorzugte Darstellung der Intervalle [SGM00] und wird auch so in SQL:2011 übernommen.

Es gibt drei elementare Zeitbegriffe. Diese sind orthogonal zueinander: es kann keiner, einer oder alle drei in ein und derselben Relation verwenden [SGM00].

Benutzerdefinierte Zeit (user-defined time)

Ein beliebiger temporaler Wert, der vom Benutzer festgelegt werden kann, z.B. ein Geburtsdatum. Dies ist aber kein temporaler Wert im Sinne der Historisierung, da es meist ein fixer Zeitpunkt ist und dieser für den Datensatz „für immer“ gilt. Die Anwesenheit eines temporalen Attributs heißt nicht gleich automatisch, dass diese Tabelle eine temporale Tabelle ist. [SGM00]

Gültigkeitszeit (valid time)

Diese Zeit definiert die anwendungsbezogene Gültigkeit von Tupeln in der Datenbank [SSH13]. Sie beschreibt die Gültigkeit von Fakten aus der modellierten Realität [SGM00]. Es werden zwei Attribute für die Start- und Endzeit in der Relation angelegt. Diese werden mit Werten aus der Anwendung heraus besetzt. Es muss lediglich die Bedingung gelten, dass die Endzeit nach der Startzeit liegt [SSH13].

Die Gültigkeitszeit beantwortet also die Frage, wann welcher Datensatz gültig war bzw. noch ist.

Transaktionszeit (transaction time)

Diese vom Datenbanksystem vergebene Zeit speichert, wann ein Datensatz in der Relation anwesend ist bzw. war [SGM00]. Sie definiert systemintern den Zeitpunkt, an dem das jeweilige Tupel in die Datenbank eingetragen (Startzeit) bzw. wieder logisch entfernt [SSH13] oder logisch verändert wurde [SGM00] (Endzeit).

Die Transaktionszeit beantwortet somit die Frage, welche Informationen in der Datenbank an einem Zeitpunkt vorhanden waren bzw. noch sind.

Bitemporale Relation

Wird sowohl die Gültigkeitszeit als auch Transaktionszeit zusammen in einer Relation verwendet, so wird diese auch bitemporale Relation genannt [SSH13, SGM00].

Anhand von Beispielen erfolgt in den weiteren Abschnitten die Vorstellung der drei möglichen temporalen Tabellenarten. Es wird gezeigt, wie die Relationen aussehen und wie diese erzeugt werden. Zuerst ohne auf den Primärschlüssel einzugehen, später wird dann aufgezeigt, wie ein neuer Primärschlüssel gefunden werden kann. Zum Schluss wird auf temporale Abfragen teils in SQL:2011 Syntax und teils ohne eingegangen.

3.3.2 Aktueller Zeitpunkt

Den aktuellen Zustand einer temporalen Relation in SQL:2011 ohne die Angabe von jeglichen zusätzlichen Zeitinformationen zu ermitteln, ist mit Aufwand verbunden [SSH13]. In [SGM00] werden drei Ansätze beschrieben, wie der gerade aktuelle Zeitpunkt dargestellt werden kann.
Als erste Methode könnte das kleinste darzustellende Datum des DBMS für das Endzeit-Attribut verwendet werden. Ein Vorteil davon wäre, dass aktuelle Datensätze sortiert nach dem Enddatum ganz oben erscheinen. Nachteilig ist aber, dass die gesamte Anwendung diesen speziellen Wert behandeln muss. Die WHERE-Klausel mit dem Prädikat des kleinsten dazustellenden Datums um damit die aktuellen Daten zu ermitteln, sieht in MySQL Syntax folgendermaßen aus:

```
WHERE vt_end = DATE '1001-01-01'
```

Jeder Benutzer der Datenbank muss über diesen speziellen Wert frühzeitig aufgeklärt werden, damit es nicht zu Missverständnissen kommt.

Ein anderer Ansatz wäre, den Wert NULL zu benutzen:

```
WHERE vt_end IS NULL
```

Nachteile sind:

- Irritation der Nutzer, wenn diese mit einem NULL-Wert in einem DATE-Attribut konfrontiert werden.
- SQL liefert bei Vergleichen mit NULL immer FALSE zurück.
- Weitere Verwendungen von NULL sind nicht mehr möglich, da nun z.B. nicht mehr zwischen einem nicht vorhandenen Enddatum oder einem Enddatum, das dem „jetzt“ entspricht, unterschieden werden kann.

Der am weitesten verbreitete Ansatz ist die Verwendung des größtmöglichen Datums des DBMS für das Endzeit-Attribut. Dies erlaubt das aktuelle Datum in der WHERE-Klausel zu nutzen, um die aktuellen Datensätze zu bekommen (SQL:2011):

```
WHERE ValidTime CONTAINS CURRENT_DATE
```

Oder ohne SQL:2011 Syntax:

```
WHERE vt_begin <= CURRENT_DATE
AND vt_end > CURRENT_DATE
```

Dieser Ansatz wird so auch in anderer Literatur, z.B. [KM12], und in dieser Arbeit benutzt. Ein kleiner Überblick über die minimalen und maximalen unterstützten DATE Werte in den gängigsten DBMS ist in Tabelle 3.7 zu finden[1].

	von	**bis**
MySQL	1001-01-01	9999-12-31
PostgreSQL	4713 BC	5874897 AD
Oracle	4712-01-01 BC	4712-12-31 AD
DB2	0001-01-01	9999-12-31

Tabelle 3.7: Minimale und maximale Datumswerte

3.3.3 Tabellen mit Gültigkeitszeit

Eine temporale Tabelle mit zwei ausgezeichneten Spalten (*vt_begin*, *vt_end*) vom Typ DATE, die einen Gültigkeitszeitraum darstellen, könnte folgendermaßen aussehen:

id	**name**	**vt_begin**	**vt_end**
4711	Scrobicularia alba	2000-01-03	2004-12-31
4711	Abra alba	2004-12-31	9999-12-31

Tabelle 3.8: Tabelle mit Gültigkeitszeit

Hier wird eine Namensänderung dargestellt. Anstelle einer einfachen UPDATE-Anweisung, welche den ersten Datensatz ändert, wird dieser auf „ungültig“ gesetzt und ein neuer Datensatz eingefügt. Der erste Datensatz mit alten Namen war demnach gültig vom 03.01.2000 bis 30.12.2004. Zu beachten ist, dass das Zeitintervall *closed-open* ist. Das bedeutet, dass der Endzeitpunkt nicht mehr im Zeitintervall enthalten ist. Danach folgte die Namensänderung, welche ab 31.12.2004 bis „heute“ gültig ist. Das Datum 31.12.9999 symbolisiert hier den aktuell gültigen Datensatz in der Relation. Warum dieses Datum gewählt wurde, kann in Abschnitt 3.3.2 nachgelesen werden. Das

[1] Ermittelt aus den jeweiligen gültigen Dokumentationen zum Erstellzeitpunkt dieser Arbeit.

dazugehörige CREATE-Statement sieht in SQL:2011 Syntax folgendermaßen aus:

```
CREATE TABLE taxa_names (
  id INT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  vt_begin DATE
  vt_end DATE
  PERIOD FOR ValidTime (vt_begin, vt_end)
)
```

SQL:2011 führt eine PERIOD-Klausel in den CREATE TABLE- und ALTER TABLE-Anweisungen ein. In dieser werden das Zeitintervall für die Gültigkeitszeit benannt und die Attribute für die Start- und Endzeit eingetragen. [SSH13]

War zu Beginn der Tabelle ohne temporale Attribute der künstliche Primärschlüssel *id* ausreichend, so ist er es nun nicht mehr, da dieser mehrfach auftreten kann. Um einen gültigen Primärschlüssel zu definieren, werden die beiden Attribute Start- und Endzeit mit in den ursprünglichen Primärschlüssel genommen [SSH13]:

```
CREATE TABLE taxa_names (
  id INT,
  name VARCHAR(100) NOT NULL,
  vt_begin DATE
  vt_end DATE
  PERIOD FOR ValidTime (vt_begin, vt_end),
  PRIMARY KEY (id, vt_begin, vt_end)
)
```

Hierbei ergibt sich aber das Problem, dass es zu überlappenden Zeiträumen kommen kann. So könnten z.B. zwei Namen gleichzeitig gültig sein, was prinzipiell nicht erwünscht ist. Es kann nur ein Fakt zu genau einer Zeit gültig sein. SQL:2011 bietet hier die WITHOUT OVERLAPS-Klausel an, die spezifiziert, dass die Zeitintervalle sich nicht überlappen dürfen [SSH13]. Dazu werden nicht mehr die Attribute *vt_ begin* und *vt_ end* sondern die PERIOD *ValidTime* selber in den Primärschlüssel aufgenommen:

```
CREATE TABLE taxa_names (
  id INT,
  name VARCHAR(100) NOT NULL,
  vt_begin DATE
  vt_end DATE
  PERIOD FOR ValidTime (vt_begin, vt_end),
  PRIMARY KEY (id, ValidTime WITHOUT OVERLAPS)
)
```

Steht dagegen SQL:2011 nicht zur Verfügung, so müssen diese Konsistenzbedingungen per Hand behandelt werden. In [SGM00] wird gezeigt, wie so ein sequenzieller Primärschlüssel mittels einer ASSERTION umgesetzt werden kann. Ein Primärschlüssel kann somit ohne SQL:2011 Unterstützung durch eine ASSERTION oder durch ein Tabellen-CONSTRAINT ausgedrückt werden [SGM00].

Nachdem ein gültiger Primärschlüssel gefunden wurde, müssen auch die Besonderheiten bei Fremdschlüsseln betrachtet werden. Zur Gewährleistung der referentiellen Integrität wird in der FOREIGN KEY-Klausel die Angabe der PERIOD-Klausel erlaubt. So wird sichergestellt, dass das Zeitintervall der referenzierenden Relation vollkommen in der Zeitperiode eines oder mehrerer Tupel der referenzierten Relation enthalten ist [Pet13]. Dies bedeutet, dass für einen gegebenen Kind-Datensatz es nicht zwingend genau einen übereinstimmenden Vater-Datensatz geben muss, der das Zeitintervall des Kindes überdeckt. Solange das Zeitintervall des Kind-Datensatzes mittels Vereinigung von zwei oder mehrere zusammenhängenden übereinstimmenden Zeitintervallen aus der Vater-Tabelle überdeckt wird, ist die referentielle Integritätsbedingung erfüllt [KM12].

Angenommen es gibt eine weitere Tabelle „taxa_details“, welche zusätzliche Informationen zu einem Taxon aufnehmen kann:

```
CREATE TABLE taxa_details (
  id INT,
  detail1 VARCHAR(200),
  detail2 VARCHAR(200),
  detail3 VARCHAR(200),
  vt_begin DATE
  vt_end DATE
  PERIOD FOR TaxonDetails_ValidTime (vt_begin, vt_end),
  PRIMARY KEY (id, TaxonDetails_ValidTime WITHOUT OVERLAPS)
)
```

Durch das Hinzufügen des Primärschlüssels aus der Details-Tabelle und der Angabe eines Fremdschlüssels, sieht die angepasst CREATE-Anweisung für die ursprüngliche Relation folgendermaßen aus:

```
CREATE TABLE taxa_names (
  id INT,
  name VARCHAR(100) NOT NULL,
  taxa_details_id INT,
  vt_begin DATE
  vt_end DATE
  PERIOD FOR ValidTime (vt_begin, vt_end),
  PRIMARY KEY (id, ValidTime WITHOUT OVERLAPS),
  FOREIGN KEY (taxa_details_id, PERIOD ValidTime)
    REFERENCES taxa_details (id, PERIOD TaxonDetails_ValidTime)
)
```

Somit verweist ein Detail-Datensatz auf ein Taxon, welches im passenden Zeitraum vorhanden sein muss.

Das Einfügen eines neuen Datensatzes in die „taxa_names“-Relation geschieht weiterhin wie gewohnt mit der INSERT-Anweisung. Taucht keine Primärschlüsselverletzung durch ein überlappendes Zeitintervall auf, wird der neue Datensatz in die Relation eingefügt. In SQL:2011 wurden die UPDATE- und DELETE-Anweisungen um die FOR PORTION OF-Klausel erweitert [KM12]. Dies erlaubt effektiv Daten innerhalb von Zeitintervallen zu ändern. Wurde z.B. zwischenzeitlich im Zeitraum vom 09.06.2002 bis 31.10.2003 ein anderer Name „Syndesmya alba“ verwendet, kann dies mit folgender UPDATE-Anweisung in die Relation eingetragen werden:

```
UPDATE taxa_names
  FOR PORTION OF ValidTime (
    FROM DATE '2002-06-09'
    TO DATE '2003-11-01'
  )
  SET name = 'Syndesmya alba'
  WHERE id = 4711
```

Die Ergebnisse der Aktualisierung sind in Tabelle 3.9 veranschaulicht.

id	name	vt_begin	vt_end
4711	Scrobicularia alba	2000-01-03	2002-06-09
4711	Syndesmya alba	2002-06-09	2003-11-01
4711	Scrobicularia alba	2003-11-01	2004-12-31
4711	Abra alba	2004-12-31	9999-12-31

Tabelle 3.9: Tabelle mit Gültigkeitszeit nach UPDATE-Anweisung

Zu erkennen ist, dass der Zeitraum des ersten Datensatzes unterbrochen und der neue Datensatz im spezifizierten Zeitraum eingefügt wurde.

Das Löschen mit der neuen Klausel geschieht äquivalent. Wird innerhalb eines Zeitintervalls ein Datensatz entfernt, so wird dieses Zeitintervall dementsprechend in zwei Intervalle aufgeteilt und es entsteht eine temporale Lücke.

Ohne SQL:2011 Unterstützung müssen die Transformationen der DML-Operationen manuell durchgeführt und könnten in einer Stored Procedure gekapselt werden. Wie diese Transformationen in einzelne SQL-Anweisungen bewerkstelligt werden können, wird in [SGM00] ausführlich beschrieben.

Weiterhin stellt SQL:2011 temporale Prädikate zum Abfragen über Zeitintervalle in einem SFW-

Block bereit: CONTAINS, OVERLAPS, EQUALS, PRECEDES, SUCCEEDS, IMMEDIATELY RECEDES und IMMEDIATELY SUCCEEDS [KM12, SSH13]. Diese sind in der WHERE-Klausel anwendbar und basieren auf den Allen-Relationen für Zeitintervalle [All83], setzen diese aber nicht 1:1 um [SSH13].

Das folgende Beispiel ermittelt, welcher Name am 09.06.2001 gültig war:

```
SELECT name
FROM taxa_names
WHERE id = 4711
AND ValidTime CONTAINS DATE '2001-06-09'
```

Als Ergebnis würde hier „Scrobicularia alba“ geliefert werden. Ohne die temporalen Prädikate von SQL:2011 ist eine solche Anfrage auszuformulieren:

```
SELECT name
FROM taxa_names
WHERE id = 4711
AND vt_begin <= DATE '2001-06-09'
AND vt_end > DATE '2001-06-09'
```

Weitere Beispiele und Abfragen ohne SQL:2011 Unterstützung sind in [SGM00] zu finden.

Normalerweise enthalten Relationen immer den aktuellen Datenbestand. Die Abfrage nach Tupeln liefert also immer einen aktuellen Stand der Daten. In einer temporalen Relation dagegen muss die Anfrage nach den aktuell gültigen Daten um ein Prädikat erweitert werden [SGM00]. Somit sieht die Abfrage nach aktuell gültigen Daten in dem Beispiel so aus[1]:

```
SELECT name
FROM taxon_names
WHERE id = 4711
AND ValidTime CONTAINS CURRENT_DATE
```

Unter Annahme, dass CURRENT_DATE den Wert 01.01.2015 besitzt, liefert diese Abfrage das Tupel mit dem Namen „Abra alba“ zurück.

[1] IBM hat hier, ähnlich wie bei Transaktionszeit-Tabellen, eine Klausel für die Gültigkeitszeit eingeführt. Das SQL-Statement in IBM DB2 Syntax kann im Anhang unter A.2.1 angesehen werden.

3.3.4 Tabellen mit Transaktionszeit

Tabellen mit vom System gesteuerten Zeitangaben sind dafür gedacht, einen exakten Verlauf der Datenänderungen zu verwalten [KM12]. Eine solche Tabelle besitzt zwei temporale Attribute (*tt_start*, *tt_stop*) und könnte folgendermaßen aussehen:

id	name	tt_start	tt_stop
4711	Scrobicularia alba	2000-01-03 09:00:00	2002-06-09 11:00:00
4711	Syndesmya alba	2002-06-09 11:00:00	9999-12-31 23:59:59

Tabelle 3.10: Tabelle mit Transaktionszeit

Diese Relation hat sehr viel Ähnlichkeit mit der Relation aus dem Abschnitt 3.8 Gültigkeitszeit. Vom System verwaltete temporale Tabellen müssen aber einigen wichtigen Anforderungen genügen [KM12]:

- Jede UPDATE- oder DELETE-Anweisung muss automatisch den alten Zustand der betroffenen Tupel erhalten, bevor diese Anweisungen durchgeführt werden.
- Das Datenbanksystem ist alleine dafür zuständig, die Start- und Endzeiten der Zeitintervalle der Tupel zu verwalten. D.h. insbesondere, dass der Benutzer keine Möglichkeit haben soll, bereits in der Vergangenheit liegende Tupel zu ändern.

Letzteres garantiert die Integrität der aufgezeichneten historischen Daten im Sinne der Datensicherheit.

Das CREATE-Statement für Tabelle 3.10 sieht in SQL:2011 Syntax folgendermaßen aus:

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

Zu beachten ist, dass hier eine Spalte vom Typ TIMESTAMP mit höchster Präzision gewählt wurde. Diese Präzision wird aber, in den in dieser Arbeit abgebildeten Relationen, nicht dargestellt.

Jede Tabelle, welche als Zeitintervall-Definition den Standard-Namen SYSTEM_TIME und die Schlüsselwörter WITH SYSTEM VERSIONING besitzt, ist eine Transaktionszeit-Tabelle

[KM12]. In solchen Tabellen ist ein aktuell gültiger Datensatz ein genau solcher, dessen Systemzeit-Intervall den aktuellen Zeitpunkt enthält; alle anderen Tupel gelten damit als historisierte System Daten [KM12]. Dieser Punkt ist wichtig für das Finden des Primärschlüssels. Folgende Informationen sind in [KM12] beschrieben und gelten für SQL:2011. Die Primärschlüssel-Bedingungen müssen nur auf aktuelle Datensätze angewendet werden. Jegliche Bedingungen, die in Kraft traten, als ein historisiertes Tupel erzeugt wurde, wurden auch schon überprüft, als dieser Datensatz noch eine aktueller war. Somit besteht keine Notwendigkeit diese Primärschlüssel-Bedingungen auf historisierte Datensätze anzuwenden. Demnach kann der Primärschlüssel einfach aus dem *id*-Attribut bestehen bleiben, ohne weitere Attribute, wie die Start- und Endzeit, mit in diesen aufzunehmen. Gleiches geschieht mit Fremdschlüssel zur Wahrung der referentiellen Integrität.

Ist keine SQL:2011 Unterstützung gegeben, so lässt sich der Primärschlüssel als zusammengesetzter Schlüssel aus den Attributen *id* und *tt_ stop*, also der Endzeit, erzeugen [SGM00].

Unter Fortführung des Beispiel, der Änderung des Namens, muss ein UPDATE ausgeführt werden (SQL:2011):

```
UPDATE taxa_names
SET name = 'Abra alba'
WHERE id = 4711
```

Angenommen die Änderung erfolgt zur Transaktionszeit 01.11.2003 um 08:00 Uhr, so entsteht folgende Tabelle:

id	name	tt_start	tt_stop
4711	Scrobicularia alba	2000-01-03 09:00:00	2002-06-09 11:00:00
4711	Syndesmya alba	2002-06-09 11:00:00	2003-11-01 08:00:00
4711	Abra alba	2003-11-01 08:00:00	9999-12-31 23:59:59

Tabelle 3.11: Tabelle mit Transaktionszeit nach UPDATE-Anweisung

Somit wurde die zweite Reihe historisiert und die aktuelle gültige Reihe ist die neu eingefügte.

Wiederum ohne SQL:2011 muss diese Änderung, welche aus einem UPDATE und einem INSERT besteht, manuell, z.B. mittels Trigger, behandelt werden. Drei solche Trigger für die Operationen INSERT, UPDATE und DELETE sind in [SGM00] zu finden. In dem Buch wird allerdings mit einer zusätzlichen Tabelle gearbeitet, welche nur historische Daten enthält. Somit wird die ursprüngliche Tabelle, welche nur aktuelle Daten enthält, überwacht.

SQL:2011 stellt für Abfragen auf Transaktionszeit-Tabellen neue syntaktische Erweiterungen bereit [KM12]. Sollen die Daten von einem bestimmten Zeitpunkt ermittelt werden, so kann die

FOR SYSTEM_TIME AS OF-Klausel verwendet werden:

```
SELECT name
FROM taxa_names
FOR SYSTEM_TIME AS OF TIMESTAMP '2002-02-05 14:00:00'
```

Diese Abfrage liefert alle Tupel zurück, deren Startzeit kleiner gleich dem angegeben Zeitpunkt und deren Endzeit größer dem angegeben Zeitpunkt ist [KM12]. Somit ist diese Abfrage eine Verkürzung für folgende Abfrage, welche zu nutzen ist, wenn kein SQL:2011 zur Verfügung steht [SGM00]:

```
SELECT name
FROM taxa_names
WHERE tt_start <= TIMESTAMP '2002-02-05 14:00:00'
AND tt_stop > TIMESTAMP '2002-02-05 14:00:00'
```

Die beiden weiteren Erweiterungen ermitteln Tupel, welche zwischen zwei Zeitpunkten liegen [KM12]:

```
SELECT name
FROM taxa_names
FOR SYSTEM_TIME
  FROM TIMESTAMP '2002-02-05 14:00:00'
  TO TIMESTAMP '2002-04-05 16:00:00'
```

Diese Abfrage benutzt ein *closed-open* Zeitintervall. Im Gegensatz dazu liefert folgende Abfrage die Daten in einem *closed-closed* Zeitintervall [KM12]:

```
SELECT name
FROM taxa_names
  FOR SYSTEM_TIME
    BETWEEN TIMESTAMP '2002-02-05 14:00:00'
    AND TIMESTAMP '2002-04-05 16:00:00'
```

Um nun aktuelle Daten in Transaktionszeit-Tabellen zu ermitteln, ist kein besonderer Aufwand notwendig. Es kann anhand des Primärschlüssels das aktuelle Tupel ermittelt werden. Zurückgeliefert wird aber nur jenes Tupel, was nicht in der Vergangenheit liegt, also dessen Systemzeitintervall das aktuelle Datum enthält. Wird nämlich keine der drei genannten Erweiterungen angegeben, so entspricht dies der FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP-Klausel und es werden eben jene aktuellen Daten zurückgegeben [KM12].

In [SGM00] sind weitere Abfragen beschrieben, die genutzt werden können, wenn SQL:2011 nicht zur Verfügung steht.

3.3.5 Bitemporale Tabellen

Diese Relationen besitzen sowohl Attribute für die Gültigkeitszeit (*vt_begin*, *vt_end*) als auch für die Transaktionszeit (*tt_start*, *tt_end*). SQL:2011 selbst spezifiziert keinen eigenen Begriff, aber in der Literatur und in einigen Produkten werden diese Tabellen bitemporale Relationen genannt [KM12]. Eine bitemporale Tabelle könnte wie folgt aussehen:

id	name	vt_begin	vt_end	tt_start	tt_stop
4711	Scrobicularia alba	2000-01-03	9999-12-31	2000-01-03 09:00:00	9999-12-31 23:59:59

Tabelle 3.12: Bitemporale Tabelle

Das CREATE-Statement in SQL:2011 Syntax:

```
CREATE TABLE taxa_names (
  id INT,
  name VARCHAR(100),
  vt_begin DATE
  vt_end DATE,
  tt_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
  tt_stop TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
  PERIOD FOR ValidTime (vt_begin, vt_end),
  PERIOD FOR SYSTEM_TIME (tt_start, tt_end),
  PRIMARY KEY (id, ValidTime WITHOUT OVERLAPS)
) WITH SYSTEM VERSIONING
```

Der Primärschlüssel wird hier von dem Gültigkeitszeitintervall, so wie bei der Gültigkeitszeit-Tabelle aus Abschnitt 3.3.3, erzeugt [KM12]. Ohne SQL:2011 kann kein eindeutiger Primärschlüssel gefunden werden und es muss wieder, ähnlich wie bei der Gültigkeitszeit-Tabelle, auf eine ASSERTION zurückgegriffen werden [SGM00].

Da bitemporale Relationen die Eigenschaften von Gültigkeitszeit- und Transaktionszeit-Tabellen vereinen, können auch dieselben Erweiterungen in der Abfragesprache genutzt werden. So können sowohl das einfache UPDATE oder DELETE als auch jene mit der Erweiterung FOR PORTION OF verwendet werden [KM12]. Bei Betrachtung der Transaktionszeit ist zu erkennen, dass für jedes aktuelle Tupel, welches verändert (aktualisiert oder gelöscht) wird, automatisch eine versionierte Zeile erzeugt wird [KM12].

Das Beispiel der Namensänderung wird hier wieder aufgegriffen, wobei die Transaktionszeit 01.11.2003 um 08:00 Uhr sei. Die Aktualisierung erfolgt mit folgender Anweisung:

```
UPDATE taxa_names
FOR PORTION OF ValidTime FROM CURRENT_DATE TO DATE '9999-12-31'
SET name = 'Abra alba'
WHERE id = 4711
```

Die Variable *CURRENT_DATE* liefert hierbei nur das aktuelle Datum ohne Uhrzeit, welches dem Datum der Transaktionszeit entspricht. Es entsteht folgende Tabelle:

id	name	vt_begin	vt_end	tt_start	tt_stop
4711	Scrobicularia alba	2000-01-03	9999-12-31	2000-01-03 09:00:00	2003-11-01 08:00:00
4711	Scrobicularia alba	2000-01-03	2003-11-01	2003-11-01 08:00:00	9999-12-31 23:59:59
4711	Abra alba	2003-11-01	9999-12-31	2003-11-01 08:00:00	9999-12-31 23:59:59

Tabelle 3.13: Bitemporale Tabelle nach UPDATE-Anweisung

Zu erkennen ist, dass die erste Zeile versioniert wurde, da die Transaktionszeit besagt, dass diese Zeile ab dem 01.11.2003 um 08:00 Uhr als Information als solche nicht mehr in der Tabelle vorhanden ist. Es wurde die zweite Zeile eingefügt, welche zwar laut Transaktionszeit in der Tabelle präsent ist, aber anhand der Gültigkeitszeit ist zu sehen, dass dieser Datensatz nicht mehr gültig ist. Die dritte Zeile ist dann die eigentliche Änderung: sie ist gültig und laut der Transaktionszeit auch der aktuelle Datensatz für die Tupel mit der *id=4711* in dieser Relation.
Bei der Betrachtung des Gültigkeitszeitintervall fällt auf, dass dieser die Primärschlüsselbedingung verletzt. Es kommt zu Überlappungen aller drei Datensätze. Da die Relation aber auch eine mit System-Versionierung ist, gilt die Primärschlüsselbedingung nur für aktuelle Datensätze, welche hier die beiden letzten Reihen sind. Dort ist die Primärschlüsselbedingung intakt. Um nun den aktuell gültigen Datensatz aus dieser Relation zu ermitteln, muss wieder die WHERE-Klausel, wie bei der reinen Gültigkeitszeit-Tabelle, erweitert werden:

```
SELECT *
FROM taxa_names
WHERE id = 4711
AND ValidTime CONTAINS CURRENT_DATE
```

Durch das Weglassen der FOR SYSTEM_TIME AS OF-Klausel werden wieder automatisch diejenigen Datensätze ermittelt, deren Transaktionszeitintervall den aktuellen Zeitpunkt beinhalten. Es würde hier also die letzte Zeile geliefert werden.

Steht wiederum SQL:2011 nicht zur Verfügung, muss diese vermeintlich einfache UPDATE-Anweisung in fünf SQL-Anweisungen transformiert und in der richtigen Reihenfolge ausgeführt

werden. Ein Beispiel dazu findet sich in [SGM00]. Die Umwandlung erfolgt dabei in zwei Stufen: die erste Stufe wendet die manuellen Operationen aus der Gültigkeitszeit-Tabelle und die zweite Stufe die der Transaktionszeit-Tabelle an [SGM00]. Diese Transformationen können in einer Stored Procedure gekapselt werden.

3.3.6 Zusammenfassung

Die neuen temporalen Erweiterungen von SQL:2011 machen es möglich, zeitbezogene Daten und Zeitdimensionen für Tabellen zu speichern, Integritätsbedingungen einzuhalten und diese temporalen Aspekte in Anfragen zu berücksichtigen. SQL:2011 stellt aber zunächst einmal einen noch unvollständigen Ansatz bei der Umsetzung von temporalen Konzepten dar [SSH13]. Einige Erweiterungen, wie z.B. Verbunde von temporalen Relationen, Gruppierungen und Aggregate oder die Anwendung von mehr als einer Gültigkeitszeit in einer Relation, sind für spätere Versionen des Standards in Vorbereitung [SSH13, KM12]. Im Gegensatz zum ersten Ansatz zur Schaffung von temporalen Erweiterungen in SQL (TSQL2 von [Sno95]) verhält sich SQL:2011 genau entgegengesetzt. Um auf aktuellen Tupeln zu arbeiten und diese zu ermitteln, sollten eigentlich keine zusätzlichen Zeitinformationen notwendig sein. Auch die temporalen Attribute sollten vor dem Benutzer versteckt werden. Beides ist in SQL:2011 nicht der Fall: die Attribute sind sichtbar und um aktuelle Datensätze zu erfragen, muss ein temporales Prädikat in die WHERE-Klausel mitgeführt werden [SSH13]. Dennoch bietet SQL:2011 einen minimalen Aufwand für die Hersteller von Datenbanksystemen und verzichtet auf tiefgreifende Eingriffe in die bestehenden Systeme.

Die temporalen Relationen können auch in extra dafür bereitgestellte Tabellen aufgeteilt werden, so dass diese History-Tabellen die eigentlichen Tabellen überwachen. Die überwachten Relationen enthalten somit nur aktuelle Daten. Ein Beispiel dafür wäre die Umsetzung der Transaktionszeit-Tabellen oder der Tracking Log in [SGM00]. Weiterhin ist es möglich, Zusatzinformationen über die Art der Operation (INSERT, UPDATE, DELETE, siehe Backlog in [SGM00]) oder über den Benutzer, der diese ausgeführt hat, zu speichern. Ein weiteres Beispiel für eine History-Tabelle ist im Wiki der PostgreSQL-Gemeinschaft [Pos15a] zu finden.

Reine temporale Datenbanken sind bisher ein aktives Forschungsgebiet und in der Praxis so gut wie nicht vorhanden[1]. Mit SQL:2011 wird aber schon heute ein temporales Konzept zur Verfügung gestellt, welches diejenigen Techniken simuliert, die schon von vielen Anwendern als Ersatz für fehlende temporale Fähigkeiten selbst in ihren Anwendungen implementiert wurden [SSH13].

[1] Als Beispiel-Ausnahme sei hier Teradata genannt [SSH13]

3.4 Stand der Technik

Es existieren viele verschiedene Datenbanken zur Speicherung von taxonomischen Daten. Als Beispiele dafür sind WoRMS, BioLib oder ITIS zu nennen. Ein Ableger von ITIS ist der „Catalogue of life“[1], welcher versucht, die vielen verteilten und meist lokalen Datenbanken zu konzentrieren und eine einheitliche Schnittstelle auf die Daten anzubieten.

Keine der oben erwähnten Datenbanken beinhaltet aber den Aspekt der dynamischen Versionierung mittels temporaler Techniken. Dies erkennt man einerseits am Datenmodell (sofern verfügbar), andererseits anhand der Beschreibung auf der Homepage. Als Beispiel sei hier der „Catalogue of life“ genannt: „...and there is no tracking of those changes“ [Cat15].

Der Tatsache der fehlenden temporalen Techniken ist wohl dem Umstand geschuldet, dass dieses Gebiet ein relativ junges ist. Die Anfänge der Datenbanken reichen dabei zurück bis 1996 (ITIS), 2000 (CoL) oder 2008 (WoRMS). Eine Anpassung des Datenmodells und der Migration der teils Millionen Datensätze ist daher keine triviale Aufgabe. Weiterhin ist die Unterstützung des SQL:2011-Standards noch nicht weit verbreitet und in bisher nur einigen kommerziellen Datenbanksystemen, wie IBM DB2 V10 oder Oracle 12c vorhanden. In einigen Open-Source Datenbanksystemen, z.B. PostgreSQL, wird über die Einführung temporaler Konzepte diskutiert [Pos15c]. Derweil entstehen Erweiterungen von Open-Source-Entwicklern [Pos15b].

[1] http://www.catalogueoflife.org/

4 Konzeption

In diesem Kapitel wird ein Konzept entworfen, was das Abbilden einer Hierarchie und die zeitliche Komponente kombiniert und somit zur Problemstellung aus Abschnitt 2.3 eine Lösung liefert. Es wird von einem verfügbaren SQL:2011-Standard sowie von einem idealen Datenbankprodukt ausgegangen, welches alle erforderlichen Operationen, insbesondere die Rekursion, unterstützt.

4.1 Hierarchie-Modell

Um eine Taxa-Liste im Benthos-Projekt zu implementieren, muss zuerst ein Modell gefunden werden, welches die Anforderungen aus 2.3 erfüllt. Anhand der vorgestellten Beispiel-Operationen in Kapitel 2 Abschnitt 2.1.2 wurde festgestellt, dass

- das Finden von Nachfolgern,
- das Einfügen neuer Knoten sowie
- das Aktualisieren der Knoten der Nachfolger

immer wiederkehrende Operationen auf Hierarchien in der biologischen Systematik sind.

Anhand des Vergleichs der Tabelle 3.6 aus der Zusammenfassung der Hierarchie-Modelle mit den herausgestellten häufig auftretenden Operationen aus Abschnitt 2.1.2, kann ein Modell direkt ausgeschlossen werden: das Nested-Sets-Modell. Es schneidet schlecht bei den Einfüge- und Löschoperationen ab und unterstützt keine referentielle Integrität. Dies resultiert daraus, dass das Neuberechnen der linken und rechten Werte bei Strukturänderungen in der Hierarchie sehr komplex ist. Aufgrund dieser Nachteile wird das Modell nicht weiter betrachtet.

Die Modelle Adjazenz-Liste und Closure-Table sind sich hinsichtlich der Operationen ebenbürtig. Beide unterstützen die referentielle Integrität und besitzen effiziente Operationen. Das Modell der Closure-Table arbeitet aber mit zwei Tabellen. Dies stellt einen, wenn auch nicht bedeutenden, Nachteil dar, weil hierfür viel Speicherplatz benötigt wird[1]. Aufgrund der zwei Tabellen wird sich daher gegen das Closure-Table-Modell entschieden.

[1] Im schlimmsten Fall $O(n^2)$, in der Realität aber sehr viel weniger [Kar10a].

Das Path-Enumeration-Modell weist eine sehr gute Effizienz bei allen Operationen auf. Dennoch unterstützt es keine referentielle Integrität, sodass es, alleine betrachtet, hier keine Anwendung findet. Es ist aber für User Interfaces gut geeignet, vor allem zur sogenannten Brotkrümelnavigation (engl. breadcrumbs), welche anzeigt, in welcher Ebene man sich innerhalb der Anwendung befindet. Das Path-Enumeration-Modell kann parallel mit anderen Modellen benutzt werden, sodass ein hybrides Modell entsteht[2].

Aus diesen Gründen wird sich für ein hybrides Modell, bestehend aus Adjazenz-Liste und Path-Enumeration, entschieden. Es können somit die Operationen beider Modelle benutzt und sich damit, zwischen eventuell auftretenden Effizienz-Problemen, immer für die effizienteren Operationen des jeweiligen Modells entschieden werden. Zwar gibt es keine Ordnung der Taxa in der biologischen Systematik, dennoch kann mittels Path-Enumeration-Modell eine Ordnung gewährleistet werden, falls diese gewünscht ist. Steht im Datenbankprodukt keine Rekursion zur Verfügung, so kann auf die Operationen des Path-Enumeration-Modells ausgewichen werden. Da aber nur eine Hierarchietiefe von sechs Rangstufen geplant ist, kann selbst ohne Rekursion ein 6-fach verschachtelter Verbund, sowie die entsprechenden Vereinigungen, fest programmiert werden, ohne dass es zu Performance-Problemen kommt. Als Alternative dazu ist hier das CURSOR-Prinzip zu nennen. Die feste Tiefe der Hierarchie sorgt also dafür, dass nur wenig Effizienz verloren geht.

Tabelle 4.1 stellt noch einmal die Modelle den häufigen Operationen gegenüber. Basierend auf den eingetragenen Bewertungen und unter Berücksichtigung, dass das Closure-Table-Modell zwei Tabellen nutzt, sowie der daraus resultierende erhöhte Speicherbedarf, wurde ein hybrides Modell aus Adjazenz-Liste und Path-Enumeration gewählt.

Modell	Finden von Nachfolgern	Einfügen/Löschen	Aktualisieren
Nested-Sets	–	-/–	–
Closure-Table	++	+/++	++
Path-Enumeration	++	++	++
Adjazenz-Liste	+	++	++

Tabelle 4.1: Hierarchie-Modelle im Vergleich zu den am häufigsten auftretenden Operationen

Aus der Beispiel-Taxonomie A.1.1 entsteht die folgende Relation 4.2. Es wurde vorerst auf die Normalisierung verzichtet, sodass es keine eigene Relation für die Rangstufen gibt.

[2] So wird es z.B. in ITIS verwendet. Das Datenmodell kann für verschiedene Datenbanksysteme unter http://www.itis.gov/downloads/index.html heruntergeladen werden. (zum Aufrufzeitpunkt: 24.04.2015)

id	parent_id	path	name	rank
1	NULL	1/	Bivalvia	Klasse
2	1	1/2/	Veneroida	Ordnung
3	1	1/3/	Myoida	Ordnung
4	2	1/2/4/	Cyrenidae	Familie
5	2	1/2/5/	Semelidae	Familie
6	5	1/2/5/6/	Abra	Gattung
7	5	1/2/5/7/	Scrobicularia	Gattung
8	6	1/2/5/6/8/	Abra alba	Art
9	6	1/2/5/6/9/	Abra prismatica	Art
10	7	1/2/5/7/10/	Scrobicularia plana	Art

Tabelle 4.2: Hierarchie-Tabelle mit Adjazenz-Liste und Path-Enumeration-Modell

4.2 Temporalität

Um die Anforderung der Versionierung und Änderungsverfolgung aus Abschnitt 2.3 zu erfüllen, muss der Hierarchie-Relation 4.2 die zeitliche Komponente hinzugefügt werden. Das Konzept sieht vor, bitemporal zu arbeiten, während in der Umsetzung nur die Gültigkeitszeit betrachtet wird. Die Verwendung des bitemporalen Ansatzes ermöglicht eine lückenlose Änderungsverfolgung der Daten. Jeder Datensatz wird, bevor dieser in irgendeiner Art und Weise verändert wird, vorher versioniert bzw. historisiert. Dies erlaubt eine transparente Dokumentation und kann zur späteren Auswertung genutzt werden. Auch im Falle eines Bedienfehlers ist dies von Vorteil.

Tabelle 4.3 zeigt die neue bitemporale Relation. Es wurden vier Attribute hinzugefügt: *vt_begin*, *vt_end* und *tt_start*, *tt_stop*. Die Präfixe *vt* und *tt* bedeuten *valid time* und *transaction time* und beziehen sich auf die Gültigkeits- bzw. Transaktionszeit.

id	parent_id	path	name	rank	vt_begin	vt_end	tt_start	tt_end
1	NULL	1/	Bivalvia	Klasse	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
2	1	1/2/	Veneroida	Ordnung	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
3	1	1/3/	Myoida	Ordnung	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
4	2	1/2/4/	Cyrenidae	Familie	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
5	2	1/2/5/	Semelidae	Familie	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
6	5	1/2/5/6/	Abra	Gattung	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
7	5	1/2/5/7/	Scrobicularia	Gattung	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
8	6	1/2/5/6/8/	Abra alba	Art	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
9	6	1/2/5/6/9/	Abra prismatica	Art	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
10	7	1/2/5/7/10/	Scrobicularia plana	Art	2015-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59

Tabelle 4.3: Bitemporale Hierarchie-Tabelle mit Adjazenz-Liste und Path-Enumeration-Modell

Das Create-Statement in SQL:2011 Syntax sieht folgendermaßen aus:

```
CREATE TABLE taxa_hierarchy (
  id INT,
  parent_id INT,
  path VARCHAR(255),
  name VARCHAR(100),
  rank VARCHAR(8),
  vt_begin DATE
  vt_end DATE,
  tt_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
  tt_stop TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
  PERIOD FOR ValidTime (vt_begin, vt_end),
  PERIOD FOR SYSTEM_TIME (tt_start, tt_end),
  PRIMARY KEY (id, ValidTime WITHOUT OVERLAPS),
) WITH SYSTEM VERSIONING
```

Wie aus Tabelle 4.3 zu sehen, starten die Gültigkeits- und Transaktionszeitintervalle am Anfang des Jahres 2015. Die Migration von den Excel-Protokollen sieht vor, dass eine aktuelle, dort vorhandene, Artenliste in die Hierarchie-Tabelle rekonstruiert wird. Die Rekonstruktion geschieht mittels eines PHP-Scripts, welches in Abschnitt 5.3 im Umsetzungskapitel kurz erläutert wird. Die Gültigkeits- und Transaktionsdaten resultieren daher aus dem Aufrufzeitpunkt dieses Scripts.

4.3 Datenbank-Schema

Das Datenbankschema in Abbildung 4.1 für die Verwaltung von Artenlisten besteht aus vier Relationen. Die bitemporale „taxa_hierarchy“-Relation enthält die Taxonomie Informationen. Eine Taxa-Liste besitzt eine Menge an Taxa aus der Hierarchie und wird durch die beiden Tabellen „taxa_list“ und „taxa_list_has_taxa_hierarchy“ abgebildet Diese Menge wird, zu mindestens einmalig, per Hand angelegt. Die Liste selbst besitzt einen Namen sowie ein Anlagedatum. Zusätzlich gibt es eine „ranks“-Tabelle die die beabsichtigten Hauptrangstufen aufnimmt.

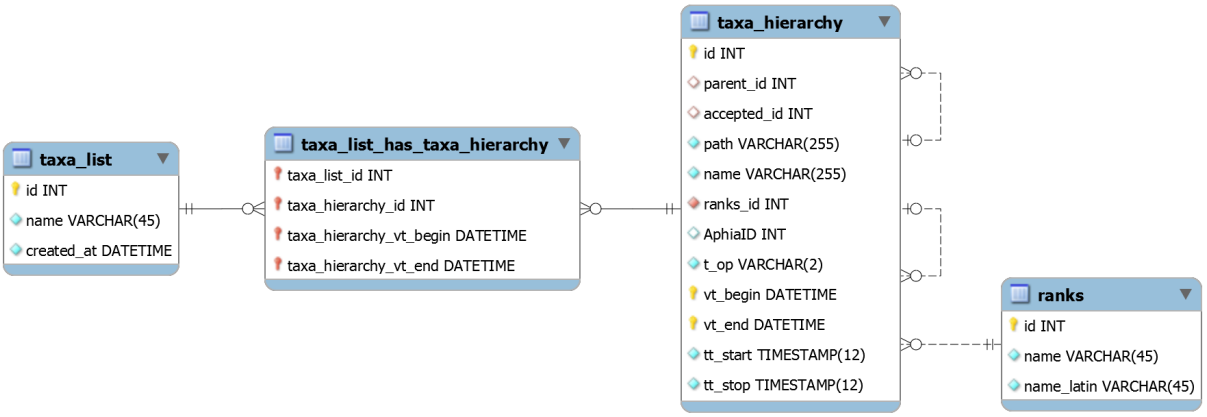

Abbildung 4.1: Erweitertes ER-Diagramm in MySQL-Workbench-Syntax

Einige neu eingeführte Attribute in der „taxa_hierarchy“-Relation bedürfen der näheren Erläuterung. Das Attribut *AphiaID* entstammt aus der WoRMS Datenbank und wird dort als Primärschlüssel verwendet. Durch das Mitführen dieser Information kann jedes Taxon in WoRMS referenziert werden. Allerdings gibt es in der Artenliste im Excel-Protokoll nicht zu jedem Eintrag eine AphiaID. Diese wird jedoch benötigt, um eine initiale Hierarchie mittels WoRMS zurückzubauen. Wie das bewerkstelligt wird, ist im Umsetzungskapitel in Abschnitt 5.3 beschrieben.

Das Attribut *accepted_id* gibt an, ob ein Taxon gültig im Sinne der biologischen Nomenklatur ist (vgl. Abschnitt 2.1.2). Es referenziert ein vorhandenes Taxon in der Hierarchie anhand dessen *id*-Attributs. Ist das *accepted_id*-Attribut leer, hat also keinen Wert (NULL), so ist dieses Taxon gültig und besitzt den Status „accepted“. Jeder andere Wert weist das Taxon als ungültig („unaccepted“) aus. In diesem Fall zeigt der Wert des *accepted_id*-Attributs auf ein aktuell gültiges und akzeptiertes Taxon. Dies ist somit eine Art Synonym, siehe dazu Abschnitt 4.4.3.

Zusätzlich soll im *t_op*-Attribut der „taxa_hierarchy“-Relation die Art der Operation mit abgespeichert werden. Die Operationsarten können z.B. eine Namensänderung, ein Anlegen oder das Umhängen eines Taxons sein. Vorgesehen sind hier Zeichenketten-Kürzel der Länge zwei. Tabelle 4.4 listet die Kürzel auf. Diese werden hauptsächlich automatisch beim Abgleich mit WoRMS gesetzt. Wird die Hierarchie per Hand geändert, so muss ermittelt werden, um welche Änderung es sich handelt und dementsprechend das Kürzel gesetzt werden. Wegen der temporalen Eigenschaften der „taxa_hierarchy“-Relation und der darauf ausgeführten temporalen Anweisungen, die teils aus mehreren impliziten Operationen bestehen, können die Operationsarten nicht immer klar voneinander getrennt werden. Dieses optionale Attribut wird hier mitgeführt, um später anhand dessen besser nachvollziehen zu können, wie sich die Hierarchie über die Zeit verändert hat. Diese Änderungsverfolgung und gegebenenfalls die Rekonstruktion und Vergleich von Hierarchien zu verschiedenen Zeiten ist aber nicht Gegenstand dieser Arbeit.

Kürzel	**Bedeutung**
IN	INSERT
NU	NAME UPDATE
AC	ACCEPTED UPDATE/STATUS UPDATE
RE	REALLOCATE
DE	DELETE

Tabelle 4.4: Kürzel für die Operationsarten

Fremdschlüssel

Die Fremdschlüsselbeziehungen zwischen der „taxa_list_has_taxa_hierarchy“-Relation und der Hierarchie-Relation sind hier nicht korrekt. Dies resultiert daraus, dass das Gültigkeitszeitintervall dem Anwender unterliegt und er diesen manipulieren kann. Zwar bleibt die Primärschlüsselbedingung in der Hierarchie-Relation unverletzt, es kann aber vorkommen, dass Datensätze

aus der „taxa_list_has_taxa_hierarchy“-Relation später nicht mehr eindeutig zuzuordnen sind. Durch die bitemporalen Eigenschaften der Hierarchie-Tabelle wird aber jeder Datensatz historisiert, so dass dieser immer noch zur Verfügung steht und referenziert werden kann. Wie dies bewerkstelligt wird, ist in Abschnitt 4.5 beschrieben.

Ähnlich verhält es sich mit den Fremdschlüsseln der Attribute *parent_id* und *accepted_id*. Da Schlüsselbedingungen nur auf aktuelle Transaktionszeit-Datensätze angewendet werden, kommt es vor, dass zum Beispiel das Attribut *parent_id* auf ein *id*-Attribut zeigt, welches doppelt als aktueller Datensatz vorhanden ist, aber dennoch unterschiedliche Gültigkeitszeitintervalle besitzt. Hier muss bei der Abfrage darauf geachtet werden, dass der aktuelle Datensatz im Sinne der Gültigkeitszeit erfragt wird (siehe Abschnitt 3.3.3). Gleiches gilt für das *accepted_id*-Attribut.

Die Fremdschlüssel in Abbildung 4.1 sind also mehr als Hinweise und Visualisierung der Beziehungen zwischen den Tabellen und Attributen zu verstehen, anstelle von korrekten Fremdschlüsseln im Sinne der Datenbanktheorie. Angesichts dieser Einschränkungen aufgrund der temporalen Eigenschaften fällt der Vorteil der referentiellen Integrität des Adjazenz-Listen Modells weg.

Darüber hinaus wurde bei den Gültigkeitszeit-Attributen *vt_begin* und *vt_end* anstelle des DATE-Typs ein DATETIME-Typ gewählt. Diese feinere Granularität des Zeitintervalls ermöglicht damit mehrere Änderungen eines Datensatzes innerhalb eines Tages, welche anderenfalls aufgrund des DATE-Wertebereichs ausgeschlossen worden wären.

4.4 Operationen

In diesem Abschnitt wird gezeigt, wie die häufig auftretenden Operationen auf der neu eingeführten bitemporalen Hierarchie-Relation ausgeführt werden können. Durch das Hinzufügen der zeitlichen Komponente müssen einige Abfragen erweitert und angepasst werden. Weiterhin wurde das *ranks_id*-Attribut in diesem Abschnitt durch das Attribut *rank* ersetzt, welches den Namen der Hauptrangstufe enthält. Des Weiteren ist die *AphiaID* auch ausgeblendet. Beides soll hier zum Verständnis der Tabellen beitragen und die Lesbarkeit fördern.

4.4.1 Einfügen & Aktualisieren

Da es sich beim Aktualisieren von Datensätzen immer um eine Versionierung handelt, ist somit auch immer die Einfüge-Operation beteiligt. Deshalb werden beide Operationen hier zusammen an Beispielen veranschaulicht.

In diesem Abschnitt wird eine andere, absichtlich veraltete, Taxonomie genutzt. Diese wird Schritt für Schritt modifiziert und aktualisiert, um wieder zur aktuellen Taxonomie A.1.1 zu gelangen. Das Beispiel ist aus der Biologie motiviert und mit WoRMS auf Korrektheit abgeglichen. Ob aber die zeitliche Reihenfolge derart war, wie die vorgestellte, war anhand von WoRMS nicht genau nachvollziehbar und soll an dieser Stelle auch nicht relevant sein. Es sollen nur die Lösungen für die angesprochenen Operationen aufgezeigt werden. Die Abbildung 4.2 stellt die hier verwendete Taxonomie dar. Die bitemporale Relation dazu ist in Tabelle 4.5 zu finden.

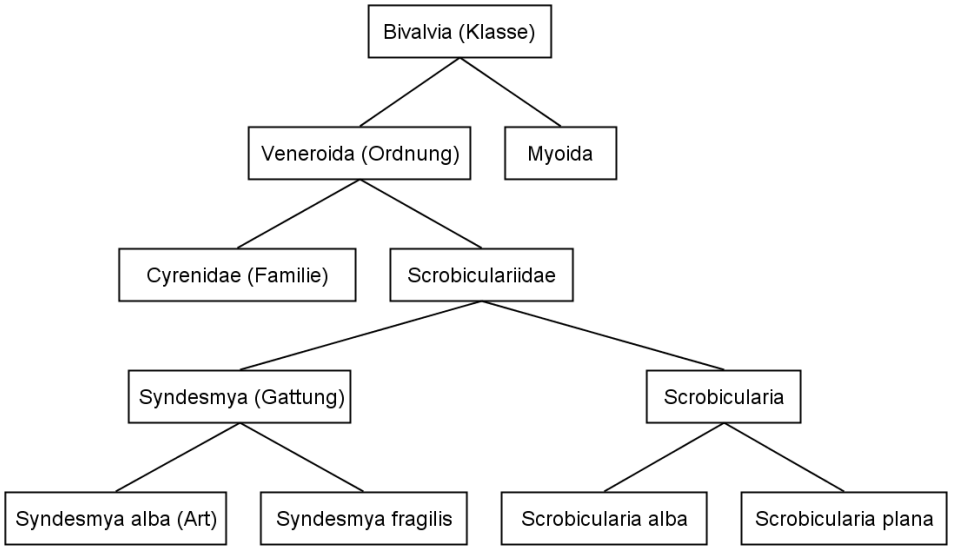

Abbildung 4.2: Konzeption: veraltete Taxonomie, auf die die Operationen ausgeführt werden

id	parent_id	accepted_id	path	name	rank	t_op	vt_begin	vt_end	tt_start	tt_stop
1	NULL	NULL	1/	Bivalvia	Klasse	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
2	1	NULL	1/2/	Veneroida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
3	1	NULL	1/3/	Myoida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
4	2	NULL	1/2/4/	Cyrenidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
6	5	NULL	1/2/5/6/	Syndesmya	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
7	5	NULL	1/2/5/7/	Scrobicularia	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
8	6	NULL	1/2/5/6/8/	Syndesmya alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
9	6	NULL	1/2/5/6/9/	Syndesmya fragilis	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
10	7	NULL	1/2/5/7/10/	Scrobicularia alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
11	7	NULL	1/2/5/7/11/	Scrobicularia plana	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00

Tabelle 4.5: Bitemporale Hierarchie-Tabelle mit Adjazenz-Liste, Path-Enumeration-Modell und der veralteten Taxonomie

Umbenennung einer Familie

Als erstes wird die Umbenennung der Familie „Scrobiculariidae“ nach „Semelidae“ vorgenommen, gemäß der Vorgehensweise, wie sie in der Abbildung 2.5 aus Abschnitt 2.1.2 veranschaulicht ist. Dies passiert in zwei Schritten:

1. Aktualisierung des Namens mittels temporalem UPDATE-Statement:

```
UPDATE taxa_hierarchy
  FOR PORTION OF ValidTime FROM CURRENT_DATE TO DATE '9999-12-31'
SET
  name = 'Semelidae',
  t_op = 'NU'
WHERE name = 'Scrobiculariidae'
```

2. Aktualisierung des *accepted_ id*-Attributs. Hier wird keine temporale Anweisung gebraucht, da es nach der Aktualisierung des Namens durch vorangegangenes UPDATE-Statement keinen Datensatz gibt, der im Gültigkeitszeitintervall liegt:

```
UPDATE taxa_hierarchy
SET
  accepted_id = id,
  t_op = 'AC'
WHERE name = 'Scrobiculariidae'
```

Die temporalen Eigenschaften bewirken, dass die direkten Kinder nicht mehr aktualisiert werden müssen, so wie es laut Abbildung 2.5 der Fall gewesen wäre. Da das *id*-Attribut sich nicht geändert hat, zeigt das *parent_ id*-Attribut der direkten Kinder weiterhin auf das aktuell gültige Taxon. Tabelle 4.6 zeigt das Ergebnis der Umbenennung. Die ausgegrauten Reihen stellen historisierte Datensätze dar. Dies soll wieder die Übersichtlichkeit erhöhen und zur besseren Lesbarkeit beitragen. Es ist zu erkennen, dass die zwei UPDATE-Anweisungen zwei historisierte (grau) und zwei aktuelle Tupel (**fett**) generiert haben. Der Datensatz mit dem Namen „Semelidae“ ist hier aufgrund der Gültigkeitszeit der aktuell gültige. Der nicht historisierte Datensatz mit dem Namen „Scrobiculariidae“ ist zwar ein aktueller Datensatz im Sinne der Transaktionszeit, aber kein gültiger mehr im Sinne der Gültigkeitszeit. Weiterhin ist er auch nicht mehr gültig im Sinne der biologischen Nomenklatur, da das *accepted_ id*-Attribut nicht mehr NULL ist. Es zeigt auf den aktuell gültigen Datensatz, in diesem Fall „Semelidae“. Diese Änderungen sind grundlegend für die hier verwendeten temporalen Tabellen und tauchen auch so in den im Folgenden behandelten Operationen auf.

id	parent_id	accepted_id	path	name	rank	t_op	vt_begin	vt_end	tt_start	tt_stop
1	NULL	NULL	1/	Bivalvia	Klasse	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
2	1	NULL	1/2/	Veneroida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
3	1	NULL	1/3/	Myoida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
4	2	NULL	1/2/4/	Cyrenidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:47:54
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	2015-04-15	2015-04-23 14:47:54	2015-04-23 14:47:54
5	**2**	**5**	**1/2/5/**	**Scrobiculariidae**	**Familie**	**AC**	**2015-01-01**	**2015-04-15**	**2015-04-23 14:47:54**	**9999-12-31 00:00:00**
5	**2**	**NULL**	**1/2/5/**	**Semelidae**	**Familie**	**NU**	**2015-04-15**	**9999-12-31**	**2015-04-23 14:47:54**	**9999-12-31 00:00:00**
6	5	NULL	1/2/5/6/	Syndesmya	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
7	5	NULL	1/2/5/7/	Scrobicularia	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
8	6	NULL	1/2/5/6/8/	Syndesmya alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
9	6	NULL	1/2/5/6/9/	Syndesmya fragilis	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
10	7	NULL	1/2/5/7/10/	Scrobicularia alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
11	7	NULL	1/2/5/7/11/	Scrobicularia plana	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00

Tabelle 4.6: Bitemporale Datensätze nach Änderung des Familiennamens

Umbenennung einer Gattung

Als nächstes wird eine Umbenennung der Gattung „Syndesmya“ nach „Abra“ vorgenommen, siehe dazu auch Abschnitt 2.1.2 Abbildung 2.4. Das Umbenennen einer Gattung zieht auch immer die Umbenennung der Arten dieser Gattung nach sich. Die Arten sind die unterste Rangstufe, daher wird ein Self-JOIN anstelle der Rekursion benutzt, um die direkten Nachfolger (Kinder) zu ermitteln. Ähnlich wie bei der Rangstufe Familie wird hier zunächst der Name der Gattung durch eine temporale UPDATE-Anweisung und danach das *accepted_id*-Attribut geändert.

```
UPDATE taxa_hierarchy
  FOR PORTION OF ValidTime FROM CURRENT_DATE TO DATE '9999-12-31'
SET
  name = 'Abra',
  t_op = 'NU'
WHERE name = 'Syndesmya'


UPDATE taxa_hierarchy
SET
  accepted_id = id,
  t_op = 'AC'
WHERE name = 'Syndesmya'
```

Es folgt die Umbenennung der Kinder. Artennamen setzen sich vereinfacht immer aus vorangestelltem Gattungsnamen und einem Artnamen zusammen. Also wird der alte Gattungsname als Teil des Artnamens durch den neuen Gattungsnamen ersetzt und die Art aktualisiert.

```
UPDATE taxa_hierarchy
  FOR PORTION OF ValidTime FROM CURRENT_DATE TO DATE '9999-12-31'
SET
  name = REPLACE(name, 'Syndesmya', 'Abra'),
  t_op = 'NU'
WHERE id IN (
  SELECT t2.id
  FROM taxa_hierarchy t1
    JOIN taxa_hierarchy t2
      ON t2.parent_id = t1.id
      AND t2.ValidTime CONTAINS CURRENT_DATE
  WHERE t1.name = 'Abra'
  AND t1.ValidTime CONTAINS CURRENT_DATE
)
```

Jetzt muss wieder das *accepted_id*-Attribut für alle veralteten Kinder (Arten) angepasst werden. Dazu wird wieder der Self-JOIN verwendet. In diesem Fall wird aber anstelle des „jetzigen“ Datums ’9999-12-31’ das aktuelle Datum durch die System-Variable *CURRENT_DATE* genom-

men, da sich das Datum zwischen der Änderung des Gattungsnamen und der Änderung der zugehörigen Artennamen nicht ändert[1].

```
UPDATE taxa_hierarchy
SET
  accepted_id = id,
  t_op = 'AC'
WHERE id IN (
    SELECT t2.id
    FROM taxa_hierarchy t1
      LEFT OUTER JOIN taxa_hierarchy t2
        ON t2.parent_id = t1.id
        AND t2.vt_end = CURRENT_DATE
    WHERE t1.name = 'Syndesmya'
)
AND vt_end = CURRENT_DATE
```

Eine einfachere und übersichtliche Alternative wäre folgende Anweisung, die auf einen Namensvergleich hinausläuft. Zu Beachten ist das zwingende Leerzeichen zwischen dem Gattungsnamen und dem Wildcard-Zeichen, da sonst auch die veraltete Gattung gefunden und diese somit auch fälschlicherweise aktualisiert werden würde.

```
UPDATE taxa_hierarchy
SET
  accepted_id = id,
  t_op = 'AC'
WHERE name LIKE 'Syndesmya %'
```

Basierend auf Tabelle 4.6 zeigt Tabelle 4.7 das Ergebnis der Umbenennung der Gattung und deren zugehörigen Arten.

[1] Es existiert natürlich der unwahrscheinliche Fall, dass die Änderung einer Gattung und deren Kinder so unglücklich um Mitternacht angestoßen werden kann, dass die Transaktion bis in den neuen Tag läuft und sich somit die Daten doch unterscheiden.

id	parent_id	accepted_id	path	name	rank	t_op	vt_begin	vt_end	tt_start	tt_stop
1	NULL	NULL	1/	Bivalvia	Klasse	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
2	1	NULL	1/2/	Veneroida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
3	1	NULL	1/3/	Myoida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
4	2	NULL	1/2/4/	Cyrenidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:47:54
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	2015-04-15	2015-04-23 14:47:54	2015-04-23 14:47:54
5	2	5	1/2/5/	Scrobiculariidae	Familie	AC	2015-01-01	2015-04-15	2015-04-23 14:47:54	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Semelidae	Familie	NU	2015-04-15	9999-12-31	2015-04-23 14:47:54	9999-12-31 00:00:00
6	5	NULL	1/2/5/6/	Syndesmya	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
6	5	NULL	1/2/5/6/	Syndesmya	Gattung	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
6	**5**	**6**	**1/2/5/6/**	**Syndesmya**	**Gattung**	**AC**	**2015-01-01**	**2015-04-15**	**2015-04-23 14:50:02**	**9999-12-31 00:00:00**
6	**5**	**NULL**	**1/2/5/6/**	**Abra**	**Gattung**	**NU**	**2015-04-15**	**9999-12-31**	**2015-04-23 14:50:02**	**9999-12-31 00:00:00**
7	5	NULL	1/2/5/7/	Scrobicularia	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
8	6	NULL	1/2/5/6/8/	Syndesmya alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
8	6	NULL	1/2/5/6/8/	Syndesmya alba	Art	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
8	**6**	**8**	**1/2/5/6/8/**	**Syndesmya alba**	**Art**	**AC**	**2015-01-01**	**2015-04-15**	**2015-04-23 14:50:02**	**9999-12-31 00:00:00**
8	**6**	**NULL**	**1/2/5/6/8/**	**Abra alba**	**Art**	**NU**	**2015-04-15**	**9999-12-31**	**2015-04-23 14:50:02**	**9999-12-31 00:00:00**
9	6	NULL	1/2/5/6/9/	Syndesmya fragilis	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
9	6	NULL	1/2/5/6/9/	Syndesmya fragilis	Art	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
9	**6**	**9**	**1/2/5/6/9/**	**Syndesmya fragilis**	**Art**	**AC**	**2015-01-01**	**2015-04-15**	**2015-04-23 14:50:02**	**9999-12-31 00:00:00**
9	**6**	**NULL**	**1/2/5/6/9/**	**Abra fragilis**	**Art**	**NU**	**2015-04-15**	**9999-12-31**	**2015-04-23 14:50:02**	**9999-12-31 00:00:00**
10	7	NULL	1/2/5/7/10/	Scrobicularia alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
11	7	NULL	1/2/5/7/11/	Scrobicularia plana	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00

Tabelle 4.7: Bitemporale Datensätze nach Änderung einer Gattung

Umbenennung einer Art

Es folgt die Umbenennung der Art „Abra fragilis“ in „Abra prismatica“ wie in Abbildung 2.2 aus Abschnitt 2.1.2 dargestellt. Die Vorgehensweise ist wieder dieselbe wie zuvor und besteht aus zwei Schritten. Als erstes wird die eigentliche Namensänderung vollzogen, danach findet die Aktualisierung des für den Status zuständigen *accepted_id*-Attributs statt.

```
UPDATE taxa_hierarchy
  FOR PORTION OF ValidTime FROM DATE '2015-04-13' TO DATE '9999-12-31'
SET
  name = 'Abra prismatica',
  t_op = 'NU'
WHERE name = 'Abra fragilis'

UPDATE taxa_hierarchy
SET
  accepted_id = id,
  t_op = 'AC'
WHERE name = 'Abra fragilis'
```

Aufbauend auf Tabelle 4.7 zeigt Tabelle 4.8 das Ergebnis dieser Umbenennung.

id	parent_id	accepted_id	path	name	rank	t_op	vt_begin	vt_end	tt_start	tt_stop
1	NULL	NULL	1/	Bivalvia	Klasse	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
2	1	NULL	1/2/	Veneroida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
3	1	NULL	1/3/	Myoida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
4	2	NULL	1/2/4/	Cyrenidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:47:54
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	2015-04-15	2015-04-23 14:47:54	2015-04-23 14:47:54
5	2	5	1/2/5/	Scrobiculariidae	Familie	AC	2015-01-01	2015-04-15	2015-04-23 14:47:54	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Semelidae	Familie	NU	2015-04-15	9999-12-31	2015-04-23 14:47:54	9999-12-31 00:00:00
6	5	NULL	1/2/5/6/	Syndesmya	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
6	5	NULL	1/2/5/6/	Syndesmya	Gattung	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
6	5	6	1/2/5/6/	Syndesmya	Gattung	AC	2015-01-01	2015-04-15	2015-04-23 14:50:02	9999-12-31 00:00:00
6	5	NULL	1/2/5/6/	Abra	Gattung	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
7	5	NULL	1/2/5/7/	Scrobicularia	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
8	6	NULL	1/2/5/6/8/	Syndesmya alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
8	6	NULL	1/2/5/6/8/	Syndesmya alba	Art	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
8	6	8	1/2/5/6/8/	Syndesmya alba	Art	AC	2015-01-01	2015-04-15	2015-04-23 14:50:02	9999-12-31 00:00:00
8	6	NULL	1/2/5/6/8/	Abra alba	Art	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
9	6	NULL	1/2/5/6/9/	Syndesmya fragilis	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
9	6	NULL	1/2/5/6/9/	Syndesmya fragilis	Art	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
9	6	9	1/2/5/6/9/	Syndesmya fragilis	Art	AC	2015-01-01	2015-04-15	2015-04-23 14:50:02	9999-12-31 00:00:00
9	6	NULL	1/2/5/6/9/	Abra fragilis	Art	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	2015-04-23 14:52:14
9	6	NULL	1/2/5/6/9/	Abra fragilis	Art	NU	2015-04-15	2015-04-16	2015-04-23 14:52:14	2015-04-23 14:52:14
9	**6**	**9**	**1/2/5/6/9/**	**Abra fragilis**	**Art**	**AC**	**2015-04-15**	**2015-04-16**	**2015-04-23 14:52:14**	**9999-12-31 00:00:00**
9	**6**	**NULL**	**1/2/5/6/9/**	**Abra prismatica**	**Art**	**NU**	**2015-04-16**	**9999-12-31**	**2015-04-23 14:52:14**	**9999-12-31 00:00:00**
10	7	NULL	1/2/5/7/10/	Scrobicularia alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
11	7	NULL	1/2/5/7/11/	Scrobicularia plana	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00

Tabelle 4.8: Bitemporale Datensätze nach Änderung des Artennamens

Statusänderung einer Art

Es kann vorkommen, dass eine Art in eine andere bestehende aufgeht. Somit wird eine Art ungültig und erhält einen Verweis auf die andere, weiterhin gültige, Art. Diese Operation ist in Abbildung 2.3 in Abschnitt 2.1.2 dargestellt. An dieser Stelle steigt die Komplexität: es werden insgesamt vier SQL-Anweisungen ausgeführt. Im Prinzip werden hier die temporalen UPDATE-Anweisungen manuell durchgeführt. Als erstes erfolgt ein nicht-temporales UPDATE, welches unter anderem den Endzeitpunkt der Gültigkeitszeit auf das aktuelle Datum der Ausführung dieser Anweisung setzt. Hier wird angenommen, dass das aktuelle Datum der 16.04.2015 ist und daher anstelle von CURRENT_DATE eingesetzt.

```
UPDATE taxa_hierarchy
SET
  accepted_id = 8, -- Abra alba
  t_op = 'AC',
  vt_end = DATE '2015-04-16'
WHERE name = 'Scrobicularia alba'
AND vt_begin <= DATE '2015-04-16'
AND vt_end > DATE '2015-04-16'
```

Somit wird diese Art auf „unaccepted“ für den spezifizierten Zeitraum gesetzt, indem das *accepted_id*-Attribut auf die gültige Art gesetzt wird. Danach folgt, im Gegensatz zu den Operationen davor, eine nicht temporale INSERT-Anweisung, die den neuen Datensatz gültig im Sinne der Gültigkeitszeit macht.

```
INSERT INTO taxa_hierarchy
  (id, parent_id, accepted_id, path, name, rank, t_op, vt_begin, vt_end)
  SELECT id, parent_id, accepted_id, path, name, rank, t_op,
    vt_end AS vt_begin, '9999-12-31' AS vt_end
  FROM taxa_hierarchy
  WHERE name = 'Scrobicularia alba'
  AND vt_begin <= DATE '2015-04-16'
  AND vt_end = DATE '2015-04-16'
```

Danach müssen bereits bestehende Verweise, die nun auf das veraltete Taxon zeigen, auf das neue Taxon aktualisiert werden, da ein ungültiges Taxon im Sinne der biologischen Nomenklatur immer auf dasjenige zeigt, welches aktuell gültig, also akzeptiert ist.

```
UPDATE taxa_hierarchy
  FOR PORTION OF ValidTime FROM DATE '2015-04-16' TO DATE '9999-12-31'
SET
  accepted_id = 8, -- Abra alba
  t_op = 'AC'
WHERE accepted_id = 10; -- Scrobicularia alba
```

Weiterhin müssen alle anderen aktuellen Datensätze im Sinne der Transaktionszeit auf den neuen Verweis zeigen und dementsprechend aktualisiert werden. Dabei werden nur diejenigen Daten aktualisiert, welche nicht bereits auf das neue „accepted“ Taxon zeigen. Dieses letzte UPDATE-Statement ist wichtig für die Behandlung mehrfacher Statusänderungen eines Taxons. So wird sichergestellt, dass alle veralteten Taxa im Sinne der Biologie, aber gültig im Sinne der Gültigkeitszeit, auf das nun aktuell gültige und „accepted“ Taxon zeigen.

```
UPDATE taxa_hierarchy
  FOR PORTION OF ValidTime FROM DATE '2015-04-16' TO DATE '9999-12-31'
SET
  accepted_id = 8,
  t_op = 'AC'
WHERE name = 'Scrobicularia alba'
AND accepted_id != 8
```

Diese Operation, also das Ändern des *accepted_id*-Attributs, bewirkt nun, dass auch ungültige Taxa im Sinne der Biologie bei der Abfrage nach aktuell gültigen Daten im Sinne der Gültigkeits- und Transaktionszeit zurückgeliefert werden. Es reicht nun nicht mehr die alleinige Angabe der CONTAINS-Klausel, um sowohl aktuelle gültige Daten, als auch biologisch gültige Daten zu ermitteln. Weiteres dazu kann in Abschnitt 4.4.2 nachgelesen werden.

Die finalen Daten nach dieser Operation können in Tabelle 4.9 nachgeschlagen werden. Daneben zeigt die Tabelle 4.10 die aktuellen gültigen Daten, ohne die historisierten Datensätze. Wird der Datensatz mit dem Namen „Scrobicularia alba“ ausgeblendet, da dieser biologisch nicht gültig ist, ist die Beispiel-Hierarchie A.1.1 aus den restlichen Datensätzen zu erkennen.

id	parent_id	accepted_id	path	name	rank	t_op	vt_begin	vt_end	tt_start	tt_stop
1	NULL	NULL	1/	Bivalvia	Klasse	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
2	1	NULL	1/2/	Veneroida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
3	1	NULL	1/3/	Myoida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
4	2	NULL	1/2/4/	Cyrenidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:47:54
5	2	NULL	1/2/5/	Scrobiculariidae	Familie	IN	2015-01-01	2015-04-15	2015-04-23 14:47:54	2015-04-23 14:47:54
5	2	5	1/2/5/	Scrobiculariidae	Familie	AC	2015-01-01	2015-04-15	2015-04-23 14:47:54	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Semelidae	Familie	NU	2015-04-15	9999-12-31	2015-04-23 14:47:54	9999-12-31 00:00:00
6	5	NULL	1/2/5/6/	Syndesmya	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
6	5	NULL	1/2/5/6/	Syndesmya	Gattung	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
6	5	6	1/2/5/6/	Syndesmya	Gattung	AC	2015-01-01	2015-04-15	2015-04-23 14:50:02	9999-12-31 00:00:00
6	5	NULL	1/2/5/6/	Abra	Gattung	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
7	5	NULL	1/2/5/7/	Scrobicularia	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
8	6	NULL	1/2/5/6/8/	Syndesmya alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
8	6	NULL	1/2/5/6/8/	Syndesmya alba	Art	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
8	6	8	1/2/5/6/8/	Syndesmya alba	Art	AC	2015-01-01	2015-04-15	2015-04-23 14:50:02	9999-12-31 00:00:00
8	6	NULL	1/2/5/6/8/	Abra alba	Art	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
9	6	NULL	1/2/5/6/9/	Syndesmya fragilis	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:50:02
9	6	NULL	1/2/5/6/9/	Syndesmya fragilis	Art	IN	2015-01-01	2015-04-15	2015-04-23 14:50:02	2015-04-23 14:50:02
9	6	9	1/2/5/6/9/	Syndesmya fragilis	Art	AC	2015-01-01	2015-04-15	2015-04-23 14:50:02	9999-12-31 00:00:00
9	6	NULL	1/2/5/6/9/	Abra fragilis	Art	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	2015-04-23 14:52:14
9	6	NULL	1/2/5/6/9/	Abra fragilis	Art	NU	2015-04-15	2015-04-16	2015-04-23 14:52:14	2015-04-23 14:52:14
9	6	9	1/2/5/6/9/	Abra fragilis	Art	AC	2015-04-15	2015-04-16	2015-04-23 14:52:14	9999-12-31 00:00:00
9	6	NULL	1/2/5/6/9/	Abra prismatica	Art	NU	2015-04-16	9999-12-31	2015-04-23 14:52:14	9999-12-31 00:00:00
10	7	NULL	1/2/5/7/10/	Scrobicularia alba	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:53:58
10	**7**	**8**	**1/2/5/7/10/**	**Scrobicularia alba**	**Art**	**AC**	**2015-01-01**	**2015-04-16**	**2015-04-23 14:53:58**	**9999-12-31 00:00:00**
10	**7**	**8**	**1/2/5/7/10/**	**Scrobicularia alba**	**Art**	**AC**	**2015-04-16**	**9999-12-31**	**2015-04-23 14:53:58**	**9999-12-31 00:00:00**
11	7	NULL	1/2/5/7/11/	Scrobicularia plana	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00

Tabelle 4.9: Bitemporale Datensätze nach Änderung des Status einer Art

id	parent_id	accepted_id	path	name	rank	t_op	vt_begin	vt_end	tt_start	tt_stop
1	NULL	NULL	1/	Bivalvia	Klasse	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
2	1	NULL	1/2/	Veneroida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
3	1	NULL	1/3/	Myoida	Ordnung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
4	2	NULL	1/2/4/	Cyrenidae	Familie	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
5	2	NULL	1/2/5/	Semelidae	Familie	NU	2015-04-15	9999-12-31	2015-04-23 14:47:54	9999-12-31 00:00:00
6	5	NULL	1/2/5/6/	Abra	Gattung	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
7	5	NULL	1/2/5/7/	Scrobicularia	Gattung	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
8	6	NULL	1/2/5/6/8/	Abra alba	Art	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
9	6	NULL	1/2/5/6/9/	Abra prismatica	Art	NU	2015-04-16	9999-12-31	2015-04-23 14:52:14	9999-12-31 00:00:00
10	7	8	1/2/5/7/10/	Scrobicularia alba	Art	AC	2015-04-16	9999-12-31	2015-04-23 14:53:58	9999-12-31 00:00:00
11	7	NULL	1/2/5/7/11/	Scrobicularia plana	Art	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00

Tabelle 4.10: Aktuelle gültige bitemporale Datensätze nach allen Änderungen

4.4.2 Suchen

Da die Einführung der Gültigkeitszeit zum Abfragen gerade aktueller Daten ein zusätzliches Prädikat bedingt, muss dieses in der SQL-Anweisung mit angegeben werden. Aufgrund der Eigenschaften der Transaktionszeit werden nur Datensätze geliefert, die aktuell in der Relation vorhanden sind. Zu beachten ist, dass die Klausel FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP an dieser Stelle weggelassen wird.

Suche und Aktuelle Daten

Um die aktuellen und gültigen Daten im Sinne der Gültigkeits- und Transaktionszeit zu ermitteln, wird das *vt_end*-Attribut mitgeführt. Durch die folgende Abfrage ist Tabelle 4.10 entstanden.

```
SELECT *
FROM taxa_hierarchy
WHERE ValidTime CONTAINS CURRENT_DATE
```

Durch die Einführung des *accepted_id*-Attributs reicht nun die alleinige Angabe der CONTAINS-Klausel nicht mehr aus. Das *accepted_id*-Attribut bewirkt, dass auch Datensätze zurückgegeben werden, die im Sinne der biologischen Nomenklatur nicht mehr gültig sind. Um diese „unaccepted“ Daten auszublenden, kann die WHERE-Klause um

```
AND accepted_id IS NULL
```

erweitert werden. Diese Änderung bewirkt, dass nur aktuelle gültige und „accepted“ Datensätze zurückgegeben werden.

Eine allgemeine Suche, sowohl nach veralteten, „unaccepted“ als auch nach aktuell gültigen „accepted“ Taxa, kann durch eine Abfrage absteigend sortiert nach dem Endzeitpunkt des Gültigkeitszeitintervalls realisiert werden. Die Ergebnismenge muss auf genau einen Datensatz beschränkt werden. Da der neueste Datensatz oben steht, wird auch nur dieser zurückgegeben.

```
SELECT t.*
FROM taxa_hierarchy t
WHERE name = 'Scrobiculariidae'
ORDER BY t.vt_end DESC
FETCH FIRST 1 ROWS ONLY
```

Basierend auf der aktuellen Hierarchie-Relation 4.9 wird dieser Datensatz geliefert (gekürzt um die Attribute *path* und *rank*):

id	parent_id	accepted_id	name	t_op	vt_begin	vt_end	tt_start	tt_stop
5	2	5	Scrobiculariidae	AC	2015-01-01	2015-04-15	2015-04-23 14:47:54	9999-12-31 00:00:00

Tabelle 4.11: Ergebnis der Suche nach dem Taxon „Scrobiculariidae“

Um die CONTAINS-Klausel und das *accepted_id*-Attribut nicht ständig mitführen zu müssen, um auf aktuellen biologischen Daten Abfragen zu stellen, bietet sich die Erzeugung eines VIEW an.

Finden von Nachfolgern

Direkte Nachfolger können beim Adjazenz-Listen-Modell mit einem Self-JOIN gefunden werden. In der äußeren Abfrage nach „Semelidae“ wird der aktuell gültige Datensatz gesucht. In der inneren Abfrage werden die aktuell gültigen Kinder ermittelt.

```
SELECT t2.*
FROM taxa_hierarchy FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP t1
  LEFT OUTER JOIN taxa_hierarchy FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP t2
  ON t2.parent_id = t1.id
  AND t2.ValidTime CONTAINS CURRENT_DATE
WHERE t1.name = 'Semelidae'
AND t1.ValidTime CONTAINS CURRENT_DATE
```

Durch das Entfernen der erweiterten Klausel für die Systemzeit vereinfacht sich die Abfrage:

```
SELECT t2.*
FROM taxa_hierarchy t1
  LEFT OUTER JOIN taxa_hierarchy t2
  ON t2.parent_id = t1.id
  AND t2.ValidTime CONTAINS CURRENT_DATE
WHERE t1.name = 'Semelidae'
AND t1.ValidTime CONTAINS CURRENT_DATE
```

Mit Tabelle 4.3 als Basis, wird dieser Datensatz geliefert:

id	parent_id	path	name	rank	vt_begin	vt_end	tt_start	tt_end
6	5	1/2/5/6/	Abra	Gattung	2014-01-01	9999-12-31	2015-01-01 00:00:00	9999-12-31 23:59:59
7	5	1/2/5/7/	Scrobicularia	Gattung	2014-01-01	9999-12-31	2014-01-01 00:00:00	9999-12-31 23:59:59

Tabelle 4.12: Bitemporaler Datensatz: direkte Nachfolger

Um alle Nachfolger, also den gesamten Teilbaum, zu ermitteln, kann folgende rekursive Abfrage verwendet werden:

```
WITH RECURSIVE GetTaxaSubTree AS (
  SELECT *
  FROM taxa_hierarchy
  FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP
  WHERE name = 'Semelidae'
  AND ValidTime CONTAINS CURRENT_DATE
UNION ALL
  SELECT children.*
  FROM
  taxa_hierarchy FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP children,
  GetTaxaSubTree subtree
  WHERE subtree.id = children.parent_id
  AND children.ValidTime CONTAINS CURRENT_DATE
)
SELECT *
FROM GetTaxaSubTree
ORDER BY id
```

Wieder kann die Klausel für die System-Zeit weggelassen werden. Basierend auf Tabelle 4.3 werden die Datensätze aus Tabelle 4.13 geliefert:

id	parent_id	path	name	rank	vt_begin	vt_end	tt_start	tt_end
5	2	1/2/5/	Semelidae	Familie	2014-01-01	9999-12-31	2014-01-01 00:00:00	9999-12-31 23:59:59
6	5	1/2/5/6/	Abra	Gattung	2014-01-01	9999-12-31	2014-01-01 00:00:00	9999-12-31 23:59:59
7	5	1/2/5/7/	Scrobicularia	Gattung	2014-01-01	9999-12-31	2014-01-01 00:00:00	9999-12-31 23:59:59
8	6	1/2/5/6/8/	Abra alba	Art	2014-01-01	9999-12-31	2014-01-01 00:00:00	9999-12-31 23:59:59
9	6	1/2/5/6/9/	Abra prismatica	Art	2014-01-01	9999-12-31	2014-01-01 00:00:00	9999-12-31 23:59:59
10	7	1/2/5/7/10/	Scrobicularia plana	Art	2014-01-01	9999-12-31	2014-01-01 00:00:00	9999-12-31 23:59:59

Tabelle 4.13: Bitemporaler Datensatz: alle Nachfolger

Würde diese Abfrage aber auf der aktuellen Hierarchie-Relation 4.9 ausgeführt werden, würde auch der im biologischen Sinne nicht gültige Datensatz mit *accpeted_id = 18* und Namen „Scrobicularia alba“ in der Ergebnismenge enthalten sein. Dies kann unterbunden werden, indem die SQL-Anweisung um das *accpeted_id*-Attribut erweitert wird (Listing 4.4.2).

Die Abfrage nach den direkten Nachfolgern im Path-Enumeration-Modell basiert auf dem Vergleich zwischen den Pfaden und des Ausgangspfades mit angehängter ID dynamischer Länge.

```
SELECT t2.*
FROM
  taxa_hierarchy FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP t1,
  taxa_hierarchy FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP t2
WHERE t1.name = 'Semelidae'
```

```
AND t1.ValidTime CONTAINS CURRENT_DATE
AND t2.path LIKE CONCAT(t1.path, REPEAT('_', CHAR_LENGTH(t2.id)), '/')
AND t2.ValidTime CONTAINS CURRENT_DATE
```

Der gesamte Teilbaum mit allen Nachfolgern kann anhand des Vergleichs mit dem Wildcard-Zeichen „%“ ermittelt werden:

```
SELECT *
FROM taxa_hierarchy FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP t1
WHERE t1.path LIKE CONCAT((
  SELECT t2.path
  FROM taxa_hierarchy FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP t2
  WHERE t2.name = 'Semelidae'
  AND t2.ValidTime CONTAINS CURRENT_DATE
), '%')
AND t1.ValidTime CONTAINS CURRENT_DATE
```

Auch hier kann in beiden Fällen die FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP-Klausel weggelassen werden. Die Ergebnisse sind die Gleichen, wie die aus den Operationen des Adjazenz-Listen-Modells.

4.4.3 Synonyme

Es gibt zwei Arten von Synonymen in der biologischen Taxonomie: „accepted“ und „unaccepted“. Ein „accepted“, also biologisch aktuell gültiges, Taxon besitzt eine Menge an Synonymen. Diese Synonyme sind „unaccepted“ und daher nicht mehr im biologischen Sinne gültig. Solch ein veraltetes Taxon besitzt keine Menge an Synonymen. Es hat vielmehr ein Synonym in der Form eines einzelnen Verweises auf das aktuell gültige Taxon, welches dieses alte Taxon ersetzt. Diese Art von Synonymen ist durch das Attribut *accepted_id* abgebildet. Beide Varianten von Synonymen können mittels dieses Attributs gefunden werden.

Um die in Abschnitt 2.1.2 angesprochenen Synonyme bzw. Synonym-Liste zu einem „accepted“ Taxon zu ermitteln, kann folgende Abfrage genutzt werden:

```
SELECT *
FROM taxa_hierarchy
WHERE accepted_id = 8 -- Abra alba
ORDER BY id ASC, vt_end
```

Basierend auf der Hierarchie-Tabelle 4.9 werden drei Datensätze geliefert. Tabelle 4.14 (gekürzt um die Attribute *path* und *rank*) stellt diese dar. Zu erkennen ist, dass es bedingt durch eine

Statusänderung zwei Taxa mit dem Namen „Scrobicularia alba“ zurückgegeben werden. Um das zu verhindern, kann auf ausgewählte Attribute mit der DISTINCT-Klausel gearbeitet werden. Als Alternative sei hier die temporale Normalisierung, also das Zusammenfassen von zusammenhängenden Gültigkeitszeitintervallen zu nennen.

id	parent_id	accepted_id	name	t_op	vt_begin	vt_end	tt_start	tt_stop
8	6	8	Syndesmya alba	AC	2015-01-01	2015-04-15	2015-04-23 14:50:02	9999-12-31 00:00:00
10	7	8	Scrobicularia alba	AC	2015-01-01	2015-04-16	2015-04-23 14:53:58	9999-12-31 00:00:00
10	7	8	Scrobicularia alba	AC	2015-04-16	9999-12-31	2015-04-23 14:53:58	9999-12-31 00:00:00

Tabelle 4.14: Synonyme für das Taxon „Abra prismatica“

Durch Ermitteln des *accepted_id*-Attributs eines „unaccepted“ Taxons kann dessen gültiger Nachfolger gefunden werden. Die Unterabfrage ist ähnlich der allgemeinen Suche nach einem Taxon (Listing 4.4.2), es ist lediglich die Einschränkung nach dem *accepted_id*-Attribut hinzugekommen. Diese Vorgehensweise ist notwendig, da es passieren kann, dass es mehrere Datensätze gibt, die aber unterschiedliche Werte für das *accepted_id*-Attribut liefern. So wird sichergestellt, dass es immer genau einen Datensatz gibt. Dabei wird immer auf den letzten bekannten Stand der Daten zugegriffen und dementsprechend das aktuelle gültige und „accepted“ Taxon ermittelt.

```
SELECT *
FROM taxa_hierarchy
WHERE id = (
  SELECT accepted_id
  FROM taxa_hierarchy
  WHERE name = 'Syndesmya fragilis'
  AND accepted_id IS NOT NULL
  ORDER BY vt_end DESC
  FETCH FIRST 1 ROWS ONLY
)
AND ValidTime CONTAINS CURRENT_DATE
```

Das Ergebnis der Abfrage basierend auf Tabelle 4.9 ist in Tabelle 4.15 dargestellt.

id	parent_id	accepted_id	name	t_op	vt_begin	vt_end	tt_start	tt_stop
9	6	NULL	Abra prismatica	NU	2015-04-16	9999-12-31	2015-04-23 14:52:14	9999-12-31 00:00:00

Tabelle 4.15: Das aktuell gültige Taxon für das ungültige Taxon „Syndesmya fragilis“

4.5 Erzeugung einer Artenliste

An dieser Stelle wird gezeigt, wie eine Artenliste erstellt werden kann. Die „taxa_list“-Relation enthält hier drei Datensätze mit verschiedenen Anlagedaten. Diese sind in Tabelle 4.16 zu finden. Des Weiteren wurden für die Taxa-Liste mit Namen „version 2“ die Datensätze aus Tabelle 4.17 in die „taxa_list_has_taxa_hierarchy“-Relation eingefügt.

id	name	created_at
1	version 1	2015-02-02
2	version 2	2015-04-15
3	version 3	2015-04-16

Tabelle 4.16: Angelegte Taxa-Listen

taxa_list_id	taxa_hierarchy_id	vt_begin	vt_end
2	1	2015-01-01	9999-12-31
2	2	2015-01-01	9999-12-31
2	3	2015-01-01	9999-12-31
2	4	2015-01-01	9999-12-31
2	5	2015-04-15	9999-12-31
2	6	2015-04-15	9999-12-31
2	7	2015-01-01	9999-12-31
2	8	2015-04-15	9999-12-31
2	9	2015-04-15	2015-04-16
2	10	2015-01-01	9999-12-31
2	11	2015-01-01	9999-12-31

Tabelle 4.17: Beispieldaten aus der Zuordnungstabelle „taxa_list_has_taxa_hierarchy“

Es existieren zwei Varianten, um eine Taxa-Liste zu generieren. Als Beispiel wird hier die Taxa-Liste mit *id=2* bzw. Namen „version 2“ verwendet. Variante 1 erzeugt anhand des kompletten Transaktionszeitintervalls die Taxa-Liste:

```
SELECT th.*
FROM taxa_list_has_taxa_hierarchy tl
  JOIN taxa_hierarchy
    FOR SYSTEM_TIME
      FROM '0001-01-01-00.00.00'
      TO '9999-12-31-00.00.00' th
    ON tl.taxa_hierarchy_id = th.id
    AND tl.vt_begin = th.vt_begin
    AND tl.vt_end = th.vt_end
```

```
    AND th.accepted_id IS NULL
WHERE tl.taxa_list_id = 2
ORDER BY th.id ASC, tt_start
```

Diese Variante arbeitet unter der Annahme, dass keine „unaccepted“ Taxa in die Liste mit aufgenommen werden. Durch die Selektion des Attributs *accepted_id IS NULL* wird dies verhindert. Basierend auf Tabelle 4.9, welche die zurzeit aktuelle Hierarchie darstellt, werden die Datensätze aus Tabelle 4.18 zurückgegeben (gekürzt um die Attribute *path* und *rank*).

id	parent_id	accepted_id	name	t_op	vt_begin	vt_end	tt_start	tt_stop
1	NULL	NULL	Bivalvia	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
2	1	NULL	Veneroida	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
3	1	NULL	Myoida	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
4	2	NULL	Cyrenidae	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
5	2	NULL	Semelidae	NU	2015-04-15	9999-12-31	2015-04-23 14:47:54	9999-12-31 00:00:00
6	5	NULL	Abra	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
7	5	NULL	Scrobicularia	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
8	6	NULL	Abra alba	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
9	6	NULL	Abra fragilis	NU	2015-04-15	2015-04-16	2015-04-23 14:52:14	2015-04-23 14:52:14
10	7	NULL	Scrobicularia alba	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	2015-04-23 14:53:58
11	7	NULL	Scrobicularia plana	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00

Tabelle 4.18: Generierte Taxa-Liste (Variante 1)

Variante 2 nutzt das Gültigkeitszeitintervall zum Zeitpunkt der Erstellung dieser Liste. Diese Vorgehensweise macht die „taxa_list_has_taxa_hierarchy“-Relation überflüssig. Durch folgende temporale SQL-Anweisung wird anhand des *created_at*-Attributs die Taxa-Liste aus der „taxa_hierarchy“-Relation erstellt.

```
SELECT *
FROM taxa_hierarchy
WHERE ValidTime CONTAINS DATE '2015-04-15'
ORDER BY id ASC, tt_start
```

Diese Variante liefert im Unterschied zur Variante 1 aktuelle Transaktionszeit-Datensätze. Das bedeutet, dass Taxa geliefert werden, die bereits nicht mehr gültig im Sinne der biologischen Nomenklatur sind. Diese Taxa sind am *accepted_id*-Attribut zu erkennen, das nicht NULL ist. Basierend auf Tabelle 4.9 werden die Datensätze aus Tabelle 4.19 zurückgegeben (gekürzt um die Attribute *path* und *rank*).

Beide Varianten besitzen eine nicht zu vernachlässigende Einschränkung: Werden Datensätze im Nachhinein manipuliert und der Gültigkeitszeitraum verändert, so ändert sich auch die generierte Taxa-Liste. Dieser Punkt ist nicht unerheblich im Blick auf die eindeutige Zuordnung zwischen Taxa-Liste und Prüfbericht. Für Variante 1 kann dieses Problem gelöst werden, indem ein Taxon in dem bereits vergangenen Gültigkeitszeitraum als Synonym angelegt wird. Diese Variante liefert nämlich nur „accepted“-Taxa und würde dieses neue Synonym nicht anzeigen.

id	parent_id	accepted_id	name	t_op	vt_begin	vt_end	tt_start	tt_stop
1	NULL	NULL	Bivalvia	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
2	1	NULL	Veneroida	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
3	1	NULL	Myoida	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
4	2	NULL	Cyrenidae	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
5	2	NULL	Semelidae	NU	2015-04-15	9999-12-31	2015-04-23 14:47:54	9999-12-31 00:00:00
6	5	NULL	Abra	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
7	5	NULL	Scrobicularia	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00
8	6	NULL	Abra alba	NU	2015-04-15	9999-12-31	2015-04-23 14:50:02	9999-12-31 00:00:00
9	6	9	Abra fragilis	AC	2015-04-15	2015-04-16	2015-04-23 14:52:14	9999-12-31 00:00:00
10	7	8	Scrobicularia alba	AC	2015-01-01	2015-04-16	2015-04-23 14:53:58	9999-12-31 00:00:00
11	7	NULL	Scrobicularia plana	IN	2015-01-01	9999-12-31	2015-04-23 14:40:43	9999-12-31 00:00:00

Tabelle 4.19: Generierte Taxa-Liste (Variante 2)

Bei Variante 2 gibt es keine derartige Lösung. Als allgemeine Lösung sollte verhindert werden, dass Gültigkeitszeitintervalle im Nachhinein verändert werden dürfen. Da diese Veränderung nur per Hand geschehen kann und nicht durch den automatischen Abgleich mit WoRMS, müssen diejenigen Personen darauf geschult werden, die diese Verwaltungsarbeit vornehmen. Somit wird davon ausgegangen, dass der Startzeitpunkt des Gültigkeitszeitintervalls immer dem Zeitpunkt der Bearbeitung entspricht. Weiterhin sollten Aktualisierungen immer temporal erfolgen.

4.6 Zusammenfassung

Das vorgestellte Konzept nutzt ein hybrides Hierarchie-Modell, bestehend aus der Adjazenz-Liste und der Path-Enumeration. Es wurde um die zeitlichen Komponenten der Gültigkeits- und Transaktionszeit ergänzt und arbeitet somit bitemporal. Die vorgestellten Operationen aus Abschnitt 2.1.2 können damit vollständig abgedeckt werden. Es waren dafür Anpassungen an den DML- und DDL-Anweisungen nötig.

Eine Taxa-Liste kann auf zwei Arten erzeugt werden. Variante 1 nutzt den „Fremdschlüssel“, also die Attribute *id*, *vt_begin*, *vt_end* aus der „taxa_list_has_taxa_hierarchy“-Relation, um über das gesamte Transaktionszeitintervall die Liste aus der „taxa_hierarchy“-Relation zu rekonstruieren. Die zweite Variante überspringt die „taxa_list_has_taxa_hierarchy“-Relation. Es wird hier der Gültigkeitszeitintervall und das Anlagedatum der Liste aus der “taxa_list“-Relation genutzt, um die Taxa-Liste zu erzeugen.

Bei der Verwaltung der Artenlisten muss aber darauf geachtet werden, dass jede Änderung temporal erfolgt. Das bedeutet, dass der Startzeitpunkt des Gültigkeitszeitintervalls dem Zeitpunkt der Bearbeitung entspricht und der Endzeitpunkt in der Zukunft liegt. Weiterhin ist zu beachten, dass ein Gültigkeitszeitintervall im Nachhinein nicht mehr verändert werden darf.

5 Umsetzung

Dieses Kapitel beschreibt die prototypische Implementierung des Konzepts. Als Datenbanksystem kommt MySQL zum Einsatz, da es die Basis für die zurzeit im Benthos-Projekt in Entwicklung befindliche Benthos-Datenbank bildet (siehe Abschnitt 1.2). Des Weiteren wird hier nur mit der Gültigkeitszeit, anstelle des bitemporalen Ansatzes, gearbeitet. Darüber hinaus wird die Annahme gemacht, dass Daten immer nur ab dem Anlage-Zeitpunkt gültig sind. Es können somit keine Daten in der Vergangenheit, noch in der Zukunft angelegt werden. Daher entspricht die Umsetzung im Prinzip der der Transaktionszeittabellen. Alle hier beschriebenen SQL-Anweisungen und PHP-Scripte sind auf der beigelegten CD im Verzeichnis „Umsetzung“ zu finden.

5.1 Hierarchie-Modell in MySQL

Das hybride Hierarchie-Modell kann in MySQL mit folgender SQL-Anweisung erzeugt werden:

```
CREATE TABLE taxa_hierarchy (
  id INT(10) NOT NULL AUTO_INCREMENT,
  parent_id INT(10) DEFAULT NULL,
  accepted_id INT(10) DEFAULT NULL,
  path VARCHAR(255) NOT NULL,
  name VARCHAR(100) NOT NULL,
  ranks_id INT(10) NOT NULL,
  AphiaID INT(10) DEFAULT NULL,
  t_op VARCHAR(2) NOT NULL,
  vt_begin DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  vt_end DATETIME NOT NULL DEFAULT '9999-12-31 00:00:00',
  PRIMARY KEY (id, vt_begin, vt_end),
  INDEX idx_ranks (ranks_id),
  FOREIGN KEY fk_ranks (ranks_id) REFERENCES ranks (id)
) ENGINE = InnoDB
```

Zu beachten ist, dass hier bereits die Attribute für die Temporalität aus Abschnitt 5.2 mit aufgeführt sind. Dies vermeidet ALTER TABLE-Anweisungen im Nachhinein. Der Primärschlüssel setzt sich , so wie in Abschnitt 3.3.3 beschrieben, aus dem *id*-Attribut und dem Gültigkeitszeitintervall zusammen. Überschneidungen sind aber noch nicht ausgeschlossen. Wie dies verhindert

wird, ist in Abschnitt 5.2 über Temporalität beschrieben. Weiterhin muss die „ranks“-Relation (siehe Abschnitt 4.3) bereits vorhanden sein, da sonst das Anlegen aufgrund der Fremdschlüsselbedingung fehl schlägt. Das SQL-Statement dafür befindet sich auf der CD und hat den Namen „create_ranks.sql“.

Da MySQL keine *Common Table Expressions* aus dem SQL:1999-Standard und damit keine Rekursion beherrscht [MyS15b], müssen die Abfragen zum Suchen und Finden von Nachfolgern umgeschrieben werden. Einerseits können dazu die Methoden des Path-Enumeration-Modells verwendet werden. Andererseits besteht die Hierarchie aus maximal sechs Ebenen, d.h. dass eine Anweisung, bestehend aus mehreren UNION- und Self-JOIN-Anweisungen bis zur maximalen Tiefe von 6, fest programmiert werden kann. Dieses Statement sowie weitere Anweisungen zum Suchen von Nachfolgern und Vorgängern, sowohl mittels Methoden des Adjazenz-Listen-Modells, als auch des Path-Enumeration-Modells, sind auf der CD in der Datei „suche_im_hierarchiemodell.sql“ zu finden.

5.2 Temporalität in MySQL

MySQL unterstützt keine temporalen SQL-Anweisungen gemäß dem SQL:2011-Standard. Dieses Problem kann durch das Programmieren der einzelnen Anweisungen und der Kapselung dieser in einer Stored Procedure gelöst werden. Snodgrass beschreibt in [SGM00] wie Primärschlüssel ohne Überlappungen sowie Einfüge-, Änderungs- und Löschoperationen ohne SQL:2011-Standard implementiert werden können.

Primärschlüssel

Um die Bedingung zu simulieren, dass sich kein Gültigkeitszeitintervall in der „taxa_hierarchy“-Relation überlappen darf, kann dies durch eine ASSERTION oder CHECK CONSTRAINT implementiert werden. Da MySQL keines von beiden unterstützt[1], kann alternativ dazu ein Trigger erstellt werden. Einen einzigen Trigger für mehrere Trigger-Events, wie INSERT oder UPDATE, in MySQL zu erstellen ist nicht möglich. Daher werden zwei separate Trigger erzeugt, deren Funktionalität aber dieselbe ist. Die Trigger erzeugen bei Überschneidungen einen ähnlichen Fehler, wie ihn MySQL selbst bei einer Primärschlüsselverletzung werfen würde. Die Trigger befinden sich im Unterverzeichnis „sql Skripte“ in der Datei „trigger.sql“. Um zu überprüfen, ob die Trigger richtig arbeiten, liegt eine SQL-Datei namens „pk_unit_tests.sql“ im gleichen Verzeichnis. Alle darin enthaltenen Tests müssen einen Fehler werfen. Dies stellt sicher, dass es nicht zu Überschneidungen der Zeiträume kommen kann.

[1] „The CHECK clause is parsed but ignored by all storage engines“ [MyS15a]

Einfügen - INSERT

Unter der Annahme, dass Daten immer nur ab dem Zeitpunkt gültig sind, an dem sie angelegt wurden, vereinfacht sich das Einfügen. Es besteht somit beim Einfügen neuer Datensätze kein Unterschied zu herkömmlichen, also nicht temporalen Tabellen. Darüber hinaus besitzen die temporalen Attribute *vt_begin* und *vt_end* Default-Werte, so dass diese bei der INSERT-Anweisung nicht angegeben werden müssen.

```
INSERT INTO taxa_hierarchy
  (parent_id, path, name, ranks_id, AphiaID, t_op)
VALUES
  (7, '1/2/5/7/11/', 'Scrobicularia', 6, 828053, 'IN')
```

Aktualisieren - UPDATE

Auch hier wird von der Annahme ausgegangen, dass Veränderungen bzw. Aktualisierungen eines Datensatzes den Gültigkeitszeitraum nicht rückwirkend modifizieren dürfen. Dennoch muss ein einfaches UPDATE-Statement in zwei Anweisungen, nämlich einem UPDATE und einem INSERT aufgeteilt werden. Das neue UPDATE beendet den alten Datensatz, indem das *vt_end*-Attribut auf den Zeitpunkt des Aufrufs gesetzt wird. Die INSERT-Anweisung fügt einen neuen Datensatz mit den gewünschten Änderungen auf Basis des gerade aktualisierten Datensatzes ein. Zusätzlich legt es den Startzeitpunkt auf den Aufrufzeitpunkt und den Endzeitpunkt auf die Repräsentation des aktuellen Zeitpunkts '9999-12-31' fest. Da die Anweisungen die Fremdschlüsselbedingungen verletzen und diese erst nach Ausführung beider Anweisungen wieder intakt sind, müssen die Trigger temporär deaktiviert werden. Da es keinen eingebauten Mechanismus in MySQL gibt, der dies bewerkstelligt, muss auf eine andere Lösung zurückgegriffen werden. Es wird dazu eine Benutzer-Variable pro Trigger angelegt, welche anzeigt, ob dieser Trigger aktiviert oder deaktiviert ist. Im Trigger selbst wird diese Variable geprüft und nur wenn diese anzeigt, dass der Trigger aktiviert ist, wird dieser auch weiter ausgeführt. Die Datei „stp_temporal_update.sql“ enthält die Stored Procedure „temporal_update“, welche die beiden oben genannten Anweisungen, also UPDATE und INSERT, enthält. Kommentare in der SQL-Datei erläutern und beschreiben diese Prozedur detailliert.

Löschen - DELETE

Der Vollständigkeit halber wird hier die Löschoperation mit angegeben. Ein temporales Löschen wird mittels einer UPDATE-Anweisung umgesetzt. Diese beendet den Datensatz, indem der Endzeitpunkt auf den Zeitpunkt des Aufrufs gesetzt wird.

```
UPDATE taxa_hierarchy
SET
  vt_end = CURRENT_TIMESTAMP,
  t_op = 'DE'
WHERE id = 11
AND vt_end = '9999-12-31'
```

temporale Abfragen

Mit Einführung der Temporalität müssen wieder alle Abfragen erweitert werden (vgl. Abschnitt 3.3.3). Das betrifft auch die Suchabfragen aus Abschnitt 5.1. Um aktuell gültige Daten zu ermitteln, kann folgende Abfrage genutzt werden:

```
SELECT *
FROM taxa_hierarchy
WHERE vt_begin <= CURRENT_TIMESTAMP
AND vt_end > CURRENT_TIMESTAMP
```

Diese Abfrage ermittelt auch „unaccepted“ Datensätze. Um dies auszuschließen, muss die WHERE-Klausel um

```
AND accepted_id IS NULL
```

erweitert werden.

Auf der CD befindet sich dazu die Datei „temporale_suche_im_hierarchiemodell.sql“, welche die beiden oben genannten Abfragen enthält. Darüber hinaus sind dort alle Abfragen aus der Datei „suche_im_hierarchiemodell.sql“ enthalten, nur erweitert um die temporalen Prädikate.

5.3 Erzeugen der initial Hierarchie

Um einen Startpunkt für die Daten in der Datenbank zu definieren, wird die Artenliste des Excel-Protokolls „Dateneingabe-Muster-ab 2010-Akreditiert und gemessen.xlsm“ in eine Hierarchie im Sinne einer biologischen Taxonomie in das hybride Hierarchie-Modell überführt. Alle an dieser Stelle verwendeten Dateien befinden sich auf der CD im Unterverzeichnis „Umsetzung/Hierarchie Rekonstruktion“. Benutzt wurden Windows 7 x64, Microsoft Excel 2007, MySQL 5.6.24 sowie PHP 5.6.8. Es folgt die Beschreibung der Schritte, zur Erzeugung der Anfangshierarchie.

1. Exportieren der Mappe „Artenliste_Version2“ in das CSV-Format. Der Name der Datei lautet hier „artenliste_version2.csv“.

2. Die CSV-Datei in einem beliebigen Editor öffnen und alle leeren Zeilen am Ende der Datei entfernen. Leere Zeilen sind diejenigen, welche nur aus dem Semikolon bestehen, aber sonst keine Daten besitzen (;;;;;;;). Eine komplett leere Zeile ohne Semikola am Ende der Datei belassen. Datei speichern.

3. Eine temporäre Tabelle „artenliste_excel“ erstellen, mit den Attributen *id*, *AphiaID*, *valid_name* und *not_found*. Das CREATE-Statement befindet sich in der Datei „create_inital_hierarchy.sql“. Das *not_found*-Attribut dient nach der Rekonstruktion zur Ermittlung von nicht gefundenen Taxa.

4. Mittels LOAD DATA-Klausel die CSV-Datei in diese Tabelle importieren. Die Datei lag an dieser Stelle direkt auf der Festplatte mit dem Laufwerksbuchstaben „D“. Es interessieren hierbei nur die CSV-Spalten „AphiaID“ und „valider Artname_ohne_Autor_Jahr“. Alle anderen Spalten werden mittels der Technik der benutzerdefinierten Variablen übersprungen. Diese Anweisung ist ebenso in der Datei „create_inital_hierarchy.sql“ zu finden.

5. Das PHP-Script „create_start_taxonomie.php“ rekonstruiert die eigentliche Hierarchie und füllt die Hierarchie-Relation mit Daten. Dazu kann es auf einer Konsole (Windows Syntax) mittels

   ```
   php.exe create_start_taxonomie.php
   ```

   aufgerufen werden. Anhand der Ausgaben in der Konsole ist zu erkennen, welches Taxon gerade bearbeitet wird. Die Arbeitsweise und der Ablauf des Scriptes ist in der Datei selbst dokumentiert. Dieses Script lief auf dem Test-Rechner[2] rund 6 Minuten und erzeugte 3019 Datensätze.

6. Erzeugung des Pfades als Zeichenkette. Dazu gibt es zwei Hilfsprozeduren: „CreatePathFromLeafToRoot“ und „ReversePath“.
   Die Prozedur „CreatePathFromLeafToRoot“ erzeugt einen Pfad, indem sie ausgehend von einem Blatt, also einem Taxon der Rangstufe Art, die Hierarchie hoch traversiert, bis die Wurzel erreicht ist. Sie nutzt die Funktionalität der „UpTreeTraversal“-Prozedur aus [Cel04]. Da diese Vorgehensweise einen Pfad erzeugt, der verkehrt herum aufgebaut ist, muss dieser noch umgedreht werden, bevor er abgespeichert werden kann. Diese Aufgabe übernimmt die Prozedur „ReversePath“. Die auszuführende SQL-Anweisung hat die Form:

[2] Windows 7 x64, i5-2500K, 8GB RAM, DSL20000

```
UPDATE taxa_hierarchy
SET
  path = ReversePath(CreatePathFromLeafToRoot(id))
```

Beide Prozeduren sind in den jeweiligen SQL-Dateien, die mit „stp_“ beginnen, zu finden. Die Ausführung dauerte auf dem Test-Rechner ca. 2 Sekunden.

Anhand des *not_found*-Attributs der temporären „artenliste_excel“-Relation sind Taxa zu erkennen, welche in WoRMS weder anhand der AphiaID, noch anhand des Namens gefunden wurden. Diese Datensätze müssen manuell behandelt werden. Nachdem das Erzeugen der Anfangs-Hierarchie abgeschlossen ist, kann die „artenliste_excel“-Relation gelöscht werden.

5.4 Automatisierter Abgleich mit WoRMS

Dieser Abschnitt dient der Vorstellung eines Proof-Of-Concepts in Form des PHP-Scripts „worms_abgleich.php“, welches im Verzeichnis „WoRMS Abgleich“ zu finden ist. Das Script vergleicht die aktuell vorhandenen und biologisch gültigen Taxa in der lokalen Datenbank anhand des von WoRMS angebotenen Webservices. Der Vergleich erfolgt nur auf den von WoRMS zurück gelieferten Status des Taxons. Ist der WoRMS-Status „unaccepted“, so ist das Taxon in der lokalen Datenbank veraltet und wird aktualisiert. Dabei gibt es zwei unterschiedliche Formen der Aktualisierung. Welche Art der Aktualisierung gewählt wird, hängt davon ab, ob das neue „accepted“ Taxon in der lokalen Datenbank bereits vorhanden ist oder nicht. Dabei liefert WoRMS bei einem „unaccepted“ Taxon den Verweis auf das nun „accepted“ Taxon in Form einer gültigen AphiaID mit. Das Suchen in der lokalen Datenbank erfolgt unter der Annahme, dass das „accepted“ Taxon von WoRMS auch lokal „accepted“ und gültig ist. Wurde das Taxon lokal gefunden, so erfolgt ein Status-Update (Operationsart AC, vgl. Tabelle 4.4). Ist es nicht vorhanden, wird eine Namens-Änderung (Operationsart NU) durchgeführt. In beiden Fällen erfolgt der Aufruf der Stored Procedure „temporal_update“ aus Abschnitt 5.2.

Im selben Verzeichnis, in dem auch das PHP-Script liegt, ist eine weitere Datei namens „worms_abgleich.sql“. Mit dieser Datei kann die veraltete Taxonomie 4.2 aus Abschnitt 4.4.1 erstellt werden. Durch Aufruf des PHP-Scriptes in der Konsole

php.exe worms_abgleich.php

wird diese veraltete Taxonomie per WoRMS automatisch abgeglichen und aktualisiert. Vergleicht man diese automatisch aktualisierte Taxonomie mit den manuellen Aufrufen aus Datei „konzept.sql“ im „Umsetzung/sql Skripte/Konzept-Operationen“-Verzeichnis, so ist zu erkennen, dass das Taxon „Abra fragilis“ durch den automatischen Abgleich übersprungen wurde, da das Taxon „Syndesmya fragilis“ in WoRMS auf das aktuell gültige und „accepted“ Taxon „Abra prismatica“

zeigt. Das hier vorgestellte Script soll der Veranschaulichung der Machbarkeit eines automatischen Abgleichs mit WoRMS dienen und erhebt kein Anspruch auf Vollständigkeit.

5.5 Erzeugung einer Artenliste

Aufgrund der fehlenden Transaktionszeit kann die Variante 1 aus Abschnitt 4.5 nicht verwendet werden. Das bedeutet, dass die „taxa_list_has_taxa_hierarchy“-Relation nicht erzeugt werden muss. Folgendes SQL-Statement erstellt dagegen die benötigte „taxa_list“-Relation:

```
CREATE TABLE taxa_list (
  id INT(10) NOT NULL AUTO_INCREMENT,
  name VARCHAR(45),
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id)
)
```

Das dynamische Erzeugen der Artenliste mittels Variante 2 (vgl. Abschnitt 4.5) erfolgt durch folgende SQL-Anweisung:

```
SELECT th.*
FROM taxa_hierarchy AS th, taxa_list AS tl
WHERE th.vt_begin <= tl.created_at
AND th.vt_end > tl.created_at
AND tl.id = 2 -- 'version 2'
```

Die CREATE-Anweisung, Beispieldaten sowie zwei Beispiel-Abfragen sind in der Datei „create_taxa_list.sql“ zu finden.

6 Bewertung

In diesem Kapitel werden das Konzept und die Umsetzung kritisch betrachtet. Es geht auf die Vor- und Nachteile der genutzten Methodiken ein. Noch bestehende Probleme werden aufgezeigt und Alternativen angesprochen.

6.1 Konzept

Das vorgestellte Konzept versucht durchgängig den SQL:2011-Standard zu nutzen. Da dieser vergleichsweise neu und aktuell zum Zeitpunkt der Erstellung dieser Arbeit ist, findet er wenig bis gar keine Unterstützung in aktuellen Datenbank-Produkten. Dadurch kommt es bei der Umsetzung zu Schwierigkeiten.

Die hybride Variante des Hierarchie-Modells funktioniert fehlerlos, obwohl das Modell der Adjazenz-Liste in [Kar10b] als ein Anti-Pattern aufgeführt wird. Dies impliziert, dass das Modell ein schlechter Lösungsansatz zum Abbilden von Hierarchien in relationalen Datenbanken darstellt. Nachteile seien unter anderem, das Finden aller Nachfolger und die allgemeine Navigation durch den Graph mittels prozeduralem Stil [Cel11]. Aber mit aufkommender Unterstützung der *Common Table Expressions* aus dem SQL:1999-Standard und damit auch der rekursiven Abfragen, werden die Nachteile abgeschwächt oder verlieren gar ihre Gültigkeit. Das Hinzufügen des Path-Enumeration-Modells fügt sich gut in die Anforderung eines Webinterfaces ein, da somit gleich alle *id*-Attribut-Werte der Vorgänger eines Taxons zur Verfügung stehen. Damit lässt sich leicht in der Weboberfläche zu den Vorgängern navigieren.

Vergleicht man das hier vorgestellte Konzept mit anderen verfügbaren Datenmodellen, z.B. ITIS[1] oder WoRMS[2], so verfolgt bisher keines von diesen einen temporalen Ansatz. Die Kopplung des Hierarchie-Modells mit der temporalen Komponente bringt jedoch auch Probleme mit sich. Das Fehlen korrekter Fremdschlüsselbeziehungen zerstört die durch das Adjazenz-Listen-Modell gegebene referentielle Integrität. Es können somit nur unzureichende oder teilweise gar keine Integritätsbedingungen eingesetzt werden. Dies ist aus Sicht der Datenbanktheorie bedenklich,

[1] Das Datenmodell kann für verschiedene Datenbanksysteme unter http://www.itis.gov/downloads/index.html heruntergeladen werden. (Aufrufzeitpunkt: 24.04.2015)
[2] Das Datenmodell ist unter http://www.marinespecies.org/structure/ zu finden. (Aufrufzeitpunkt: 24.04.2015)

weil es die Datenintegrität bzw. -konsistenz des Datenmodells gefährdet. Darüber hinaus ist das Konzept der Fremdschlüssel, sei es von nicht-temporalen zu temporalen Tabellen oder nur zwischen temporalen Tabellen, nicht vollkommen durchdacht. Während der Erstellung des Konzepts wurde mit IBM DB2 10.5 Express-C in einer virtuellen Maschine gearbeitet. Auch dort gelang es nicht, Fremdschlüsselbeziehungen auf dem *id*-Attribut und dem Gültigkeitszeitraum anzulegen. Bedenkt man, dass das Gültigkeitszeitintervall theoretisch geteilt und anschließend der referenzierte Datensatz nicht mehr eindeutig zugeordnet werden kann (siehe Abschnitt 4.5), ist das eine sinnvolle Einschränkung. IBM geht selber auf Fremdschlüsselbeziehungen mit temporalem Kontext in der Dokumentation [IBM15] nicht ein. Der Artikel [NS12] von IBM erläutert das Konzept temporaler Fremdschlüssel. Zusammenfassend ist zu sagen, dass es zwar grundlegende und vielversprechende Ansätze der temporalen Speicherung von Daten gibt, diese aber noch nicht ausgereift sind.

Durch die fehlenden Fremdschlüsselbeziehungen ist eine Artenliste nur lose an die eigentliche Taxonomie gekoppelt. Variante 1 aus Abschnitt 4.5 versucht eine Beziehung mittels Zuordnungstabelle herzustellen. Dies funktioniert nur unter der Bedingung, dass es nicht möglich sein soll, den Gültigkeitszeitraum eines Datensatzes im Nachhinein[3] zu verändern. Das bedeutet, dass die Datensätze in der Zuordnungstabelle referenziert werden können, obwohl keine Fremdschlüssel im Sinne der Datenbanktheorie existieren. Es können auch, wie weiter oben ausgeführt, keine Fremdschlüssel angelegt werden, da das Datenbanksystem nichts von dieser Einschränkung durch den Anwender wissen kann.

Variante 2 aus Abschnitt 4.5 basiert dagegen nur auf einer temporalen Abfrage auf die gesamte Hierarchie selbst. Es sind also keine eigenen physischen Datensätze für diese Art von Artenliste vorhanden. Sie wird auf Anfrage dynamisch anhand des Anlagedatums generiert. Diese Variante spart einerseits Speicherplatz, andererseits fehlt hier die logische Zuordnung zwischen einer Liste und deren Inhalt. Wird die Hierarchie-Relation durch ein externes Ereignis beschädigt und es kommt zum Datenverlust, wird das von Variante 2 nicht bemerkt. Variante 1 dagegen würde einige Datensätze zu viel besitzen, anhand derer man den Datenverlust bemerken würde und geeignete Maßnahmen ergreifen könnte. Als Zusammenfassung kann festgehalten werden, dass eine feste Kopplung, am besten mit Integritätsbedingungen durch Fremdschlüssel, bei der Abbildung und der Generierung einer Artenliste zu bevorzugen ist.

Viele temporale SQL-Anweisungen konnten in dieser Arbeit mit Hilfe der DB2 10.5 Express-C Datenbank von IBM nachvollzogen werden. Aufgrund dieser Nachvollziehbarkeit wurde bestätigt, dass das Konzept unter den oben genannten Einschränkungen grundlegend funktioniert. Verbesserungspotential besteht in den genannten Einschränkungen, also den fehlenden Fremdschlüsselbeziehungen und dem im Nachhinein unveränderlichen Gültigkeitszeitintervall.

[3] Im „Nachhinein“ bedeutet, dass das Gültigkeitszeitintervall eines angelegten Taxons, welches bereits in einer Artenliste referenziert wird, nicht mehr verändert werden darf.

6.2 Umsetzung

Das für die prototypische Umsetzung erforderliche Datenbanksystem ist MySQL. Dies ist die Vorgabe vom IOW im Zuge der Datenmigration. Es gibt MySQL in zwei Varianten: Open-Source und als kommerzielles Produkt. MySQL ist als Open-Source Version in der Webentwicklung weit verbreitet. Benutzt wurde Version 5.6.24.

Das Datenbanksystem bietet keine Unterstützung des SQL:2011-Standards. Das bedeutet, dass temporale Aspekte hier komplett fehlen. Es wurde deshalb versucht, mit Hilfe von [SGM00] mindestens die Gültigkeitszeit abzubilden. Dies stellte sich als sehr schwierig heraus. Allein für die Primärschlüsselbedingung der Hierarchie-Relation sind zwei Trigger notwendig, die bei jeder INSERT- und UPDATE-Anweisung prüfen, ob es zu Überschneidungen kommt. Trigger mussten hier zur Anwendung kommen, da MySQL keine ASSERTIONS oder Abfragen in CHECK-CONSTRAINTS unterstützt. Darüber hinaus war es notwendig, die einzelnen SQL-Anweisungen, die bei dem SQL:2011-Standard automatisch im Hintergrund ausgeführt werden, manuell zu implementieren. Diese Umsetzung führte zu verschiedenen Problemen, unter anderem, dass die Primärschlüsselbedingungen zwischen zwei Anweisungen, also während einer Transaktion, verletzt wurden, da die Trigger bei jeder einzelnen SQL-Anweisung auslösten. Trigger können in MySQL nicht deaktiviert und wieder aktiviert werden. Um die Prüfung der Primärschlüssel temporär zu deaktivieren, musste auf eine entsprechende Variable geprüft werden, welche markiert, ob der Trigger-Body ausgeführt werden soll oder nicht. Dieses Beispiel allein verdeutlicht die Unzulänglichkeiten, die es mit MySQL gibt. Die Umsetzung des Konzepts ist damit sehr aufwändig. Es entsteht schwer wartbarer Code, bestehend aus Stored Procedures, User Defined Functions und Triggern. Diese im Datenbanksystem verankerten prozeduralen Elemente bilden eine definierte Schnittstelle, um temporale Anweisungen auf der Datenbank auszuführen. Dies schränkt die Freiheit zur Benutzung beliebiger Abfragen seitens der Anwendung ein. Wird ein Datensatz nicht über diese Schnittstelle aktualisiert, kommt es zu Inkonsistenzen und falschen Daten. Daraus folgt die Gefährdung der gesamten Datenintegrität.

Fasst man die angesprochenen Punkte zusammen, ergibt sich die Schlussfolgerung, dass man die Wahl des Datenbanksystems überdenken sollte. Als Alternative, die den SQL:2011-Standard grundlegend unterstützt, ist zum Beispiel das kommerzielle System IBM DB2 zu nennen. Eine andere Möglichkeit wäre das Open-Source Datenbanksystem PostgreSQL, welches durch Anwender und Programmierer aus der Community Unterstützung erfährt. Es existieren bereits Ansätze, die temporalen Konzepte in das Datenbanksystem aufzunehmen bzw. durch Erweiterungen von Open-Source Entwicklern einzuführen (siehe auch Abschnitt 3.4).

7 Zusammenfassung und Ausblick

Dieses Kapitel fasst diese Arbeit noch einmal zusammen und beschreibt knapp die Vorgehensweise zur Erstellung eines Konzepts, welches in der Lage ist, Taxonomien und deren Änderungen über Zeiträume zu erfassen und zu verwalten. Der Ausblick wirft noch unbeantwortete Fragen auf und erstellt ein Fazit über die temporalen Konzepte des SQL:2011-Standards.

7.1 Zusammenfassung

Die aus der Biologie motivierten Stammbäume unterliegen stetigen Änderungen. Neue Forschungsergebnisse bedingen die ständige Anpassung dieser Taxonomien. Diese Arbeit liefert ein Konzept zur Pflege und Verwaltung solcher Taxonomien mit Versionierung über die Zeit. Sie zeigt die Möglichkeiten auf, wie als Ausschnitt dieser Taxonomie eine Taxa-Liste für beliebige Zeitpunkte generiert werden kann. Dazu wurden verschiedene Modelle, welche Baumstrukturen in relationalen Datenbanken abbilden, vorgestellt und verglichen. Die Wahl fiel auf ein hybrides Model, welches die Adjazenz-Liste mit dem Path-Enumeration-Modell koppelt. Um die Anforderung der Versionierung umzusetzen, wurde, aufbauend auf dem SQL:2011-Standard, das Hierarchie-Modell bitemporal, d.h. mit Gültigkeits- und Transaktionszeit, erweitert. Anhand der daraus entstehenden Relation wurde gezeigt, dass die typischen Operationen auf einer Taxonomie darauf ausgeführt werden können. Durch Verwendung eines durchgängigen Beispiels konnte nachvollzogen werden, welche Datensätze neu generiert, verändert und historisiert wurden. Es folgte die Vorstellung von zwei Varianten der Generierung der Artenliste anhand eines entworfenen Datenbank-Schemas. Die dabei auftretenden Probleme wurden herausgestellt und Lösungsmöglichkeiten aufgezeigt. Das Konzept wurde anschließend prototypisch, unter Verwendung des vorgegebenen MySQL-Datenbanksystems, implementiert.

Durch das Konzept wurde festgestellt, dass der SQL:2011-Standard vielversprechende, aber noch nicht komplett durchdachte, temporale Erweiterungen besitzt, die es ermöglichen, auf einfache Art und Weise temporale Aspekte in das relationale Datenbankmodell einzubringen. Aufgrund der fehlenden Fremdschlüsselbeziehungen durch die temporalen Eigenschaften verliert man aber die referentielle Integrität des Datenmodells. Weiterhin hat sich herausgestellt, dass sich MySQL nur bedingt eignet, um eine temporale Version einer Taxonomie in einer relationalen Datenbank abzubilden. Es fehlt an Unterstützung des SQL:2011-Standards, so dass dieser im Rahmen der

Möglichkeiten im prozeduralen Stil wiedergegeben werden muss.

Zusammenfassend ist festzuhalten, dass das Konzept eine Lösung der temporalen Speicherung von Taxonomien liefert, es aber bei der Umsetzung Probleme bereitet. Selbst DB2 von IBM, mit denen die Beispieldaten aus dem Konzept generiert wurden, liefert keine Lösung des Fremdschlüsselproblems.

7.2 Ausblick

Auf Basis des optional mitgeführten *t_ op*-Attributs aus Abschnitt 4.3, sind noch einige wichtige Fragen offen:
Kann mithilfe des *t_ op*-Attributs eine genaue Nachverfolgung der Änderungen und der Operationen, die zu diesen Änderungen führten, vollzogen werden? Wenn ja, lassen sich Taxonomien aus unterschiedlichen Zeiträumen vergleichen und die Unterschiede darstellen? Falls nicht, was fehlt an Informationen? Kann das Prinzip der Nachverfolgung auf andere, nicht biologische, Daten angewendet werden? Wenn ja, auf welche?

Mit DB2 als Referenz von IBM und den damit gesammelten Erfahrungen während der Erstellung dieser Arbeit ist festzuhalten, dass die temporalen Konzepte grundlegend funktionieren. IBM arbeitet hierbei mit eigenen Tabellen speziell für die historisierten Datensätze, so dass aktuelle Daten nur in der eigentlichen Tabelle zu finden sind. Dies erhöht die Übersichtlichkeit enorm. Durch die temporalen Klauseln in den SQL-Abfragen war es ohne Probleme möglich, die Tabellen temporal zu manipulieren.

Der SQL:2011-Standard definiert zwar temporale Aspekte und integriert diese in den alten SQL-Standard, er hinterlässt aber noch viele offene Probleme [DDL14], die in den nächsten Versionen des Standards geklärt werden müssen. Es fehlt zum Beispiel noch ein wichtiger Bestandteil: Integritätsbedingungen in Form von Fremdschlüsseln. Diese sind grundlegende Eckpfeiler eines jeden Datenbankprodukts. Schlussendlich darf man gespannt sein, wie die Entwicklung temporaler Konzepte in SQL verläuft und was die Forschung an temporaler Datenhaltung oder ganzer temporaler Datenbanken hervorbringt.

Literaturverzeichnis

[All83] Allen, James F.: Maintaining Knowledge About Temporal Intervals. In: *Commun. ACM* 26 (1983), November, Nr. 11, 832–843. http://dx.doi.org/10.1145/182.358434. – DOI 10.1145/182.358434. – ISSN 0001–0782

[Cat15] Catalogue of Life: *Frequently Asked Questions | Catalogue of Life*. http://www.catalogueoflife.org/content/frequently-asked-questions#4. Version: April 2015

[Cel04] Celko, Joe: *Joe Celko's Trees and Hierarchies in SQL for Smarties*. 1. Aufl. San Francisco, Calif : Morgan Kaufmann, 2004. – ISBN 978–0–080–49169–1

[Cel11] Celko, Joe: *Joe Celko's SQL for Smarties - Advanced SQL Programming*. 4. Auflage. Amsterdam : Elsevier, 2011. – ISBN 978–0–123–82022–8

[DDL14] Date, C.J. ; Darwen, Hugh ; Lorentzos, Nikos: *Time and Relational Theory - Temporal Databases in the Relational Model and SQL*. 2. Aufl. San Francisco, Calif : Morgan Kaufmann, 2014. – ISBN 978–0–128–00675–7

[Fre04] Freudig, Doris: *Lexikon der Biologie : in fünfzehn Bänden*. Spektrum, 2004. – ISBN 978–3–7653–4126–7

[GDS15] Glück, F. ; Darr, A. ; Schiele, K.: *Leibniz-Institut für Ostseeforschung Warnemünde - Was ist Benthos? - AG Ökologie benthischer Organismen*. Poster, Februar 2015

[IBM15] IBM Corporation: *IBM Knowledge Center - Time Travel Query (TTQ) über temporale Tabellen*. http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.1.0/com.ibm.db2.luw.admin.dbobj.doc/doc/c0058476.html. Version: April 2015

[Kar10a] Karwin, Bill: *Models for Hierarchical Data with SQL and PHP*. Präsentation/PDF. http://www.slideshare.net/billkarwin/models-for-hierarchical-data. Version: Mai 2010

[Kar10b] Karwin, Bill: *SQL Antipatterns - Avoiding the Pitfalls of Database Programming*. 1. Aufl. Pragmatic Bookshelf, 2010. – ISBN 978–1–934–35655–5

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

[KZN00] Kraus, O. ; Zoological Nomenclature, International C.: *Internationale Regeln für die Zoologische Nomenklatur: angenommen von International Union of Biological Sciences : offizieller deutscher Text*. Goecke & Evers, 2000 (Abhandlungen des Naturwissenschaftlichen Vereins in Hamburg). – ISBN 978–3–93137–436–5

[Lei15a] Leibniz-Institut für Ostseeforschung Warnemünde: *Arbeitsgruppe Ökologie benthischer Organismen - Arbeitsschwerpunkte*. http://www.io-warnemuende.de/arbeitsschwerpunkte.html. Version: Februar 2015

[Lei15b] Leibniz-Institut für Ostseeforschung Warnemünde: *Das Leibniz-Institut für Ostseeforschung Warnemünde*. http://www.io-warnemuende.de/de_index.html. Version: Februar 2015

[Lin59] Linné, Carl von, and Salvius, Lars: *Caroli Linnaei...Systema naturae per regna tria naturae: secundum classes, ordines, genera, species, cum characteribus, differentiis, synonymis, locis. Editio decima, Reformata.*. Bd. v.2. Holmiae, Impensis Direct. Laurentii Salvii, 1759. – 564 S. http://www.biodiversitylibrary.org/item/10278. – http://www.biodiversitylibrary.org/bibliography/542

[Mey15] Meyer, Frank: *Verwaltung von Taxonomien in objektrelationalen Datenbanken*, Universität Rostock, Literaturarbeit, 2015

[MyS15a] MySQL / Oracle Corporation: *MySQL :: MySQL 5.6 Reference Manual :: 13.1.17 CREATE TABLE Syntax*. http://dev.mysql.com/doc/refman/5.6/en/create-table.html. Version: April 2015

[MyS15b] MySQL / Oracle Corporation (reported by Stuart Friedberg): *MySQL Bugs: #16244: SQL-99 Derived table WITH clause (CTE)*. https://bugs.mysql.com/bug.php?id=16244. Version: April 2015

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

[Pet13] Petković, Dušan: Was lange währt, wird endlich gut: Temporale Daten im SQL-

Standard. In: *Datenbank-Spektrum* 13 (2013), Nr. 2, 131-138. http://dx.doi.org/10.1007/s13222-013-0120-3. – DOI 10.1007/s13222–013–0120–3. – ISSN 1618–2162

[Pos15a] PostgreSQL: *SQL2011Temporal - PostgreSQL wiki*. https://wiki.postgresql.org/wiki/SQL2011Temporal. Version: April 2015

[Pos15b] PostgreSQL Extension Network: *temporal_ tables: Temporal Tables Extension / PostgreSQL Extension Network*. http://pgxn.org/dist/temporal_tables/. Version: April 2015

[Pos15c] PostgreSQL Global Development Group (Vlad Arkhipov): *Temporal features in PostgreSQL*. http://www.postgresql.org/message-id/50D99278.3030704@dc.baikal.ru. Version: April 2015

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

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

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

[TS06] Türker, Can ; Saake, Gunter: *Objektrelationale Datenbanken - ein Lehrbuch*. 1. Aufl. Köln : Dpunkt-Verlag, 2006. – ISBN 978–3–898–64190–6

[ZGDS15] Zettler, M.L. ; Glück, F. ; Darr, A. ; Schiele, K.: *Leibniz-Institut für Ostseeforschung Warnemünde - Bodenlebende Wirbellose als Indikatoren in der Umweltüberwachung - AG Ökologie benthischer Organismen*. Poster, Februar 2015

[Zie14] Zierke, Jessica: *Konzeption der Datenintegration für eine zu entwickelnde Benthos-Datenbank*, Universität Rostock, Masterarbeit, 2014

A Anhang

A.1 Hierarchie-Modelle

A.1.1 Beispiel-Taxonomie

Die Beispiel-Hierarchie aus der Literaturarbeit [Mey15]:

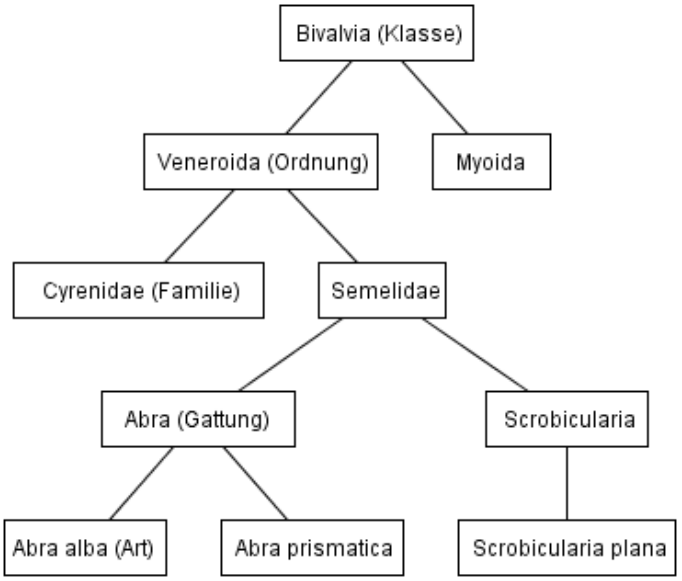

Abbildung A.1: Vereinfachte Systematik der Muscheln (Bivalvia)

A.1.2 Verschachtelte Mengen / Nested-Sets

Ablauf der modifizierten Tiefensuche:

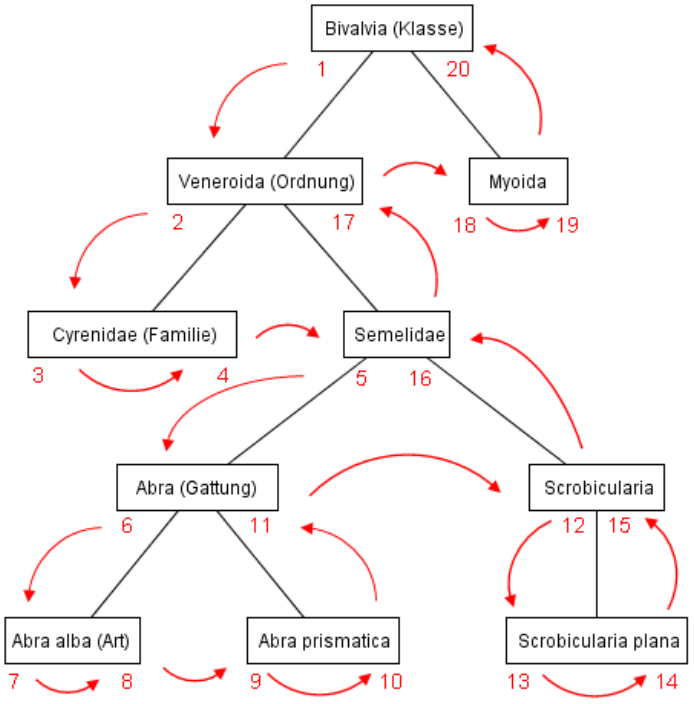

Abbildung A.2: Modifizierte Tiefensuche zur Ermittlung der Nummerierung

A.2 Temporale Daten

A.2.1 Gültigkeits-Tabellen

Die Abfrage nach aktuell gültigen Daten in IBM DB2 Syntax geschieht durch die FOR BUSINESS_TIME AS OF-Klausel:

```
SELECT *
FROM taxa_names
FOR BUSINESS_TIME AS OF '9999-12-31'
WHERE id = 4711
```

A.3 CD

Dieser Arbeit liegt eine CD mit der Beschriftung „BA Frank Meyer“ bei. Darauf ist folgende Ordnerstruktur (nur erste Ebene) vorhanden:

Literatur

Einige, aber nicht jede benutzte Quelle aus dieser Arbeit. Zusätzlich alle erwähnten Webseiten als PDF Version mit daraus ersichtlichem Zugriffszeitpunkt.

Materialien

In diesem Verzeichnis befinden sich Bilder sowie die yed-Dateien des frei verfügbaren yEd Graph Editors[1], aus denen die Grafiken erstellt wurden.

Umsetzung

Alle am Umsetzungskapitel 5 beteiligten Dateien sind in diesem Verzeichnis zu finden.

[1] http://www.yworks.com/

Selbstständigkeitserklärung

Hiermit erkläre ich an Eides statt, dass ich

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

Rostock, d. 27.04.2015 _____

Frank Meyer