Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Philip Hundt

Relationale Intervallbäume über dynamischen Trägermengen in skalierbaren Datenbanken

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Philip Hundt

Relationale Intervallbäume über dynamischen Trägermengen in skalierbaren Datenbanken

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Dr.-Ing. Sabine Schumann

Eingereicht am: 19. November 2015

Philip Hundt

Thema der Arbeit
Relationale Intervallbäume über dynamischen Trägermengen in skalierbaren Datenbanken

Stichworte
RI-Tree, Relationaler Intervallbaum, Intervalle, Relationale Datenbanken, Zeitintervalle

Kurzzusammenfassung
Diese Arbeit befasst sich mit der Indizierung von Intervallen in relationalen Datenbankmanagementsystemen mit Hilfe des relationalen Intervallbaumes. Es wird der besondere Fall von sich verschiebenden Trägermengen wie z.B. einem Zeitfenster der letzten X Wochen betrachtet.

Philip Hundt

Title of the paper
Relational Interval Trees over a dynamic data space in scalable Databases

Keywords
RI-Tree, Relational Interval Tree, Intervals, Relational Databases, Time Intervalls

Abstract
This work is concerned with the indexing of intervals in relational database management systems using the relational interval tree. The special case of shifting dataspace, for example a time window of the last X weeks, is considered.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

DB	Datenbank
DBMS	Datenbankmanagementsystem
IST	Interval-Spatial Transformation Index
ORDBMS	Objektrelationales Datenbankmanagementsystem
PL/SQL	Procedural Language/Structured Query Language
RDBMS	Relationales Datenbankmanagementsystem
RI-Baum	Relationaler Intervallbaum
SQL	Structured Query Language

1 Einleitung

1.1 Motivation

In den letzten Jahren hat die Menge an gespeicherten und zu verarbeitenden Daten in allen Bereichen der Informatik weiter unvermindert zugenommen. Sowohl in der Wirtschaft als auch in der Wissenschaft ist daher die effektive Handhabung großer Datenmengen ein aktuelles Thema. Eines der Teilprobleme ist dabei die Laufzeit von Anfragen auf großen Datenmengen. Um diese zu beschleunigen hat sich der Einsatz von Indexstrukturen bewährt.

Ein häufig in Datentypen abgebildetes Konstrukt aus der Mathematik sind Intervalle. Diese werden dazu genutzt, verschiedenste Sachverhalte darzustellen. Dies reicht von Zeitintervallen zur Darstellung von Zeiträumen bis hin zu domänenspezifischen Interpretationen wie Frequenzbereichen oder Distanzmaßen.
Eine der wichtigen Relationen zwischen Intervallen ist die Überlappung von Intervallen. So kann bei Zeitintervallen interessant sein, welche Intervalle einen bestimmten Zeitabschnitt betreffen. Diese Art von Abfragen ist allerdings aufwändig, da sowohl das untere, als auch das obere Ende der Intervalle berücksichtigt werden muss.
Gerade im Bereich der Massendaten kann die Beantwortung dieser Anfragen dadurch länger dauern, als es die Systemanforderungen zulassen. Daher gibt es verschiedene Indexkonzepte, die auch das Finden von Überlappungen auf Intervallen beschleunigen sollen.
Eine dieser Indexstrukturen ist der Intervallbaum von Edelsbrunner [Edelsbrunner, 1980]. Um die Vorteile des Intervallbaumes auch auf bestehenden relationalen Datenbankmanagementsystemen nutzen zu können, wurde das Konzept des relationalen Intervallbaumes [Kriegel u. a., 2000] entworfen.

Ist der Bereich, aus dem die Intervalle stammen, nicht statisch sondern dynamisch, entstehen jedoch Probleme. So ist es am Beispiel der Zeitintervalle ungünstig, alle Zeitintervalle, die innerhalb der letzten x Wochen liegen, zu speichern. Der Bereich, in dem die Intervalle liegen, verschiebt sich kontinuierlich auf der Zeitachse. In solchen Szenarien entsteht beim

relationalen Intervallbaum schnell ein nicht unerheblicher Verwaltungsaufwand.
Im Bereich von Sensordaten entsteht regelmäßig die Situation, dass die Daten nur eine bestimmte Zeit von Interesse sind und die Daten, auf Grund der großen Menge, nicht länger als unbedingt nötig gespeichert werden. Es werden daher, regelmäßig Daten, die ein bestimmtes Alter überschritten haben, gelöscht. Gleichzeitig fallen dauerhaft aktuelle Daten an. Dabei muss sichergestellt sein, dass neue Daten sofort in Abfragen berücksichtigt werden. Im Beispiel des Zeitfensters müssen auch Anfragen der aktuell letzten Minute konsistent beantwortet werden. Die Menge an neu hinzukommenden Daten kann dabei enorm sein. So gibt es Systeme, die pro Minute eine hohe fünfstellige Anzahl an Intervallen persistieren müssen.

Eine Nicht-Verfügbarkeit der Systeme ist oft keine Option. Einige Systeme müssen in kurzer Zeit auf Ereignisse reagieren, so dass auch das Zwischenspeichern von neuen Daten in Puffern während solcher Wartungsphasen nicht den Anforderungen genügt. Zusätzlich müssen auch die neuesten Datensätze jederzeit konsistent angefragt werden können.

1.2 Ziel der Arbeit

In dieser Arbeit wird eine Möglichkeit erarbeitet, Überlappungen bei einer großen Anzahl an Intervallen, in einem hoch dynamischen Kontext, bestimmen zu können. Ein besonderes Augenmerk liegt auf dem Anwendungsfall der sich kontinuierlich verschiebenden Trägermenge, am Beispiel eines Zeitfensters. In solch ein Fenster fließen beständig neue Intervalle ein, während hinten Intervalle wegfallen. Das Augenmerk wird dabei auf die Anpassung des bestehenden relationalen Intervallbaums gelegt. Der überarbeitete relationalen Intervallbaums soll auf längere Umstrukturierungsphasen verzichten und neue Intervalle sofort berücksichtigen, sowie weggefallene Intervalle nicht weiter berücksichtigen. Die mögliche Integration in bestehende RDBMS wird dabei als Kernvoraussetzung für die Praxistauglichkeit des Konzeptes betrachtet.

Neben dem relationalen Intervallbaum gibt es weitere Konzepte, die zum Indizieren von Intervallen geeignet sind. Diese werden nicht Teil dieser Arbeit sein. Für vergleichende Tests zum relationalen Intervallbaum sei auf „The Relational Interval Tree: Manage Interval Data Efficiently in Your Relational Database“ [Kriegel u. a., 2000] verwiesen. Die Autoren geben in ihrer Arbeit auch einen Überblick über andere Ansätze zur Indizierung von Intervallen und

führen vergleichende Tests mit Implementationen der Konzepte durch. Außerdem wird sich auf das Handling von Intervallen in relationalen Datenbankmanagementsystemen beschränkt.

1.3 Gliederung der Arbeit

In Kapitel 2 wird auf die für diese Arbeit benötigen Grundlagen eingegangen. Dies umfasst Intervalle (2.1) und relationale Datenbankmanagementsysteme (2.2). Im Zuge relationaler Datenbankmanagementsysteme werden auch Indizes und das Konzept der Partitionierung erläutert werden.

Anschließend erfolgt in Kapitel 3 eine Analyse der in dieser Arbeit betrachteten Problematik. Zu Beginn wird kurz auf den Einsatz von Intervallen in Informationssystemen (3.1) eingegangen. Darauf hin wird sich mit Indexkonzepten zum Indizieren von Intervallen befasst. Dafür werden der Intervallbaum (3.2) und der relationale Intervallbaum (3.3) detaillierter betrachtet. Es wird das Problem einer wandernden Trägermenge erläutert (3.5) und aufbauend auf der Erläuterung der Einsatz des relationalen Intervallbaums im Kontext dieses Problems geprüft (3.6). Aufbauend auf den Ergebnissen dieser Prüfung werden Anforderungen und Ziele für einen an das Problem angepassten relationalen Intervallbaum formuliert (3.7).

Im Kapitel 4 wird eine Anpassung des bestehenden relationalen Intervallbaums an die in Kapitel 3 erläuterte Problematik, erfolgen. Dazu wird es ermöglicht, den relationalen Intervallbaum frei im Wertebereich eines Datentyps zu platzieren (4.1). Anschließend wird darauf aufbauend eine Möglichkeit gezeigt, wie eine wandernde Trägermenge durch die Änderung unterstützt werden kann (4.2).

In Kapitel 6 wird die Implementation des angepassten RI-Baumes in verschiedenen Tests einer Evaluation unterzogen.

Abschließend wird in Kapitel 7 eine Zusammenfassung (7.1) der Inhalte der Arbeit erfolgen und ein Ausblick (7.2) auf mögliche zukünftige Entwicklungen und noch offene Forschungsfragen gegeben.

2 Grundlagen

Bevor sich in den nächsten Kapiteln mit relationalen Intervallbäumen beschäftigt wird, sollen in diesem Kapitel die dafür notwendigen Grundlagen erläutert werden. Leser, die bereits mit Intervallen und relationalen Datenbankmanagementsystem vertraut sind, können dieses Kapitel überspringen.

2.1 Intervalle

Intervalle sind Teilmengen geordneter Körper.
„Sei K ein angeordneter Körper und seien $a \leq b$ Elemente von K. Das abgeschlossene Intervall [a, b] ist die Menge $[a, b] = \{x \in K : a \leq x \leq b\}$. [Deitmar, 2014, S. 39]
Das Intervall wird durch seine Schranken[1] beschrieben und enthält diese, sowie alle in der Ordnung zwischen den Schranken liegenden Elemente. Der total geordnete Körper, dessen Teilmengen die Intervalle sein können, wird als Trägermenge bezeichnet. Die Mächtigkeit einer Trägermenge beschreibt die Anzahl der enthaltenen Elemente.

In Abbildung 2.1 sind zur Visualisierung der Begriffe die Intervalle [2,4], [5,9] und [8,10] über der Trägermenge $M = \{x \wedge x \in \mathbb{R} \wedge x \geq 1 \wedge x \leq 10\}$ abgebildet. Die Trägermenge hat in dem Beispiel eine Mächtigkeit von 10.

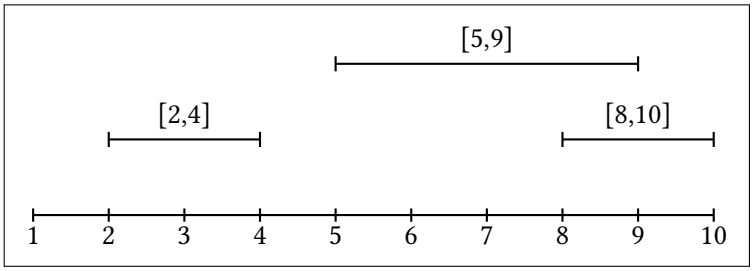

Abbildung 2.1: Drei Intervalle über der Trägermenge: 1 bis 10

[1] In Literatur wird teilweise auch der Begriff Grenzen statt Schranken verwendet.

Neben Intervallen über die klassischen, mathematischen Körper, wie den relationalen Zahlen, sind auch Intervalle über total geordnete Mengen möglich. Beispielsweise Zeitintervalle als Teilmengen der Trägermenge aller Millisekunden vom 01.01.2000 00:00:00 bis zum 31.12.2010 23:59:59.

Bei allen Berechnungen, Formeln und Codebeispielen, die zu dieser Arbeit gehören, wird von geschlossenen Intervallen ausgegangen. Dies bedeutet, dass die Schranken des Intervalls im Intervall enthalten sind. (Notation: „$[a, b]$“) Dazu abgegrenzt gibt es noch offene Intervalle, bei denen die Schranken selbst nicht mehr Teil des Intervalls sind. (Notation: „(a, b)“) Halboffene Intervalle bezeichnen eine Notation, bei der eine Schranke (obere oder untere) Teil des Intervalls ist und die andere Schranke nicht. (Notation: „$(a, b]$“ und „$[a, b)$“)
Ein Intervall kann auch unbeschränkt sein, die entsprechende Schranke wird dann als unendlich notiert. Die Angabe der als unendlich definierten Schranke erfolgt dabei in der offenen Form der Art $[x, \infty)$. (vgl. [Deitmar, 2014, S. 39 ff.]) Alle Formeln, Berechnungen und Beispiele dieser Arbeit gehen davon aus, dass die Intervalle in geschlossener Form angegeben werden. Vergleiche und andere Berechnungen, welche die Schranken betreffen, müssen für die Verwendung der offenen Form entsprechend angepasst werden. Geschlossene Intervalle, deren obere und untere Schranke gleich sind, werden als Punktintervalle bezeichnet.

Intervalle können mit anderen Intervallen auf Relationen wie die Überlappung, Gleichheit, „liegt vor“, „liegt hinter“ und „liegt in“ untersucht werden.
Für diese Arbeit besonders wichtig ist dabei die Frage der Überlappung. Eine Überlappung liegt vor, wenn die zu vergleichenden Intervalle mindestens ein gemeinsames Element haben. Mathematisch lässt sich Intervall [a,b] überlappt Intervall [c,d] als $\exists x : x \in [a, b] \wedge x \in [c, d]$ beschreiben. In Abbildung 2.1 Überlappen sich die Intervalle [5,9] und [8,10] im Bereich von 8 bis 9. Die Überlappungsrelation ist reflexiv und symmetrisch. Jedes Intervall überlappt mit sich selbst und es gilt: Wenn [a,b] mit [c,d] überlappt dann gilt auch [c,d] überlappt mit [a,b].

2.2 Relationale Datenbankmanagementsysteme

„Ein Datenbanksystem besteht aus einer Datenbank und einem Datenbankmanagementsystem (DBMS) und dient zur Speicherung und Verwaltung von Daten. Als Datenbank wird der strukturierte, von einem DBMS verwaltete Datenbestand bezeichnet. Das DBMS enthält die Struktur in der die Daten abgelegt werden und umfasst die gesamten Softwaremodule zur

Verwaltung der Datenbank.“ [Saake u. a., 2013, S. 8 f.]

Datenbanken entkoppeln somit die Daten von einzelnen Applikationen und speichern diese zentral in vereinheitlichter Form. Dies hat den Vorteil, dass die selben Daten einfacher für verschiedene Anwendungen verwendet werden können, doppelte Speicherung/Verwaltung für mehrere Anwendungen entfällt und der Lebenszyklus der Daten vom Lebenszyklus der Software entkoppelt wird.
Die physische Verwaltung der Daten wird durch ein Datenbankmanagementsystem erledigt, über das auch alle Zugriffe und Änderungen an den Daten in der Datenbank erfolgen. Das verbreitetste DBMS-Modell ist das relationale Modell, welches 1970 durch Edgar F. Codd vorgestellt wurde und den „De-facto-Standard“ (vgl. [Vossen, 2008]) für DBMS darstellt. Das relationale Modell basiert dabei auf mathematischen Relationen.
Die visuelle Darstellung erfolgt in Tabellenform. Alle Einträge in einer Spalte entsprechen dabei einem für die Spalte vergebenen Datentyp. Ein Vorteil der Verwendung des relationalen Modells ist seine Beweisbarkeit. Da auch die auf den Daten angebotenen Operationen eines RDBMS den mathematischen Operationen auf Relationen entsprechen, können die vielfach bereits durch Mathematiker geführten Beweise und Beweisverfahren analog angewendet werden. Dies erleichtert die Optimierung der Ausführung.

Die Steuerung des RDBMS erfolgt über eine von der ANSI in mehreren Versionen normierten Sprache: der Structured Query Language (SQL). Die Umsetzung der SQL auf einzelnen RDBMS entspricht dabei meistens nicht genau den ANSI Spezifikationen, sondern ist an diese angelehnt. SQL ist eine deklarative Sprache, allerdings bieten viele RDBMS mittlerweile prozedurale Erweiterungen der SQL. Im Zuge dieser Arbeit wird als prozedurale SQL-Erweiterung die Sprache PL/SQL von Oracle für das Oracle-DBMS verwendet.

2.2.1 Partitionierung

Partitionierung beschreibt im Bereich der RDBMS das Unterteilen von Tabellen in Teiltabellen. Die Tabelle wird dabei horizontal in mehrere Teiltabellen unterteilt. Jede Zeile wird über den Partitionierungsschlüssel genau einer Partition zugeordnet. Der Partitionierungsschlüssel besteht aus einer oder mehren Spalten deren Wert/Werte das Kriterium für die Zuordnung zu einer Partition darstellt/darstellen. Beispielsweise könnte in einer Tabelle mit Personen Einträgen das Geburtsjahr der Partitionierungsschlüssel sein und über die Zuordnung zu einer Partition bestimmen. Die Größe der Partitionen kann dabei frei gewählt werden, so dass für jedes Jahr oder auch für jedes Jahrzehnt eine andere Partition gewählt werden kann.

Mittels Partitionierung können Anfragen, die den Partitionsschlüssel enthalten, häufig auf eine Partitionen beschränkt werden. Partitionen sind im Vergleich zur gesamten Tabelle meist deutlich kleiner. Außerdem ist es möglich, die Partitionen getrennt voneinander zu verwalten. Dies bedeutet unter anderem, dass es möglich ist, einzelne Partitionen zu sperren, während andere Partitionen der selben Tabelle weiter verfügbar sind. Dies erhöht die mögliche Parallelität.
Ein weiter Vorteil ist auch der deutlich geringere Aufwand zum Löschen einer Partition im Vergleich zum Löschen der gleichen Zeilen aus einer großen unpartitionierten Tabelle. Gerade bei regelmäßig zu löschenden Teilen des Datenbestandes hat es sich daher bewährt, nach dem Löschkriterium zu partitionieren und so den Aufwand für das Löschen zu minimieren.

Einige RDBMS unterstützen auch Partitionierung über mehrere Ebenen. Dies bedeutet, dass Partitionen der ersten Ebene wiederum über einen weiteren Partitionsschlüssel aufgeteilte Partitionen enthalten. Partitionierung wird allerdings nicht von allen RDBMS unterstützt und kostet in manchen RDBMS zusätzliche Lizenzgebühren.

2.2.2 Indizes

Datenbank Indizes sind Strukturen, welche neben dem eigentlichen Datenbestand bestehen und das Auffinden von Elementen des Datenbestandes beschleunigen sollen, indem die Anzahl benötigter Ein- und Ausgabe-Operationen (I/O) reduziert wird.
Dies ist möglich, da Indexstrukturen die Daten in einer Struktur ordnen, welche das Auffinden gesuchter Daten erleichtert. Diese Einsparung geht zu Lasten von Schreiboperationen, da die Pflege des Index Aufwand erzeugt.

Indizes sind elementarer Bestandteil von RDBMS, und es werde verschiedene Indexstrukturen angeboten. Am verbreitetsten ist die Verwendung von B+-Bäumen.[2] Dies sind B-Bäume, bei denen die Werte nur in Blättern gespeichert sind. Für eine ausführliche Erläuterung zur Funktion des B+-Baumes, wie er in dem für diese Arbeit verwendeten Oracle 11.2g RDBMS implementiert wurde, siehe [Oracle, 2015].

Indizes können auch partitioniert werden. Besonders von Bedeutung ist dies, wenn Indizes über den selben Partitionierungsschlüssel und das selbe Partitionierungskriterium wie die Tabelle, zu der sie gehören, partitioniert werden. Dann besteht eine eins-zu-eins Beziehung zwischen den Partitionen der Tabelle und den Partitionen des Index.

[2] Auch beim B+-Baum ist das „-“als Bindestrich und nicht als Minus zu interpretieren.

Bei Anfragen mit dem Partitionierungsschlüssel als Kriterium, muss nur die passende Indexpartition betrachtet werden. Dies reduziert auch den Aufwand der Indexpflege. Wird eine Tabellenpartition geändert muss auch nur die dazugehörige Indexpartition geändert werden. Mit dem Löschen einer Tabellenpartition kann auch die dazugehörige Indexpartition gelöscht werden, ohne das die anderen Indexpartitionen angepasst werden müssen.

3 Analyse

Im Folgenden soll der Einsatz von Intervallen in Informationssystemen erläutert werden und eine für den Einsatz auf RDBMS geeignete Indexstruktur für Intervalle vorgestellt werden. Im Anschluss wird das in dieser Arbeit besonders betrachtete Problem einer wandernden Trägermenge charakterisiert werden. Aufbauend darauf wird die Verwendung geeigneter, bestehender Indexstrukturen im Kontext dieses Problems analysiert.

3.1 Einsatz von Intervallen

Intervalle werden in der Praxis zur Darstellung verschiedenster Sachverhalte verwendet. Der verbreitest Anwendungsfall dürften dabei Zeitintervalle sein. Prinzipiell eignen sich Intervalle immer dann, wenn zusammenhängende Teile geordneter Mengen betrachtet werden. So können physikalische Größen wie Temperaturbereiche (10 bis 12°C), Frequenzenbereiche (1000-2000 Hz) oder noch stärkere domänenspezifische Einheiten betrachtet werden.

Zeitintervalle sind in vielen Domänen von Bedeutung, da sie Zeiträumen beschreiben und eine Zuordnung selbiger zu weiteren Daten ermöglichen. Dies kann vielfältig genutzt werden. Wird z.B. der Verlauf eines Wertes erfasst, der sich nicht bei jeder Erfassung ändert, ist es mit Hilfe von Zeitintervallen möglich, den Speicherbedarf erheblich zu senken. Dies ist möglich, indem nicht jede Messung als Kombination aus Messwert und Messzeitpunkt gespeichert wird, sondern für Zeiträume in denen der Wert konstant bleibt, nur ein Zeitintervall samt zugeordnetem Messwert gespeichert wird. Beispielsweise könnte in einem intelligenten Stromnetz ermittelt werden, welcher Verbraucher zu welchem Zeitpunkt welche Menge Strom verbraucht. Die Verbraucher würden regelmäßig den durch sie verbrauchten Stromverbrauch melden. Da jedoch nicht jede Messung einen anderen Verbrauchswert liefern wird, lassen sich die Verbrauchsmengen Zeiträumen zuordnen und so der Speicherplatzbedarf verringern.

Gerade in Szenarien, in denen Zeitintervallen Werte zugeordnet werden, benötigt man die Überlappungs- und „enthält“-Anfragen. Um den Wert zu einem Bestimmten Zeitpunkt zu bestimmen, wird der Zeitpunkt als Punktintervall betrachtet und alle Intervalle gesucht, in

denen dieser Punktintervall liegt. Im Beispiel des intelligenten Stromnetzes entspricht dies der Anfrage nach dem verbrauch zu einem Bestimmten Zeitpunkt. Werden alle aktiven Verbraucher innerhalb eines Zeitraumes gesucht, so wird die Frage zu einem Überlappungsproblem. Im Beispiel des intelligenten Stromnetzes müssen alle Zeitintervalle, die in den Suchzeitraum hineinreichen, gesucht werden. Es werden alle Verbraucher mit einer überlappenden Aktivität gesucht da wichtig ist, ob überhaupt Strom im Suchzeitraum abgegeben wurde.

In der Praxis erfolgt die Persistierung großer Datenmengen meistens in RDBMS. Dies gilt auch für Intervalldaten. Die Beantwortung von Überlappungsanfragen kann in RDBMS lange dauern, da sämtliche Intervalle geprüft werden müssen. So kann z.B. ein am unteren Ende der Trägermenge beginnendes Intervall eine obere Schranke haben, die das Intervall in ein Intervall nahe des oberen Endes der Trägermenge Überlappen lässt. Die Zeitanforderungen die an eine Überlappungsanfrage gestellt werden kann stark variieren.
Die Anforderungen an die Verarbeitungsdauer von Anfragen lassen sich in verschiedene Kategorien einteilen.

1. Keinerlei Zeitanforderung
2. Quasi-Echtzeit-Anforderungen
3. Weiche Echtzeit-Anforderungen
4. Feste Echtzeit-Anforderungen
5. Harte Echtzeit-Anforderungen

Als Beispiel für keinerlei Zeitanforderungen wäre es für die Erstellung einer Studie zur Verteilung des Stromverbrauchs über die Tageszeiten und den Wochenverlauf nicht dramatisch, wenn die Abfrage der Daten sogar Tage benötigt. Eine Optimierung ist auch hier für den Nutzer wünschenswert, aber die Studie kann auch erstellt werden, wenn die Ergebnisse erst nach einiger Zeit eintreffen.

Quasi-Echtzeit-Anforderungen werden im Zuge dieser Arbeit an weiche Echtzeitanforderungen angelehnt. Damit sind Systeme gemeint, deren Laufzeiten nicht völlig egal sind, die jedoch auch keine Zusicherung über Laufzeit treffen wie Echtzeit-Systeme dies tun. Ziel der Systeme ist, möglichst oft die Zeitanforderung zu erreichen ohne dabei Garantien dafür geben zu können. So gibt es viele Systeme die im Normfall Reaktionszeiten im Bereich weniger Sekunden erwarten allerdings keine Maßnahmen ergreifen um diese Zeiten auch zu garantieren.

Echtzeit-Systeme sollen wiederum die Einhaltung vorgegebener Laufzeiten für Berechnungen bzw. eine maximale Reaktionszeit auf Ereignisse garantieren. Bei einer weichen Echtzeit Anforderungen ist das Ergebnis nach Überschreiten der Zeitanforderung nicht sofort wertlos. Die geforderten Zeiten dürfen innerhalb einer definierten Toleranz überschritten werden. Eine automatische Lichtsteuerung wäre ein Beispiel für eine solche weiche Anforderung. Betritt jemand einen dunklen Raum, so sollte das Licht in einer für den Betretenden angenehmen Zeit eingeschaltet werden. Das System verliert jedoch nicht sofort seinen Nutzen, wenn in Einzelfällen eine kurze Verzögerung zu bemerken ist. Als weiterer Aspekt ist hier noch zu benenn, dass das Ergebnis mit dem Überschreiten der Zeit immer weiter an Wert verlieren kann.

Bei einer festen Echtzeit-Anforderung verliert das Ergebnis nach Ablauf der Zeit seinen Wert. Die Rechnung kann abgebrochen und verworfen werden. Die stärksten Anforderungen sind harte Echtzeit-Anforderungen. In dieser Klasse führt ein Überschreiten der Zeitanforderungen nicht nur dazu, dass das Ergebnis seinen Wert verliert, sondern ein Schadensfall eintritt. Diese Anforderung wird besonders oft im maschinennahen Bereich gestellt, wo verspätetes Reagieren zu realen Schäden an der Hardware führen kann. Im Extrembeispiel des Stromnetzes werden Harte Echtzeit-Anforderungen an die Steuerung der Kühlung eines Kraftwerkes gestellt. Ein nicht oder zu spätes Reagieren kann in einer Katastrophe enden. (vgl. [Kuhrt, 2013])

Diese Arbeit beschäftigt sich mit Problemen der Quasi-Echtzeit-Anforderung. Echtzeit-Probleme können mit den hier vorgestellte Mitteln nicht erfüllt werden. So sind die gängigen relationalen Datenbanken, die Grundlage der hier verwendeten Konzepte sind, nicht geeignet, um Echtzeitanforderungen zu erfüllen.

Auf RDBMS sind die in 2.2.2 bereits erläuterten Indizes zum Optimieren von Anfragen ein wichtiger Bestandteil. So gibt es auch Index-Konzepte, die das Auffinden von gesuchten Intervallen vereinfachen sollen. Das Problem besteht darin das viele dieser bestehenden Konzepte jedoch nicht ohne weiteres auf bestehenden RDBMS umgesetzt werden können. Bekannte Konzepte deren Entwurf auch eine Umsetzung auf bestehenden RDBMS ermöglicht sind:

- **Time Index** (vgl. [Elmasri u. a., 1990]) Der Time Index kann auf bestehenden RDBMS implementiert werden. Das Einfügen neuer Intervalle hat jedoch eine Komplexität von O(n) und das Löschen von Indexeinträgen von $O(n^2)$.

- **Tile Index** (vgl. [Oracle, 1999]) Beherrscht auch Überlappunsganfragen auf mehrdimensionalen Konstrukten. Der Tile Index ist auf bestehenden RDBMS umsetzbar. Er basiert auf dem Linear Quadtree.
- **Interval-Spatial Transformation** (vgl. [Goh u. a., 1996]) Interval-Spatial Transformation (IST) nutzt bestehende Index Konzepte. Für eindimensionale Intervalle erfolgt die Implementation als kombinierter Index mit (unter Schranke, obere Schranke) oder (obere Schranke, untere Schranke). Die Performance zum bestimmen von Überlappungen nimmt jedoch ab, je stärker die Auswahl von der zweiten Indexspalte abhängt.
- **Relationaler Intervallbaum** (vgl. [Kriegel u. a., 2000]) Der relationale Intervallbaum ist eine Weiterentwicklung des Intervallbaumes, um diesen auf bestehenden RDBMS verwenden zu können.

Diese Aufzählung erhebt keinen Anspruch auf Vollständigkeit und für genauere Informationen zu den Indizes sei auf die angegebenen Quellen verwiesen. Für weitere, jedoch nicht auf RDBMS umsetzbare Intervallindizes sei noch ein mal auf [Kriegel u. a., 2000] verwiesen.

Neben Faktoren wie der Geschwindigkeit und dem Speicherplatzbedarf können sich für die Praxistauglichkeit der Indizes verschiedene Problem ergeben. So gibt es statische Indizes, die darauf ausgelegt sind, eine statische Menge an Intervallen zu speichern, jedoch nicht auf das nachträgliche Hinzufügen und Entfernen von Intervallen ausgelegt sind. Andere Indexstrukturen können Teile der Operationen, wie z.B. das bestimmen von Überlappungen, nur sehr ineffektiv Lösen.

Aus der oben genannten Aufzählung, erscheint der RI-Baum als erfolgversprechendster Kandidat, zum Bestimmen von Überlappungen in einem dynamischen Kontext. Im Folgenden werden daher der Intervallbaum und der relationale Intervallbaum näher erläutert.

3.2 Intervallbäume

Eine der zentralen Grundlagen für diese Arbeit ist der 1980 von H. Edelsbrunner in „Dynamic Rectangle Intersection Searching “ [Edelsbrunner, 1980] vorgestellte Intervallbaum.
Der Intervallbaum ist ein auf mehreren Strukturen basierendes Konzept zur Indexierung von Intervallen, mit dem Ziel auch Überlappungen über den Index bestimmen zu können. Die Struktur teilt sich dabei in mehrere Teilstrukturen. Diese sind in Abbildung 3.1 dargestellt.
Ziel der Strukturen ist es, über die Einordnung in eine Baumstruktur bei der Bestimmung von

Überlappungen direkt einen großen Teil der nicht überlappenden Intervalle von der Betrachtung ausschließen zu können.

Die erste Struktur ist die Primärstruktur. Es existieren mehrere Instanzen der Primärstruktur. Um die erste Instanz zu bilden wird der Median aller Intervall-Schranken als Knoten der Primärstruktur gewählt. Diesem werden alle, den Knoten schneidenden, Intervalle zugeordnet. Jeder Knoten der Primärstruktur verweist auf eine eigene Instanz der Sekundärstruktur. In dieser Instanz der Sekundärstruktur werden alle Intervalle eingeordnet, die dem Knoten der Primärstruktur zugeordneten wurden. Die Sekundärstruktur kann durch verschiedene Strukturen repräsentiert werden.
Edelsbrunner schlägt als Sekundärstruktur eine weitere Baumstruktur vor. Diese wird, wie in Abbildung 3.1 gezeigt, als Binärbaum aus den Schranken aller dem Knoten zugeordneten Intervalle aufgebaut.

Der Aufbau der Sekundärstruktur erfolgt auf einer geordneten Liste der in ihr gespeicherten Schranken und nach folgenden Regeln:

- Wenn die Liste mehr als eine Schranke enthält und eine gerade Anzahl von Elementen enthält, wird der Durchschnitt der beiden mittleren Elemente als Knoten gewählt. Zum Bestimmen des linken Tochterknotens werden die Regeln auf die linke Hälfte der geordneten Liste angewendet. Zum Bestimmen des rechten Tochterknotens werden die Regeln auf die rechte Hälfte der geordneten Liste angewendet.
- Wenn die Liste mehr als eine Schranke und eine ungerade Anzahl von Elementen enthält, wird der Durchschnitt aus mittlerem und folgendem Elemente als Knoten gewählt. Zur Bestimmung des linken Tochterknotens werden die Regeln auf die linke Hälfte der Liste, inklusive des mittleren Elements, angewendet. Der rechte Tochterknoten wird, mit Hilfe der Regeln, auf Basis der rechten Hälfte, exklusive des mittleren Elements, ermittelt.
- Enthält die Liste nur ein Wert wird dieser als Blatt eingetragen. Im Blatt werden alle durch den Wert des Blattes beschränkten Intervalle vermerkt.

Nach dem Erzeugen des Baumes werden alle Blätter der Sekundärstruktur von links nach rechts verkettet. Dies entspricht einer doppelt-verkettenden Liste aller Schrankenwerte. Zusätzlich erhält die Wurzel der Sekundärstruktur Verweise auf die größte und kleinste Schranke (die Blätter ganz links und ganz rechts im Baum). Die Sekundärstruktur wird zum Bestimmen von Überlappungen später nur noch als doppelt-verkette Liste verwendet.

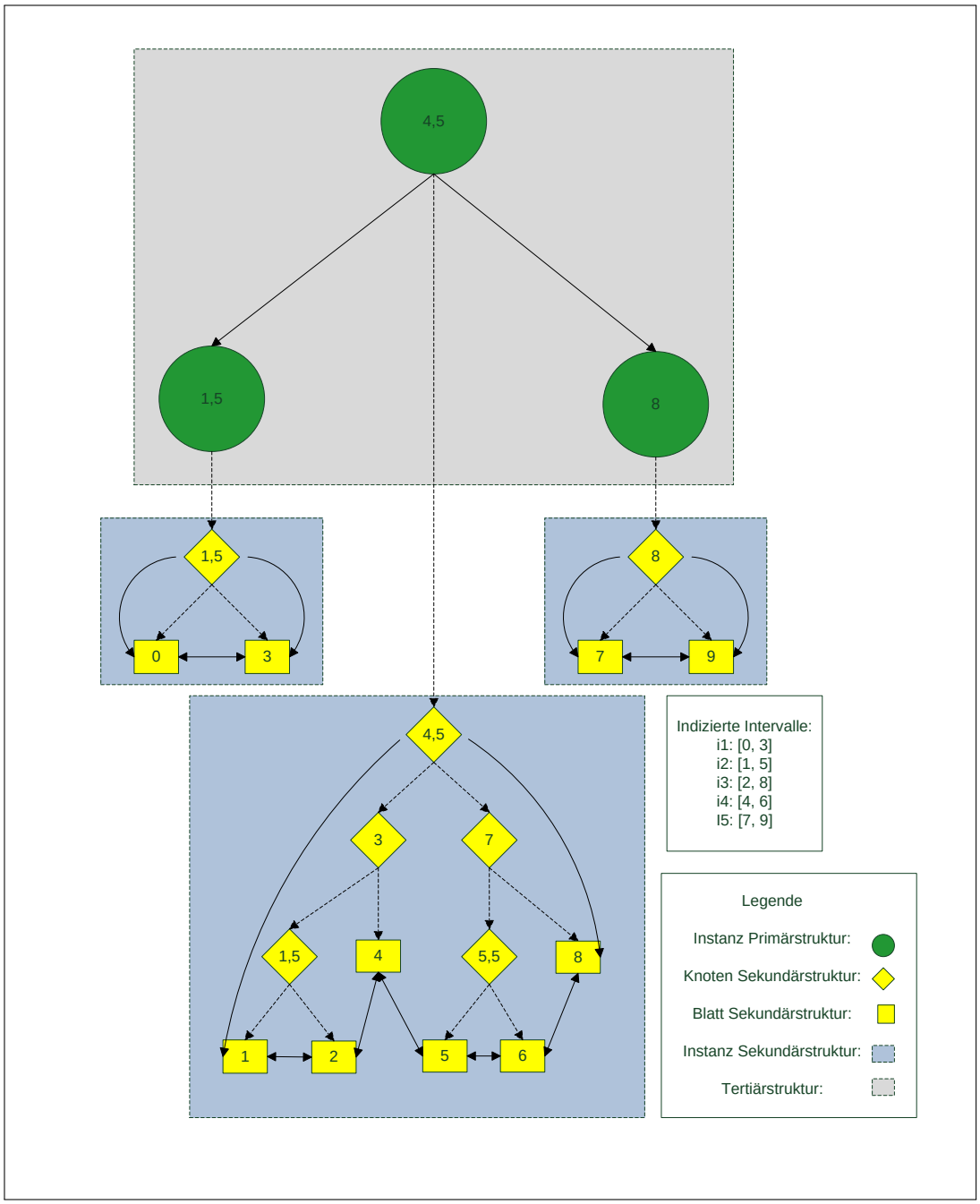

Abbildung 3.1: Strukturen des Intervallbaumes (vgl. [Edelsbrunner, 1980])

In der Literatur finden sich häufig weitere Vorschläge zur Verwendung anderer Strukturen als Sekundärstruktur. In Lehrbüchern finden sich oft zwei einfache sortierte Listen der unteren und oberen Schranke als Sekundärstruktur. Diese enthalten:

- Aufsteigend alle unteren Schranken der Intervalle, die den Knoten der Primärstruktur schneiden.
- Absteigend alle unteren Schranken der Intervalle, die den Knoten der Primärstruktur schneiden.

Außerdem enthalten Einträge in beiden Listen einen Verweis auf die Identität des zugehörigen Intervalls, um eine Zuordnung von oberer und unterer Schranke zum Intervall zu ermöglichen.

Unabhängig von der gewählten Sekundärstruktur, werden als erstes in der Primärstruktur alle Intervalle, die den Primär-Knoten schneiden, diesem zugeordnet. Diese zugeordneten Intervalle werden danach in die vom Primär-Knoten referenzierte Sekundärstruktur eingeordnet. Nach dem Erzeugen eines Elementes der Primärstruktur und der dazugehörigen Instanz der Sekundärstruktur, teilen sich alle nicht diesem Knoten der Primärstruktur zugeordneten Intervalle in zwei Gruppen:

1. Intervalle, deren obere Schranke kleiner als der Knoten ist und die somit unterhalb des Knotens liegen.
2. Intervalle, deren untere Schranke größer als der Knoten ist und die somit oberhalb des Knotens liegen.

Auf die in beiden Listen verbleibenden Intervalle wird jeweils der Algorithmus zur Erzeugung eines Elementes der Primärstruktur und der dazugehörigen Sekundärstrukturen angewendet. Der mit den Intervallen der 1. Gruppe gebildete Knoten wird als linker Tochterknoten an den bestehenden Knoten angehängt. Der mit den Intervallen der 2. Gruppe gebildete Knoten wird als rechter Tochterknoten an den bestehenden Knoten angehängt. Das gesamte Vorgehen wiederholt sich rekursiv mit den jeweils unterhalb und oberhalb des Knoten liegenden Intervallen, bis alle Intervalle einem Knoten zugewiesen wurden.

Der durch die Verknüpfung der Instanzen der Primärstruktur (Knoten) entstehende Binärbaum wird in [Edelsbrunner, 1980] als Tertiärstruktur bezeichnet. Zusammengefasst ist der Intervallbaum in der Lage, Überlappungsrelationen für ein Suchintervall mit logarithmischer

Abhängigkeit von der Menge der gespeicherten Intervalle zu bestimmen. So ergibt sich insgesamt für eine Anfrage die zu einem Intervall aus n Intervallen, r Überlappende Intervalle liefert, eine Komplexität von $O(\log n + r)$. Edelsbrunner verwendet den vorgestellten Intervallbaum als Grundlage, für einen mehrdimensionalen Intervallbaum, der in n-dimensionalen Systemen Überlappungen finden kann.

3.3 Relationale Intervallbäume

Beim relationalen Intervallbaum, im Folgenden als RI-Baum bezeichnet, handelt es sich um ein Konzept von Kriegel, Pötke und Seidl (vgl. [Kriegel u. a., 2000]), das es ermöglichen soll, die Ansätze des Intervallbaumes auf RDBMS zu nutzen. Die Implementierung soll direkt auf dem DBMS persistiert werden und die vom DBMS angebotenen Indizes nutzen. Der dem RI-Baum zugrunde liegende Baum wird dabei nicht als Baum in der Datenbank gespeichert, sondern bei Bedarf virtuell erzeugt. Deshalb wird nur die Wurzel gespeichert und die Knoten algorithmisch bei Bedarf berechnet.

Im Gegensatz zum Intervallbaum wird der Baum als geordneter, vollständiger balancierter Binärbaum über der Trägermenge aufgebaut. In Abbildung 3.2 ist der für die Trägermenge der Zahlen 1 bis 15 benutzte virtuelle Baum exemplarisch abgebildet. Der Knoten, dem ein Intervall im virtuellen Baum zugeordnet wird, wird als Forknode des Intervalls bezeichnet. Intervalle werden einem Forknode zugeordnet, indem der Baum von der Wurzel nach folgendem Schema durchlaufen wird:

- Liegt der Knoten im Intervall, wird das Intervall dem Knoten zugeordnet.
- Ist die obere Schranke kleiner als der Knoten und das Intervall liegt somit unterhalb des Knotens, wird das Vorgehen mit dem linken Tochterknoten wiederholt.
- Ist die untere Schranke größer als der Knoten und das Intervall liegt somit oberhalb des Knotens, wird das Vorgehen mit dem rechten Tochterknoten wiederholt.

In Abbildung 3.3 ist der gewählte Weg zum Einordnen des Intervalls [9,10] in den Baum aus 3.2 als Beispiel gezeigt. Zur Umsetzung wird in [Kriegel u. a., 2000] der in Listing 3.1 gezeigte Pseudocode vorgeschlagen.

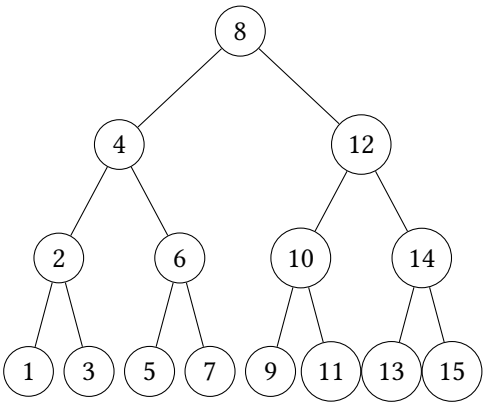

Abbildung 3.2: Geordneter, voller Binärbaum für die Trägermenge der Zahlen 1 bis 15.

```
FUNCTION forkNode (lowerBound, upperBound) {

        SELECT root
           INTO node
           FROM Metadaten_Tabelle;

        for(step = node/2; step >=1; step = step/2){
                if(upperBound < node) node -=step;
                elsif(node < lowerBound) node += step;
                else break;
        } //now node is the forknode
        return node
}
```

Listing 3.1: Pseudocode zur Bestimmung des Forknodes. nach: [Kriegel u. a., 2000]

Die Persistierung erfolgt, indem die Tabelle, welche die Intervalle beinhaltet, um eine Spalte erweitert wird. In dieser neuen Spalte wird der, mit Hilfe des virtuellen Baumes berechnete, Forknode gespeichert. Auf der erweiterten Tabelle werden zwei kombinierte Indizes aus je zwei Spalten erzeugt:

- Forknode und untere Schranke
- Forknode und obere Schranke

Diese Indizes werden automatisch durch das DBMS verwaltet und enthalten alle beim Intervallbaum in primärer und sekundärer Struktur gespeicherten Informationen.

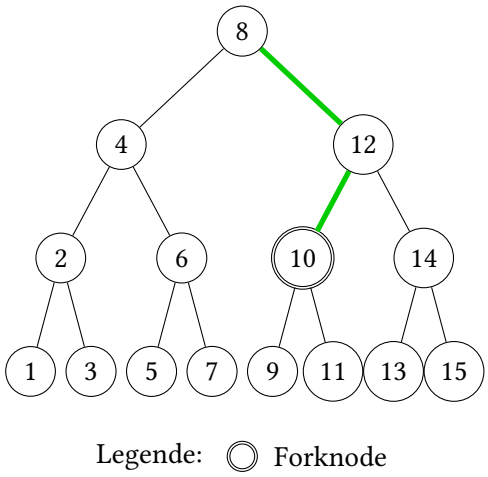

Abbildung 3.3: Pfad durch den Baum zur Bestimmung der Forknode für [9,10] in der Baumstruktur aus Abbildung 3.2.

Die Aufteilung der Information auf 2 verschiedene Strukturen entfällt. Das Einfügen neuer Intervalle erfolgt in 2 Teilen. Einem algorithmischen Teil in dem der Forknode berechnet wird und dem SQL Teil, in dem das Intervall in die Tabelle des DBMS eingetragen wird. Der „virtuelle“ Binärbaum wird nur arithmetisch zur Berechnung der Knoten bei Schreib- und Leseoperationen benötigt und nicht als Datenbank-Element gespeichert. Dafür werden von der Wurzel aus nur die benötigten Knoten berechnet. Die Größe der beiden erzeugten Indizes ist daher auch nur von den in der Datenbank gespeicherten Intervallen abhängig, da keine Einträge für Knoten ohne zugeordnete Intervalle angelegt werden.

Auch Suchanfragen auf den RI-Baum teilen sich in zwei Teile. In der ersten Phase wird arithmetisch berechnet, welchen Knoten Intervalle zugeordnet sein können, die relevant für die Anfrage sind. Dabei können Knoten in drei Fällen für die Anfrage relevant sein.

1. Der Knoten liegt im Vergleichsintervall, dann sind automatisch alle zugeordneten Intervalle, mindestens im durch den Knoten bezeichneten Punkt, überlappend.

2. Knoten die kleiner als die untere Schranke des Intervalls sind und auf ihrer Baumebene die geringste Differenz zur unteren Schranke haben. Diese können Intervalle enthalten, deren obere Schranke größer oder gleich der unteren Schranke des Vergleichs-Intervalls ist.

3. Knoten die größer als die obere Schranke sind und auf ihrer Baumebene die geringste Differenz zur oberen Schranke haben. Diese können Intervalle enthalten, deren untere Schranke kleiner oder gleich der oberen Schranke des Vergleichs-Intervalls ist.

Auf Grund der Eigenschaften des Binärbaumes sind in den Fällen zwei und drei nur Knoten relevant, die auf ihrer Baumebene die geringsten Differenz zur unteren bzw. oberen Schranke haben. Die Liste der im Suchintervall liegenden Knoten wird nicht explizit im algorithmischen Anteil berechnet, sondern ergibt sich bei der Anfrage durch die Bedingung: „untere Schranke des Suchintervalls $\leq$ Knoten $\wedge$ Knoten $\leq$ obere Schranke des Suchintervalls “.

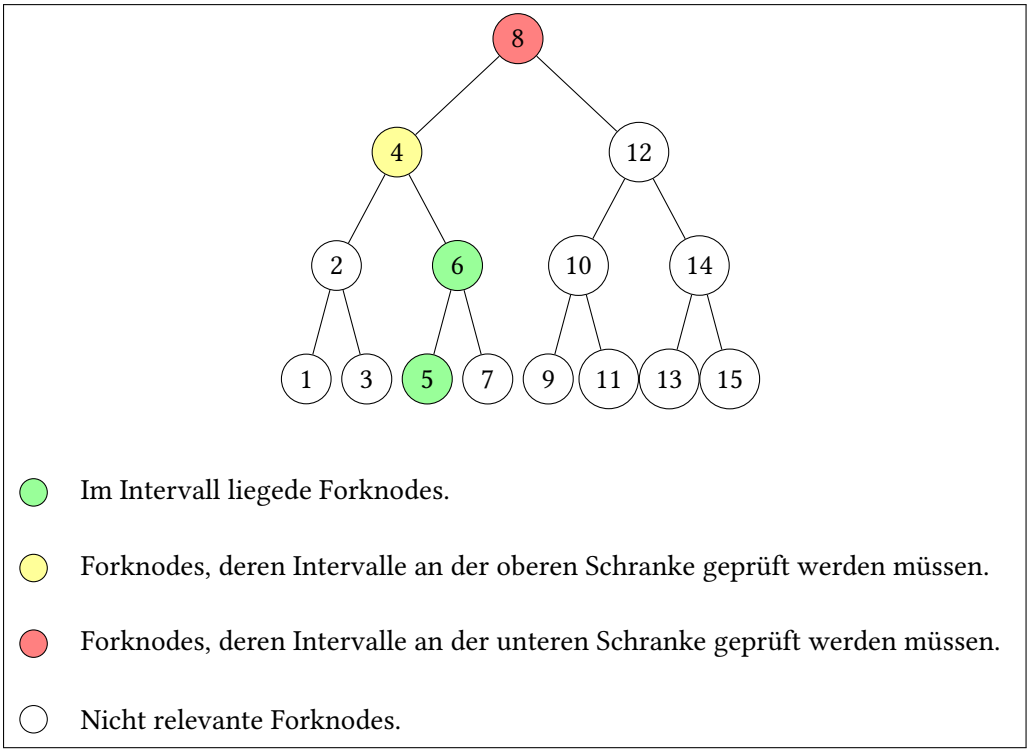

Abbildung 3.4: Knoten für die Überlappungs-Abfrage des Intervalls [5,6]

In Abbildung 3.4 ist für das Intervall [5,6] aus der Trägermenge 1 bis 15 abgebildet, welche Forknodes relevant sind. Mit den Listen der relevanten Knoten kann unter Verwendung der kombinierten Indizes die Suche auf der Datenbank beschleunigt werden. Dabei müssen die drei weiter oben genannten Gruppen an interessanten Knoten getrennt behandelt werden.

- Die im Intervall liegenden Knoten lassen sich für das DBMS mit beiden Datenbank-Indizes effizient lösen, da nur die Forknode Einträge geprüft werden.
- Der zweite Fall wird über den kombinierten Index der Forknode und der oberen Schranke gelöst.
- Der dritte Fall wird über den kombinierten Index der Forknode und der unteren Schranke gelöst.

Die Komplexität der gesamten Überlappungssuche ergibt sich aus folgenden Anteilen. Für jeden Eintrag in den Listen der relevanten Forknodes wird auf eine der beiden Indizes ein Range-Scan der Komplexität $O(\log_b n)$ ausgeführt. b steht dabei für die verwendete Blockgröße und n für die Menge der gespeicherten Intervalle. Die Listen der relevanten Forknodes enthalten dabei höchstens so viele Einträge wie der virtuelle Baum hoch ist. In Formeln steht h für die Höhe des virtuellen Baumes. Die im Range Scan gefundenen Ergebnisse müssen aus der Tabelle geladen werden. Dafür benötigt RDBMS für r_ρ Ergebnisse im schlimmsten Fall $O(r_\rho/b)$ Speicherzugriffe.

Insgesamt ergibt sich eine Komplexität von $O(h * \log_b n + r_\rho/b)$ (vgl. [Kriegel u. a., 2000]). Der Einfluss der gespeicherten Intervalle ist dabei nur logarithmisch. Mit der reinen Übertragung ins relationale Schema lassen sich allerdings noch nicht alle relevanten Anwendungsfälle lösen, bzw. es besteht noch weiteres Optimierungspotential. Deshalb werden in [Kriegel u. a., 2000] weitere, teilweise optionale, Änderungen und Erweiterungen vorgeschlagen. Diese werden im Folgenden kurz vorgestellt.

3.3.1 Trägermengen, die nicht bei 1 beginnen

Für eine gewählte Wurzel wird einen Baum erzeugt, der eine Trägermenge von 1 bis $Wurzel * 2 - 1$ abdeckt. Je nach Anwendungsszenario liegen die zu speichernden Intervalle allerdings über einer Trägermenge, die nicht bei 1 beginnt. Diese liegt vor, wenn die kleinste untere Schrank größer als 1 ist. Die Mächtigkeit der Trägermenge bestimmt die Höhe des Baumes. Der Baum wird in diesem Fall daher höher als zum reinen Abdecken der Trägermenge nötig, da auch der nicht genutzten Bereiche von 1 bis zur kleinsten unteren Schranke abgedeckt wird. So ist es möglich, dass beispielsweise Intervalle der Trägermenge $M = \{x \mid x \varepsilon N \wedge x \geq 101 \wedge x \leq 125\}$ betrachtet werden. Ohne Offset wird 64 als Wurzel gewählt, um auch alle Zahlen von 101 bis 125 abzudecken. Allerdings wird ein großer Teil der durch den Baum abgedeckten Trägermenge nicht genutzt. Gelöst wird dies, indem der Baum um einen Offset verschoben wird. Der Offset

ergibt sich aus der unteren Schranke der Trägermenge - 1.

Im oben genannten Beispiel ist das eine Wurzel von 16 und ein Offset von 100. Die Höhe[1] des Baumes ist somit von 6 auf 4 reduziert. Allgemein gilt: Von $\lfloor \log_2(\text{obere Schranke Trägermenge}) \rfloor$ wird die Höhe des Baumes auf
$\lfloor \log_2(\text{obere Schranke Trägermenge} - (\text{untere Schranke Trägermenge} - 1)) \rfloor$ reduziert.

3.3.2 Minstep

Weiteres Optimierungspotential besteht beim Berechnen der für eine Anfrage relevanten Forknodes. Es kann passieren, dass keinem Forknode unterhalb einer bestimmten Tiefe Intervalle zugeordnet sind. Bei der Bearbeitung von Anfragen wird eine Berechnung dieser Forknodes keine Intervalle liefern. Kriegel, Pötke und Seidl schlagen daher vor, die tiefste Ebene, der ein Intervall zugeordnet wurde, zu speichern. Die Speicherung des Minstep erfolgt dabei direkt als Wert der Step-Variable der Berechnung, um ohne Umrechnen mit diesem in der Funktion zur Berechnung des Forknodes arbeiten zu können. Die Berechnung der Forknodes kann bereits früher beendet werden und leere Knoten werden nicht in die Liste der relevanten Konten aufgenommen. Es besteht ein Zusammenhang zwischen dem Minstep und der minimalen Länge der gespeicherten Intervalle. So können nur Punktintervalle auf der tiefsten Ebene einem Forknode zugeordnet werden. Dies ergibt sich daraus, dass in einem vollständigem balancierten Binärbaum genau jeder zweite Knoten auf der tiefsten Ebene liegt. Ein Intervall mit einer Mächtigkeit von zwei muss mindestens einen weiteren Forknode enthalten, der nicht auf der tiefsten Ebene liegen kann. Allgemein gilt, dass Intervalle mit x = (obere Schranke - untere Schranke) nicht tiefer als $\lfloor \log_2(x) \rfloor$ eingeordnet werden können.

3.3.3 Erweiterung der Trägermenge an der unteren Schranke

Ein auf dem Offset aufbauender Ansatz ist der von LeftRoot und RightRoot. Es besteht das Problem, dass beim Einfügen des ersten Intervalls möglicherweise noch nicht klar ist, welches die niedrigste untere Schranke sein wird. Ein Offset muss aber spätestens beim Einfügen des ersten Intervalls bestimmt werden. Soll ein Intervall eingefügt werden dessen untere Schranke kleiner als der Offset ist, muss der Offset korrigiert und alle Forknodes neu berechnet werden. Um dies zu vermeiden werden die LeftRoot und RightRoot eingeführt. Es wird eine globale Wurzel mit dem Wert 0 gewählt. Diese globale Wurzel verweist auf die LeftRoot und RightRoot.

[1] Es sei darauf Hingewiesen das es zwei übliche Definitionen für die Höhe von Bäumen gibt. In dieser Arbeit entspricht die Höhe der Tiefe des tiefsten Knotens. Dies bedeutet ein nur aus der Wurzel bestehender Baum hat die Höhe 0.

Diese beiden Wurzeln lassen sich unabhängig voneinander verwalten. Wird ein Intervall eingefügt, so wird der beim Einfügen des ersten Intervalls bestimmte Offset angewendet. Intervalle die nach Abzug des Offsets die 0 schneiden, werden an die globale Wurzel gehängt. Intervalle, deren untere Schranke nach dem Abzug des Offsets positiv ist, werden dem Baum der RightRoot zugeordnet. Intervalle, deren Schranken beide negativ sind, werden mit dem Betrag der Schranken[2] in den Baum der LeftRoot einsortiert. Die LeftRoot stellt somit die Wurzel eines zweiten virtuellen Baumes dar. So ist es möglich, durch Erweitern des Baumes der LeftRoot, die untere Schranke der Trägermenge weiter nach unten zu Verschieben.

3.3.4 Unbeschränkte Intervalle

In Kapitel 2.1 Intervalle wurden bereits die unbeschränkten Intervalle erwähnt. Diese müssen im RI-Baum gesondert behandelt werden. Das Erweitern der Metadaten auf den gesamten Wertebereich eines Datentyps ist dabei keine gute Option, da so ein sehr großer, evtl. gar nicht benutzter Bereich von Trägermenge abgedeckt wird, der aber beim Einordnen der Intervalle und Beantworten von Anfragen arithmetisch durchlaufen werden muss. Die Höhe des virtuellen Binärbaumes wird höher als nötig. Daher wird in [Kriegel u. a., 2000] vorgeschlagen einen zusätzlichen Forknode für Intervalle ohne obere Schranke einzuführen. Dieser Forknode wird bei der Berechnung nicht in den virtuellen Baum eingeordnet, sondern die Zugehörigkeit des neuen Intervalls gesondert geprüft. Außerdem wird der „unbeschränkt Forknode“bei jeder Anfrage in die Liste für den am rechten Rand liegenden Forknode aufgenommen. Dies führt dazu, dass immer die untere Schranke mit der oberen Schranke des Vergleichsintervalls verglichen wird und so Überlappungen sicher festgestellt werden.

Dieses Konzept kann entsprechend gespiegelt auch auf die untere Schranke angewendet werden. Der „unbeschränkt Forknode“muss durch einen außerhalb des virtuellen Baumes liegenden Wert des Forknode-Datentyps dargestellt werden. Es bieten sich z.B. der maximale und minimale Wert des Datentyps an.

3.3.5 „Jetzt“ als Schranke von Zeitintervallen

Auch in [Kriegel u. a., 2000] wurden Zeitintervalle bereits als relevanter Anwendungsfall identifiziert. Bei Zeitintervallen kann es vorkommen, dass als Endzeitpunkt „jetzt“(in [Kriegel u. a., 2000] als „now“bezeichnet) vorkommt. Das besondere an „jetzt“ist, dass der tatsächliche Wert sich kontinuierlich verändert. Beschreibt man dies als konkreten Zeitpunkt, so müsste das

[2] Das nun negative Vorzeichen der Schranken wird weggelassen

Intervall regelmäßig geändert und die Einordnung in den virtuellen Baum geprüft und ggf. angepasst werden.

Um diesen Aufwand zu vermeiden wird ein weiterer reservierter Forknode außerhalb des virtuellen Baumes eingeführt. Diesem werden alle Intervalle mit der oberen Schranke „jetzt“zugeordnet. Ist bei einer Abfrage die untere Schranke eines Vergleichsintervalls kleiner (also zeitlich vor) oder gleich dem jetzigen Zeitpunkt wird der Forknode für „jetzt“in die Liste der oberhalb des Suchintervalls liegenden Forknodes aufgenommen. Dies führt dazu, dass die unteren Schranken der „jetzt“zugeordneten Intervalle mit der oberen Schranke des Suchintervalls verglichen werden.

Entsprechend gespiegelt ist es auch möglich das Konzept auf „Jetzt“ als untere Schranke an zu wenden.

3.3.6 Zusammenfassung RI-Baum

Zusammengefasst stellt der RI-Baum eine Übertragung des Konzeptes des Intervallbaumes auf RDBMS dar. Diese nutzt existierende Indizes des RDBMS so wie sie sind und ist dabei effektiver in der Bestimmung von Überlappungen, als andere Verfahren die existierende Indizes verwenden. Die Komplexität der Überlappungsbestimmung ist dabei nur logarithmisch von der Menge der gespeicherten Intervalle abhängig. Damit ist der RI-Baum sehr gut zum Senken der Bearbeitungszeit von Überlappungsanfragen geeignet. Zusätzlich ist es möglich, auch anwendungsspezifische Fälle wie „Jetzt“ zu integrieren. Der RI-Baum ist auf normalen RDBMS, wie sie im Kontext dieser Arbeit verwendet werden, jedoch nicht geeignet, um Echtzeitanforderungen zu erfüllen. Hierfür müssten Laufzeiten garantiert werden können.

3.4 Andere Arbeiten zum RI-Baum

Es gibt eine Reihe weiterer Arbeiten, die sich mit dem RI-Baum beschäftigen. Diese greifen die Konzepte des RI-Baumes auf und entwickeln sie weiter.

3.4.1 Implementation mit Hilfe des Extensible Index Interfaces von DB2

Eine davon ([Brochhaus u. a., 2005]) beschäftigt sich mit der Implementation des RI-Baumes auf IBMs DB2. Die Autoren gehen dabei auf eine Implementation des RI-Baumes unter Ausnutzung

des vom DB2 DBMS angebotenen Extensible Index Interfaces ein. Das Extensible Index Interface von DB2 ermöglicht es, dem DBMS eigene Methoden zum Verwalten eigener Datentypen in einem DB-internen Index zu ermöglichen. Das DB2 RDBMS generiert aus einer bereitgestellten Methoden die Index Schlüssel eines Eintrags und pflegt den Datenbank-internen Index auf Grundlage dieser Schlüssel.

Die weiteren Methoden werden im Zuge von Abfragen dazu benötigt, über die Verwendung des Index entscheiden können, die passende Suche auf dem Index zu definieren und bei Bedarf die Ergebnisse der Indexsuche nachträglich zu filtern. Die bereitgestellten Methoden stellen damit den algorithmischen Anteil des RI-Baums dar. Im Zuge ihrer Arbeit identifizieren die Autoren Besonderheiten die für die Implementation des RI-Baums auf dem DB2 RDBMS relevant sind und im Folgenden kurz erläutert werden sollen.

Der Einsatz des Extensible Index Interfaces ermöglicht nur die Verwendung eines einzelnen B+-Baums. Dies kollidiert mit dem ursprünglichen Konzept des RI-Baums, welches zwei B+-Baum basierte Indizes vorsieht.[3] Dieses Problem wird durch die Autoren gelöst, indem der Ansatz zur Partitionierung von B-Bäumen aus [Graefe, 2003] aufgegriffen wird. Der Lösungsansatz sieht vor, den vom DBMS genutzten B+-Baum in zwei Bereiche aufzuteilen. Die Bereiche trennen sich dabei in einen Bereich für Forknodes und untere Schranken und eine Bereich für Forknodes mit obere Schranken. Dies wird erreicht, indem den Forknode-Einträgen eines Intervalls unterschiedliche Vorzeichen zugewiesen werden. So ist es möglich, die zwei Indizes in einem einzigem Datebankindex abzubilden. In diesen werden für jedes hinzugefügte Intervall durch die im Extensible Index Interfaces hinterlegten Methoden die Einträge (-Forknode, untere Schranke) und (Forknode, -obere Schranke) eingefügt. Das negative Vorzeichen vor der oberen Schranke im 2. Index Eintrag benötigen die Autoren, um eine Vereinfachung der SQL-Anfragen vornehmen zu können. Für eine Erläuterung dieser wird auf das Paper verwiesen. Für komplexere Datentypen ohne Vorzeichen ist dieses Vorgehen allerdings nicht ohne Weiteres übertragbar. Für Zeitpunkte als Schranken von Zeitintervallen funktioniert die direkte Übertragung z.B. nur wenn diese durch natürliche Zahlen abgebildet werden, indem z.B. die Millisekunden seit 1. Januar 1970 gespeichert werden. Dies verschlechtert allerdings die Lesbarkeit der Rohdaten für Menschen. Zusätzlich bieten RDBMS auf zeit spezifischen Datenstrukturen häufig unterstützende Möglichkeiten, um z.B. Zeitzonen besser verwalten zu können.

[3] (Forknode, untere Schranke) und (Forknode, obere Schranke)

Die zweite Einschränkung besteht darin, dass das Extensible Index Interface keinen Zugriff auf Tabellen und andere dynamische Inhalte des RDBMS hat. Dies beschränkt den Einsatz auf feste, beim Anlegen des Index anzugebende Metadaten. Im Falle des RI-Baum betrifft dies die Wurzel, sowie die Konzepte des Offsets, der RightRoot und LeftRoot sowie des Minstep.
Da der Schwerpunkt dieser Arbeit auf der dynamischen Anpassung der Metadaten liegt, sind die Konzepte dieser Arbeit nicht mit dem Extensible Index Interface des DB2 umsetzbar.

3.4.2 Einsatz des RI-Baumes zum Bestimmen weiterer Intervall relationen

Eine weitere Veröffentlichung ([Kriegel u. a., 2001]) modifiziert den Ansatz des RI-Baumes, um auf diesem neben Überlappungen auch weitere Anfragen effizient Lösen zu können. Die Autoren beziehen sich dabei auf 13 in 3.5 definierte Beziehungen zwischen Intervallen.

Im ersten Schritt definieren die Autoren 6 Gruppen, denen ein Knoten auf einem für eine Anfrage berechneten Pfad im Baum zugeordnet werden kann. Bei diesen handelt es sich um:

- TopLeft: Knoten, die auf dem Pfad durch den Baum über dem Forknode liegen und kleiner als die untere Schranke des Intervalls sind.
- TopRight: Knoten, die auf dem Pfad durch den Baum über dem Forknode liegen und kleiner als die größer Schranke des Intervalls sind.
- InnerLeft: Knoten, die auf dem Pfad von Forknode zur unteren Schranke und innerhalb des Intervalls liegen.
- BottomLeft: Knoten, die auf dem Pfad von Forknode zur unteren Schranke und unterhalb des Intervalls liegen.
- InnerRight: Knoten, die auf dem Pfad von Forknode zur oberen Schranke und innerhalb des Intervalls liegen.
- BottomRight: Knoten, die auf dem Pfad von Forknode zur oberen Schranke und außerhalb des Intervalls liegen.

Die Vereinigung der Gruppen BottomLeft und TopLeft entspricht dabei den links vom Intervall liegenden relevanten Knoten für Überlappungsanfragen. Entsprechend ergibt die Vereinigung von TopRight und BottomRight die rechts vom Intervall liegenden relevanten Knoten für Überlappungsanfragen. Zusätzlich gibt es noch die gesondert gehandelten Knoten: Forknode, untere Schranke und obere Schranke. Sowie die Bereiche aller im Intervall liegenden Knoten

und alle links und alle rechts vom Intervall liegenden Knoten. Um diese zu bestimmen ist es jedoch nicht nötig, auf den virtuellen Baum zurück zu greifen.

Im nächsten Schritt ordnen die Autoren die Gruppen den möglichen Beziehungsanfragen für die diese eine Relevanz besitzen zu. Die Gruppen-Beziehungen sind in Abbildung 3.5 dargestellt.

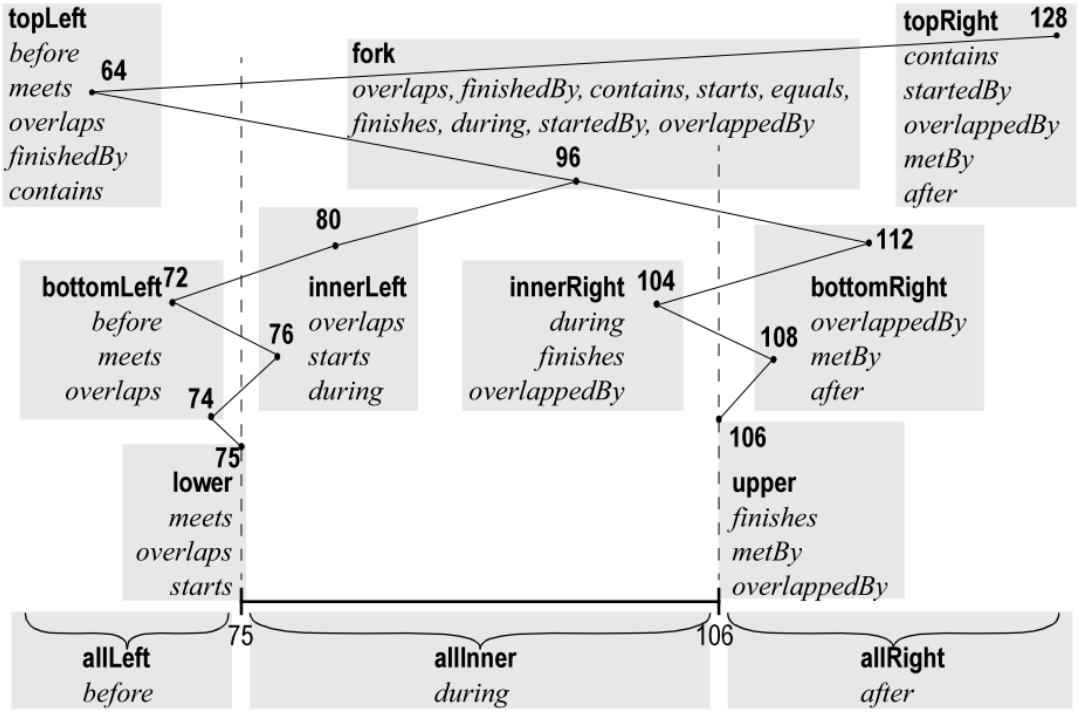

Abbildung 3.5: Beispielhafte Darstellung der Gruppen aus [Kriegel u. a., 2001] für das Intervall [75,106] in einem Baum mit 128 als Wurzel.

Da eine Erläuterung aller Zuordnungen den Rahmen dieser Arbeit überschreiten würde, wird für einer Erläuterung der Zuordnung auf [Kriegel u. a., 2001] verwiesen.

Die Berechnung der relevanten Knoten erfolgt weiter mit den gleichen Algorithmen. Nur die Zuordnung der Knoten zu den Gruppen wird entsprechend angepasst. Für die Abfragen auf das Datenbankschema müssen je nach abgefragter Beziehung verschiedene Gruppen von Knoten berücksichtigt werden und auf diesen Gruppen verschieden Vergleiche auf die zweite Spalte des Indizes. Die Bearbeitung einer Beziehungsanfrage besteht damit aus einer Menge an SQL-Abfragen, deren Ergebnis vereinigt werden muss.

Um die Menge dieser Abfragen und damit den Aufwand für Abfragehandling und Vereinigungsoperationen zu verringern erweitern die Autoren die Datenbank-Indizes jeweils um eine dritte Spalte. Diese enthält die der zweiten Spalte entgegen liegende Schranke. Aus dem Index (Forknode, untere Schrank) wird der Index (Forknode, untere Schranke, obere Schranke) und aus dem Index (Forknode obere Schranke) wird der Index (Forknode, obere Schranke, untere Schranke).

Da die grundlegenden Algorithmen dieses erweiterten RI-Baumes denen des original RI-Baumes entsprechen funktionieren die in dieser Arbeit erarbeiteten Mittel auch mit dem auf weitere Intervall-Beziehungen erweiterten RI-Baum.

3.5 Das Problem einer wandernden Trägermenge

In dieser Arbeit sollen vor allem Probleme mit dynamischen Trägermengen betrachtet werden. Eine dynamische Trägermenge liegt vor, wenn der Bereich in dem die Intervalle liegen können, sich zur Laufzeit verändern kann.
Die mindestens zu unterstützende Trägermenge bezeichnet zu jedem Zeitpunkt die Menge der Elemente, die gleich oder größer der kleinsten gespeicherten unteren Schrank und gleich oder kleiner der größten gespeicherten oberen Schranke sind. Dabei soll insbesondere eine „wandernde“ Trägermenge thematisiert werden.
Dies bedeutet, es wird eine gleich groß bleibende Trägermenge betrachtet, deren Grenzen sich im Bereich der möglichen Werte verschieben. Die Verschiebung erfolgt dabei in kleineren Schritten, als die Trägermenge groß ist, so dass davon auszugehen ist, dass Teile der vor dem Verschieben vorhandenen Intervalle präsent bleiben.

Ein Beispiel für ein wanderndes Fenster ist im Kontext von Zeitintervallen eine Trägermenge der letzten X Tage. Die Mächtigkeit dieser Trägermenge bleibt mit X Tagen konstant, die obere und untere Grenze verschieben sich jedoch regelmäßig. In Abbildung 3.6 ist zur Veranschaulichung eine die letzten 3 Tage umfassende Trägermenge zu verschiedenen Zeitpunkten dargestellt. Die Zeitintervalle liegen zu den angegeben Zeitpunkten nur innerhalb der markierten Trägermenge. Das Verschieben folgt dabei einer bekannten Regel. Die echte Verschiebung erfolgt dabei synchron mit dem Voranschreiten der Zeit. In Computersystemen wird dieses Verschieben meist in festen Abständen durchgeführt, z.B. jede Stunde. Alle Intervalle, die älter sind als die maximale Speicherdauer, werden nach dem Verschieben gelöscht und müssen

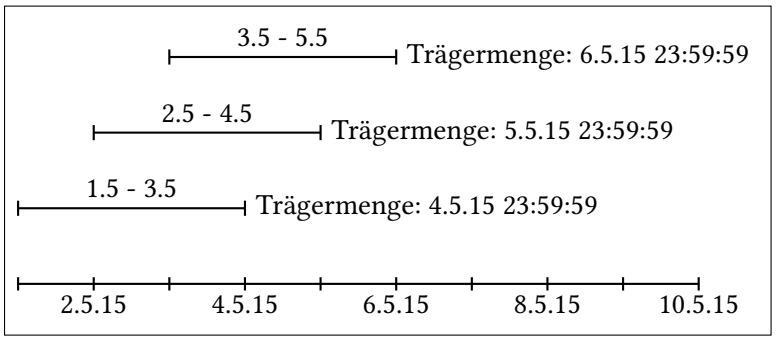

Abbildung 3.6: Trägermenge, die ein Zeitfenster der letzten 3 Tage abdeckt, zu verschiedenen Zeitpunkten.

damit nicht weiter indiziert werden.

Das Verschieben kann nicht in kleineren Schritten erfolgen, als die Genauigkeit des verwendeten Datentyps ermöglicht. Bildet der Datentyp Zeiten bis auf Millisekunden ab, kann das Anpassen der Trägermenge, bzw. Voranschreiten der Zeit nicht in kleineren Schritten als Millisekunden-Schritten dargestellt werden. Im Beispiel des intelligenten Stromnetzes wäre eine mögliche Umsetzung, dass die Verbraucher mit Voranschreiten der Zeit immer neuere Zeitstempel liefern, während stündlich alle Intervalle die älter sind, als die Speicherdauer von X Wochen, gelöscht werden.

Bei der Definition von „sind älter als“ ist darauf zu achten, ob dies über die obere oder untere Schranke definiert wird. Werden Intervalle erst gelöscht, wenn ihr Enddatum älter ist als X Wochen, fällt die zu unterstützende Trägermenge größer aus als X Wochen. Die zu unterstützende Trägermenge vergrößert sich um den Zeitraum zwischen der kleinsten unteren Schranke und dem Beginn des X Wochen Zeitfensters. Dieser kann im ungünstigsten Fall der maximalen Mächtigkeit der Intervalle entsprechen. Die Wahl der abgedeckten Trägermenge muss entsprechend angepasst werden. Wird die untere Schranke als Löschkriterium verwendet oder die untere Schranke von Intervallen, die früher als vor X Wochen begonnen haben, beim Löschen auf den Zeitpunkt vor X Wochen korrigiert, besteht dieses Problem nicht.

3.6 Anwendbarkeit des RI-Baumes auf eine wandernde Trägermenge

Der RI-Baum ist durch seine leichte Integration in viele bestehende RDBMS, seine effektive Laufzeit und den geringen Platzbedarf eine sehr geeignete Struktur zum indizieren von Intervallen in RDBMS. Im Folgenden soll daher der Einsatz des RI-Baumes im Kontext einer wandernde Trägermenge analysiert werden.

Der RI-Baum ist konzeptuell bereits in der Lage, mit dynamischen Trägermengen umzugehen. Es ist auch möglich, den virtuellen Baum an wachsende Trägermengen anzupassen. Das Wachsen an der oberen Schranke der Trägermenge lässt sich durch ein Verdoppeln der Wurzel des virtuellen Baumes lösen. Mit Hilfe der Erweiterung um die LeftRoot und RightRoot lässt sich das selbe Konzept auch nutzen, um ein Wachsen der Trägermenge an der unteren Schranke zu ermöglichen. Da es sich in beiden Fällen nur um Anpassungen der Metadaten handelt, lässt sich dies in konstanter Zeit lösen. Das Vergrößern der durch den virtuellen Baum abgedeckten Trägermenge löst das Problem zwar, allerdings nimmt mit der Zeit die Höhe des virtuellen Baumes zu. Dies liegt daran, dass nur die obere Schranke angepasst wird, die Trägermenge also wächst.

Um dies zu vermeiden ist es nötig, auch die untere Schranke der abgedeckten Trägermenge anzupassen. Diese muss allerdings nicht nach unten verschoben werden, sondern nach oben, so dass die abgedeckte Trägermenge konstant groß bleibt. Dies ist im ursprünglichen Konzept durch eine Anpassung des Offsets möglich. Die Anpassung des Offsets hat den Nachteil, dass eine Neuberechnung der Forknodes aller bereits gespeicherten Intervalle nötig wird. Dies stellt einen erheblichen zeitlichen Aufwand darf. Während dieser Anpassung ist es nicht möglich, konsistente Anfragen an die Tabelle zu stellen.

Zum Evaluieren der Relevanz wurde der RI-Baum auf einem Oracle 11.2 DBMS in PL/SQL Implementiert und die benötigte Zeit zur Neuberechnung der Forknodes für eine wachsende Anzahl an Intervallen gemessen. Die Implementation bezieht sich auf Zeitintervalle. Es wurde einmal die benötigte Zeit zum direkten Verschieben und einmal zum Verwerfen der beiden Indizes, neu berechnen der Forknodes und anschließendem Indizes neu erstellen gemessen. In Abbildung 3.7 werden die gemessenen Zeiten dargestellt. Die Laufzeit kann je nach verwendetem System und besonders je nach verwendeter Speicherform variieren. Es zeigt sich deutlich, dass zum Ändern des Offsets eine erhebliche Zeit benötigt wird.

In der Testumgebung dauerte die Neuberechnung ab etwa 10 Millionen Intervallen etwa eine Viertelstunde. Allgemein steigt die benötigte Zeit linear mit der Menge der gespeicherten Intervalle an. In dieser Zeit sind Anfragen und Schreiboperationen auf die Intervalldaten-Tabelle nicht möglich. Dies stellt im Kontext ständig verfügbarer Systeme, wie dem beispielhaften intelligenten Stromnetz, eine nicht hinnehmbare Einschränkung da.

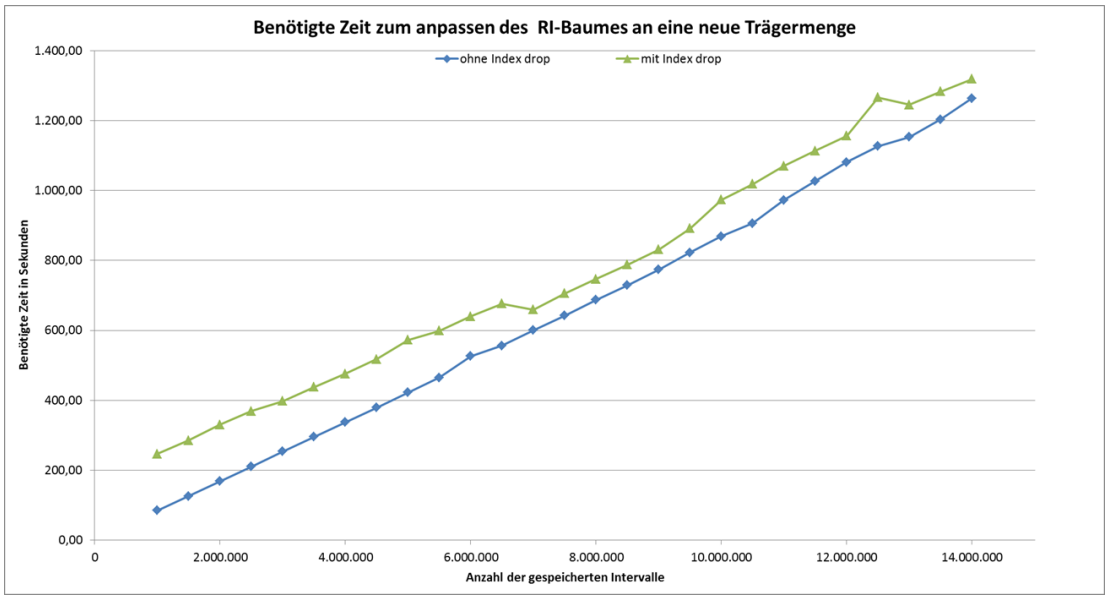

Abbildung 3.7: Benötigte Zeit zum Anpassen der Forknodes nach einer Änderung des Offsets.

3.7 Erwartungen an einen angepassten RI-Baum

Die Übertragung zum relationalen RI-Baum und Erweiterungen des RI-Baumes um die LeftRoot und RightRoot, sowie Konzepte wie „jetzt“und „unendlich“zeigen, dass der Intervallbaum und der RI-Baum potenziell gut anpassbare Konzepte sind.

Ziel ist es daher, den RI-Baum so anzupassen, dass es möglich ist, die in Kapitel 3.5 beschriebene wandernde Trägermenge zu unterstützen. Dazu ist es nötig, dass der veränderte RI-Baum die untere Schranke der Trägermenge nach oben anpassen kann, ohne dass die Forknodes aller gespeicherten Intervalle neu berechnet werden müssen. Dabei ist es wichtig, dass die Möglichkeit den RI-Baum auf bestehenden RDBMS umzusetzen, nicht eingeschränkt wird. Es sollen weiterhin die vom RDBMS angebotenen Indizes verwendet werden und in diese auch nur tatsächlich in der DB gespeicherte Kombinationen aus Forknode und unterer bzw.

oberer Schranke eingetragen werden. Genau so soll die logarithmische Komplexitätsklasse zum Berechnen der Forknodes erhalten bleiben. Das Verschieben soll im Optimalfall durch Anpassen der Metadaten erfolgen und damit unabhängig von der Anzahl der in der DB gespeicherten Intervalle sein. Auch soll der benötigte Speicherplatz in der selben Komplexitätsklasse bleiben und im Optimalfall nur weitere Metadaten benötigt werden. Der veränderte RI-Baum soll außerdem zu den bestehenden Erweiterungen für „jetzt“(siehe 3.3.5) und unbeschränkte Intervalle (siehe 3.3.4) kompatibel bleiben.

3.8 Relevanz für die Zukunft

Das Problem der Intervall Indizierung über dynamischen Trägermengen wird auch in Zukunft von Bedeutung sein. So sind Systeme, die durch massiven Einsatz von Sensoren dauerhaft Informationen über ihre Umwelt erfassen, z.B. in Form von intelligenten Autos und Wohnungen, Fokus vieler Forschungsprojekte. Auch in solchen Systemen können große Datenmengen an Zeitintervallen und anderen Intervallen anfallen, die nur für eine bestimmte Zeit von Interesse sind. Ziel dieser Systeme ist es oft auf Grundlage der gesammelten Daten auf ihre Umwelt zu reagieren oder gar mit dieser zu Interagieren. Diese Interaktion und Reaktion mit/auf die reale Welt stellt oft Echtzeit- oder Quasi-Echtzeit-Anforderungen an die verarbeitenden Systeme. In diesem Kontext ist daher von einem wachsenden Interesse an effizienten Mitteln zum Durchsuchen großer Mengen an Intervalldaten auszugehen. Auf Grund der weiterhin steigenden Datenmengen wird auch die Weiterentwicklung der Hardware das Problem nicht lösen.

4 Anpassung des RI-Baumes an eine dynamische Trägermenge

Im Folgenden wird ein Konzept zur Anpassung des RI-Baumes zur besseren Unterstützung dynamischer Trägermengen und im besonderen einer wandernden Trägermenge, vorgestellt werden. Um das Verschieben des virtuellen Baumes zu ermöglichen, wird als erstes ein Konzept zur freien Platzierung der Trägermenge vorgestellt und darauf aufbauend erläutert, wie die Trägermenge sich dynamischer verwalten lässt.

4.1 Freie Platzierung der Baumstruktur über dem Wertebereich

Wie bereits in Kapitel 3.6 erläutert, ist das Vergrößern der Trägermenge bereits an beiden Schranken möglich. Das Verkleinern ist jedoch an der unteren Schranke nur mit erheblichem Aufwand möglich. Dies liegt an der nötigen Neuberechnung und Persistierung aller Forknodes nach dem Ändern des Offsets. Daher wird im ersten Schritt eine Alternative zum Offset eingeführt werden, mit der es möglich wird, den virtuellen Intervallbaum frei im Wertebereich zu platzieren.

Der in 3.3 vorgestellte relationale Intervallbaum baut den virtuellen Baum der Forknodes über die Differenz der Wurzel zur 0 auf. Um den virtuellen Baum frei aufspannen zu können, wird dieser von der 0 entkoppelt. Dies geschieht, indem ein weiteres Metadatum eingeführt wird. Das neue Metadatum beschreibt die Differenz der Wurzel zu ihren Tochterknoten. Mathematisch lässt sich das neue Metadatum als „Wurzel minus linker Tochterknoten“ sowie als „rechter Tochterknoten minus Wurzel“ beschreiben. Das neue Metadatum lässt sich zusätzlich durch Wurzel $-$ (kleinstes Element des virtuellen Bames $-$ $1)/2$ und Wurzel $-$ (größtes Element des virtuellen Baumes $+$ $1)/2$ berechnen.

Das neue Metadatum wird im Folgenden als Step-Metadatum bezeichnet. Mit dem Step-

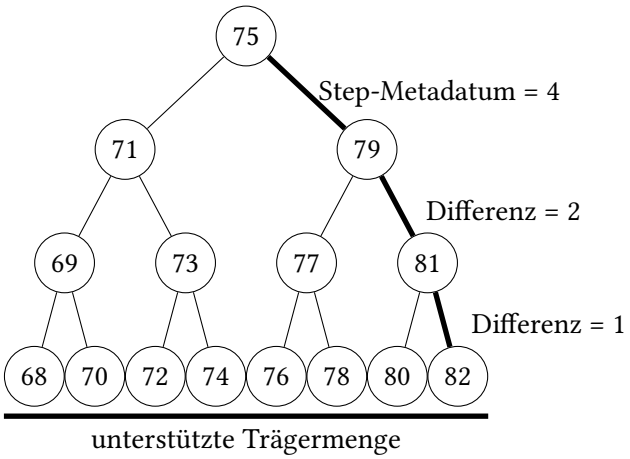

Abbildung 4.1: Virtueller Baum des RI-Tree, der eine Trägermenge von 68 bis 82 abdeckt.

Metadatum lassen sich die Tochterknoten der Wurzel durch Subtraktion des Step-Metadatums von der Wurzel (linker Tochterknoten) und Addition des Step-Metadatums auf die Wurzel (rechter Tochterknoten) bestimmen. Für jeden anderen Knoten des virtuellen Baumes lässt sich die Differenz des Knotens zu seinen Tochterknoten durch „Step-Metadatum/$(2 *$ Tiefe des Knotens im Baum)“ bestimmen. Zum Bestimmen der Tochterknoten muss diese Differenz jeweils auf den Knoten addiert (rechter Tochterknoten) oder subtrahiert (linker Tochterknoten) werden. Die Differenz von einem Knoten zu seinen beiden Tochterknoten wird mit jeder Ebene des Baumes halbiert. Der mit dem Step-Metadatum aufgebaute virtuelle Baum für die Trägermenge der natürlichen Zahlen von 68 bis 82 ist in Abbildung 4.1 dargestellt. Das Step-Metadatum als Abstand der Wurzel zu seinen Tochterknoten ist zur Veranschaulichung ausgewiesen.

Damit ein vollständig balancierter Binärbaum entsteht, wird empfohlen, den virtuellen Baum immer eine Trägermenge mit der Mächtigkeit einer 2er-Potenz - 1 abdecken zu lassen. Liegt eine Trägermenge mit einer Mächtigkeit ungleich einer 2er-Potenz - 1 vor, sollte der virtuelle Baum mit der Mächtigkeit der nächst-größeren 2er-Potenz - 1 gewählt werden.

Da es sich bei dem virtuellen Baum um einen vollständig balancierten Binärbaum handelt, entspricht die Höhe des virtuellen Baumes, der eine Trägermenge der Mächtigkeit $2^x + 1$ abdeckt, genau x (vgl. [Kriegel u. a., 2000]). Die benötigte Höhe h des Baumes zum Abdecken einer gegebene Trägermenge von y bis z (inklusive y und z) ergibt sich daher aus $h = \lfloor \log_2(z - (y - 1)) \rfloor$. Die Mächtigkeit m des gewählten, virtuellen Binärbaumes der die Trägermenge abdeckt ent-

spricht somit $m = 2^{h+1} - 1 = 2^{(\lfloor \log_2(z-(y-1)) \rfloor)+1} - 1$.
Das zu Baum der Mächtigkeit M passende Step-Metadatum lässt sich durch 2^{h-1} bestimmen.
Ist die vom virtuellen Baum abgedeckte Trägermenge größer als die Trägermenge, die abgedeckt werden soll, gibt es verschiedene Möglichkeiten den Baum über der abzudeckenden Trägermenge zu platzieren. Dies beeinflusst die Wahl der Wurzel für eine abzudeckende Trägermenge. Es gibt drei Fälle, die je nach Anwendungskontext sinnvoll für die Platzierung des Baumes und damit auch die Wahl der Wurzel sind.

Der Baum wird so platziert, dass:

1. die untere Schranke der abzubildenden Trägermenge und die untere Schranke der durch den virtuellen Baum abgedeckten Trägermenge gleich sind.
2. die Wurzel des virtuellen Baumes mittig in der abzudeckenden Trägermenge liegt.
3. die obere Schranke der abzubildenden Trägermenge und die obere Schranke der durch den virtuellen Baum abgedeckten Trägermenge gleich sind.

Der Anwendungskontext bestimmt, welches die sinnvollste Wahl ist. Wird von dem in dieser Arbeit hauptsächlich betrachteten Fall einer sich in eine Richtung verschiebenden Trägermenge ausgegangen, ist es sinnvoll, den Baum in dieser Richtung möglichst weit überhängen zu lassen. Dadurch ist ein Anpassen des virtuellen Baumes möglichst lange nicht nötig.
Es wird daher Variante 1 oder Variante 3 gewählt und das Ende des virtuellen Baumes an der dem Verschieben entgegengesetzten Seite passend angesetzt. Kann die Trägermenge sich in beide Richtungen verschieben oder vergrößern, empfiehlt es sich, die Wurzel und damit den Baum mittig zu platzieren, so dass zu beiden Seiten ein gleichmäßiger Überhang entsteht.

Da der virtuelle Baum von der Wurzel ausgehend aufgebaut wird, bestimmt die Wahl der Wurzel den entstehenden Baum. Es muss daher aus der abzudeckenden Trägermenge die Wurzel berechnet werden. Für die folgenden Formeln sei x die berechnete benötigte Tiefe des virtuellen Baumes.

- In Variante 1 wird die Wurzel auf $(\text{untere Schranke } - 1) + 2^{x-1}$ gesetzt.
- In Variante 2 wird die Wurzel auf $\lfloor \text{untere Schranke} + (\text{obere Schranke} - \text{untere Schranke})/2 \rfloor$ gesetzt.
- in Variante 3 wird die Wurzel auf $(\text{obere Schranke} + 1) - 2^{x-1}$ gesetzt.

Das neue Metadatum entkoppelt somit den Aufbau des Baumes von der 0 und ermöglicht ein Aufspannen des Baumes an beliebiger Stelle. Selbst ein Aufspannen im Bereich negativer Zahlen ist kein Problem.

Werden Intervalle mit einer oberen Schranke oberhalb der abgedeckten Trägermenge eingefügt, kann der Baum mit dem selben Mechanismus wie der unveränderte RI-Baum nach oben vergrößert werden. Das Vergrößern der Trägermenge an der oberen, sowie das Vergrößern an der unteren Schranke wird in Abbildung 4.2 dargestellt.
Die Wurzel wird beim Vergrößern an der oberen Schranke auf $\text{Wurzel} = \text{Wurzel} + 2 * \text{Step}$ gesetzt, anstatt wie im originalen RI-Baum die Wurzel mit 2 zu multiplizieren. Wird die Wurzel verändert, muss auch das Step-Metadatum angepasst werden. Dazu muss das Step-Metadatum verdoppelt werden. Beide Teiloperationen können in konstanter Zeit durchgeführt werden.

Analog zum Vergrößern an der oben Schranke kann der Baum auch an der unten Schranke erweitert werden, indem die Wurzel auf $\text{Wurzel} = \text{Wurzel} - 2 * \text{Step}$ gesetzt wird. Wieder ist das Step-Metadatum, durch verdoppeln, anzupassen. Um ein vergrößern an der unteren Schranke zu ermöglichen war im originalen RI-Baum das Konzept der LeftRoot und RightRoot nötig.
Zusätzlich kann der virtuelle Baum durch das neue Metadatum, auf jeden seiner Teilbäume verkleinert werden, ohne dass die Forknodes der bereits gespeicherten Intervalle neu berechnet werden müssen. Dazu wird die in der Metadaten Tabelle gespeicherte Wurzel auf die Wurzel des zu erhaltenden Teilbaumes gesetzt. Das neue Step-Metadatum wird mit der Formel $\text{Step} = \text{Step}/(2^{\text{Tiefe der neuen Wurzel im alten Baum}})$ berechnet. Alternativ kann die Wurzel auch über die weiter oben genannten Formeln in Abhängigkeit zu den neuen Schranken, oder der Höhe des Baumes berechnet werden.

Das neue Step-Metadatum stellt eine Erweiterung des Konzeptes des RI-Baumes dar und ist kompatibel zu seinen bestehenden Konzepten. Sowohl die Minstep-Optimierung, als auch die Optionen für unbeschränkte Intervalle (siehe: 3.3.4) sowie “jetzt“ als Schranke von Zeitintervallen (siehe: 3.3.5) sind zur Verwendung des Step Metadatums kompatibel. Selbst eine Kombination mit den nun nicht mehr benötigten Konzepten des Offset, sowie der RightRoot und LeftRoot wäre möglich.

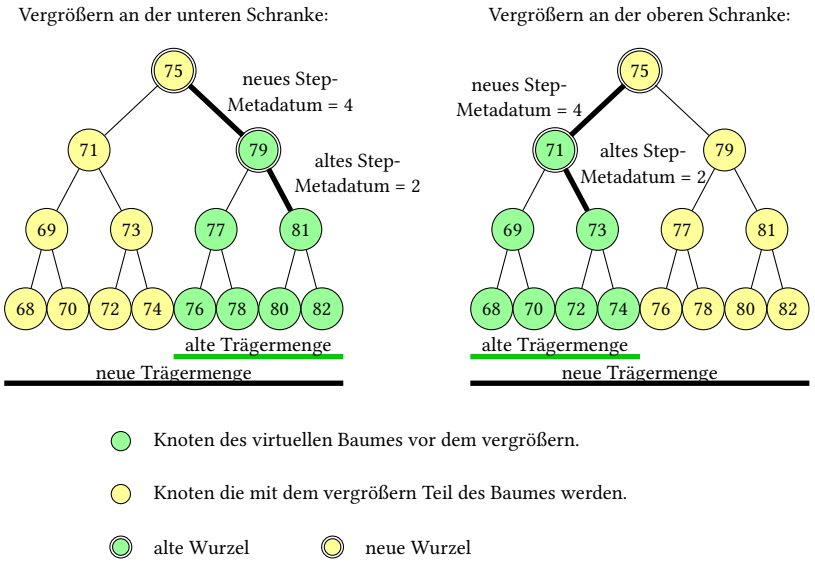

Abbildung 4.2: Vergrößern der Trägermenge mit dem Step-Metadatum

4.2 Anpassung der Baumstruktur bei wandernder Trägermenge

Die wandernde Trägermenge ist ein in 3.5 beschriebener, besonderer Fall einer dynamischen Trägermenge. Durch das planbare Wandern der Trägermenge besteht die Möglichkeit, die Metadaten pro-aktiv anzupassen. Dabei wird vor allem die Möglichkeit, den virtuellen Baum auf einen seiner Teilbäume verkleinern zu können, benutzt werden. Da auch die untere Schranke der Trägermenge angepasst werden soll, ohne eine Neuberechnung der Forknode nötig zu machen, wird der in Kapitel 4.1 vorgestellte, um das Step-Metadatum erweiterte, RI-Baum verwendet.

In diesem Kapitel ist mit „genutzte Trägermenge“ die Trägermenge gemeint, die genau alle tatsächlich gespeicherten Intervalle abdeckt. Abbildung 4.3 visualisiert das in diesem Absatz erläuterte Konzept graphisch. Im ersten Schritt wird die Trägermenge bestimmt, welche auf die zu Beginn genutzte Trägermenge passt. Diese wird dann mit den in 4.1 gezeigten Mitteln nach oben vergrößert.[1] Die vom virtuellen Baum unterstütze Trägermenge wird die Mächtigkeit der genutzen Trägermenge $* 2 + 1$ haben. Dargestellt ist dies in Abbildung 4.3 oben links. Die zu Beginn genutzte Trägermenge fängt somit am Anfang der vom virtuellen Baum unterstützten

[1] Wurzel = Wurzel + 2 * Step-Metadatum und Step-Metadatum = Step-Metadatum * 2

Trägermenge an und geht bis ein Element vor die Wurzel des virtuellen Baumes.

Durch das Verschieben der Trägermenge wandert diese im Baum nach rechts und über die Wurzel. Dies wird in Abbildung 4.3 oben rechts dargestellt. Dabei ist es unerheblich, ob das Verschieben wie beim Voranschreiten der Zeit in kleinen Millisekunden-Schritten erfolgt oder ob die Trägermenge in größeren Schritten verschoben wird. Zu jeder Zeit können für Schreib- und Lese-Zugriffe benötigte Forknodes mit Hilfe des virtuellen Baumes berechnet werden. Es wird anschließend zu der Situation kommen, dass die untere Schranke der genutzten Trägermenge über die Wurzel hinweg gewandert ist. Dies wird in Abbildung 4.3 unten links dargestellt. Ist die untere Schranke des Fenster über die Wurzel gewandert, sind in der Datenbank nur noch Einträge gespeichert, die einem Forknode aus dem Teilbaum rechts der Wurzel zugeordnet wurden. Im Beispiel aus Abbildung 4.3 wären nur noch Intervalle die einem Forknode größer 75 und kleiner 83 zugeordnet wurden, in der Datenbank gespeichert. Nur der rechte Teilbaum muss noch erhalten werden, die aktuelle Wurzel und der linke Teilbaum werden nicht mehr benötigt. Der virtuelle Baum wird daher auf den rechten Teilbaum der Wurzel verkleinert. Bevor die obere Schranke der Trägermenge über den größten Knoten des verkleinerten Baumes hinaus wandert, muss der virtuelle Baum wieder an der oberen Schranke vergrößert werden. Anderenfalls kann nicht mehr garantiert werden, dass für jedes Intervall der richtige Forknode berechnet werden kann. Das Vergrößern ist in Abbildung 4.3 unten rechts dargestellt. Der ehemals rechts der alten Wurzel liegende Teilbaum ist jetzt der linke Teilbaum der aktuellen Wurzel. Die Situation stellt sich wieder wie zu Beginn dar. Die genutzt Trägermenge beginnt am linken Ende des virtuellen Baumes und endet ein Element vor der Wurzel.

Die Schritte des Verkleinerns auf einen Teilbaum und das darauf folgende Vergrößern können direkt hintereinander ausgeführt werden oder getrennt. Es ist zwischen dem Vergrößern und Verkleinern möglich, Lese- (z.B. Überlappunsanfragen) und Schreib-Operationen konsistent durchzuführen, solange dass Vergrößern auf dem ehemals rechten Teilbaum stattfindet, bevor die obere Schranke der genutzten Trägermenge über die obere Schranke des Baumes hinaus wandert. Das vergrößern kann daher auch durch die Schreibprozedur angestoßen werden sobald ein Intervall mit einer Schranke oberhalb der Trägermenge eintrifft.

Nach dem Aktualisieren der Metadaten und damit der virtuellen Baumstruktur hat die Trägermenge wieder Platz, um weiter nach rechts verschoben zu werden, ohne aus dem vom Baum abgedeckten Bereich zu wandern. Für dieses Verschieben ist nur das Ändern von Metadaten nötig. Gleichzeitig wird sichergestellt, dass auch bei einer sich dauerhaft verschiebenden

Trägermenge, wie z.B. einem Zeitfenster der letzten x Stunden, alle Forknodes bei Anfragen korrekt berechnet werden und für neue Intervalle der richtige Forknode berechnet werden kann. Ein Anpassen auch während des kontinuierlichen Weiterlaufens der Zeit ist nur in bestimmbaren Abständen nötig. Die Höhe des Baumes erhöht sich im Vergleich zum genau auf die Trägermenge passenden virtuellen Baum nur um eine Ebene. Außerdem muss der Datenbank mitgeteilt werden, zu welchem Zeitpunkt die untere Schranke der genutzten Trägermenge größer als die Wurzel ist. Kann dies zu nicht-deterministischen Zeitpunkten der Fall sein, muss nach dem Löschen von Intervallen geprüft werden ob die Bedingung „untere Schranke der genutzten Trägermenge $>$ Wurzel“ erfüllt ist.

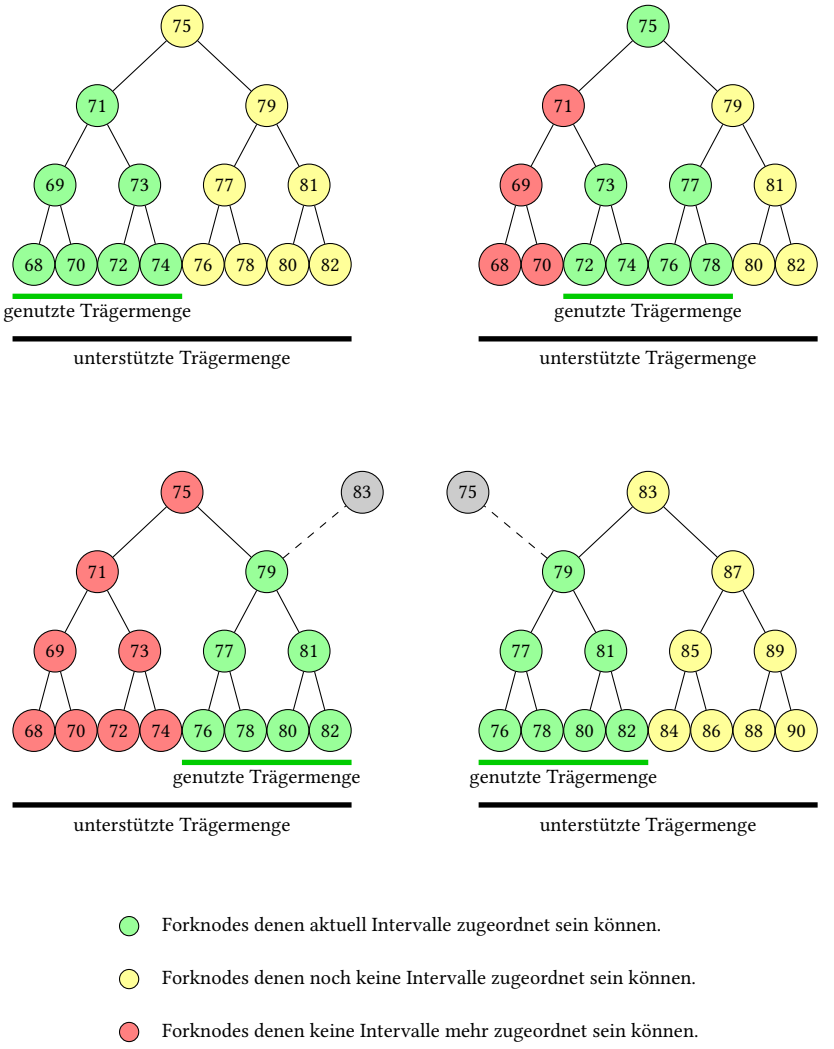

Abbildung 4.3: Wandern der genutzten Trägermenge über den Intervallbaum

4.3 Partitionierung der wanderenden Trägermenge

Im Kontext des wandernden Zeitfenster ist es sinnvoll, das Löschen der zu alten Intervalle durch die Verwendung von Partitionierung zu beschleunigen. Das Löschen einer Partition gleicht in der Geschwindigkeit dem Löschen einer Tabelle und ist damit deutlich schneller vollzogen als ein Löschen der Intervalle per „Delete“ SQL-Statement.

Als Besonderheit bei der Partitionierung der Intervalldaten ist zu beachten, dass die Intervall-Schranken jedoch in verschiedenen Partitionen liegen können. Die Intervalle also in mehrere Partitionen hineinreichen. Dies führt zu einigen Besonderheiten. So ist zu entscheiden, über welche der Schranken die Intervalle einer Partition zugeordnet werden. Sollten auf eine der beiden Schranken nachträglich noch Updates erfolgen, empfiehlt es sich, die konstante Schranke zu verwenden. Ansonsten kann ein geänderter Schrankenwert zu einer Zuordnung in eine andere Partition führen. Dies erzeugt zusätzlichen Lese- und Schreibaufwand, da der Datensatz in eine andere Partition verschoben werden muss. Wird über das Löschen ganzer Partitionen gelöscht entscheidet der Partitionierungsschlüssel auch darüber über welche der beiden Schranken gelöscht wird.

Da Intervalle sich über mehrere Partitionen erstrecken können, kann es auch Überlappungen zu Intervallen aus anderen Partitionen geben. Zusätzlich kann auch das Anfrage-Intervall mehrere Partitionen betreffen. Damit auch diese Überlappungen gefunden werden können und damit der RI-Baum die Intervalle wie verlangt komplett abdeckt, ist es nicht möglich, für jede Partition einen eigenen RI-Baum anzulegen. Stattdessen wird der virtuelle Baum weiter so gewählt, dass er die Trägermenge, die sich aus der Gesamtheit der Partitionen ergibt, abdeckt. Partitioniert wird nur die Tabelle mit den Intervallen und die beiden vom RDBMS angebotenen kombinierten Indizes aus dem Forknode und einer der Schranken. Die Partitionen von Tabellen und Indizes stehen dabei in direkter Beziehung zueinander.

Dieser Weg der Partitionierung führt dazu, dass die Partitionierung für den algorithmischen Anteil des RI-Baumes vollständig transparent ist und sich nur im Datenbank-Schema widerspiegelt. Es ist jedoch zu beachten, dass die für das Starten des Löschens verantwortliche Instanz auch für das Anlegen neuer Partitionen verantwortlich ist.

5 Implementation

Um den in Kapitel 4 entworfenen, erweiterten RI-Baum einer Evaluation unterziehen zu können, wird er implementiert. Es wird das Step-Metadatum (siehe 4.1) verwendet und eine wandernde Trägermenge betrachtet. Für die Implementation werden Zeitintervalle innerhalb eines sich verschiebenden Zeitraumes als dynamische Trägermenge als Beispiel gewählt, da hier der wichtigste Anwendungsfall für die Praxis gesehen wird. In Abbildungen dieses Kapitels werden Intervalle der natürlichen Zahlen über der bereits aus anderen Abbildungen bekannten Trägermenge von 68 bis 82 verwendet. Dies soll die Nachvollziehbarkeit der Abbildungen und Pseudocodes für den Leser verbessern.

5.1 Implementation auf dem Oracle Database RDBMS

Die Implementation wird auf dem Oracle 11.2g RDBMS in der Enterprise Edition durchgeführt. Wie bereits vorher erwähnt ist die Implementation des RI-Baumes auch auf anderen RDBMS möglich. Die Implementation des RI-Baumes ist zweigeteilt. So muss das Datenbank-Schema zur Unterstürzung des RI-Baumes im Vergleich zu einem Schema, welches nur die Intervalldaten enthält, leicht angepasst werden. Zusätzlich werden die Algorithmen zum Berechnen der algorithmischen Anteile in PL/SQL Implementiert.

5.1.1 Datenbankschema

Um den RI-Baum auf einem DB-Schema für Intervalle zu nutzen muss die Tabelle mit den Intervalldaten erweitert und eine Metadaten-Tabelle eingeführt werden. In dieser Implementation werden Intervalle als Tabelleneintrag mit je einer Spalte für untere und obere Schranke umgesetzt. Es ist jedoch in ORDBMS auch möglich, einen Objekttyp „Intervall“ zu definieren und diesen zu verwenden. Die Intervall-Tabelle wird um eine zusätzliche Spalte „Forknode“erweitert, welche die Forknode des Intervalls enthält. Diese hat den selben Datentyp wie obere und untere Schranke.
Die implementierte Tabelle sieht damit wie in Abbildung 5.1 aus. Jedoch in der konkreten

Implementierung mit einem Zeittyp in Millisekunden-Auflösung für untere Schranke, obere Schranke und Forknode.

ID	untere Schranke	obere Schranke	Forknode
1	68	69	69
2	77	81	79
3	72	76	75
4	74	74	74

Tabelle 5.1: Für den RI-Baum erweiterte Intervall-Tabelle mit beispielhaften Intervalldaten. Eingeordnet in den virtuellen Baum der Trägermenge: 68 bis 82.

Auf der Tabelle werden 2 kombinierte Indizes angelegt:

- lowerBoundIndex: (Forknode, untere Schranke)
- upperBoundIndex: (Forknode, obere Schranke)

Außerdem wurde die Konsistenz der Tabelle sichergestellt, indem untere „Schranke $\leq$ obere Schranke“ als Constraint definiert wird.

Zusätzlich wird eine Tabelle zur Haltung der Metadaten benötigt. Diese enthält, wie im unveränderten RI-Baum, die Wurzel und den Minstep. Auf den Offset wird allerdings verzichtet und stattdessen das Step-Metadatum aufgenommen. Die Tabelle wird nur eine Zeile haben. Auch die Möglichkeit eine linke und eine rechte Wurzel des LeftRoot und RightRoot Konzeptes zu verwenden ist nicht nötig.
Um bei der Evaluierung schnell zwischen verschiedenen hinterlegten Metadaten für verschiedene RI-Bäume wechseln zu können wurde außerdem eine identifizierende Spalte eingefügt, die den richtigen Metadatensatz identifiziert. Soll dauerhaft nur ein RI-Baum konfiguriert sein, ist es auch möglich auf diese Spalte zu verzichten, da die Metadaten Tabelle in diesem Fall nur einen Eintrag haben wird. Die entsprechende Where-Bedingung würde damit in den Pseudocodes entfallen. Die Metadaten-Tabelle hat die in Tabelle 5.2 gezeigte Struktur. Die Wurzel entspricht demselben Datentyp wie untere und obere Schranke. Der Step entspricht dem Datentyp, der Differenzen aus oberer und unterer Schranke beschreibt. In der Beispiel Implementation daher Zeitintervallen mit Millisekunden Genauigkeit. Die Informationen zur Wurzel und dem Step-Metadatum reichen aus um den kompletten Baum zu beschreiben. Sowohl die Grenzen der Trägermenge, als auch die Maximale Baumtiefe ergeben sich daraus.

User_Schema	Root	Step	Minstep
1	75	4	1

Tabelle 5.2: Geänderte Metadaten-Tabelle des relationalen Intervallbaumes

5.1.2 Algorithmischer Anteil

Neben den Anpassungen am Datenbankschema, muss der algorithmische Anteil des RI-Baums implementiert werden. Der algorithmische Teil enthält die Funktionen und Prozeduren zum Schreiben und Lesen. Benutzer der Datenbank sollen nur über diese Lese- und Schreib-Operationen auf die Intervalldaten zugreifen und nicht mehr direkt auf der Tabelle arbeiten. Ein anderer Weg wäre es, die Prozedur zur Berechnung der Forknodes an einen Datenbank Trigger zu binden. Dieser würde die Aufgabe der Schreibfunktion erledigen, sobald der Nutzer in die Tabelle schreibt. Um Schreib- und Lese-Operationen möglichst dicht beieinander zu halten, wird auf diese Möglichkeit verzichtet.

Die Implementation erfolgt mittels PL/SQL [Oracle, 2014] in einem PL/SQL-Package. Außerdem wird das PL/SQL-Package um Methoden zum initialen Füllen der Metadaten-Tabelle erweitert. PL/SQL-Packages teilen sich in zwei Teile. Der Head beschreibt dabei das nach außen angebotene Interface des Packages und der Body enthält die Implementationen der Prozeduren und Funktionen so wie interne Hilfsfunktionen/prozeduren.

Wie bereits in Kapitel 3.3 erläutert, muss zum Einfügen von Intervallen der Forknode des Intervalls berechnet werden. Dafür wird beim original RI-Baum der in Listing 3.1 abgebildete Algorithmus vorgeschlagen.

Dieser muss nur geringfügig abgewandelt werden, um für den veränderten RI-Baum genutzt zu werden. Die in der Schleife verwendete Variable Step beschreibt den Abstand des aktuellen Knotens zu seinen Tochterknoten. Zu Beginn wird die Differenz der Wurzel zu ihren Tochterknoten beschrieben, was dem neu eingeführten Step-Metadatum entspricht. Der Algorithmus wird daher nur wie in Listing 5.1 abgebildet verändert. Step wird mit dem Wert des Step-Metadatums initialisiert. Die Initialisierung von Step als Wurzel/2 entfällt. Zur Laufzeit wird das in der Metadaten Tabelle hinterlegte Step-Metadatum beim Aufruf der Prozedur verwendet.
Die gesamte, angepasste Schreibprozedur für Intervalle ist in Listing 5.2 dargestellt. Die verwendete Funktion zur Bestimmung des Forknodes wird angepasst, sowie die geladenen Metadaten,

ansonsten entspricht die Prozedur im Aufbau der des original RI-Baumes. Der Ablauf folgt weiter den Schritten:

1. Lesen der Metadaten. (Wurzel, Step, Minstep)
2. Prüfen ob virtuelle Baum das Intervall abdeckt.
3. Berechnen der Forknodes des Intervalls.
4. In die Tabelle schreiben des Intervalls samt Forknode.
5. Wird die Minstep Optimierung verwendet, falls nötig den Minstep anpassen.

Anders als im original RI-Baum wird in der Testimplementation keine Anpassung des Baumes bei nicht abgedeckten Intervallen vorgenommen, sondern diese als ungültige Eingabe betrachtet und ein Fehler geworfen.

```
FUNCTION forkNode (lowerBound, upperBound) {
        //laden der Metadaten
        SELECT root, step, minstep
           INTO node, intial_step, minstep
           FROM Metadaten_Tabelle
           WHERE User_Schema = testSchema;


        if(upperBound < lowerBound
                        || !(node - 2*step < lowerBound)
                        || !(upperBound < node + 2*step)){
                throw Exception;
        }


        for (step = intial_step; step >= 1; step = step/2) {
                if (upperBound < node) node -= step;
                elsif (node < lowerBound) node += step;
                else  break;
        } //now node is Forknode
        return node
}
```

Listing 5.1: Pseudocode zur Bestimmung des Forknodes bei Verwendung des Step-Metadatums.

```
PROCEDURE insertInterval(lowerBound, upperBound, id) {


        //laden der Metadaten
        SELECT root, step, minstep
           INTO node, intial_step, minstep
           FROM Metadaten_Tabelle
           WHERE User_Schema = testSchema;

        if(upperBound < lowerBound
                        || !(node - 2*step < lowerBound)
                        || !(upperBound < node + 2*step)){
                throw Exception;
        }

        //berechnen der Forknode
        for (step = intial_step; step >= 1; step = step/2) {
                if (upperBound < node) node -= step;
                elsif (node < lowerBound) node += step;
                else break;
        } //now node is Forknode

        INSERT INTO Intervals VALUES
        (node, lowerBound, upperBound, id);

        if(node != root && step < minstep){
                UPDATE Metadaten_Tabelle  SET minstep = step;
        }
        commit;
}
```

Listing 5.2: Pseudocode der Schreibprozedur des ans Step-Metadatum angepassten RI-Baumes.

Neben der Prozedur zum Schreiben von Intervallen wird auch eine zum Finden der Überlappungen benötigt. Die Implementation dieser Leseoperation zum Finden von Überlappungen entspricht dem bekannten Schema des unveränderten RI-Baumes.

1. Laden der Metadaten.
2. Prüfen ob das Suchintervall vom virtuellen Baum abgedeckt wird.
3. Berechnen der für die Anfrage relevanten Forknodes per PL/SQL.

4. SQL-Abfrage auf die Intervall-Tabelle unter Ausnutzung der beiden Indizes.

5. Rückgabe der erhaltenen Intervalle.

Die geladenen Metadaten werden um das Step-Metadatum erweitert und im Gegenzug auf den Minstep, sowie eine linke und rechte Wurzel verzichtet. Auch bei der Berechnung muss geprüft werden ob das Suchintervall vom virtuellen Baum abgedeckt wird. Auch beim Lesen wird in der Testimplementation ein Fehler geworfen wenn dies nicht der Fall ist. Wie in Kapitel 3.3 erläutert und in Abbildung 3.4 dargestellt, teilen sich die für die Überlappung relevanten Forknodes in drei Gruppen auf: Die Forknodes im Suchintervall sind bereits durch das Intervall selbst gegeben. Die relevanten Forknodes unterhalb der unteren Schranke sowie oberhalb der oberen Schranke müssen algorithmisch berechnet werden.
Diese werden im folgenden Pseudocodes in leftTupels und rightNodes gespeichert. Bei den rightNodes handelt es sich um eine Liste der relevanten Knoten, die oberhalb der oberen Schranke des Suchintervalls liegen. Bei leftTupels handelt es sich um eine Liste der relevanten Knoten, die unterhalb der unteren Schranke liegen, allerdings werden hier Tupel mit (Knoten, Knoten) gespeichert. Dies wird für eine später erläuterte Optimierung der SQL-Anfrage benötigt.

Nicht nur die möglichen Ergebnisse teilen sich in 3 Kategorien, sondern auch der Algorithmus zur Berechnung der relevanten Forknodes. Die Berechnung setzt sich aus einem Teil zusammen, der den virtuellen Baum bis zur Forknodes des Suchintervalls durchläuft und je einem Teil für die beide durch die Tochterknoten der Forknode bezeichneten Forknodes. Die PL/SQL-Prozeduren wurden an die Nutzung des Step-Metadatum angepasst.

Der Algorithmus zum Finden der relevanten Forknodes im linken Teilbaum unterhalb der Forknode entspricht dem Pseudocode in Listing 5.3. Der Algorithmus geht von der Forknode aus so lange zum linken Tochterknoten, bis er auf einen Knoten trifft, der nicht mehr innerhalb des Suchintervalls liegt. Dieser wird in die Liste der Tupel aufgenommen. Von diesem Knoten aus wird so lange der rechte Tochterknoten gewählt, bis ein Knoten innerhalb des Suchintervalls erreicht wird. Alle passierten Knoten außerhalb des Suchintervalls werden der Tupelliste hinzugefügt. Vom ersten Knoten innerhalb des Suchintervalls wird das Vorgehen wiederholt. Die Berechnung kann nach der Minstep Ebene oder dem Erreichen der unteren Schranke abgebrochen werden. Wird die Minstep Ebene unterschritten, ist in tieferen Ebenen kein Intervall mehr zugeordnet. Wird die untere Schranke erreicht, sind Intervalle, die in das Suchintervall überlappen diesem Knoten entweder zugeordnet oder liegen im rechten Teilbaum,

der als im Intervall liegender Knoten sowieso berücksichtigt wird. Im linken Teilbaum liegende Intervalle müssen bereits unterhalb der unteren Schranke enden, ansonsten wären sie dem Knoten der unteren Schranke zugewiesen worden.

```
leftTuples computeLeftNodes(node, step, lowerBound, minstep){

 for( ;step >= minstep && node != lowerBound ; step/2){
        if(node < lowerBound){
                leftTuples.add(node,node);
                node = node + step;
        }
        elsif(lowerBound < node){
                node = node - step;
        }
 }
 return leftTuples;
}
```

Listing 5.3: Pseudocode zum Finden der relevanten Knoten im linken Teilbaum der Forknode.

Der Algorithmus zum Finden der relevanten Forknodes im rechten Teilbaum unterhalb der Forknode entspricht dem Listing 5.4. Der Algorithmus verwendet das selbe Vorgehen wie der für den linken Teilbaum unterhalb der Forknode. Die Vergleiche und Links und Rechts sind allerdings entsprechend gespiegelt. Außerdem wird eine einfache Liste mit Knoten gefüllt und nicht eine Liste von Tupeln.

```
rightNodes computeRightNodes(node, step, upperBound, minstep){
  for( ;step >= minstep && node != upperBound ; step/2){
        if(node < upperBound){
                node = node + step;
        }
        elsif(upperBound < node){
                rightNodes.add(node);
                node = node - step;
        }
 }
 return rightNodes;
}
```

Listing 5.4: Pseudocode zum Finden der relevanten Konten im rechten Teilbaum der Forknode.

Der in Listing 5.5 gezeigte Algorithmus für den Weg zum Forknode ist dem zum Finden des Forknode sehr ähnlich. Allerdings werden auf dem Weg zum Forknode alle passierten Knoten jeweils entsprechend der Liste der rechten und linken relevanten Knoten hinzugefügt. Mit Erreichen des Forknode werden die Algorithmen für die beiden Teilbäume unterhalb des Forknode verwendet.

```
nodeListen computeForknodesOverlapping(node, intial_step,
                                lowerBound, upperBound, minstep) {
 for(step = intial_step ; ; step/2){
        if(node < lowerBound){
                leftTuples.add(node,node)
                node = node + step;
        }
        elsif(upperBound < node){
                rightNodes.add(node);
                node = node - step;
        }
        else{
                break; //now node is Forknode
        }
 }



 rightNodes.addAll(computeRightNodes(node, step,
                        upperBound, minstep));
 leftTuples.addAll(computeLeftNodes(node, step,
                        lowerBound, minstep));


 return (leftTuples, rightNodes);
}
```

Listing 5.5: Pseudocode zum Finden der für die Überlappung relevanten Knoten bis zur Forknode und delegation der suche in den Teilbäumen unterhalb der Forknode.

Die gesamte Prozedur zum Bestimmen von Überlappungen ist in Listing 5.6 als Pseudocode aufgeführt. Da die Algorithmen zum Bestimmen der relevanten Knoten auf den gleichen 2 Listen arbeiten wird darauf verzichtet, diese in Unterfunktionen aufzuteilen. Durch die Verwendung der Tupel-Liste und dem Hinzufügen des Suchintervalls als Tupel (untere Schranke, obere

Schranke) in die Tupel-Liste kann das DBMS interne Abfrage-Management vereinfacht werden, da nur noch 2 statt 3 Teilabfragen durchgeführt und vereinigt werden müssen.

```
Function ref_cursor overlappingIntervals(lowerBound, upperBound) {
  SELECT root, step, minstep
           INTO node, intial_step, minstep
           FROM Metadaten_Tabelle
           WHERE User_Schema = testSchema;

  if(upperBound < lowerBound
                        || !(node - 2*step < lowerBound)
                        || !(upperBound < node + 2*step)){
                throw Exception;
        }

  for(step = intial_step ; ; step/2){
        if(node < lowerBound){
                leftTuples.add(node,node)
                node = node + step;
        }
        elsif(upperBound < node){
                rightNodes.add(node);
                node = node - step;
        }
        else{
                break; //now node is Forknode
        }
  }

  forknode = node;
  forknode_step = step;

  for(; step >= minstep && node != lowerBound; step/2){
        if(node < lowerBound){
                leftTuples.add(node,node);
                node = node + step;
        }
        elsif(lowerBound < node){
                node = node - step;
        }
```

```
38   }
39
40
41
42   node = forknode;
43   step = forknode_step;
44   for( ;step >= minstep && node != upperBound ; step/2){
45         if(node < upperBound){
46                 node = node + step;
47         }
48         elsif(upperBound < node){
49                 rightNodes.add(node);
50                 node = node - step;
51         }
52   }
53   //Einfuegen des Suchintervalls so dass die im Intervall
54   //liegenden Forknode beruecksichtigt werden.
55   leftTuples.add(lowerBound, upperBound);
56
57   return SELECT * FROM Intervals i, leftTupels left
58     WHERE i.Forknode BETWEEN left.min AND left.max
59         AND i.upperBound >= lowerBound
60     UNION ALL
61         SELECT * FROM Intervals i, rightNodes right
62         WHERE i.Forknode = right.node AND i.lowerBound <= upperBound;
63 }
```

Listing 5.6: Pseudocode zum Finden von überlappenden Intervallen.

In der Implementation erfolgt die Rückgabe des Ergebnisses als Cursor. Dies ermöglicht ein einfaches Verarbeiten der Ergebnisse in Schleifen. Der Aufruf der Prozedur zum Finden von Überlappungen und die anschließende Verarbeitung der im Cursor zurückgegebenen Intervalle kann sowohl lokal auf dem RDBMS aus einer PL/SQL-Prozedur/Funktion heraus erfolgen, als auch über die von Oracle angebotene API aus einer anderen Programmiersprache.

Das Verwenden der PL/SQL-Packages bringt zusätzlich den Vorteil einer Entkopplung zwischen dem Datenbank-Schema und der Software, welche die Datenbank verwendet. Anwendungsentwickler müssen nur noch Aufrufe gegen das im Head des PL/SQL-Packages definierte Interface entwickeln und nicht mehr selbst SQL-Abfragen schreiben oder sich auf durch einen ORM-Mapper erstellte SQL-Abfragen verlassen. Dies bündelt die Verantwortlichkeit für die Anfragen

im PL/SQL-Package und damit innerhalb der Datenbank. Detailwissen über das Datenbank-Schema ist nicht nötig, um gegen das Interface zu programmieren. Viele Anpassungen an ein geändertes Datenbank-Schema können innerhalb des PL/SQL-Package-Bodys erfolgen, ohne dass eine Änderung der Anwendungen nötig wäre. Die Anpassung des PL/SQL-Package-Bodys stellt somit eine Alternative zum Einführen einer View dar.

Eine weitere Oracle-spezifische Optimierung ist die Nutzung des Optimizer-Hinweises „/*+ result_cache */“ für alle Metadaten-Tabellen-Anfragen. Dieser weist Oracle dazu an, das Ergebnis dieser Anfrage im Cache abzulegen und die selbe Anfrage so lange aus dem Cache zu laden, bis sich die Tabelle ändert. Dies beschleunigt die Abfrage, der die meiste Zeit gleich bleibenden Metadaten-Tabelle deutlich. Aufgrund der geringen Größe der Metadaten-Tabelle ist es auch auf kleinen Systemen kein Problem die Daten der Metadaten-Tabelle dauerhaft im Cache zu halten. Das Oracle DBMS wird bei Änderungen auf der Tabelle automatisch dafür sorgen, die im Cache liegenden Daten als ungültig zu markieren und bei Bedarf die Änderungen in diesen nachzuladen.

5.1.3 Verschieben des virtuellen Baumes

Es gibt für Zeitintervalle verschiedene Möglichkeiten das Verschieben der Forknodes zu starten. Da diese Implementation von einem festen, wandernden Zeitfenster ausgeht, kann das Verschieben zeitgesteuert erfolgen. Dafür wird ein vom Scheduler der Datenbank angestoßener Job verwendet. Alternativ kann das Verschieben auch umgesetzt werden, indem nach jedem Durchlauf des Löschens geprüft wird, ob die untere Schranke der Trägermenge über die Wurzel gewandert ist und es damit Zeit zum Verschieben der virtuellen Baumstruktur ist.
In der gewählten Implementation wird davon ausgegangen, dass das Löschen in kleineren Schritten, also häufiger als das Verschieben erfolgt. Der Algorithmus zum Verschieben der virtuellen Baumes ist dabei simpel gestaltet, da wie in Kapitel 4.2 erläutert nur ein Verkleinern auf den Teilbaum rechts der Wurzel und danach ein Vergrößern dieses Teilbaumes durchgeführt werden muss. Der Pseudocode für das Verschieben ist in Listing 5.7 dargestellt.

Für einen Langzeit Test wurde ein Partitionmanager konfiguriert, welcher alle X Stunden, alle Intervalle die älter als die Speicherdauer sind, löscht. Der Partitionmanager wurde dabei nicht selbst Implementiert sondern als bestehendes PL/SQL Script übernommen. Durch die bekannte Speicherdauer kann der Scheduler so konfiguriert werden, dass er nach jedem zweiten Ablauf der Speicherdauer direkt hinter dem Löschjob läuft und den virtuellen Baum verschiebt.

```
Procedure moveTree() {
            SELECT Root, Step, Minstep
            INTO root, intial_step, minstep
            FROM Metadaten_Tabelle;

            //verkleinern und vergrößern in einem Schritt
            root =  root + 2*step;

            UPDATE Metadaten_Tabelle SET Root = root;
            commit;
}
```

Listing 5.7: Pseudocode zum Verschieben des virtuellen Baumes.

5.1.4 Verkleinern auf den kleinsten, passenden Teilbaum

Durch die Verwendung des Step-Metadatums kann der virtuelle Baum auf jeden seiner Teilbäume verkleinert werden, ohne dass die gespeicherten Forknodes dieses Teilbaumes ungültig werden. Dies ermöglicht es, die abgedeckte Trägermenge auch nachträglich zu verkleinern, wenn z.B. das betrachtete Fenster von 12 Monaten auf 6 Monate oder 3 Monate verkleinert werden soll.
Eine Optimierung ist genau dann möglich, wenn es einen Teilbaum des virtuellen Baumes gibt, der bereits die Trägermenge abdeckt. Um den mindestens benötigten Teilbaum zur Abdeckung der gesamten genutzten Trägermenge zu bestimmen, müssen die minimale untere Schranke und die maximale obere Schranke aller gespeicherten Intervalle bestimmt werden. Der zu erhaltene Teilbaum lässt sich mit dem in Listing 5.8 dargestellten Algorithmus bestimmen.

```
FUNCTION findSubTree(minLowerBound, maxUpperBound, root, step){
        if(minLowerBound > root) {
                return findSubTree(minLowerBound, maxUpperBound,
                root + step, step/2)
        }
        elsif(maxUpperBound < root) {
                return findSubTree(minLowerBound, maxUpperBound,
                root - step, step/2)
        }
        else return (root, step);
}
```

Listing 5.8: Pseudocode zum Bestimmen des benötigten Teilbaumes.

5.2 Aktualisierung des Minstep nach dem Löschen von Intervallen

Ein Problem, welches sich mit dem eingeplanten Löschen von Intervallen ergibt, ist die komplexere Verwaltung des Minstep. Durch das Löschen der auf der Minstep-Ebene zugeordneten Intervalle kann es passieren, dass der Minstep sich weiter nach oben verschiebt. Dieses Problem besteht auch im ursprünglichen Intervallbaum, jedoch geht [Kriegel u. a., 2000] auf das Löschen nicht weiter ein. Die vorgestellten Änderungen werden benötigt, wenn mit der Zeit auch alte Intervalle wegfallen und Bereiche wieder freigegeben werden können.

Das Löschen nicht mehr benötigter Intervalle ist hier als elementarer Bestandteil des Anwendungskontextes zu betrachten. Löscht man gezielt Intervalle, ist noch einfach zu entschieden, ob eines davon einem Knoten in Minstep Tiefe zugeordnet ist. Ist dies der Fall muss geprüft werden, ob der Minstep noch gültig ist. In einem partitioniertem System wo ganze Partitionen entfernt werden, ist es allerdings gerade von Vorteil, nicht einzelne Intervalle berücksichtigen zu müssen, sondern ganze Partitionen zu entfernen und so effizient zu löschen.

Es werden zwei mögliche Lösungsansätze gesehen. Erstens ist es möglich, nach jedem Löschen einer Partition den aktuellen Minstep zu bestimmen und den global niedrigsten Minstep über alle Partition zu speichern. Dabei ist es jedoch nötig, die Tabelle komplett zu sperren um auch sicher zu stellen, dass parallel kein Intervall einem Forknode zugeordnet wird, der tiefer liegt als der Minstep der aktuell bestimmt wird.

Zweitens ist es auch möglich, den Minstep für jede Partition einzeln zu speichern und bei Anfragen den minimalen Minstep aller Partitionen als Minstep zu verwenden. Dies ist vor allem eine Option, wenn nur über das Entfernen ganzer Partitionen gelöscht wird.

Es sei noch einmal betont, dass nur der Minstep für eine Optimierung der arithmetischen Berechnung der Forknodes betroffen ist. Der Datenbank-interne Index wird durch das DBMS verwaltet und bleibt auch beim Löschen von Einträgen gültig.
Es geht nur darum, die Berechnung und Einbeziehung von im Index nicht vorhandenen Forknodes bei Abfragen zu vermeiden und so die Performance weiter zu optimieren. Außerdem ist wie bereits unter 3.3.2 Minstep genannt, nicht in jedem System ein Minstep sinnvoll, da dieser z.B. in Anwendungsfällen mit vielen Punktintervallen zu 1 wird.

5.3 Erweiterbarkeit

Neben der einfachen Implementierung ist auch eine Erweiterbarkeit des Konzeptes wichtig. So sind besonders weitere Where-Bedingungen für andere Spalten von Bedeutung. Dies kann entweder in Form weiterer, vorgegebener Argumente als Parameter erfolgen oder aber direkt durch Text, der als SQL Code an die Where-Klausel der Abfrage angehängt wird. Bei der Verwendung von dynamischen SQL Statements ist die Gefahr einer SQL-Injektion zu bedenken. Unter Oracle muss in PL/SQL selbst das Vorkommen gefährlicher Schlüsselwörter geprüft werden, da Oracle keine eigene Validierung gegen SQL-Injections für dynamischen SQL-Code bereitstellt. (vgl. [Oracle, 2008])

5.4 Partitionierungsmanager

Wie in 4.3 bereits erläutert ist eine Partitionierung der Intervalldaten-Tabelle im Kontext von löschenden Systemen sinnvoll. Die Verwaltung der Partitionen wird dabei jedoch nicht durch die Algorithmen des RI-Baumes vorgenommen. Hier bietet sich ein weiteres PL/SQL-Package an, welches als Partitionierungsmanager verwendet wird. Dieser ist für das Löschen alter Partitionen und das anlegen neuer Partitionen zuständig. Für die Tests, die mit partitionierten Tabellen durchgeführt wurden, wurde ein bestehenden Partitionierungsmanager eines mittelständischen Hamburger Unternehmens bereitgestellt.

6 Tests und Auswertung

Mit der Implementation aus Kapitel 5 soll im Folgenden eine Evaluation des geänderten RI-Baumes anhand praktischer Tests durchgeführt werden. Das verwendete Oracle 11.2g RDBMS in der Enterprise Edition wird für die Tests auf einer Maschine mit Intel Core i7-2600K CPU sowie 16GB Arbeitsspeicher und Windows Betriebssystem aufgesetzt.

6.1 Verschieben der Trägermenge

Die wichtigste Messgröße ist die beim Verschieben der Intervalle eingesparte Zeit. Hauptziel der Arbeit war es, dieses Zeit zu reduzieren. Es wird daher die benötigte Zeit für das Verschieben des virtuellen Baumes gemessen. Für den original RI-Baum enthält dies die benötigte Zeit zum Neuberechnen aller Forknodes. Die Messewerte für das Neuberechnen aller Forknodes sind bereits aus Abbildung 3.7 bekannt. Diese Werte werden mit der benötigten Dauer zum Verschieben der Trägermenge mit dem in dieser Arbeit vorgestellten Konzept in Relation gesetzt. In Abbildung 6.1 ist die experimentell ermittelte Zeit, die zum Verschieben des virtuellen Baumes benötigt wird, im Verhältnis zur Anzahl der gespeicherten Intervalle angegeben. Die Anzahl der Intervalle wurde für die Messungen in 500.000er Schritten erhöht und die Werte zwischen zwei Messpunkten durch lineares Verbinden interpoliert.
Im Diagramm ist klar zu erkennen, dass es im original RI-Baum einen linearen Zusammenhang zwischen vorhandenen Intervallen und zum Verschieben benötigter Zeit gibt. Der neue RI-Baum muss für jedes gespeicherte Intervall den Forknode neu berechnen um sicher zu gehen das dieser richtig ist. Der auf das Step-Metadatum angepasste RI-Baum hingegen muss keine Forknodes neu berechnen und operiert nur auf der Metadaten-Tabelle. Dies ermöglicht dem veränderten RI-Baum das Verschieben in konstanter Zeit zu lösen. Die dafür benötigte Zeit lag auf dem Testsystem dauerhaft im Bereich von 3 bis 5 Millisekunden. Im Vergleich mit dem neu Berechnen der Forknodes ist die Differenz zur X Achse daher im Diagramm nicht darstellbar. Deshalb finden sich die gemessenen Zeiten, für das verschieben mit dem Step-Metadatum, noch einmal gesondert in Abbildung 6.2. Das Ziel, das Verschieben des virtuellen Baumes in konstanter Zeit zu erreichen, darf damit als erreicht betrachtet werden.

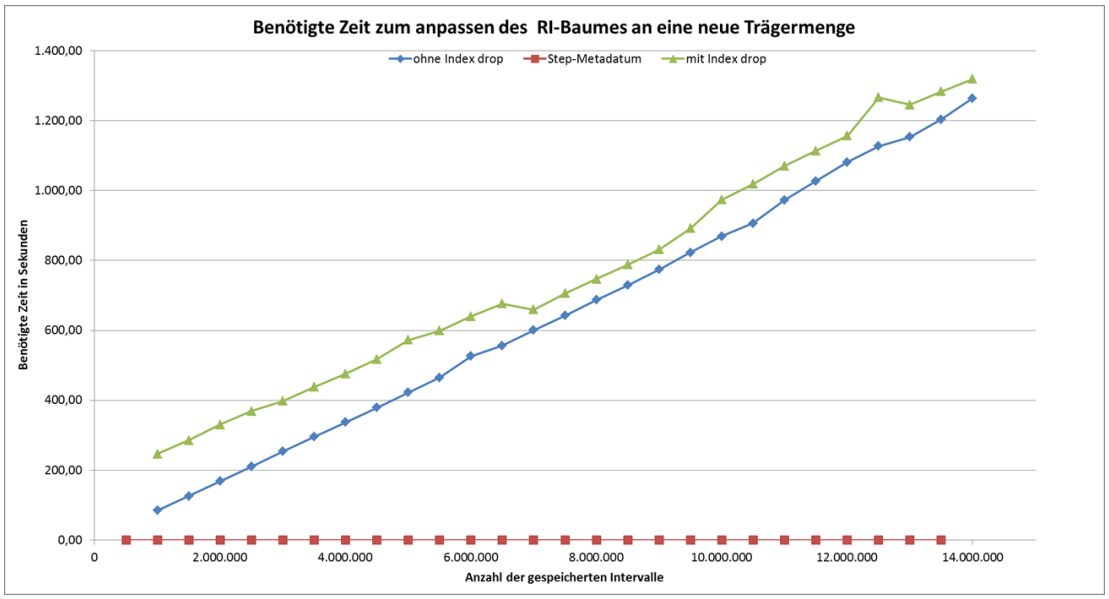

Abbildung 6.1: Benötigte Zeit zum Anpassen des RI-Baumes an eine neue Trägermenge.

In diesem Zuge sei auch festgestellt, dass auch der angestrebte Platzbedarf des angepassten RI-Baumes erreicht wird. Für jedes gespeicherte Intervall sind, wie im unveränderten RI-Baum, zwei Indexeinträge nötig. Es erfolgt ein Eintrag für den kombinierten Index aus Forknode und unterer Schranke, sowie ein Eintrag für den kombinierten Index aus Forknode und oberer Schranke.

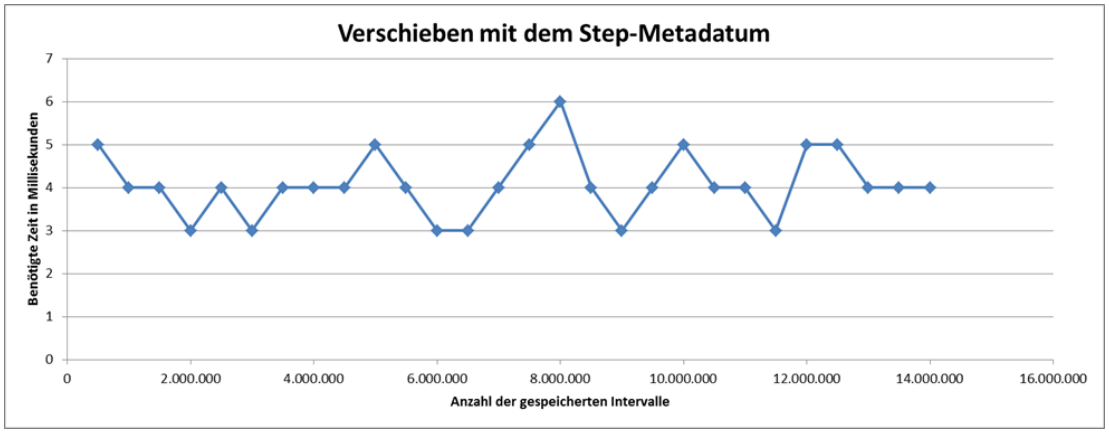

Abbildung 6.2: Benötigte Zeit zum Anpassen des RI-Baumes an eine neue Trägermenge.

6.2 Langzeit-Test

Der Mechanismus der Verschiebens der Trägermenge wird auch einem Langzeit-Test unterzogen. Auf dem Testsystem wird eine passende, dynamische Trägermenge, die ein Zeitfenster von einem Tag abdeckt, eingerichtet. Die Intervalldaten-Tabelle samt Indizes wird partitioniert. Insgesamt existieren zu jedem Zeitpunkt 12 in die Vergangenheit reichende 2-Stunden-Partitionen. Zusätzlich wird eine 13. Partition als Puffer in die Zukunft anlegt. Alle 2 Stunden wird durch einen Partitionmanger die älteste Partition gelöscht und eine neue Partition für die 2 Stunden vor der neuesten Partition angelegt.

In dieses 24-Stunden-Zeitfenster werden dauerhaft zufällig generierte Intervalle mit einer Mächtigkeit von einer Millisekunde bis zu einer Sekunde geschrieben. Das System wird so über mehrere Wochen in Betrieb gelassen. Zur Kontrolle werden in unregelmäßigen Abständen mit einem in Java geschriebenen Programm Überlappungsanfragen an das System gestellt. Dabei wird zufällige ein zum Anfragzeitpunkt im Zeitfenster liegendes Intervall mit einer Mächtigkeit von 10 Sekunden bis hin zu 59 Sekunden abgefragt. Die Abfrage wird jeweils mit Hilfe des RI-Baumes und ohne diesen durchgeführt. Die Anzahl der zurückgegebenen Intervalle wird danach verglichen, um die Korrektheit der Ergebnisse zu überprüfen.

Durch den Test konnte experimentell bestätigt werden, dass der RI-Baum auch über eine längere Laufzeit und nach mehrfachen Verschieben der Trägermenge korrekte Ergebnisse liefert.

6.3 Überlappende Intervalle finden

Das Ziel beim Einsatz des RI-Baumes ist es, Such-Anfragen auf Intervalle zu beschleunigen. Ein besonderer Schwerpunkt wird dabei auf die Frage der Überlappung gelegt. Im Folgenden soll daher die Bearbeitungszeit von Überlappungsanfragen untersucht werden.
Im Vergleich zum angepassten RI-Baum wird einmal die benötigte Zeit ohne Index und einmal die benötigte Zeit mit einem Interval-Spatial Transformation Index (IST) (vgl. [Goh u. a., 1996]) gemessen. Der Umsetzung des IST ist im Beispielfall identisch zu einem kombiniertem Index auf (obere Schranke, untere Schranke) (vgl. [Kriegel u. a., 2000]. Zusätzlich wird durch die Autoren noch eine Implementation des Tile Index [Oracle, 1999] in den Vergleich aufgenommen. Allerdings wurde durch die Autoren eine Neu-Implementation vorgenommen, die effektiver als die Standard-Implementation ist. Da keine Details zu den Änderungen genannt wurden

und auch die optimierte Version in den durch die Autoren durchgeführten Tests schlechter abschneidet, als der RI-Baum, wird darauf verzichtet, selbst eine Version zu implementieren und zu testen.
Die Messung wird mit gleichmäßig auf eine 3 Tage große Trägermenge aufgeteilten Zeitintervallen durchgeführt. Die Anfragen werden mit 3 Minuten langen Suchintervallen gestellt, die zu Beginn des Zeitfensters positioniert werden. Alle Anfragen liefern 28800 Intervalle als Ergebnis zurück. Zusätzlich zu den Intervall Schranken werden auch der Forknode und die ID des Intervalls mit Abgefragt.

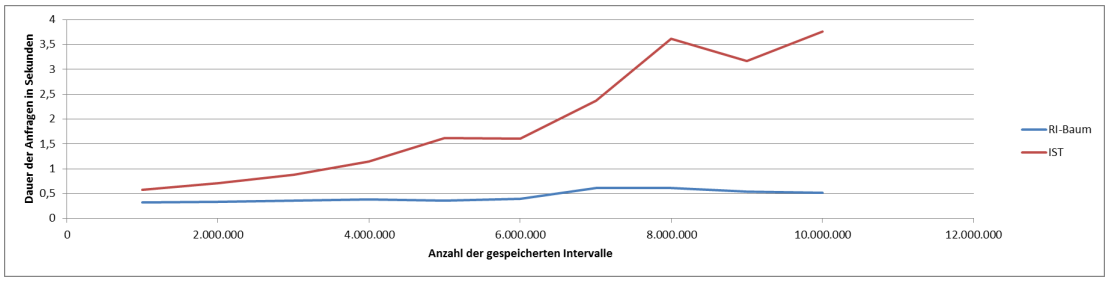

Abbildung 6.3: Benötigte Zeit zum Beantworten von Überlappungsabfragen mit verschiedenen Indizes.

In Abbildung 6.3 ist die durchschnittliche benötigte Zeit zum beantworten einer Abfrage für verschiedene Intervall Mengen abgebildet. Es werden dabei nur IST und RI-Baum abgebildet da die Messungen ohne Index Verwendungen deutlich schlechter abschneiden. Eine Angabe würde nur die Lesbarkeit des Diagramms verschlechtern. Es zeigt sich das der RI-Baum kaum durch die Menge der gespeicherten Intervalle beeinflusst wird während der IST deutlich durch diese Beeinflusst wird. In einem separaten Vergleich zwischen einem RI-Baum mit einer genau passenden Trägermenge und einem aufs Verschieben angepassten RI-Baum kann bei der Laufzeiten von Abfragen kein Unterschied festgestellt werden. Die theoretisch etwas schlechtere Performance des angepassten RI-Baumes durch den um eine Ebene höheren, virtuellen Baum ist scheinbar kleiner als die nicht beeinflussbaren Schwankung der Laufzeiten. In den in Abbildung 6.3 visualisierten Messungen zeigt sich auch, dass der RI-Baum zum Bestimmen der Überlappungen die schnellste der getesteten Möglichkeiten darstellt.

Für eine ausführlichere Performance-Analysen zum Konzept des RI-Baumes sei auf [Kriegel u. a., 2000] verwiesen. Dort wird deutlich detaillierter der Einfluss verschiedener Parameter auf die Performance des RI-Baumes, sowie die Vergleichs-Indizes untersucht und dargestellt.

6.4 Test auf echten Daten

Zusätzlich wird der RI-Baum auch auf Intervalldaten aus einem echtem System getestet. Echt bedeutet in diesem Fall, dass die Intervall-Daten in ihrer Verteilung und anderen Charakteristiken einem existierende, im Betrieb befindlichen System entsprechen. Die Daten werden netterweise von einem mittelständischen Unternehmen aus Hamburg bereitgestellt.
Es handelt sich dabei um Zeitintervalle mit Millisekunden-Auflösung. Die Intervalle stammen aus Sensorsystemen, die diese Daten produzieren um erfasste Werte Zeiträumen zuzuordnen. Das betrachtete System wird durch die folgenden Eigenschaften charakterisiert.
Als Trägermenge werden rückblickend die letzten Wochen betrachtet. Es handelt sich dabei um eine einstellige Wochenanzahl. In dem betrachteten Zeitraum herrscht ein sehr hohes Aufkommen an Intervallen. Konkret wird aus diesem Zeitraum ein Ausschnitt mit 10 Millionen Intervallen betrachtet. Diese verteilen sich auf einen Zeitraum von etwas unter 3 Stunden. Ausgehend von dieser 3 Stunden umfassenden Trägermenge, ergibt sich ein Datenaufkommen von über 55.000 neuen Intervallen pro Minute. Die aus diesem System für die Tests kopierten Intervalle verteilen sich gleichmäßig. Der Großteil der Intervalle ist kurz und hat eine Länge im Millisekunden oder einstelligen Sekunden Bereich. So gibt es Intervalle mit einer Länge von einer Millisekunde. Das längste Intervall hat eine Länge von 15 Minuten und 45,997 Sekunden. Die durchschnittliche Länge der Intervalle beträgt 2,627 Sekunden. Auf diesen Daten werden wie in Kapitel 6.3 die Performance der Anfragen mittels RI-Baum und IST gemessen.

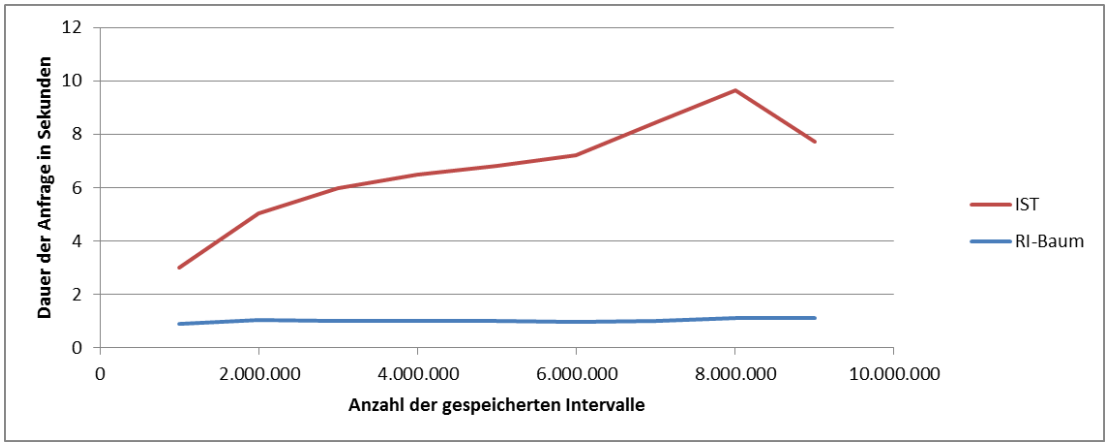

Abbildung 6.4: Benötigte Zeit zum Beantworten von Überlappungsabfragen mit verschiedenen Indizes auf echten Daten.

Die Anfragen werden mit einem Suchintervall mit einer Länge von 10 Minuten gestellt. Das Suchintervall wird dabei nahe der unteren Schranke der Trägermenge platziert. Dies führt

zu einer schlechteren Performance des IST als eine Platzierung an der oberen Schranke der Trägermenge. Dies wird so gewählt da die bestmöglichen Laufzeiten für den realen Einsatz weit weniger interessant sind als die Laufzeiten nicht optimaler Fälle. In Abbildung 6.4 wird deutlich das der Intervallbaum auch auf echten Daten seinen Vorteile bei der Abfrage Bearbeitung ausspielen kann.

6.5 Einfluss der Höhe des virtuellen Baumes

Die in dieser Arbeit vorgestellten Änderungen dienen dazu, den virtuellen Baum an die genutzte Trägermenge anzupassen und möglichst passend zu halten. In [Kriegel u. a., 2000] wird der Einfluss der Höhe des virtuellen Baumes auf Anfragen als vernachlässigbar bezeichnet. Gemessen wurde dies von den Autoren durch Anfragen auf Datensätze mit verschieden tiefliegenden Minstep in der Trägermenge $[0, 2^{20} - 1]$. Daraus ergibt sich die Annahme, dass es auch möglich ist, den RI-Baum direkt den gesamten Datentyp abdecken zu lassen oder mit einem passenden RI-Baum zu initialisieren, der bei Bedarf vergrößert wird ohne jedoch die untere Schranke anzupassen.
Die im Kontext dieser Arbeit betrachteten Systeme werden jedoch nicht nur durch eine große Anzahl an gespeicherten Intervallen charakterisiert. Zusätzlich besteht bei diesen eine hohe Fluktuation der Daten. Somit ist auch der Einfluss auf das Löschen und Schreiben von Intervallen von Bedeutung. Beim Löschen von Intervallen wird der virtuelle Baum nicht berücksichtigt.

Beim Schreiben neuer Intervalle wird der virtuelle Baum zum Bestimmen der Forknode genutzt. Daher soll im Folgenden der Einfluss der Höhe des virtuellen Baumes auf die Schreibrate überprüft werden. Es wird dabei weiter von durch Zeitstempel beschränkten Intervallen ausgegangenen. Die Messungen wird mit einem RI-Baum über einem einstündigen Zeitfenster begonnen. In dieses werden 100.000 Intervalle geschrieben. Dabei werden 20 mal jeweils 5000 Punktintervalle in Abständen von 5-Millisekunden-Schritten über die Trägermenge verteilt. Dies bewirkt, dass jedes 2. Intervall auf der tiefsten Ebene eingeordnet wird. Die anderen Intervalle werden auf unterschiedlichen Ebenen eingeordnet.
In jeder weiteren Messung wird der virtuelle Baum um einen Schritt an der oberen Schranke vergrößert (siehe. Kapitel 4.1). Mit jeder Messung werden Intervalle mit identischen Schranken in das zu Beginn festgelegte Zeitfenster geschrieben. Die Tabelle wird zwischen den Messungen gelöscht und die Indizes neu angelegt. Die Messgänge unterscheiden sich daher nur in dem durch die Metadaten beschriebenen virtuellen Baum.
In Abbildung 6.5 sind die durchschnittlich gemessenen Zeiten zum Schreiben der 100.000

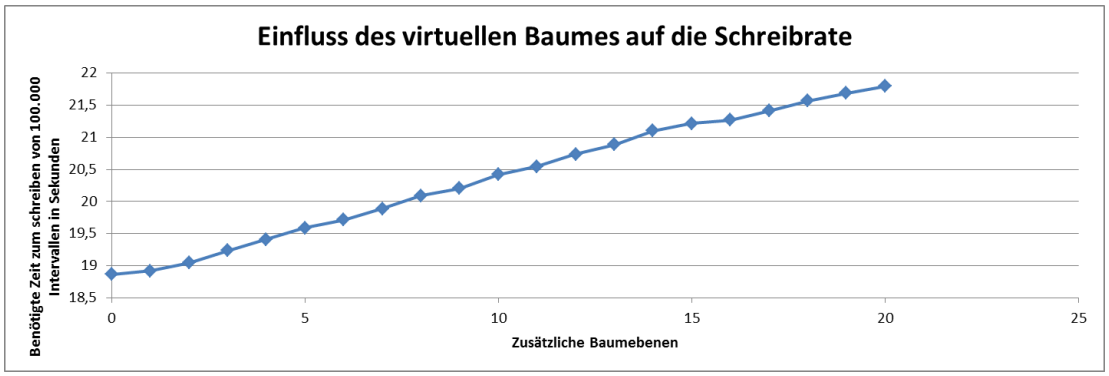

Abbildung 6.5: Durchschnittlich benötigte Zeit zum Schreiben von 100.000 Intervallen bei steigender Höhe des virtuellen Baumes.

Intervalle über 30 Messdurchläufen abgebildet. Der virtuelle Baum wurde um bis zu 20 Stufen erhöht. Ein weiteres Vergrößern des Baumes würde mit dem gewählten Datentyp zu einem Überlauf des Step-Metadatum Datentyps führen.
Gestartet wird die Messung mit einer Wurzel am 28.10.15 18:09:49,572. Ein weiteres Vergrößern an der unteren Schranke ist daher auch möglich, entspricht aber nicht der Abbildung des in dieser Arbeit betrachteten wandernden Zeitfensters.
Es zeigt sich, dass auf Grundlage der passenden Trägermenge für eine Stunde, die Schreibdauer bis zur maximalen Wurzel von durchschnittlich 18,865 Sekunden auf 21,793 Sekunden ansteigt. Dies entspricht einer Steigerung von $15,52\%$.

Es wird dadurch deutlich, dass ein Abdecken des gesamten Wertebereichs des Datentyps direkt zur Initialisierung des RI-Baumes zwar möglich ist, in Systemen mit einer hohen Schreibrate jedoch einen Performance Einbruch bedeutet.

Eine alternative Möglichkeit besteht darin, mit einem passenden, virtuellen Baum zu initialisieren und diesen bei Bedarf zu vergrößern, ohne die untere Schranke der Trägermenge anzupassen. In diesem Fall wird die Schreib-Performance mit der Lebensdauer des Systems leicht sinken. Auf Grund der quadratischen Abhängigkeit zwischen Baumhöhe und abgedeckter Trägermenge wird es sehr lange dauern, bis der maximale Performance-Einbruch erreicht ist. Jedoch verliert das System zu Beginn schneller an Schreibrate.
Um dies besser zu verdeutlichen wird in Tabelle 6.1 die Entwicklung der Wurzel und der prozentuale Anstieg der benötigten Zeit zum Schreiben der 100.000 Intervalle aufgeführt. Zur besseren Einordnung wird außerdem die Mächtigkeit der abgedeckten Trägermenge angegeben.

Baum Erhöhungen	Wurzel	|Trägermenge|	Zeitzunahme in %
0	28.10.15 18:09:49,572	00 01:09:54.303	0%
1	28.10.15 18:44:46,724	00 02:19:48.607	$0,286\%$
2	28.10.15 19:54:41,028	00 04:39:37.215	$0,959\%$
3	28.10.15 22:14:29,636	00 09:19:14.431	$1,977\%$
4	29.10.15 02:54:06,852	00 18:38:28.863	$2,889\%$
5	29.10.15 12:13:21,284	01 13:16:57.727	$3,838\%$
6	30.10.15 06:51:50,148	03 02:33:55.455	$4,5\%$
7	31.10.15 20:08:47,876	06 05:07:50.911	$5,412\%$
8	03.11.15 22:42:43,332	12 10:15:41.823	$6,504\%$
9	10.11.15 03:50:34,244	24 20:31:23.647	$7,087\%$
10	22.11.15 14:06:16,068	49 17:02:47.295	$8,242\%$
11	17.12.15 10:37:39,716	99 10:05:34.591	$8,915\%$
12	05.02.16 03:40:27,012	198 20:11:09.183	$9,928\%$
13	14.05.16 13:46:01,604	397 16:22:18.367	$10,718\%$
14	29.11.16 09:57:10,788	795 08:44:36.735	$11,852\%$
15	01.01.18 02:19:29,156	1590 17:29:13.471	$12,468\%$
16	06.03.20 11:04:05,892	3181 10:58:26.943	$12,743\%$
17	14.07.24 04:33:19,364	6362 21:56:53.887	$13,512\%$
18	30.03.33 15:31:46,308	12725 19:53:47.775	$14,339\%$
19	31.08.50 13:28:40,196	25451 15:47:35.551	$14,964\%$
20	04.07.85 09:22:27,972	50903 07:35:11.103	$15,521\%$

Tabelle 6.1: Entwicklung der Trägernmenge, Wurzel und der Schreibrate bei Betrachtung eines einstündigen Fensters.

Die Angabe erfolgt dabei nach dem Schema Tage Stunden:Minuten:Sekunden:Millisekunden.

Ein am 28.10.2015 aufgesetztes System mit einer Trägermenge, die eine Stunde abdeckt, würde bereits in diesem Jahr, ab 17.12.2015, 8,9% länger brauchen um 100.000 Intervalle zu schreiben. Um die selbe Steigerung noch ein mal drauf zu legen müsste das System danach jedoch bis 2085 laufen.

Wie sehr diese Werte auf reale Systeme übertragbar sind, ist jedoch noch weiter zu prüfen. So lassen weitere, neben den eigentlichen Intervalldaten zu schreibende Daten, den prozentualen Anteil der arithmetischen Operationen sinken. Zusätzlich beziehen sich diese Ergebnisse nur auf Oracles Zeitstempel und Zeitintervall-Datentypen. Der Einfluss der arithmetischen Operationen fällt auf einfachen nummerischen Werten wahrscheinlich geringer aus. Es ist jedoch denkbar, dass der Einfluss für komplexere Objekttypen noch größer ausfallen kann.

Die Entscheidung, die hier Vorgestellten Mittel zum Verwalten eines passenden Baumes zu verwenden, ist aufgrund des gemessenen Einflusses jedoch keine reine Performance-Entscheidung. Trotzdem sprechen folgende Argumente für die vorgestellten Ansätze.

Das Step-Metadatum stellt im ersten Schritt eine Vereinfachung dar. Anstatt der Konzepte des Offsets und der RightRoot und LeftRoot wird nur noch ein Konzept benötigt, um sowohl ein dynamisches Vergrößern an der untere Schranke, als auch an der oberen Schranke der Trägermenge zu ermöglichen. Dies vereinfacht das Verständnis und verringert damit die Wahrscheinlichkeit von Implementationsfehlern.

Zusätzlich bedeutet das Verschieben der Trägermenge mit den in dieser Arbeit vorgestellten Mitteln keinen Mehraufwand im Vergleich zum Vergrößern des Baumes. Abgesehen von einer etwas komplexeren Implementation kann hier ohne Nachteile unnötige CPU-Last vermieden werden. Ein Abdecken des Gesamten Datentyps durch den RI-Baum erfordert gar keine Verwaltung der Metadaten, bedeutet aber eine messbare Abnahme der Schreibgeschwindigkeit.

7 Zusammenfassung und Ausblick

Im Folgenden werden die Inhalte dieser Arbeit noch ein mal Zusammengefasst uns Ausblick auf zukünftige Forschungsfragen erfolgen.

7.1 Zusammenfassung

In dieser Arbeit wurde das Konzept des RI-Baum an das Szenario einer wandernden Trägermenge angepasst und realisiert. Die Realisierung wurde zusätzlich einer Evaluierung unterzogen. Zu Beginn wurden in Kapitel 2 die Grundlagen zu Intervallen 2.1 und RDBMS 2.2 erläutert. Auf diesem Wissen aufbauend wurde in Kapitel 3 der Einsatz von Intervallen in Informationssystemen beleuchtet (3.1) und der RI-Baum (3.3) als Konzept zur Intervall-Indizierung vorgestellt. Es wurde das Problem der wandernden Trägermenge (3.5) eingeführt und der Einsatz des RI-Baumes im Kontext dieses Problems untersucht. (3.6) Im Zuge dieser Untersuchung wurde ein Optimierungspotential erkannt und Ziele (3.7) für einen angepassten RI-Baum formuliert.

Zur Anpassung des RI-Baumes wurde das Step-Metadatum (4.1) eingeführt, um den RI-Baum frei im Wertebereich des Datentyps platzieren zu können. Das Step-Metadatum zeigte sich als geeignet um das Konzept des Offsets, sowie das Konzept der RightRoot und LeftRoot zu ersetzen. Als nächstes wurde ein auf dem Step-Metadatum aufbauendes Konzept zum Verschieben des RI-Baumes vorgestellt (4.2).

In Kapitel 5 wurde eine mögliche Implementation des angepassten RI-Baumes erläutert. Diese Implementation teilt sich in einen schematischen (5.1.1) und einen algorithmischen (5.1.2) Anteil. Zusätzlich wurde noch auf die Probleme der Minstep-Pflege im Zusammenhang mit dem Löschen von Intervallen (5.2) und die Erweiterbarkeit der Implementation (5.3) eingegangen.

Im Kapitel 6 wurde die Implementation einer Evaluation unterzogen. Es zeigte sich, dass die Implementation auch praktisch in der Lage ist, auch ohne Neuberechnung der Forknodes die wandernde Trägermenge zu realisieren und das Verschieben der Trägermenge in konstanter Zeit zu lösen. Auch wurde noch einmal die Laufzeit des RI-Baumes zum Bestimmen von

Überlappungen evaluiert. Dieser Test wurde zusätzlich auf Intervalldaten aus einem Produktiv-System wiederholt. Damit wurde gezeigt, dass der RI-Baum auch im Einsatz auf Daten aus einem realen System geeignet ist, um das Problem der Überlappung effizient zu lösen. Zusätzlich wurde gezeigt dass ein angepasster virtueller Baum, sich positiv auf die Schreibrate auswirkt.

7.2 Ausblick

Auch für die Zukunft stellen sich weitere interessante Fragen im Bezug auf die Indexierung von Intervallen. Auch die letzten Jahre zeigen einen unveränderten Trend im Wachstum der gespeicherten Datenmengen. Dieser Trend wird sich aller Voraussicht nach noch verstärken. So gibt es viele Forschungs- und Entwicklungsprojekte, die sich mit Systemen beschäftigen die durch massiven Einsatz von Sensorsystem-Informationen über ihre Umwelt sammeln.
Beispiele sind hier das selbst fahrende Auto oder intelligente Wohnungen. Ziel dieser Systeme ist es, auf Grundlage der gesammelten Daten auf ihre Umwelt reagieren oder sogar mit dieser interagieren zu können. Gerade die angestrebte Reaktion und Interaktion mit der realen Welt stellen oft Echtzeit- oder Quasi-Echtzeit-Anforderungen an die verarbeitenden Systeme.
Gerade Zeitintervalle werden in diesem Kontext ihre Relevanz behalten, so dass Techniken zum schnellen Verarbeiten dieser benötigt werden. Der RI-Baum hat sich dabei als geeignete Struktur zum Optimieren von Überlappungsanfragen erwiesen.

Für die weitere Forschung am RI-Baum werden verschiedene Ansatzpunkte gesehen. So ist eine Untersuchung des Einflusses der Baumhöhe in Produktiv Systemen weitere Untersuchungen Wert. Auch konnte der Einfluss der Baumhöhe auf den um weitere Intervall Relationen erweiterten RI-Baum, im Rahmen dieser Arbeit nicht untersucht werden. Um den Einsatz für noch mehr bestehende RDBMS zu ermöglichen wäre ein möglicher Ansatz, die Anforderungen an die Integration in das DBMS wider zu senken und den algorithmischen Teil in eine Software außerhalb des RDBMS zu verschieben. Dies würde die Umsetzung des RI-Baumes auf RDBMS ohne prozedurale Erweiterung von SQL ermöglichen. Hierbei würde nur der algorithmische Teil ausgelagert werden, zum Verarbeiten der Anfragen auf Datenbankebene würden weiter die RDBMS internen Indexstrukturen verwendet. Eines der zu lösenden Teilprobleme wäre effiziente Wahrung der Konsistenz der Metadaten zwischen DBMS uns externer Algorithmen-Software.

Im Zuge der Externalisierung des algorithmischen Anteils wird daher auch der Wechsel des Datenbankmodels vom relationalen Datenmodel zu einem Datenmodel aus der Familie der

sogenannten NoSQL-Datenbanken als prüfenswerter Ansatz gesehen.
Neben NoSQL-Datenbanken wird zur Steigerung der Performance besonders prüfenswertes Potential in der Verwendung von InMemory-Datenbanken gesehen. So könnten sowohl relationale InMemory-Datenbanken als auch nicht relationale InMemory-Datenbanken eine Betrachtung wert sein. Für relationale InMemory-Datenbanken wird TimesTen von Oracle als viel versprechende Möglichkeit betrachtet. So bietet TimesTen auch eine Integration des in dieser Arbeit bereits verwendeten PL/SQL. Aufgrund ihrer Struktur wären KeyValue-Datenbanken, als typische InMemory-Datenbanken der NoSQL-Familie, denkbar.
Zur Umsetzung auf einer KeyValue-Datenbank wäre die bereits genannte Externalisierung des algorithmischen Anteils in eine externe Software nötig. Der Ansatz wird darin gesehen, den Forknode extern zu berechnen und diesen in der KeyValue-Datenbank als Schlüssel zu verwenden. Der Forknode als Schlüssel verweist wiederum auf eine Liste aller ihm zugeordneten Intervalle. Die Struktur würde damit wieder der zweigeteilten Struktur des Intervallbaumes ähneln.

Eine weitere Steigerung der Performance durch die Verwendung von NoSQL und InMemory Datenbanken könnte für verschiedene Anwendungsbereiche interessant sein. Reaktive Systeme kommen in den verschiedensten Bereichen zum Einsatz und sind auf Minimale Antwortzeiten angewiesen.
Zusätzlich wurde in der Analyse auf weitere, bestehende Index-Konzepte zum Indizieren von Intervallen verwiesen. Viele dieser Konzepte basieren wie der Intervallbaum auf Baumstrukturen. Daher wäre auch bei diesen Konzepten eine ähnliche Anpassung wie beim Intervallbaum zum relationalen Intervallbaum ein prüfenswerter Ansatz, um die Integration dieser Konzepte in bestehende RDBMS zu ermöglichen.

Literaturverzeichnis

[Brochhaus u. a. 2005] Brochhaus, Christoph ; Enderle, Jost ; Schlosser, Achim ; Seidl, Thomas ; K., Stolze: Integrating the Relational Interval Tree into IBM's DB2 Universal Database Server. In: *Proc. 11th GI Conference on Database Systems for Business, Technology, and the Web (BTW 2005), Karlsruhe, Germany. GI-Edition Lecture Notes in Informatics 65*. Bonn,Germany : GI, 2005, S. 67–86. – Online verfügbar unter: http://dme.rwth-aachen.de/de/system/files/file_upload/publications/BES05-btw.pdf Abruf: 2015-09-14

[Deitmar 2014] Deitmar, Anton: *Analysis*. Wiesbaden : Springer Spektrum, 6 2014. – Online verfügbar unter: http://link.springer.com/content/pdf/10.1007%2F978-3-642-54810-9.pdf Abruf: 2015-07-03. – ISBN 978-3-642-54809-3

[Edelsbrunner 1980] Edelsbrunner, Herbert: *Dynamic Rectangle Intersection Searching* -. Graz : Institut für Informationsverarbeitung Technische Universität Graz, 1980

[Elmasri u. a. 1990] Elmasri, Ramez ; Wuu, Gene T. J. ; Kim, Yeong-Joon: The Time Index: An Access Structure for Temporal Data. In: McLeod, Dennis (Hrsg.) ; Sacks-Davis, Ron (Hrsg.) ; Schek, Hans-Jörg (Hrsg.): *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings.*, Morgan Kaufmann, 1990, S. 1–12. – Online verfügbar unter: http://www.vldb.org/conf/1990/P001.PDF Abruf: 2015-09-11. – ISBN 1-55860-149-X

[Goh u. a. 1996] Goh, Cheng H. ; Lu, Hongjun ; Ooi, Beng-Chin ; Tan, Kian-Lee: Indexing Temporal Data Using Existing B+-trees. In: *Data Knowl. Eng.* 18 (1996), März, Nr. 2, S. 147–165. – Online verfügbar unter: http://dx.doi.org/10.1016/0169-023X(95)00034-P Abruf: 2015-02-13. – ISSN 0169-023X

[Graefe 2003] Graefe, Goetz: Sorting And Indexing With Partitioned B-Trees. In: *CIDR 2003, First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, Januar, 2003, Proceedings*, www.cidrdb.org, 2003, S. unbekannt. – Online verfügbar unter: http:

//www-db.cs.wisc.edu/cidr/cidr2003/program/p1.pdf Abruf: 2015-10-28

[Kriegel u. a. 2000] Kriegel, H ; Poetke, M. ; Seidl, T.: Managing Intervals Efficiently in Object-Relational Databases. In: Kaufmann, Morgan (Hrsg.): *Twenty-sixth International Conference on Very Large Databases (VLDB 2000).* Cairo, Egypt : VLDB, 2000, S. 407–418. – Online verfügbar unter: http://dme.rwth-aachen.de/de/system/files/file_upload/publications/VLDB-01.pdf Abruf: 2015-02-05. – ISBN 1-55860-715-3

[Kriegel u. a. 2001] Kriegel, Hans-Peter ; Pötke, Marco ; Seidl, Thomas: Object-Relational Indexing for General Interval Relationships. In: Jensen, ChristianS. (Hrsg.) ; Schneider, Markus (Hrsg.) ; Seeger, Bernhard (Hrsg.) ; Tsotras, Vassilis J. (Hrsg.): *Advances in Spatial and Temporal Databases* Bd. 2121. Springer Berlin Heidelberg, 2001, S. 522–542. – Online verfügbar unter: http://www.dbs.ifi.lmu.de/Publikationen/Papers/SSTD01-Allen.pdf Abruf: 2015-11-06. – ISBN 978-3-540-42301-0

[Kuhrt 2013] Kuhrt, Sebastian: *Ein Time Triggered Ethernet basiertes Rückfahrkamerasystem vom Entwurf bis zur Integration in einen Fahrzeugdemonstrator*, HAW Hamburg, Bachelorarbeit, 5 2013

[Oracle 1999] Herbert, Jeff ; Murray, Chuck: *Oracle Spatial User's Guide and Reference.* Oracle Corporation, 12 1999. – Online verfügbar unter: https://docs.oracle.com/cd/A81042_01/DOC/inter.816/a77132.pdf Abruf: 2015-06-27

[Oracle 2008] Fallon, Mark: *How to write SQL injection proof PL/SQL.* Oracle Corporation, 12 2008. – Online verfügbar unter: http://www.oracle.com/technetwork/database/features/plsql/overview/how-to-write-injection-proof-plsql-1-129572.pdf Abruf: 2015-06-21

[Oracle 2014] Moore, Sheila ; Belden, Eric: *Oracle® Database PL/SQL Language Reference.* Oracle Corporation, 2014. – Online verfügbar unter: https://docs.oracle.com/cd/E11882_01/appdev.112/e25519.pdf Abruf: 2015-08-11

[Oracle 2015] Ashdown, Lance ; Kyte, Tom: *Oracle® Database Concepts 11g Release 2 (11.2).* Oracle Corporation, 5 2015. – Online verfügbar unter: http://docs.oracle.com/cd/E11882_01/server.112/e40540.pdf Abruf: 2015-07-14

[Saake u. a. 2013] Saake, Gunter ; Sattler, Kai-Uwe ; Heuer, Andreas: *Datenbanken: Konzepte und Sprachen.* 5. edition. Heidelberg : MITP-Verlags GmbH & Co. KG, 2013. – ISBN 978-3-8266-9453-0

[Vossen 2008] Vossen, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme* -. 5. überarbeitete und erweiterte Auflage. München : Oldenbourg Verlag, 2008. – ISBN 978-3-486-27574-2

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. November 2015 Philip Hundt