Entwicklung eines grafischen Query Designers für relationale Datenbanken

Thomas Peintner

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2013

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 28. Juni 2013

Thomas Peintner

Inhaltsverzeichnis

Kurzfassung

Abfragen auf Datenbestände erfolgen zumeist über eigens dafür vorgesehene Abfragesprachen. Für technisch unversierte Benutzer kann dies jedoch ein grundlegendes Problem darstellen, da sie in der Regel nur geringe bzw. keine Fachkenntnisse im Umgang mit Abfragesprachen aufweisen können.

Im Zuge dieser Arbeit wurde ein Konzept für einen grafischen Query Designer entwickelt, um technisch unversierten Benutzern den Zugriff auf in relationalen Datenbanken gespeicherten Daten zu ermöglichen. Dabei kann die Abfrage nach dem Baukastenprinzip flexibel durch den Benutzer formuliert werden. Die Anwendung gibt ausreichendes Feedback in natürlicher Sprache und unterstützt so den Benutzer bei der Abfragenerstellung.

Der nach dem entwickelten Konzept umgesetzte Prototyp dient als Basis für eine Machbarkeitsstudie und ermöglicht es, das Erreichen der gesetzten Ziele zu überprüfen.

Abstract

Accessing data in databases typically requires the use of specialized query languages. Technically unversed users, however, often lack the professional skills needed to utilize such programming languages.

To grant technically unversed users access to data stored in databases, the concept of a graphical query designer was developed. By making use of the modular design principle, the user should be enabled to flexibly draft database queries. In addition, the proposed concept relies on feedback based on natural language to support the user when drafting queries.

To verify that the concept has the potential for real-world application, a prototype was developed and used to measure the success of the developed concept.

Kapitel 1

Einleitung

Das Abfragen von Datenbeständen aus relationalen Datenbanken erfolgt zumeist über eigens dafür vorgesehene Abfragesprachen, welche eine spezifizierte Syntax und Semantik verwenden. Da bei technisch unversierten Personen jedoch in der Regel nur sehr geringe bzw. keine Fachkenntnisse im Umgang mit Abfragesprachen vorhanden sind, stellt dies beim Zugriff auf Datenbestände für diese Personen ein Problem dar.

Deshalb werden Anwendungen für spezielle Abfragefälle implementiert, welche die Datenbeschaffung, ohne technisches Hintergrundwissen zum Datenbestand oder zur verwendeten Abfragesprache, ermöglichen. Diese Anwendungen greifen intern über die jeweilige Abfragesprache auf den Datenbestand zu, bieten jedoch sehr wenig Flexibilität, da die Abfragen, in der jeweiligen Syntax, statisch im Anwendungscode implementiert sind.

Das Ziel dieser Arbeit besteht darin, die Abfrage auf Datenbestände in relationalen Datenbanken für technisch unversierte Personen wesentlich zu erleichtern. Dabei wird ein Konzept für eine Schnittstelle vorgestellt, das die beliebige Formulierung von Datenabfragen ermöglicht und Flexibilität bei der Verwendung von unterschiedlichen Datenbeständen gewährleistet. Um diese Zielsetzung zu erreichen, wurden bereits bestehende User Interfaces für die Datenabfrage im Vorfeld analysiert und die Erkenntnisse in die Entwicklung des Konzepts miteinbezogen. Der im Rahmen der Arbeit umgesetzte Prototyp dient als Machbarkeitsstudie und als Grundlage für eine Evaluierung, um das erstellte Konzept kritisch zu hinterfragen und Verbesserungsmöglichkeiten zu erlangen.

Diese Arbeit ist in sieben Kapitel untergliedert. Im anschließenden Kapitel 2 werden zuerst die Grundlagen beschrieben, die im Kontext mit dieser Arbeit stehen und den Leser langsam in die Thematik einführen sollen. Darin werden zuerst Möglichkeiten zur persistenten Speicherung von Daten vorgestellt. Weiters widmet sich dieses Kapitel einer Auswahl von Abfragesprachen, die den Zugriff auf die gespeicherten Daten ermöglichen.

Bereits bestehende Benutzerschnittstellen, die Abfragen auf Datenbe-

stände ermöglichen, finden sich in Kapitel 3 wieder. Diese Anwendungen werden im Hinblick auf die Bedienung durch einen technisch unversierten Benutzer getestet. Die daraus resultierenden Erkenntnisse werden am Ende des Kapitels aufgezeigt.

Kapitel 4 beschäftigt sich mit dem eigenen Lösungsansatz und stellt das entwickelte Konzept für die Umsetzung einer Schnittstelle zur Datenabfrage durch technisch unversierte Benutzer vor.

In Kapitel 5 wird auf den Aufbau und die Umsetzung des Prototyps nach dem zuvor erstellten Konzept eingegangen. Weiters werden Kernpunkte bei der Implementierung präsentiert und alternative Lösungswege diskutiert.

Der entwickelte Prototyp stellt die Grundlage für die in Kapitel 6 beschriebene Evaluierung dar. Neben dem Ablauf der Evaluierung werden die aufgetretenen Probleme beim Benutzen des Prototyps durch die Probanden präsentiert und mögliche Lösungsvorschläge aufgezeigt.

Die Zusammenfassung der Erkenntnisse dieser Arbeit befindet sich in Kapitel 7. Weiters wird dort ein kurzer Ausblick auf eine eventuelle Weiterentwicklung des Prototyps gegeben.

Der Anhang dieser Arbeit enthält die verwendeten Heuristiken, die Evaluierungstabelle für die Probanden sowie die Aufgaben, die bei der Think Aloud Methode gestellt wurden.

Kapitel 2

Grundlagen – Datenhaltung im Web

Dieser einführende Abschnitt behandelt unterschiedliche Möglichkeiten zur persistenten Speicherung von Daten. Bei der Verarbeitung der Daten durch Anwendungen (sowohl desktop- als auch webbasierte) macht es prinzipiell keinen Unterschied, ob der Datenbestand lokal bei der Anwendung oder auf einem Server im Web vorliegt, da sich lediglich der Datenpfad ändert.

Relationale Datenbanken stellen derzeit die populärste Datenhaltungsmethode dar. Bei der Umsetzung eines Konzepts zur benutzerfreundlichen Abfrage von Datenbeständen können jedoch auch andere Datenhaltungsmethoden zum Einsatz kommen. Bevor in späteren Kapiteln die Umsetzung eines Konzepts zur benutzerfreundlichen Abfrage relationaler Datenbanken erfolgt, werden diese Methoden und deren Abfragesprachen näher betrachtet.

2.1 Datenbanksysteme

Laut der Definition in [17, S. 8] besteht ein Datenbanksystem, kurz DBS, aus der Datenbank und dem Datenbankmanagementsystem. Während die Datenbank einen strukturierten Datenbestand repräsentiert, beinhaltet das Datenbankmanagementsystem (DBMS) eine Ansammlung an Softwaremodulen, die den Zugriff auf den Datenbestand sowie die Manipulation und Kontrolle der Daten ermöglichen.

Für Datenbanksysteme haben sich im Laufe der Jahre einige Anforderungen herauskristallisiert, welche in [17, S. 9] angeführt sind:

- DBMSe verwalten Daten persistent und ermöglichen den Datenzugriff und die Datenmanipulation durch Anwendungen.
- Große Datenmengen müssen durch das DBS effizient verwaltet werden können.
- Datenbankmodelle werden durch DBMSe definiert, um alle Daten ein-

heitlich beschreiben zu können.
- Verschiedene Sprachen wie Datendefinitionssprachen, Abfragesprachen sowie Datenmanipulationssprachen müssen zur Verfügung gestellt werden.
- DBMSe fassen logische, zusammenhängende Operationen zu Transaktionen zusammen und können auch parallel ausgeführt werden (Mehrbenutzerbetrieb). Die Verwaltung der Transaktionen erfolgt ebenfalls durch das System.
- Das DBS gewährleistet die Datenintegrität (Konsistenz) und und fördert die Datensicherheit.

Jedoch haben sich die Anforderungen an moderne Datenbanksysteme durch das Web 2.0-Zeitalter und die dadurch immer größer werdende zu verarbeitende Informationsflut wesentlich verändert. So wird in nicht relationalen Datenbankkonzepten mittlerweile auf verteilte und horizontale Skalierbarkeit (Erklärung in Abschnitt 2.1.2) Wert gelegt, während die strikte Transaktionskontrolle und die konsistente Datenhaltung vernachlässigt wird [2, S. 3].

Die Autoren in [2, S. 6] unterteilen Datenbanksysteme in zwei Kategorien, die relationalen Datenbanksysteme sowie die NoSQL-Systeme, wobei nicht in allen Literaturwerken die Grenzen gleich gezogen werden.

2.1.1 Relationale Datenbanksysteme

Relationale Datenbanksysteme, welche Anfang der siebziger Jahre entwickelt wurden, sind noch immer die geläufigsten Datenbankansätze.

Entity-Relationship-Modell

Das ER-Modell, ein Modell für die konzeptionelle Phase des Datenbankentwurfs, basiert auf den drei Basisbegriffen Entity, Relationship und Attribut. Entities bilden Objekte der realen Welt, wie z. B. ein Produkt oder einen Studenten, ab.

Die Beziehungen zwischen unterschiedlichen Entities werden durch Relationships repräsentiert wie z. B. ein Weingut produziert einen Wein oder ein Dozent hält eine Vorlesung. Um die Eigenschaften von Entities oder Relationen näher zu beschreiben, werden Attribute, wie z. B. Farbe und Jahrgang eines Weines oder der Titel einer Vorlesung, eingesetzt. Mehr über das Entity-Relationship-Modell findet sich unter [17, Kap. 3].

Relationenmodell

Das Relationenmodell ist im Gegensatz zum ER-Modell ein Realisierungsmodell zur Implementierung des Entwurfs und beinhaltet Konzepte zur Strukturierung der Daten. Die Objekte der abzubildenden Anwendungswelt be-

stehen aus einer Menge von Attributen (Spalten) und werden durch Relationenschemata beschrieben. Ein Datenbankschema besteht aus einer Menge von Relationenschemata und die zu einem Relationenschema passenden Daten werden durch eine Relation (Tabelle) dargestellt. Die verschiedenen Zeilen einer Relation (Tabelle) werden als sogenannte Tupel bezeichnet. Damit diese innerhalb einer Tabelle identifizierbar sind, muss eine minimale Menge an Attributen den Primärschlüssel bilden. Die Werte der Attribute müssen eindeutig sein. Als Fremdschlüssel wird die Attributmenge bezeichnet, die in einer anderen Relation (Tabelle) ein Primärschlüssel ist. Damit die Fremdschlüsselbedingung (Referenzielle Integrität) gegeben ist, müssen alle Attributwerte des Fremdschlüssels in der anderen Relation (Tabelle) als Werte des Primärschlüssels existieren. Weitere Informationen zum Relationenmodell unter [17, Kap. 4].

Datenintegrität und Transaktionen

Relationale Datenbanksysteme haben die Anforderung, die Datenbank konsistent zu halten. Transaktionen, eine Folge von Datenbankoperationen, müssen daher die ACID-Eigenschaften (Erklärung unter [17, Abschn. 12.1.1]) strikt erfüllen, damit sich die Datenbank vor und nach der Transaktionsausführung in einem zulässigen Zustand befindet. Mehr dazu in [17, Kap. 12].

2.1.2 NoSQL-Datenbanksysteme

Die Geschichte von NoSQL begann zwar bereits Anfang der 80er Jahre mit der Entwicklung einer Key/Hash-Datenbank, jedoch kam der Durchbruch erst im Jahre 2000 mit Web 2.0 und dem Versuch, große Datenmengen zu verarbeiten. Seit 2009 bildete sich durch das NoSQL-Archiv[1] eine Gruppe dieser NoSQL-Systeme als Kontrast zu den relationalen Datenbanken [2, S. 1].

Definition und Merkmale

NoSQL-Datenbanksysteme charakterisieren sich selbst durch das Berücksichtigen einiger nachfolgender Punkte [2, S. 2–5]:

- **Datenmodell nicht relational:** Die Auswahl des perfekten Datenmodells hängt zumeist von der zu lösenden Problemstellung ab. So wäre zum Beispiel das Traversieren[2] eines Graphen in einem relationalen Datenmodell mit Performanceproblemen verbunden, weshalb ein Graphmodell vorzuziehen wäre. Erklärungen zum Graphenmodell sind unter [2, S. 209–230] zu finden.

[1] NoSQL-Archiv: http://www.nosql-database.org/

[2] Ist das Durchlaufen von Knoten innerhalb eines Graphen und wird u. a. bei der Suche eines bestimmten Knotens angewendet [2, S. 219].

- **verteilte und horizontale Skalierbarkeit:** NoSQL-Systeme können Tera- oder sogar Petabyte von Daten mit Standard-Hardware verwalten, da diese Systeme von Anfang an auf Skalierbarkeit ausgerichtet sind. Bei der horizontalen Skalierbarkeit werden Knoten (Rechner) ins System zur Lastverteilung hinzugefügt, während die vertikale Skalierung die Ressourcen eines Knotens (Rechner) erweitert und dadurch eine Leistungssteigerung erzielt.
- **Open Source:** NoSQL-Systeme sollten in der Regel Open Source sein, wobei dieser Punkt nicht als Ausschlusskriterium sondern viel mehr als Wunschkriterium für NoSQL-Systeme anzusehen ist.
- **schemafrei oder weniger Schemarestriktionen:** Web 2.0-Projekte erfordern Flexibilität, da sich die Struktur der Daten (Schema) häufig verändern kann. Die Idee der NoSQL-Systeme ist Teile der Verantwortung im Umgang mit dem Schema auf die Anwendung abzugeben, da dort eine Anpassung des Schemas möglicherweise leichter durchgeführt werden kann. Während bei Schemaerweiterungen in relationalen Datenbanksystemen stundenlanges konvertieren und sperren der Tabellen notwendig ist, wird bei NoSQL-Systemen der Ansatz einer Datenversionierung verfolgt. Dadurch kann die Anwendung nach einer Schemaerweiterung von Anfang an die neuen Daten speichern, während ein Hintergrundprozess die alten Daten konvertiert. Dabei kann es zu leichten, seltenen Inkonsistenz-Zeitfenster kommen, jedoch hängt dies vom Anwendungsbereich ab, ob diese Nebeneffekte tolerierbar sind.
- **einfache Datenreplikation:** Die verteilte Architektur der meisten NoSQL-Systeme ermöglicht die einfache Replikation von Daten. Bei der Replikation werden diese mehrfach an verschiedenen Orten gespeichert und die einzelnen Replikate für unterschiedliche Einsatzgebiete verwendbar.
- **einfache API:** NoSQL-Datenbanksysteme versuchen neue Wege zu gehen und den SQL-Standard zu verbessern. Es gibt bereits einige NoSQL-APIs die tatsächlich übersichtlicher als SQL sind, jedoch werden meistens weniger mächtige Funktionen angeboten, weshalb komplexere Abfragen durch andere Technologien und Sprachen gelöst werden müssen.
- **Konsistenzmodell des Systems:** Während relationale Datenbanksystem strikte Konsistenz- und Transaktionsanforderungen (ACID) zur Verfügung stellen, werden diese von den modernen NoSQL-Systemen nicht mehr zwingend benötigt. Zum Beispiel darf es bei Social Web-Portalen, die im Normalfall keine kritischen Daten halten, durchaus für ein kurzes Zeitfenster zu Inkonsistenzen kommen. Diese Systeme werden als *Eventually Consistent* bezeichnet.

Kategorisierung der Systeme

In [2, S. 5–10] werden alle Datenbanksysteme aus dem NoSQL-Archiv (über 150 Systeme, Stand 04/2013) in NoSQL-Kernsysteme und nachgelagerte NoSQL-Systeme unterteilt. Die Kernsysteme können in vier weitere Kategorien untergliedert werden:

- **Key/Value-Systeme:** Diese Systeme besitzen ein einfaches Schema, welches sich aus einem Schlüssel und einem Wert zusammensetzt. Die Werte können neben normalen Zeichenketten auch Hashes, Sets oder Listen enthalten, wobei der Wert (Value) vom Datenbanksystem nicht interpretiert wird. Zusätzlich ist es noch möglich die Schlüssel in Namensräume und Datenbanken zu unterteilen. Dieses einfache Datenmodell ermöglicht zwar schnelle und effiziente Datenverwaltung, jedoch sind die Abfragemöglichkeiten dieser Systeme nicht zufriedenstellend. Bekannte Vertreter von Key/Value-Systemen sind *Dynamo*[3] von *Amazon* sowie *Redis*[4], *Voldemort*[5] und *Scalaris*[6]. Zusätzliche Erläuterungen unter [2, Kap. 5].
- **Column-Family-Systeme:** Column-Family-Systeme basieren auf Tabellen in denen jede Zeile durch einen Schlüssel eindeutig identifizierbar ist. Einer Zeile können beliebig viele Key/Value-Paare (Spalten) zugeordnet werden, wobei innerhalb der Tabelle (Column-Family) die Spaltenanzahl zwischen den einzelnen Zeilen variieren darf. Die typischen Vertreter sind *HBase*[7], *Cassandra*[8] und *Hypertable*[9]. Mehr dazu findet sich in [2, Kap. 3].
- **Document Stores:** Das Datenbanksystem legt eine Ansammlung von strukturierten Dokumenten wie z. B. JSON, YAML oder RDF (mehr dazu in Abschnitt 2.2.2) mit einem eindeutigen Schlüssel ab. Dabei wird nur festgelegt auf welches Datenformat der Schlüssel verweist. *CouchDB*[10], *MongoDB*[11] sowie *Riak*[12] sind bekannte Document Stores. Weitere Informationen über Document Stores in [2, Kap. 4].
- **Graphdatenbanken:** Diese NoSQL-Datenbanken beschäftigen sich mit der Verwaltung von Graph- oder Baumstrukturen, in denen die Knoten in einer Beziehung zueinander stehen und ermöglichen ein einfaches und effizientes Durchlaufen von Graphen wie z. B. die Beziehungen unter Freunden im Social Web.

[3] Dynamo: http://aws.amazon.com/de/dynamodb/
[4] Redis: http://redis.io/
[5] Voldemort: http://www.project-voldemort.com/voldemort/
[6] Scalaris: https://code.google.com/p/scalaris/
[7] HBase: http://hbase.apache.org/
[8] Cassandra: http://cassandra.apache.org/
[9] Hypertable: http://hypertable.org/
[10] CouchDB: http://couchdb.apache.org/
[11] MongoDB: http://www.mongodb.org/
[12] Riak: http://basho.com/riak/

Ergänzungen zu Graphdatenbanken sind in [2, Kap. 6] zu finden.

Können Datenbanksysteme aus dem NoSQL-Archiv nicht eindeutig in die Kategorien der NoSQL-Kernsysteme eingeordnet werden, so gehören sie der Familie der nachgelagerten NoSQL-Systeme an. Beispiele dafür sind Objektdatenbanken, XML-Datenbanken, Grid-Datenbanken sowie viele weitere zum Teil exotische und nicht relationale Datenbanksysteme.

2.2 Strukturierte Dokumente

Informationen können in verschiedenster Art und Weise strukturiert und beschrieben werden. Während HTML die Darstellungsform von Informationen beschreibt und XML Daten auf syntaktischer Ebene strukturiert, können durch RDF Informationen als Wissen repräsentiert werden [1, S. 112].

Sowohl XML- als auch RDF-Daten können neben der Verarbeitung in Dokumenten auch in datenbankähnlichen Speichersystemen verwaltet werden. Native XML-Datenbanksysteme und Triple Stores kommen dabei zum Einsatz [10, S. 664–667], [17, S. 576].

2.2.1 XML

XML (Extensible Markup Language) wurde vom *World Wide Web Consortium*[13] *(W3C)* als Weiterentwicklung der Auszeichnungssprache SGML[14] konzipiert und im Jahre 1998 veröffentlicht. XML ermöglicht als Metasprache die Definition von anwendungsspezifischen Sprachen wie z. B. XHTML und wird u. a. für den Datenaustausch zwischen verteilten Systemen eingesetzt [17, S. 572].

Klassifizierung

Dokumente des XML-Standards können auf Grund ihres Strukturierungsgrades folgendermaßen klassifiziert werden [17, S. 572], [21]:

- **dokumentzentriert:** Diese Art von Dokument ist nur sehr schwach strukturiert und nur mit wenigen XML-Elementen zur semantischen Auszeichnung versehen. Da das Dokument einem Textdokument ähnelt, ist es zwar für den Menschen durchaus lesbar, jedoch für die maschinelle Verarbeitung nahezu unbrauchbar. Beispielsweise würde beim Auszeichnen von Überschriften durch XML-Elemente innerhalb eines Fließtextes ein dokumentzentrierter Ansatz verfolgt werden.
- **datenzentriert:** Datenzentrierte Dokumente weisen ein Schema auf, welches die Struktur (Entitäten, Attribute, Beziehungen) eines Datenmodells vorgibt. Auf Grund der starken Strukturierung werden diese

[13] World Wide Web Consortium (W3C): http://www.w3.org/
[14] SGML: http://www.w3.org/MarkUp/SGML/

Dokumente hauptsächlich von Maschinen verarbeitet und sind für den Menschen schwer zu lesen. Ein datenzentriertes Dokument wäre zum Beispiel eine Konfigurationsdatei, welche verschiedene Werte beinhaltet, die durch XML-Elemente ausgezeichnet sind.

Stellen XML-Dokumente eine Mischform der beiden oberhalb genannten Dokumentklassen dar, so sind diese als semistrukturierte Dokumente zu bezeichnet.

Aufbau

XML-Dokumente können optional mit der XML-Deklaration zur Angabe der Version und der Zeichensatzkodierung sowie einer Schema-Deklaration beginnen. Anschließend folgen die XML-Daten, wobei das erste Element im Dokument immer das Wurzelelement darstellt. Elemente können ineinander verschachtelt werden, wodurch das Dokument eine hierarchische Baumstruktur erhält, die durch einen Graphen dargestellt werden kann. Attribute, bestehend aus Name und Wert, können als Eigenschaften den Elementen beliebig hinzugefügt werden.

Zusätzlich stellt der XML-Standard noch Kommentare für Entwickler, Verarbeitungsanweisungen für Anwendungen sowie Namensräume, um mehrdeutige Element- und Attributnamen zu vermeiden, zur Verfügung [20, S. 47–72].

Schema

Die Struktur eines XML-Dokumentes kann durch ein Schema definiert werden. Dieses kann direkt im Daten-Dokument enthalten sein oder von anderen Quellen bezogen werden. Dokumente, die mit keinem Schema verknüpft sind, jedoch die syntaktischen Anforderungen des XML-Standards erfüllen, werden als *well-formed* bezeichnet. Ist die Struktur eines Dokumentes durch ein Schema definiert und werden die schematischen und syntaktischen Vorgaben eingehalten, so wird das XML-Dokument *valid* genannt.

Schemata haben genau dann eine große Bedeutung, wenn XML als Datenaustauschformat verwendet wird. Durch die Definition von standardisierten Schemata für unterschiedliche Anwendungen, können Daten auf Grund der vorgegebenen Struktur modelliert und ohne Transformation ausgetauscht werden [10, S. 588–590]. Zwei bekannte Schemasprachen sind *Dokumenttypedefinition (DTD)* und *XML Schema*, wobei die modernere Sprache *XML Schema* bereits erlaubt, dass Inhalte von Elementen und Attributen auf bestimmte Datentypen beschränkt werden [21].

Verarbeitung

Grundsätzlich besteht die Möglichkeit XML auf auf zwei verschiedene Arten zu verarbeitet. Bei der Verwendung als Datenaustauschformat erfolgt die

Analyse des Dokuments durch einen geeigneten Parser. Die dadurch erhaltenen Daten können anschließend per XML-Prozessor beispielsweise in ein anderes Format transformiert oder für das Ausführen von Anfragen verwendet werden. Damit XML-Prozessoren die Dokumente verarbeiten können, müssen sie zumindest *well-formed* sein. Die datenbankbasierte Verarbeitung speichert die XML-Dokumente in einem Datenbanksystem und ermöglicht so das Ausführen von Anfragen auf die Daten [17, S. 575–576].

2.2.2 RDF

RDF (Resource Description Framework) dient der Wissensrepräsentation und dem Informationsaustausch im Web. Im Gegensatz zu XML bietet RDF die Möglichkeit Informationen semantisch zu beschreiben und in maschinenlesbare Form zu bringen.

Aufbau

RDF erlaubt es Entitäten (Ressourcen) über Uniform Resource Identifier (URIs) eindeutig zu identifizieren. Eine Aussage bildet ein Tripel aus den Komponenten Subjekt, Prädikat und Objekt, welche alle durch URIs identifiziert und dargestellt werden.

Weiters können Objekte auch konkrete Daten wie Zeichenketten oder Nummern (Literale) repräsentieren. Die textuelle Darstellung von Tripeln kann in verschiedenen Notationen erfolgen wie z. B. RDF/XML oder der leichter lesbaren Turtle-Notation. Die Beispielaussage „Thomas wohnt in Tirol“ sieht in Turtle-Notation wie folgt aus:

```
<http://thesis.com/people#Thomas>         /* Subjekt */
<http://thesis.com/properties#wohntIn>    /* Prädikat */
<http://thesis.com/states#Tirol>.         /* Objekt */
```

Die komplexere Variante in RDF/XML-Syntax sieht so aus:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:prop ="http://thesis.com/properties#">
  <rdf:Description rdf:about="http://thesis.com/people#Thomas">
    <prop:wohntIn rdf:resource="http://thesis.com/states#Tirol"/>
  </rdf:Description>
</rdf:RDF>
```

Durch das Bilden von Tripeln entstehen zum Teil auch sehr komplexe Datenstrukturen, die als Graph visualisiert werden können. Dabei werden die Subjekte und Objekte als Knoten und die Prädikate als gerichtete Kanten zwischen Subjekten und Objekten dargestellt. Die oberhalb erwähnte Beispielaussage wird in Abbildung 2.1 als Graph visualisiert.

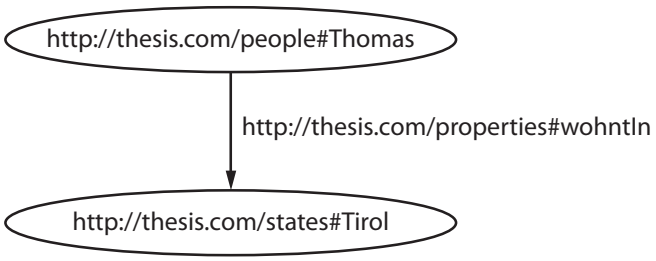

Abbildung 2.1: Beispielaussage „Thomas wohnt in Tirol“ als Graph.

Schema

Das RDF-Schema (RDFS) erlaubt eine externe Spezifikation der Semantik und verleiht so RDF mehr Ausdruckskraft beim Definieren von Wissen. Der Einsatz von RDFS ermöglicht es u. a. Ressourcen als Klassen auszuzeichnen oder Properties von bestimmten Ressourcenklassen einzuschränken.

Der Vorteil liegt darin, dass beim Informationsaustausch jede Anwendung die, im Schema definierte, spezielle Semantik richtig interpretieren kann [1, S. 109–125], [10, S. 659–661].

2.3 Abfragesprachen

Mit Hilfe von Abfragesprachen können Queries (Abfragen) formuliert werden, die eine Teilmenge des gesamten Datenbestandes als Ergebnis zurückliefern. Das Anwenden von Abfragesprachen erfolgt deklarativ. Das heißt, der Benutzer gibt nur die für ihn interessanten Daten im Query an und muss sich dabei nicht um die interne Auswertung dieser Daten kümmern [10, S. 111].

2.3.1 SQL

Grundsätzlich ist SQL (Structured Query Language) weit mehr als eine reine Abfragesprache und wird daher in der Kategorie der Datenbanksprachen für relationale Systeme eingeordnet. Neben der Erstellung von Queries zur Datenabfrage, bietet der SQL-Sprachumfang u. a. Möglichkeiten zur Datendefinition (Data Definition Language) und zur Manipulation am Datenbestand (Data Manipulation Language) an. Weiterführende Informationen zu SQL unter [17, Kap. 7].

Beispielabfrage

Ausgangspunkt ist eine Tabelle *Personen* (abgebildet in Tabelle 2.1) mit den vier Spalten *Id*, *Vorname*, *Nachname* und *Alter*. Soll eine SQL-Abfrage nur Personen als Ergebnis zurückliefern, die genau 20 Jahre alt sind, so muss das Query wie folgt formuliert sein:

Tabelle 2.1: Beispieltabelle Personen für SQL-Abfrage.

Id	*Vorname*	*Nachname*	*Alter*
1	Julia	Maier	20
2	Max	Schmidt	24
3	Thomas	Müller	20
4	Sandra	Salcher	26
5	Christian	Berger	21

```
SELECT Id, Vorname, Nachname, Alter FROM Personen WHERE Alter = 20;
```

2.3.2 XQuery

Aufbauend auf XPath[15] wurde die Abfragesprache XQuery für die Datensuche in XML-Dokumenten entwickelt. Die aus XPath bekannten Pfadausdrücke zum Adressieren von Dokumentteilen und die sogenannten FLWOR-Ausdrücke bilden das zentrale Konzept von XQuery.

FLWOR ist eine Kurzschreibweise für die Schlüsselwörter *for*, *let*, *where*, *order by* und *return*, wobei *where* und *order by* optional verwendbar sind. Die XQuery-Ausdrücke lassen sich beliebig kombinieren und verschachteln. Mehr dazu in [10, S. 597–613], [17, Kap. 18].

Beispielabfrage

Als Grundlage für diese Beispielabfrage dient ein XML-Dokument (siehe Programm 2.1), welches fünf Personen mit Id, Vorname, Nachname und Alter beschreibt. Der XQuery-Ausdruck, um als Ergebnis alle Personen im Alter von 20 Jahren zu erhalten, sieht folgendermaßen aus:

```
<Ergebnis>
    {for $p in doc("Personen.xml")//Person
     where $p/@alter = 20
     return $p}
</Ergebnis>
```

2.3.3 SPARQL

Die Sprache SPARQL ermöglicht Abfragen auf RDF-Wissensbasen auszuführen. Die Syntax ist an SQL angelehnt und verwendet ebenfalls Sprachbausteine wie *SELECT* und *WHERE*. Der *WHERE*-Teil der Abfrage besteht hauptsächlich aus Tripeln in Turtle-Syntax in denen URIs, Literale oder Variablen vorkommen können. Zusätzlich unterstützt SPARQL noch Filter, die

[15] XPath: http://www.w3.org/TR/xpath/

Programm 2.1: Beispieldokument *Personen.xml* für XQuery-Abfrage.

```
<?xml version="1.0"?>
<Personen>
  <Person id="1" vorname="Julia" nachname="Maier" alter="20" />
  <Person id="2" vorname="Max" nachname="Schmidt" alter="24" />
  <Person id="3" vorname="Thomas" nachname="Müller" alter="20" />
  <Person id="4" vorname="Sandra" nachname="Salcher" alter="26" />
  <Person id="5" vorname="Christian" nachname="Berger" alter="21" />
</Personen>
```

Programm 2.2: RDF-Wissensbasis in Turtle-Syntax für SPARQL-Abfrage.

```
1  @prefix ex: <http://thesis.com/example#>.
2  ex:person1 ex:vorname "Julia".
3  ex:person1 ex:nachname "Maier".
4  ex:person1 ex:alter "20".
5  ex:person2 ex:vorname "Max".
6  ex:person2 ex:nachname "Schmidt".
7  ex:person2 ex:alter "24".
8  ex:person3 ex:vorname "Thomas".
9  ex:person3 ex:nachname "Müller".
10 ex:person3 ex:alter "20".
11 ex:person4 ex:vorname "Sandra".
12 ex:person4 ex:nachname "Salcher".
13 ex:person4 ex:alter "26".
14 ex:person5 ex:vorname "Christian".
15 ex:person5 ex:nachname "Berger".
16 ex:person5 ex:alter "21".
```

weitere Einschränkungen der Abfragen ermöglichen. Zusätzliche Informationen zu SPARQL unter [1, S. 162–169].

Beispielabfrage

Die RDF-Wissensbasis in Turtle-Syntax (in Programm 2.2) besteht aus Tripeln, die Eigenschaften von Personen definieren. Die Angabe eines Präfix (in Zeile 1) ermöglicht eine kompaktere Schreibweise.

Wie schon in den beiden Beispielabfragen zuvor, sollen durch ein Query alle Personen im Alter von 20 Jahren abgefragt werden. In der Sprache SPARQL erhält man das gewünschte Ergebnis durch folgende Formulierung:

```
PREFIX ex: <http://thesis.com/example#>
SELECT ?vorname ?nachname ?alter
WHERE {   ?person ex:vorname ?vorname.
          ?person ex:nachname ?nachname.
          ?person ex:alter ?alter.
          FILTER (?alter = "20")    }
```

Kapitel 3

State of the Art

Die in Kapitel 2 näher betrachteten Abfragesprachen, um Datenbestände zu filtern und auszuwerten, können in der Regel nur von Benutzern verwendet werden, die das notwendige technische Hintergrundwissen über die Datenstrukturen und die anzuwendenden Sprachen besitzen. Um auch technisch unversierten Benutzern den Zugriff auf Datenbestände zu ermöglichen, gilt es benutzerfreundliche Interfaces zu entwickeln, welche eine Eingabe der gewünschten Abfrage (Query) in natürlicher Form zulassen und diese in die Syntax der jeweiligen Abfragesprache übersetzen.

Da relationale Datenbanken die populärste Datenhaltungsmethode darstellen, wird diese Methode bei der Umsetzung eines Konzepts zur benutzerfreundlichen Abfrage von Datenbeständen in dieser Arbeit zum Einsatz kommen.

Im Folgenden werden ausgewählte Benutzerschnittstellen zur Datenabfrage aus relationalen Datenbanken vorgestellt und anschließend eventuelle Erkenntnisse im Hinblick auf die Bedienung durch einen technisch unversierten Benutzer aufgezeigt.

3.1 Benutzerschnittstellen für die Datenabfrage

Bei der folgenden Untersuchung verschiedener User Interfaces steht nicht die verwendete Technologie zum Speichern der Daten im Vordergrund, sondern nur die Auswertung und Filterung der Datenbestände, die ein datenbanktechnisch unversierter Anwender durchführen könnte.

3.1.1 Microsoft Excel

Das Grundkonzept von *Excel*[1], dem Tabellenkalkulationsprogramm von *Microsoft*, ist Daten in tabellarischer Form durch Berechnungen und Formeln

[1] Microsoft Excel: aktuelle Version 2013 (Stand 04/2013), http://office.microsoft.com/de-at/excel/

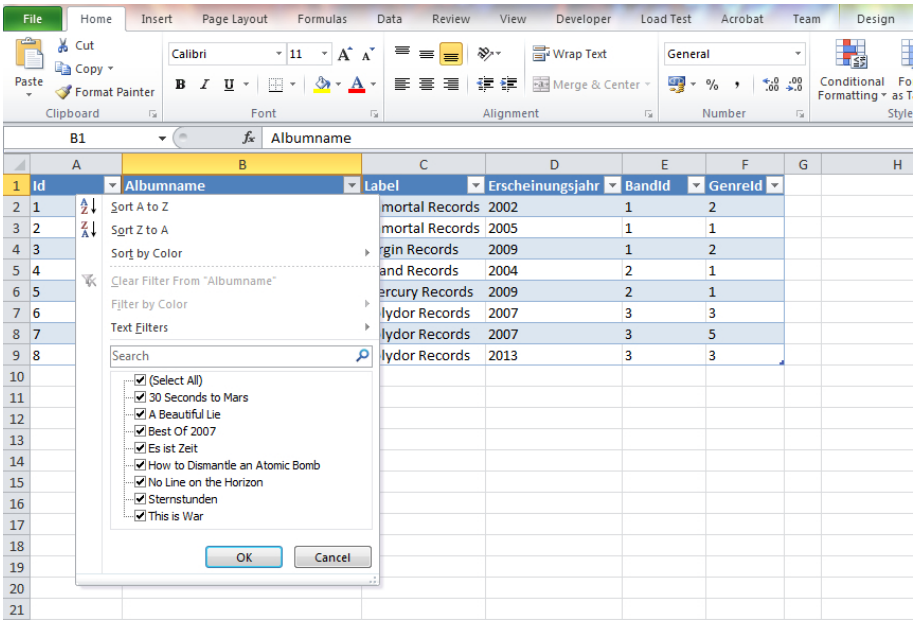

Abbildung 3.1: Microsoft Excel 2010 – Funktionen auf Spalten der Tabelle ausführen.

auszuwerten und bei Bedarf die Ergebnisse auch grafisch in Diagrammen darzustellen [22, 12].

Importieren von Daten

Die zu verarbeitenden Daten können einerseits direkt im Dokument enthalten sein oder von externen Datenquellen wie z. B. XML-Dokumenten, CSV-Dokumenten oder relationalen Datenbanken bezogen werden. In manchen Fällen ist die Installation eines Datenbanktreibers sowie die Einrichtung einer Datenverbindung zur externen Datenquelle erforderlich, weshalb diese Schritte zumeist durch einen Systemadministrator erledigt werden sollten. *Excel* interpretiert die Daten und stellt diese beim Import in tabellarischer Form dar.

Funktionen auf Spalten

Auf die einzelnen Spalten der *Excel*-Tabelle können über das Dropdown-Menü (ersichtlich in Abbildung 3.1) verschiedene Filterfunktionen angewandt werden. Filterfunktionen stellen Bedingungen bzw. Kriterien dar, die ein Datensatz erfüllen muss, um im Ergebnis der Abfrage enthalten zu sein. Dadurch lässt sich die Anzahl der angezeigten Datensätze in der Tabelle einschränken. Pro Spalte ist die gleichzeitige Anwendung von bis zu zwei Filterfunktionen möglich und die Verknüpfung der beiden Bedingungen kann

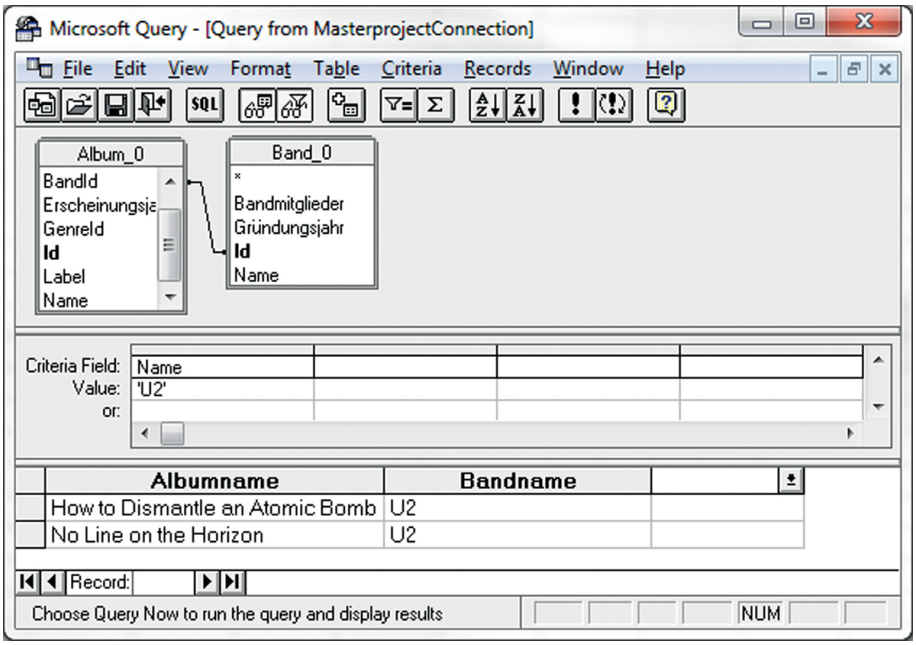

Abbildung 3.2: Microsoft Query – Erstellen einer Datenbankabfrage.

wahlweise durch die logischen Operatoren *AND* oder *OR* erfolgen.

Zusätzlich besteht die Möglichkeit Spalten in aufsteigender oder absteigender Reihenfolge zu sortieren. In der grafischen Benutzeroberfläche wird durch ein Symbol angezeigt, wenn eine Filterfunktion auf die Spalte angewendet wird, jedoch nicht welche Einschränkung diese mit sich bringt.

Abfragen via Microsoft Query

Der in *Excel* integrierte Abfrageeditor *Microsoft Query* (siehe Abbildung 3.2) bietet dem Benutzer die Möglichkeit, Daten von externen, SQL-basierenden Datenbanken abzurufen und die erhaltenen Datensätze in *Excel* zu integrieren. Nach dem Verbindungsaufbau zur Datenquelle können dem Abfrageeditor die vorhandenen Tabellen aus der Datenbank hinzugefügt werden.

Bestehende Beziehungen zwischen Tabellen importiert *Microsoft Query* nicht aus der externen Datenquelle. Erkennt der Editor beim Import eine Primärschlüssel-Spalte in einer Tabelle und eine Spalte mit demselben Namen in einer zweiten Tabelle, so wird automatisch eine Beziehung zwischen den beiden Tabellen hergestellt und im Abfrageeditor angezeigt.

Diese Vorgehensweise des Editors beim Importieren von Tabellen stellt Beziehungen zwischen Tabellen her, die in der SQL-basierenden Datenbank möglicherweise gar nicht existieren. Der Benutzer kann jedoch falsch interpretierte Beziehungen zwischen Tabellen über den Abfrageeditor selbst korrigieren.

Microsoft Query stellt einen Wizard zur Verfügung, der den Benutzer durch die einzelnen Schritte der Abfragenerstellung führt, anschließend die gewünschte Abfrage auf den Datenbestand ausführt und die Ergebnistabelle anzeigt.

Das Erstellen von Queries ohne den Wizard erfolgt durch Hinzufügen von Spalten zu einer leeren Tabelle, die dann durch Ausführen der erstellten Abfrage mit den Ergebnisdaten befüllt wird. Dadurch kann durch den Benutzer festgelegt werden, welche Daten die Abfrage enthalten soll. Fragt ein Query Daten aus unterschiedlichen Tabellen ab und sind Beziehungen zwischen diesen Tabellen vorhanden, so erfolgt eine Auflösung der Beziehungen und eine Verknüpfung der Datensätze durch den Editor. Bei Bedarf kann die Art der Datenverknüpfung durch den Benutzer festgelegt werden.

Zur Einschränkung der angezeigten Datensätze in der Abfrage, können Kriterien oberhalb der Ergebnistabelle in tabellarischer Form definiert werden. Die Bedingung beinhaltet die ausgewählte Spalte, den vergleichenden Operator sowie einen beliebigen Wert. Umfasst eine Abfrage mehrere Kriterien, so lassen sich diese mit logischen Operatoren verknüpfen.

Microsoft Query unterstützt das Speichern von Queries und ermöglicht dadurch das nachträgliche Abändern von Datenabfragen. Nach dem Schließen des Editors, werden die, durch Ausführung des Queries erhaltenen Datensätze, in *Excel* als Tabelle eingefügt.

Formeln, Funktionen und Diagramme

Durch die Verwendung von Formeln und Funktionen, die *Excel* für sämtliche Arten von Berechnungen bereitstellt, lassen sich zusätzliche Daten aus dem vorliegenden Datenbestand gewinnen und in weiterer Folge in Dokumenten zusammenfassen. Auch grafische Repräsentationen von Auswertungen können in Form von Diagrammen verwendet werden.

Verändern sich die für die Berechnungen herangezogenen Bestandsdaten, so müssen diese Dokumente nicht jedesmal neu erstellt werden. Eine neuerliche Auswertung der Formeln und Funktionen erfolgt in *Excel* automatisch. Zum Beispiel könnte *Excel* durch das Verwenden der Summenfunktion eine SQL-Abfrage, welche die Summe aller Spaltenwerte berechnet, ersetzen.

3.1.2 Microsoft Access

Die Software *Microsoft Access*[2] ist ein Datenbankmanagementsystem zur Verwaltung von Datenbanken und zur Entwicklung von datenbankspezifischen Anwendungen [23, 11].

[2]Microsoft Access: aktuelle Version 2013 (Stand 04/2013), http://office.microsoft.com/de-at/access/

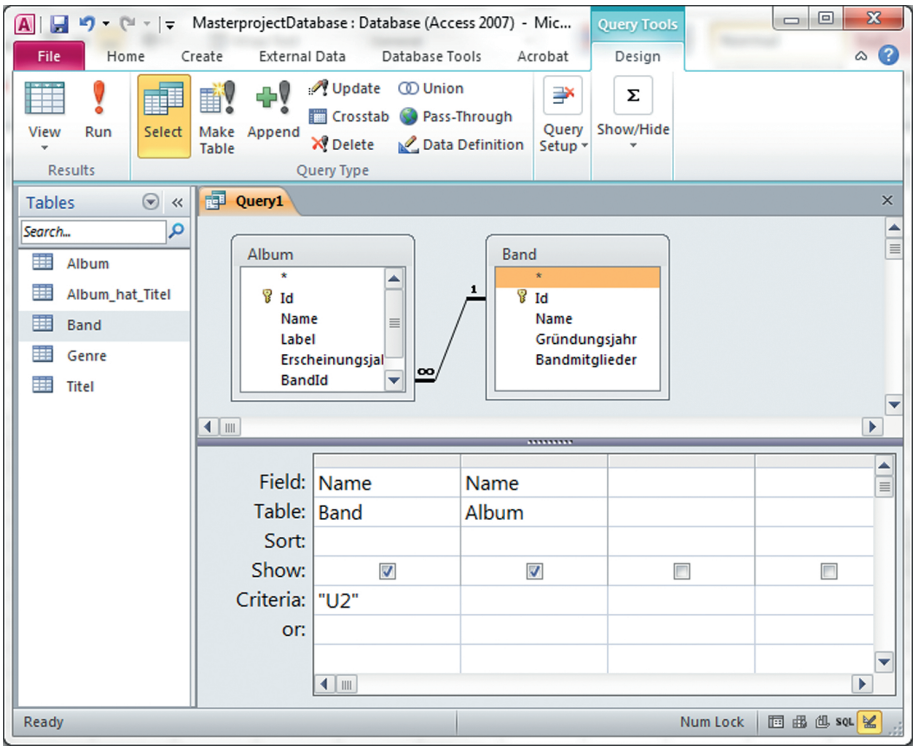

Abbildung 3.3: Microsoft Access 2010 – Erstellen einer Datenbankabfrage.

Datenbankabfragen

Access stellt dem Benutzer einen Editor zur Erstellung von Datenabfragen zur Verfügung, welcher drei verschiedene Ansichten besitzt. Neben der Ansicht zur Erstellung der Abfrage (ersichtlich in Abbildung 3.3), kann auch in die SQL-Ansicht gewechselt werden, wo die aktuelle Abfrage als SQL-Befehl dargestellt wird. Die dritte Ansicht dient zum Anzeigen der Ergebnisdaten, die durch Ausführen des Queries auf die Tabellen der *Access*-Datenbank entstehen. Neben dem Erstellen von Tabellen innerhalb der Anwendung, wird auch ein Import von Tabellen aus externen Datenbanken unterstützt. Dabei ist zu beachten, dass bestehende Beziehungen zwischen Tabellen nicht importiert werden und deshalb bei Bedarf manuell über ein Tool in *Access* zu definieren sind.

Wie schon *Microsoft Query* bietet auch *Microsoft Access* einen Wizard an, der den Benutzer zwar durch die Auswahl der Spalten für die Ergebnistabelle der Abfrage führt, jedoch die Schritte zum Definieren von Filterkriterien vernachlässigt.

Bei der Verwendung des Editors ohne den Wizard, kann der Benutzer im Menü auswählen, ob eine Datenabfrage (*SELECT*) oder eine Datenmanipulation (*UPDATE* oder *DELETE*) durchgeführt werden soll. Die Auswahl

verändert die tabellarische Eingabemaske.

Die, für das Query benötigten, Tabellen werden zur Arbeitsfläche hinzugefügt und mögliche Beziehungen zwischen den Tabellen angezeigt. Auswahlboxen in der tabellarischen Eingabemaske ermöglichen das Bestimmen der Spalten, die im Ergebnis der Abfrage enthalten sein sollen. Verwendet ein Query Spalten aus unterschiedlichen Tabellen und stehen diese Tabellen in einer Beziehung zueinander, so werden die vorhandenen Beziehungen bei der Abfrage durch den Editor aufgelöst und die Datensätze verknüpft. Die Einstellungen zur verwendeten Verknüpfungsart können im Editor getroffen werden.

Weiters besteht die Möglichkeit, Kriterien für Spalten zu definieren, wobei der Vergleichsoperator und der zu vergleichende Wert im jeweiligen Textfeld eingegeben werden muss. Beinhaltet die erstellte Abfrage mehrere Kriterien, so müssen diese durch logische Operatoren zu einer Gesamtbedingung verknüpft werden.

Zusätzlich gibt es für fortgeschrittenere Anwender die Möglichkeit, Spalten im Query zu definieren, die komplexere Ausdrücke mit logischen Operatoren und Funktionen darstellen. Diese Ausdrücke können über einen separaten Editor erzeugt werden.

Access ermöglicht das Speichern von erstellten Abfragen. Das Wechseln in die Ergebnis-Ansicht wertet die Abfrage aus und zeigt die Ergebnistabelle an.

Berichte

Berichte werden zum Anzeigen, Formatieren und Zusammenfassen von Daten aus Tabellen oder selbst erzeugten Abfragen verwendet. In *Microsoft Access* wird für die Berichterstellung auch ein Wizard angeboten, der Benutzer durch die einzelnen Schritte leitet.

Die im Bericht vorhandenen Datensätze können im Berichteditor nach verschiedenen Bereichen gruppiert oder sortiert werden. Dabei ist vor allem von Vorteil, wenn der Benutzer die Zusammenhänge der zu Grunde liegenden Bestandsdaten kennt, um sinnvolle Gruppierungen zu erzeugen.

3.1.3 phpMyAdmin

phpMyAdmin[3] dient zur Administration von *MySQL*-Datenbanken und wurde mit der Scriptsprache PHP entwickelt. Die Applikation ist plattformunabhängig und wird im Browser ausgeführt. Dadurch können auch Datenbanken auf fremden Servern verwaltet werden.

Die Anwendung stellt für alle häufig benutzten SQL-Operationen grafische Benutzeroberflächen zur Verfügung, während die direkte Formulierung und Ausführung von SQL-Befehlen zusätzlich unterstützt wird.

[3] phpMyAdmin: http://www.phpmyadmin.net/

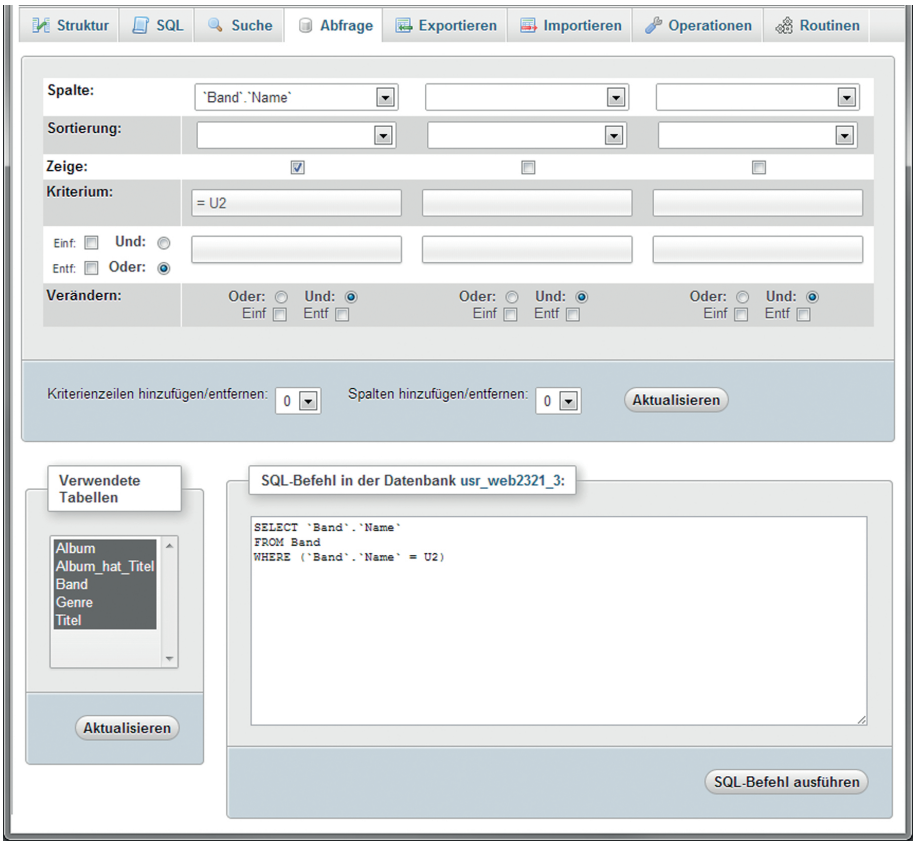

Abbildung 3.4: phpMyAdmin – Erstellen einer Datenbankabfrage.

Abfragen

Das Interface des Abfrageeditors (siehe Abbildung 3.4) sieht die Angabe von drei Spalten für die Ergebnistabelle der Abfrage vor. Diese Spalten lassen sich durch Auswahlboxen definieren. Soll die Ergebnistabelle mehr als drei Spalten besitzen, so kann das Interface von *phpMyAdmin* um weitere Eingabemöglichkeiten erweitert werden.

Der Editor ignoriert die Beziehungen zwischen Tabellen beim Generieren des SQL-Befehls. Beinhaltet das Query Datensätze aus mehreren Tabellen, so wird lediglich das Kreuzprodukt aus den gesamten Datensätzen gebildet. Dies bedeutet, dass die Ergebnistabelle alle möglichen Zeilenkombinationen aus den Tabellen enthält.

Neben der Auswahl der Sortierreihenfolge der Spalten können auch Kriterien definiert werden. Dabei muss der Benutzer den Vergleichsoperator und den zu vergleichenden Wert angeben. Werden noch zusätzliche Kriterien für die Spalte benötigt, so können weitere Eingabefelder hinzugefügt und logische Operatoren für die Verknüpfungen über Radiobuttons ausgewählt

werden. Durch Drücken des Buttons zum Aktualisieren wird der SQL-Befehl generiert und in einer Textbox angezeigt. Dadurch besteht für den Benutzer die Möglichkeit, den SQL-Befehl selbst zu ergänzen oder gegebenfalls abzuändern.

phpMyAdmin ermöglicht kein Speichern von Abfragen, um diese zu einem späteren Zeitpunkt wieder zu verwenden. Nach dem Ausführen des SQL-Befehls kann die Abfrage im Editor nicht mehr bearbeitet werden.

3.1.4 Drupal – Views

Die Content Management Plattform *Drupal*[4] wurde vom belgischen Informatiker Dries Buytaert im Jahre 2001 entwickelt. Das Open-Source-Framework unterliegt der General Public License (GPL) und wird hauptsächlich zur Verwaltung von Webinhalten (Content Management System) verwendet. Zusätzlich fungiert *Drupal* auch als datenbankbasiertes Content Management Framework, welches den Eingriff in die Anwendungsarchitektur erlaubt. Dadurch wird eine Erweiterung des Core-Systems durch die Implementierung von eigenen Modulen ermöglicht [6, Kap. 1].

In *Drupal* können Inhaltstypen (Content Types) erstellt werden, um die Struktur von Inhalten (Daten) durch Felder zu beschreiben. Die Installation von zusätzlichen Modulen ermöglicht das Definieren von Relationen zwischen Content Types. Dadurch lassen sich konkrete Inhalte, welche als Nodes bezeichnet werden, in der *Drupal*-Datenbank abbilden und verwalten.

Abfrage mit Drupalmodul Views

Mit Hilfe des Zusatzmodules *Views* besteht die Möglichkeit, die im System eingepflegten Inhalte abzufragen und anzuzeigen. In *Drupal* wird diese Art von Datenabfrage als View bezeichnet.

Beim Erstellen einer View verlangt das System zuerst die Eingabe eines Titelnamens für die Abfrage. Anschließend besteht die Möglichkeit durch Auswahl eines Inhaltstyps eine grundlegende Einschränkung der anzuzeigenden Inhalte vorzunehmen. Nach der Auswahl des gewünschten Anzeigeformats für die View, sind die Grundeinstellungen getroffen und dem Benutzer wird eine erste Darstellung vorgelegt.

Weitere Einstellungen können in der Entwurfsansicht (ersichtlich in Abbildung 3.5) erfolgen, wobei diese vom ausgewählten Anzeigeformat abhängig sind. Sollen zum Beispiel die Ergebnisse der Abfrage als Tabelle dargestellt werden, so besteht die Möglichkeit, die Spalten dieser Tabelle durch das Hinzufügen von Feldern der Inhaltstypen zu definieren.

Um die Inhalte der View weiter einzuschränken, können Filterkriterien durch den Benutzer festgelegt werden. Das Modul bietet dabei eine Vielzahl von vordefinierten Filtern.

[4]Drupal: aktuell Drupal Release 7 (Stand 04/2013), http://drupal.org/

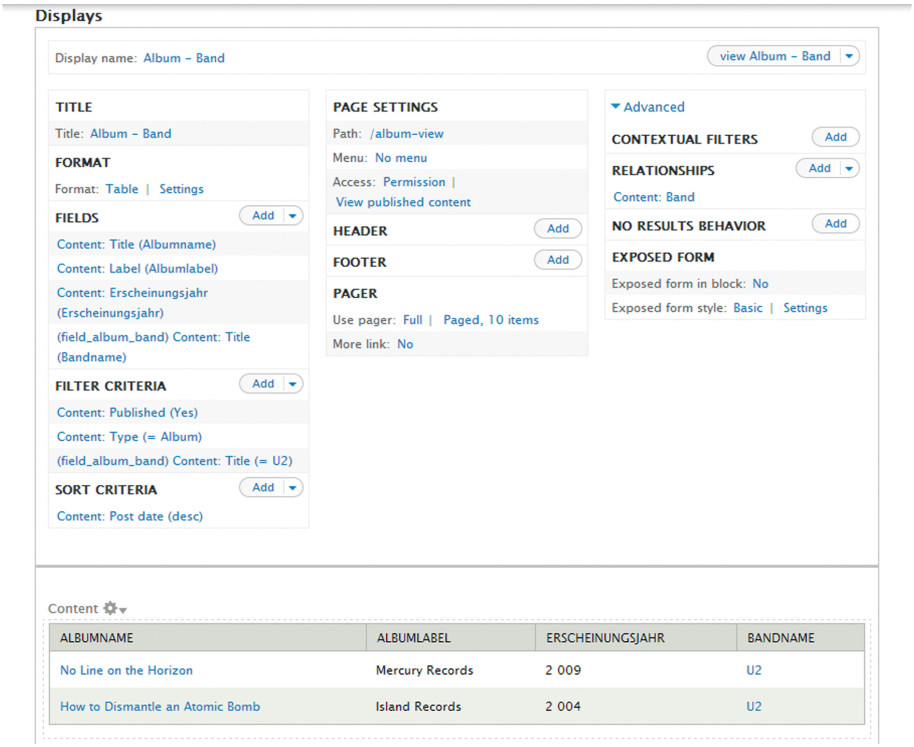

Abbildung 3.5: Drupal 7 Modul Views – Erstellen einer Datenabfrage.

Verfügen ausgewählte Inhaltstypen über eine Relation, so muss diese Verbindung in den Einstellungen unter Beziehungen angegeben werden. Dadurch wird ermöglicht, dass die Daten der beiden Inhaltstypen verknüpft und gemeinsam abgefragt werden können. Zusätzlich besteht die Möglichkeit die Darstellung der erstellten View individuell anzupassen. Die Inhaltsabfrage kann in *Drupal* abgespeichert und über einen, in den Einstellungen definierten, Pfad im Browser jederzeit abgerufen werden [18, Kap. 1].

3.1.5 Easy Query

Easy Query[5] ist ein grafischer Abfrageeditor für SQL-Datenbanken und lässt sich in Desktopanwendungen (WinForms und WPF) und webbasierte Applikationen (ASP.NET und Silverlight) integrieren.

Die Benutzeroberfläche des Editors (siehe in Abbildung 3.6) ist in Boxen unterteilt. Die Entities-Box beinhaltet alle Tabellen mit den jeweiligen Spalten, die in der, mit dem Editor verbundenen, Datenbank gespeichert sind. Vorhandene Beziehungen zwischen den unterschiedlichen Tabellen werden dem Benutzer nicht angezeigt.

[5]Easy Query: http://devtools.korzh.com/easyquery/

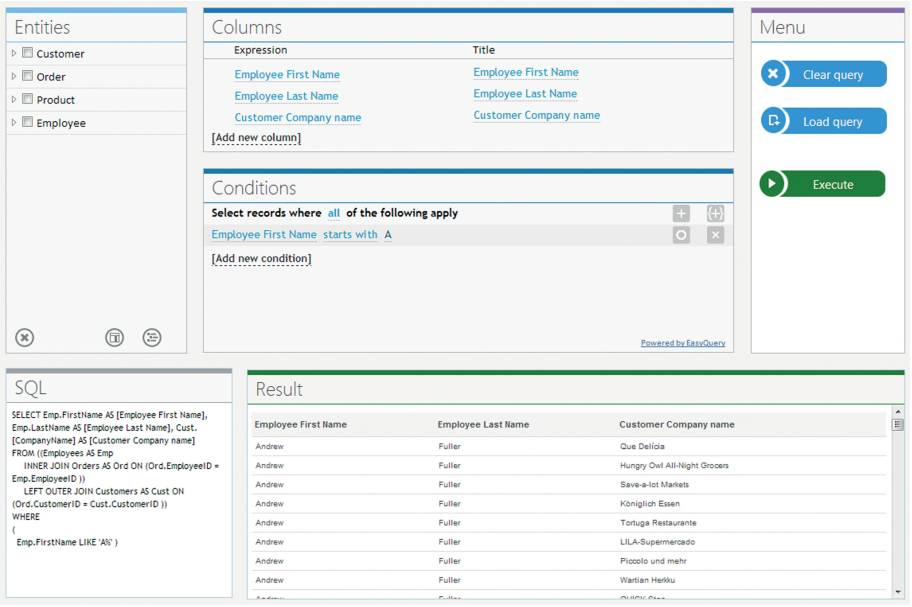

Abbildung 3.6: Easy Query für ASP.NET MVC – Erstellen einer Datenbankabfrage.

Aus dieser Box können beliebige Spalten per Drag & Drop oder über einen Button der Columns-Box oder der Conditions-Box hinzugefügt werden. Die hinzugefügten Spalten in der Columns-Box definieren die Daten, welche im Ergebnis der Abfrage vorkommen sollen. Zusätzlich können auch die Spaltennamen der Ergebnistabelle umbenannt werden.

Die Conditions-Box beinhaltet die Kriterien (Bedingungen), welche die Ergebnisdatensätze der Abfrage einschränken können. Damit die Bedingung einer Spalte vollständig ist, muss ein Vergleichsoperator ausgewählt und ein Wert angegeben werden.

Der Editor unterstützt das beliebige Verschachteln von Kriterien, wobei diese durch logische Operatoren verknüpft werden müssen. Weiters kann der Benutzer verschiedene Bedingungen durch einen Button deaktivieren. Dadurch wird dieses Kriterium beim Ausführen der Abfrage nicht berücksichtigt und lässt sich bei Bedarf auch wieder aktivieren.

Beziehungen zwischen Tabellen werden von *Easy Query* erkannt und berücksichtigt. Dies bedeutet, dass im Falle einer Beziehung zwischen zwei Tabellen die Datensätze miteinander verknüpft werden.

Die SQL-Box stellt den aus der Abfrage generierten SQL-Befehl dar und das Abfrageergebnis wird in der Result-Box als Tabelle angezeigt. *Easy Query* unterstützt das Speichern von erstellten Abfragen im XML-Format. Dadurch lässt sich das Query jederzeit wieder laden und abändern.

3.2 Erkenntnisse und Fazit

Die vorgestellten fünf Anwendungen wurden auf Funktionalität und Bedienbarkeit getestet. Vor allem die Verwendung der Abfrageeditoren durch technisch unversierte Benutzer stand bei den Tests im Vordergrund. Dabei wurden Erkenntnisse gesammelt, welche in diesem Abschnitt kurz zusammengefasst werden.

3.2.1 Auswahl der Spalten für die Abfrage

In den Abfrageeditoren von *Microsoft Excel*, *Microsoft Access* und *phpMyAdmin* werden die Spalten für die Ergebnistabelle der Abfrage nebeneinander in einer tabellarischen Eingabemaske definiert, wobei *phpMyAdmin* anfangs nur Eingabemöglichkeiten für drei Spalten vorsieht. Will der Benutzer mehr als drei Abfragespalten definieren, so ist zwar im Interface des Editors ein Button zum Erweitern der Eingabemaske vorhanden, jedoch könnte diese Funktion für einige Benutzer nicht klar ersichtlich sein. Im Allgemeinen wirkt die vorgegebene tabellarische Eingabemaske bei der Formulierung einer Abfrage in allen drei Anwendungen einschränkend auf den Benutzer.

Besser verläuft die Auswahl der Abfragespalten bei *Drupal* und *Easy Query*. Dort werden die gewünschten Spalten untereinander hinzugefügt, was auch übersichtlicher und flexibler wirkt.

3.2.2 Definieren von Kriterien

Das Hinzufügen von Kriterien zur Datenabfrage in den Anwendungen *Microsoft Excel*, *Microsoft Access* und *phpMyAdmin* erfolgt durch Eingabe eines Vergleichsoperators und eines Wertes. Dieser Wert wird anhand des Vergleichsoperators mit dem Wert der Spalte verglichen und stellt dadurch eine Bedingung in der Abfrage dar.

Dabei könnte ein Problem für Benutzer ohne SQL-Kenntnisse auftreten, da diese, um ein Kriterium zu definieren, die Schlüsselwörter für die Vergleichsoperatoren aus der Datenbanksprache SQL beherrschen müssten. Zum Beispiel muss der Benutzer, um das Vorkommen eines bestimmten Buchstabens in einem Namen zu prüfen, das Schlüsselwort *LIKE* als Vergleichsoperator beim Definieren des Kriteriums eingeben.

Beinhaltet eine Abfrage mehrere Kriterien, so müssen diese durch logische Operatoren zu einer Gesamtbedingung verknüpft werden. Dabei könnten die logischen Zusammenhänge zwischen den Kriterien, einerseits durch die unübersichtliche, tabellarische Eingabemaske und andererseits durch das fehlende Feedback beim Hinzufügen eines Kriteriums, für den Benutzer sehr schwer zu erkennen sein. Das beliebige Verschachteln von Kriterien ist in allen drei Editoren nicht möglich.

Bessere Lösungen für das Definieren von Kriterien wurden in den Edi-

toren von *Drupal* und *Easy Query* gefunden. Dort werden Bezeichnungen für die Vergleichsoperatoren in natürlicher Sprache vordefiniert und können durch den Benutzer aus einer Liste ausgewählt werden. Beispielsweise wird in beiden Editoren für das oberhalb erwähnte SQL-Schlüsselwort *LIKE* die Bezeichnung *CONTAINS* verwendet. Dadurch können Kriterien in diesen beiden Editoren wie Sätze in natürlicher Sprache gelesen und verstanden werden.

Die Verknüpfung von mehreren Kriterien durch logische Operatoren erfolgt in *Drupal* und *Easy Query* durch Verwenden von Gruppen. Der Gruppe wird ein logischer Operator zugewiesen und alle Kriterien bzw. Untergruppen innerhalb dieser Gruppe werden durch diesen Operator verbunden. Die Gruppen lassen sich untereinander wiederum durch logische Operatoren verknüpfen. Dieses Konzept ermöglicht das beliebige Verschachteln von Kriterien. Die logischen Verbindungen sind auch in der Benutzeroberfläche der Editoren gut nachvollziehbar.

3.2.3 Beziehungen zwischen Tabellen

phpMyAdmin ignoriert bei der Datenabfrage Beziehungen, die zwischen Tabellen bestehen. Dadurch werden die Datensätze bei der Abfrage nicht verknüpft und es wird lediglich das Kreuzprodukt aus den Tabellen gebildet.

Die anderen vier vorgestellten Editoren lösen automatisch vorhandene Beziehungen zwischen Tabellen bei der Datenabfrage auf, wobei in *Drupal* die Beziehungen explizit bei der Erstellung der View angegeben werden müssen. Das Berücksichtigen von Beziehungen zwischen Tabellen hat zur Folge, dass eine korrekte Verknüpfung von Datensätzen aus unterschiedlichen Tabellen stattfindet.

3.2.4 Feedback für Benutzer

Editoren sollten bei der Erstellung der Abfrage ausreichendes Feedback geben, damit der Benutzer zu jedem Zeitpunkt der Abfrageerstellung das Query auf Fehler kontrollieren und bei Bedarf auch korrigieren kann. Dieses Feedback sollte für den Benutzer möglichst leicht zu deuten sein. Dadurch würde sich die zusätzliche Formulierung der Abfrage in menschlicher und natürlicher Sprache anbieten.

Microsoft Excel, *Microsoft Access* und *phpMyAdmin* bieten in ihren Abfrageeditoren nahezu gar kein Feedback für den Benutzer. Vor allem auf Grund der unübersichtlichen Darstellung der logischen Zusammenhänge zwischen Kriterien kann es dadurch schwer sein, die Abfrage fehlerlos zu erstellen. Der Benutzer müsste die Abfrage während der Erstellung mehrmals ausführen, um anhand der Ergebnisdaten feststellen zu können, ob die Abfrage korrekt erstellt worden ist.

Bei *Drupal* und *Easy Query* wird dem Benutzer durch die benutzer-

freundlichere Oberfläche und die Darstellung der Abfragekriterien in natürlicher Sprache wesentlich mehr Feedback gegeben, als bei den anderen drei vorgestellten Editoren. Zusätzlich werden Abfragen während der Erstellung im *Easy Query*-Editor als SQL-Befehl dargestellt und können dadurch einigen Benutzern eine gewisse Art von Feedback übermitteln.

3.2.5 Fazit

Die drei Abfrageeditoren von *Microsoft Excel*, *Microsoft Access* und *phpMyAdmin* sind im Aufbau des Interfaces sowie bei der Erstellung von Abfragen fast identisch. Der Editor der webbasierten Anwendung *phpMyAdmin* sollte auf Grund seiner eingeschränkten Funktionalitäten nur für einfache Abfragen verwendet werden.

Easy Query bietet gegenüber *Drupal* durch die Darstellung der erstellten Abfrage als SQL-Befehl und die automatische Auflösung von Beziehungen zwischen Tabellen die vollständigere Lösung für einen Abfrageeditor.

Kapitel 4

Grafischer Query Designer

In diesem Kapitel werden, nach Beschreibung der Ausgangssituation, die aus dem vorherigen Kapitel gewonnenen Erkenntnisse als Grundlage für die Entwicklung eines grafischen Query Designers (in Zukunft auch gQD) herangezogen.

Anschließend werden die Anforderungen an einen benutzerfreundlichen Abfrageeditor für relationale Datenbanken formuliert und daraus ein Konzept entwickelt, das die Basis für die Implementierung eines Prototyps darstellt.

4.1 Ausgangssituation

Wie schon bereits in Abschnitt 2.3.1 erwähnt, ist SQL eine Datenbanksprache für relationale Datenbanksysteme, welche u. a. das Formulieren von Datenabfragen unterstützt.

Beispielsweise könnte eine Datenbank vorliegen, die Studenten mit ihren spezifischen Eigenschaften wie Alter, Wohnort, Studiengang u. a. abbildet. Dabei ist noch unbekannt, welche Abfragen aus dem Datenbestand später interessant sein könnten. Dieses Szenario erfordert eine Möglichkeit, die Abfragen flexibel formulieren zu können, um in weiterer Folge die gewünschten Daten zu erhalten.

Grundsätzlich können Anwendungen für technisch unversierte Benutzer implementiert werden, die intern durch SQL-Abfragen auf Daten in der Datenbank zugreifen, diese in aufbereiteter Form anzeigen oder zur weiteren Verarbeitung zur Verfügung stellen. Jedoch ist durch solche Anwendungen sehr wenig Flexibilität in der Datenabfrage gegeben, da die internen SQL-Queries statisch im Anwendungscode implementiert sind.

Benutzer können diese Flexibilität erlangen, indem sie ihre Anforderungen an das Datenbankmanagementsystem als SQL-basierte Abfragen formulieren und auf die Datenbank ausführen, um die gewünschten Daten als Ergebnis zu erhalten.

SQL als Abfragesprache erfordert das Zusammensetzen diverser Klauseln in bestimmter Reihenfolge zu einer Abfrage. Dabei muss die SQL-Syntax eingehalten werden, um ein fehlerfreies Ausführen der Abfrage auf die Datenbank zu gewährleisten.

Dies sollte zwar für Datenbankadministratoren und SQL-Programmierer keine Schwierigkeit darstellen, jedoch kann die Abfrageerstellung für technisch unversierte Benutzer ohne SQL-Kenntnisse sehr schwer werden. Zusätzlich sollte der Benutzer neben der SQL-Syntax über das zugrunde liegende Schema der Datenbank Bescheid wissen. Dadurch können semantische Fehler in der Abfrage, wie zum Beispiel die Verwendung von nicht vorhandenen Spaltennamen oder der Vergleich zweier nicht kompatibler Datentypen, vermieden werden.

Beinhalten erstellte Queries trotzdem Fehler, so werden beim Ausführen der Abfrage entsprechende Fehlermeldungen durch das Datenbankmanagementsystem (SQL-Server) generiert. Jedoch sind diese Fehlermeldungen zumeist sehr technisch formuliert und können dadurch einem technisch unversierten Benutzer nur wenig bei der Beseitigung der Fehler in der Abfrage weiterhelfen. Somit sind Benutzer oft gezwungen, zu Handbüchern oder anderen Hilfsmitteln zu greifen, damit das erstellte Query korrigiert und erfolgreich ausgeführt werden kann [3, 16].

Durch die Entwicklung eines grafischen Query Designers soll vor allem für technisch unversierte Benutzer eine Anwendung geschaffen werden, welche die Erstellung von SQL-Datenabfragen und die anschließende Ausführung auf die Datenbank wesentlich erleichtert. Dabei soll bewerkstelligt werden, dass der Benutzer keinerlei Grundkenntnisse in SQL benötigt und die Bedienung intuitiv erfolgen kann.

4.2 Anforderungen

Das grundsätzliche Ziel des geplanten Abfrageeditors, die Benutzung durch technisch wenig bewanderte Personen zu ermöglichen, kann durch die im Anschluss definierten Anforderungen erreicht werden. Bei der Definition dieser Anforderungen wurden auch die Erkenntnisse aus den, im vorherigen Kapitel getesteten, Abfrageeditoren miteinbezogen.

4.2.1 Plattformunabhängigkeit

Anwendungen, wie zum Beispiel *Microsoft Excel* oder *Microsoft Access*, sind desktopbasierte Applikationen und müssen lokal am Rechner des Benutzers installiert und konfiguriert werden. Dadurch ist der Benutzer in der Verwendung eingeschränkt, da die Applikation nur auf diesem Rechner ausgeführt werden kann. Zum Beispiel müsste die Anwendung für die Benutzung am Arbeitsplatz und zu Hause jeweils am Rechner vor Ort installiert werden.

Die Entwicklung als webbasierte Applikation stellt die Plattformunabhängigkeit her. Dies würde bedeuten, dass die Anwendung auf unterschiedlichen Betriebssystemen und Rechnerarchitekturen eingesetzt werden kann.

Wie schon in Abschnitt 2.1 erwähnt, besteht ein Datenbanksystem (DBS) aus der Datenbank, die den gesamten Datenbestand enthält, sowie dem Datenbankmanagementsystem zur Manipulation der Daten. Dabei soll es keinen Unterschied machen, ob das Datenbanksystem auf dem gleichen Server wie die Anwendung oder auf einem anderen externen Server installiert ist. In der geplanten Applikation wird die benötigte Datenbank über Parameter angegeben und die Verbindung zur Datenquelle erfolgt automatisch. Dadurch lässt sich die Datenbank einfach durch Verändern der Datenbankparameter in der Konfiguration des Editors austauschen. Der Benutzer kann sich somit auf die Erstellung der Abfrage konzentrieren und muss keine datenbankspezifischen Einstellungen vornehmen.

4.2.2 Konfiguration

Viele Abfrageeditoren, die vorwiegend für Benutzer mit SQL-Kenntnissen entwickelt wurden, bieten den gesamten Funktionsumfang von SQL an. Dadurch wirken diese für weniger technisch fortgeschrittene Benutzer sehr unübersichtlich. Zudem benötigt der technisch eingeschränkte Anwender bei der Abfrageerstellung ohnehin nur die Grundfunktionen von SQL.

Da der geplante Editor in erster Linie die Verwendung durch technisch unversierte Personen vorsieht, sollte sich dieser konfigurieren lassen. Diese Konfiguration kann in der Regel durch Systemadministratoren oder Personen mit technischem Verständnis durchgeführt werden. Dadurch lässt sich der Funktionsumfang des Editors für bestimmte Benutzer und deren Aufgabengebiete bei der Abfragenerstellung individuell anpassen.

Dies hat den Vorteil, dass der Anwender nur die Funktionalitäten bei der Abfragenerstellung zur Verfügung hat, die er auch wirklich benötigt und dadurch nicht mit, für ihn überflüssigen, Funktionen konfrontiert wird.

Die Applikation soll zusätzlich das persistente Speichern verschiedener Konfigurationen unterstützen, wobei das Laden der Einstellungen für den Abfrageeditor durch den Benutzer selbst möglich sein muss.

4.2.3 Interface und Bedienung

Die grafische Benutzeroberfläche dient der Interaktion zwischen dem Benutzer und dem Editor und ermöglicht es dem Anwender die Abfragen zu erstellen. Alle Interaktionsmöglichkeiten sollen im übersichtlich gestalteten Interface klar erkennbar sein und zusammengehörige Bereiche im Interface durch entsprechende Kategorisierung deutlich gemacht werden.

Das Interface des geplanten Systems verwendet eine natürliche Sprache, die dem Anwender vertraut ist, um die Kommunikation mit dem technisch

unversierten Benutzer zu erleichtern. Dabei ist geplant, dass vor allem auf technische Systemmeldungen und datenbankspezifische Fehlermeldungen im Interface verzichtet wird und die Darstellung von Informationen wie zum Beispiel Bezeichnungen (Labels) oder Bedienelemente (Buttons) verständlich erfolgt. Der Vorteil liegt u. a. darin, dass die Vertrautheit des Anwenders mit dem Interface schneller erlangt werden kann.

Die Bedienung der webbasierten Anwendung durch den Benutzer erfolgt intuitiv und verwendet bekannte Bedienkonzepte wie z. B. Drag & Drop, Copy & Paste, Shortcuts, Kontextmenüs u. v. m. aus desktopbasierten Anwendungen.

4.2.4 Hilfe

Grundsätzlich sollte eine Anwendung auch ohne Dokumentation durch den Benutzer verwendet werden können.

Das Interface des Abfrageeditors soll weitestgehend selbsterklärend sein und dem Benutzer die Verwendung der Anwendung ohne zusätzliche Anleitungen oder Erklärungen ermöglichen. Trotz dieser Anforderung kann eine Hilfe zum Nachlesen angeboten werden, wobei diese nur Erklärungen zu Funktionalitäten beinhalten soll, die der Editor in seiner momentanen Konfiguration unterstützt. Dadurch kann der Benutzer die Hilfestellung bei Bedarf schneller finden und wird nicht mit überflüssigen Informationen belastet.

Zusätzlich zur bekannten, klassischen Hilfe bietet die geplante Anwendung eine weitere Möglichkeit an, den Benutzer bei der Abfragenerstellung zu unterstützen. Dabei wird auf Benutzerinteraktionen reagiert und passende Tipps für den Benutzer im Interface angezeigt. Diese Tipps können den Benutzer auf Fehler oder weitere Möglichkeiten aufmerksam machen.

4.2.5 Erstellen von Abfragen

Wie schon in Abschnitt 4.2.3 erwähnt, kann der Benutzer über das grafische Interface der Anwendung die gewünschte Abfrage erstellen. Dabei gilt für den Benutzer, das Konzept der Abfragenerstellung zu verstehen und so den, in der Benutzersprache vorliegenden, Abfragewunsch im Interface als Datenabfrage zu formulieren.

Datenbankschema

Bei einigen im vorherigen Kapitel getesteten Abfrageeditoren fiel auf, dass die Datenbankstruktur durch Tabellen und deren Beziehungen zueinander im Editor grafisch dargestellt wird. Dabei stellen diese Abfrageeditoren zumeist auch vorhandene Tabellenstrukturen dar, die nur vom Datenbanksystem benötigt werden, jedoch nicht die eigentlichen Nutzdaten beinhalten. Beispielsweise könnte dies ein Primärschlüsselfeld einer Tabelle sein, das die

Datensätze durch eine fortlaufende Zahl eindeutig identifizierbar macht, jedoch für die Abfrage der Daten nicht relevant ist.

Der geplante Abfrageeditor stellt die Verbindung zur Datenbank auf Grund der vorliegenden Konfiguration her. Dadurch muss der Benutzer die Struktur und den Aufbau der Datenbank nicht zwingend kennen, um Datenabfragen definieren zu können.

Zusätzlich lassen sich die Spalten der Tabellen, auf die eine Datenabfrage durch den Benutzer ausgeführt werden darf, im Editor konfigurieren. Dadurch werden dem Benutzer nur für ihn relevante Tabellenspalten für die Datenabfrage im Interface angezeigt.

Beziehungen zwischen Tabellen

Der Benutzer soll die Möglichkeit haben, Abfragen zu formulieren, die auf Daten aus unterschiedlichen Tabellen in der verwendeten Datenbank zugreifen. Werden beim Ausführen der Abfrage vorhandene Beziehungen zwischen den Tabellen erkannt, so erfolgt eine Verknüpfung zwischen diesen Datensätzen und deren gemeinsame Darstellung in der Ergebnistabelle der Abfrage.

Somit kann der Benutzer, unabhängig von der vorliegenden Datenbankstruktur, Abfragen formulieren, ohne zu wissen, wie der Datenbestand innerhalb der Datenbank abgebildet ist.

Auswahl der Abfragedaten

Die Daten, die für die Abfrage verwendet werden sollen, müssen durch den Benutzer definiert werden. Dies geschieht in dem geplanten Editor durch die Auswahl von Spalten, die in den unterschiedlichen Tabellen der Datenbank vorhanden sind.

Der Abfrageeditor soll im Interface alle möglichen Spalten darstellen, die für die Datenabfrage verwendet werden können. Der Anwender kann diese dann über die Benutzeroberfläche beliebig zur Abfrage hinzufügen. Dadurch wird auch die Struktur der Ergebnistabelle definiert, die nach Ausführen der Abfrage die Ergebnisdaten enthält.

Für das Hinzufügen bzw. Löschen von Spalten muss keine Eingabe der Spaltennamen erfolgen. Dadurch wird gewährleistet, dass der Benutzer kein Hintergrundwissen zum Datenbankschema benötigt. Außerdem werden syntaktische Fehler durch die Eingabe falscher Spaltennamen vermieden.

Kriterien

Nachdem die Daten für die Abfrage festgelegt wurden, besteht die Möglichkeit Kriterien für die Abfragespalten zu definieren. Die Ergebnistabelle der Abfrage enthält nur jene Datensätze, die alle definierten Kriterien erfüllen.

Der Benutzer soll mit dem geplanten Editor beliebig viele Kriterien in natürlicher Sprache formulieren und an Spalten in der Abfrage binden können.

Dadurch muss der Anwender keine spezifischen Schlüsselwörter der Abfragesprache wissen und kann das Kriterium nach dem Erstellen leichter überprüfen. Beinhaltet die Abfrage mehrere Kriterien, so können diese über das Interface des geplanten Editors beliebig verschachtelt und miteinander verknüpft werden. Dies soll für den Benutzer möglichst einfach und verständlich sein.

Die Kriterien in der Abfrage sollen sich deaktivieren und bei Bedarf wieder aktivieren lassen. Der Vorteil liegt darin, dass der Benutzer relativ schnell die Auswirkungen von bestimmten Kriterien auf die Ergebnisdaten der Abfrage herausfinden kann.

Feedback

Während der Erstellung der Abfrage gilt es, dem Benutzer nach jeder Interaktion eine gewisse Art von Feedback zu geben. Tritt zum Beispiel ein Fehler in der Abfrage auf, so kann der Benutzer durch eine dementsprechende Fehlermeldung über den Fehler informiert werden. Außerdem sollte das Feedback bereits Lösungsvorschläge zur Beseitigung des Fehlers enthalten.

Die geplante Applikation interpretiert die Abfrage während der Erstellung durch den Benutzer und stellt diese zu jedem Zeitpunkt in natürlicher Sprache im Interface dar. Dadurch lässt sich die, durch den Benutzer erstellte, Abfrage besser deuten.

Außerdem kann wesentlich einfacher festgestellt werden, ob die erstellte Abfrage den Vorstellungen des Benutzers entspricht. Eventuell enthaltene Fehler in der Abfrage können ebenfalls schneller erkannt werden.

Speichern

Das Speichern von Abfragen soll dem Benutzer die Möglichkeit bieten, die Erstellung von Abfragen zu einem späteren Zeitpunkt fortzusetzen oder bestehende Abfragen wiederzuverwenden und gegebenfalls abzuändern.

4.2.6 Ausführen von Abfragen

Durch Ausführen der Abfrage im geplanten Abfrageeditor bekommt der Benutzer die Ergebnisdaten als Tabelle in der Benutzeroberfläche übersichtlich angezeigt. Zusätzlich zur Ergebnistabelle wird der, aus der erstellten Abfrage generierte, SQL-Befehl dargestellt, um den Benutzer mit der Datenbanksprache SQL vertraut zu machen und einen gewissen Lerneffekt beim Anwender zu erzielen.

Dauert das Ausführen von komplexeren Abfragen länger, so soll dies dem Benutzer über das Interface des Editors entsprechend mitgeteilt werden. Bereits ausgeführte Abfragen lassen sich auch nachträglich im geplanten Editor erweitern oder abändern.

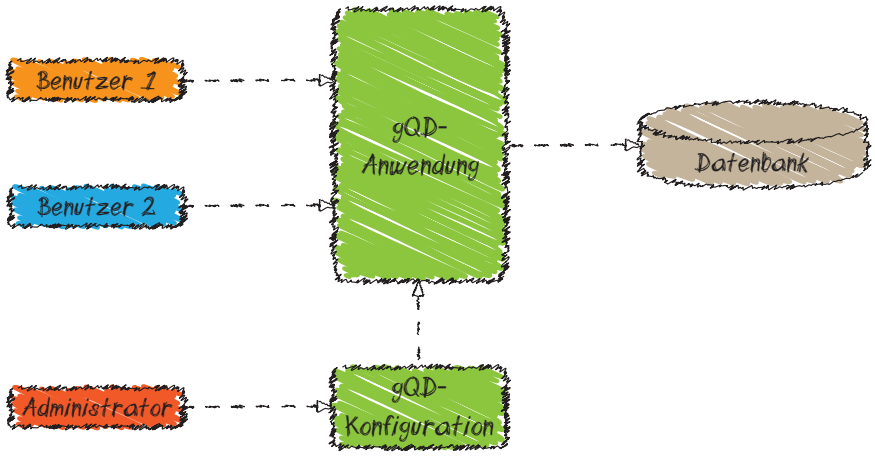

Abbildung 4.1: Grundkonzept des gQDs.

Export von Ergebnisdaten

Die, durch das Ausführen der Abfrage erhaltenen, Ergebnisdaten kann der Benutzer in bereits aufbereiteter Form durch den Abfrageeditor in verschiedene Formate exportieren lassen. Sollen die Ergebnisdaten zur Weiterverarbeitung in anderen Anwendungen integriert werden, so ist ein Export der Abfragedaten als Rohdaten durch den Editor möglich.

4.3 Konzept

Die Skizze in Abbildung 4.1 stellt das geplante Zusammenspiel zwischen den Benutzern, der Anwendung und der Datenbank grafisch dar. Die Implementierung des geplanten Query Designer soll als webbasierte Anwendung erfolgen. Dadurch können Benutzer unabhängig von der verwendeten Plattform (Betriebssystem und Rechnerarchitektur), im Browser beliebige Datenbankabfragen über das grafische Interface des gQDs formulieren. Der Zugriff auf den gQD kann durch mehrere Benutzer gleichzeitig erfolgen. Weiters ist der gQD konfigurierbar, um das dargestellte Interface und den Funktionsumfang auf die Bedürfnisse des Benutzers anzupassen. Dabei ist vorgesehen, dass die gQD-Konfiguration beispielsweise in Unternehmen von Administratoren oder von anderen technisch ausgebildeten Personen erstellt wird.

Die, durch den Benutzer erstellte, Abfrage wird durch den gQD interpretiert und in die entsprechende Datenbanksprache konvertiert. Die Anwendung führt die Abfrage auf die Datenbank aus und die erhaltenen Ergebnisdaten werden durch den gQD benutzerfreundlich aufbereitet und im Browser dargestellt.

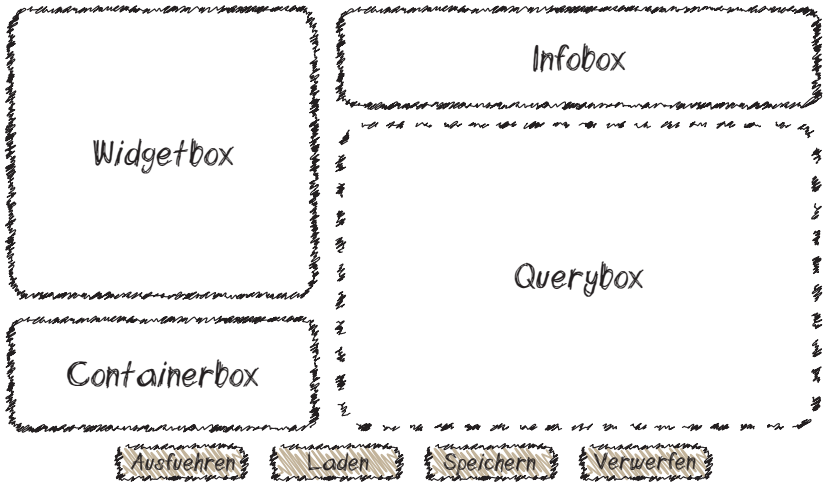

Abbildung 4.2: Interface für die Erstellung einer Abfrage im gQD.

4.3.1 Interface

Das Interface des gQDs ermöglicht die Interaktion zwischen dem Benutzer und der Anwendung und ist durch Boxen in unterschiedliche Bereiche unterteilt. Dadurch soll dem Anwender eine Benutzeroberfläche geboten werden, die übersichtlich gestaltet ist und eine geringe Einarbeitungszeit benötigt.

Die bekannten Bedienelemente wie zum Beispiel Buttons, Auswahlboxen und Textfelder werden in einem einheitlichen Design in der Benutzeroberfläche des gQDs dargestellt, um dem Benutzer alle Interaktionsmöglichkeiten aufzuzeigen.

Das Interface des gQDs für die Erstellung der Abfrage wird in folgende Bereiche untergliedert (Skizze in Abbildung 4.2):

- **Widgetbox:** Diese Box enthält interaktive Elemente, die für die Erstellung von Abfragen benötigt werden. Diese Elemente werden im gQD als Widgets bezeichnet und stellen die Spalten von Tabellen aus der Datenbank dar, die durch den Benutzer abgefragt werden können. Mehr zu den Widgets in Abschnitt 4.3.2.
- **Containerbox:** Durch Container-Elemente, die diese Box beinhaltet, kann der Benutzer bei der Abfragenerstellung beliebige Widgets verschachteln oder einfache Funktionen auf Widgets in der Abfrage ausführen. Innerhalb der Containerbox sind die Elemente zusätzlich gruppiert, damit der Benutzer den Container mit der benötigten Eigenschaft bei der Erstellung der Abfrage einfach finden und verwenden kann. Weiterführende Informationen zu diesen Elementen in Abschnitt 4.3.3.

Abbildung 4.3: Grundsätzlicher Aufbau eines Widgets.

- **Querybox:** Die Querybox ist das Herzstück des gQD-Interfaces. Der Benutzer kann dieser Box Widgets und Container hinzufügen und so die gewünschte Abfrage interaktiv und flexibel formulieren. Dabei ist es durchaus möglich ein Widget oder einen Container mehrmals in der Abfrage zu verwenden.
- **Infobox:** Diese Box dient zur Kommunikation zwischen der Anwendung und dem Benutzer. Der gQD reagiert auf Benutzerinteraktionen und gibt weiterführende Tipps und Feedback. Die Informationsübermittlung erfolgt durch den gQD innerhalb dieser Box in natürlicher Sprache. Dadurch können technisch unversierte Benutzer bei der Abfragenerstellung wesentlich unterstützt werden.
- **Buttons:** Die Buttons unterhalb der Querybox zeigen die möglichen Interaktionen zur Verarbeitung der erstellten Abfrage für den Benutzer auf. So kann die Abfrage auf die Datenbank ausgeführt oder gespeichert werden. Durch die Speicherfunktion ermöglicht der gQD, die Abfrage zu einem späteren Zeitpunkt erneut zu verwenden bzw. abzuändern. Das Laden der Abfrage erfolgt über den entsprechenden Button. Weiters kann die Querybox durch einen Button im Interface geleert werden.

4.3.2 Widgets

Wie schon oberhalb erwähnt, repräsentiert jedes Widget im Interface des gQDs eine Spalte einer Tabelle aus der Datenbank. Dadurch kann der Benutzer bei der Erstellung der Abfrage durch Hinzufügen des Widgets zur Querybox die Spalten auswählen, die in der Ergebnistabelle der Abfrage vorhanden sein sollen. Außerdem lässt sich über jedes Widget ein Filterkriterium für die zugehörige Tabellenspalte definieren, um die Ergebnisdaten der Abfrage einzuschränken.

In der Konfiguration des gQDs muss für jede Tabellenspalte, die dem Benutzer bei der Datenbankabfrage zur Verfügung stehen soll, ein Widget definiert werden.

Aufbau

Der Aufbau eines Widgets im gQD wird in Abbildung 4.3 skizziert dargestellt. Die Abbildung zeigt, dass sich ein Widget aus mehreren Elementen zusammensetzt:

- **Namen:** Der Name des Widgets deutet idealerweise auf den Inhalt der repräsentierten Spalte hin. Zum Beispiel wäre für die Spalte *Name* in der Tabelle *Band* der Widgetname „Bandname“ aussagekräftig. Dieser Widgetname wird beim Definieren von Widgets in der Konfiguration angegeben. Dadurch stellen auch gleiche Spaltennamen in unterschiedlichen Tabellen kein Problem für den gQD dar. Über dieses Element lässt sich das Widget während der Erstellung der Abfrage durch den Benutzer per Drag & Drop verschieben.
- **Button „Anzeigen“:** Dieses Bedienelement gibt einen Zustand des Widgets an und kann durch den Benutzer aktiviert oder deaktiviert werden. Bei aktiviertem Button werden die Daten der Tabellenspalte, die das Widget repräsentiert, im Ergebnis der Abfrage angezeigt. Dies bedeutet aus der Sicht von SQL, dass die *SELECT*-Klausel um den Spaltennamen erweitert wird.
- **Button „Filter“:** Das Konzept der Widgets sieht vor, dass für die verknüpfte Tabellenspalte ein Filterkriterium definiert werden kann. Die Daten in der Tabellenspalte müssen dieses Kriterium erfüllen, um in der Ergebnistabelle der Abfrage angezeigt zu werden. Durch diesen Button lässt sich die Filterfunktion des Widgets durch den Benutzer aktivieren bzw. deaktivieren.
- **Filtereinstellungen:** Ist die Filterfunktion des Widgets aktiv, so wird dem Benutzer ermöglicht, die Einstellungen für das Filterkriterium zu treffen. Der gQD sieht verschiedene Typen von Widgets vor, welche sich in den Einstellungsmöglichkeiten für den Widgetfilter unterscheiden. Durch das Definieren des Filterkriteriums wird im Bezug auf die Datenbanksprache SQL die *WHERE*-Klausel in der Abfrage erweitert.

Typen

Es werden verschiedene Typen von Widgets unterstützt, die bei der Konfiguration des gQDs ausgewählt werden können. Diese unterscheiden sich in der Art und Weise wie ein Filterkriterium für die Tabellenspalte durch den Benutzer definiert werden kann. Alle Daten der, vom Widget repräsentierten, Spalte, die dieses Filterkriterium erfüllen, scheinen in den Ergebnisdaten der Abfrage auf.

Der gQD stellt für die Abfragenerstellung mehrere Widgettypen zur Verfügung. Diese werden in Abbildung 4.4 skizziert dargestellt und im folgenden Abschnitt näher beschrieben:

- **Textbox-Widget:** Durch dieses Widget hat der Benutzer die Möglichkeit, einen Wert über die Textbox zu definieren. Das Filterkriterium gilt dann als erfüllt, wenn der eingegebene Wert mit den Daten der verknüpften Tabellenspalte übereinstimmt.
- **Operator-Widget:** Während beim Textbox-Widget nur auf Gleich-

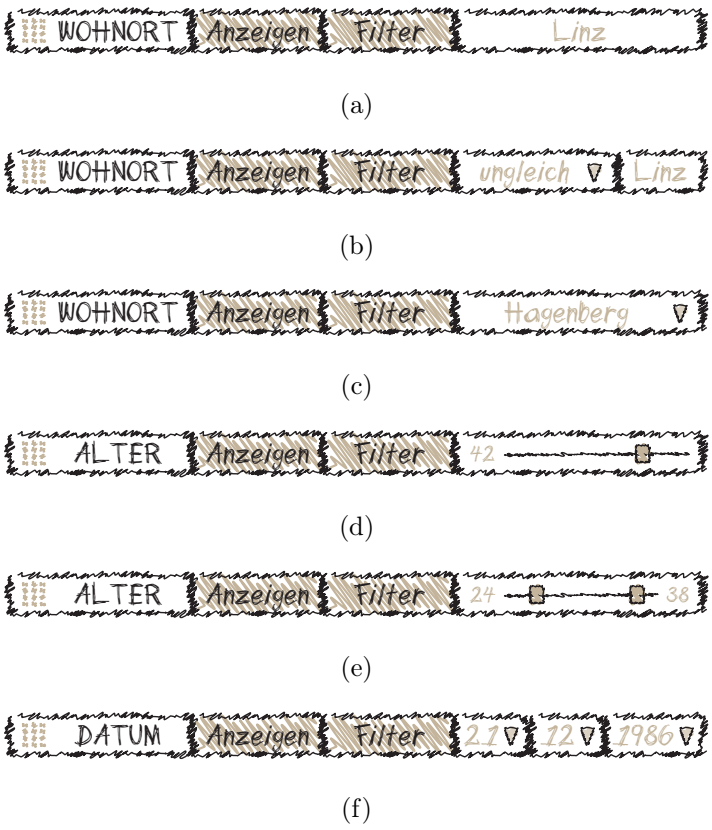

Abbildung 4.4: Unterschiedliche Widgettypen für den gQD. Textbox-Widget (a), Operator-Widget (b), Selectbox-Widget (c), Slider-Widget (d), Range-Widget (e), Date-Widget (f).

heit überprüft wird, muss beim Operator-Widget der Vergleichsoperator vom Benutzer aus einer Selectbox ausgewählt werden. Das Widget bietet neben der Textbox für die Eingabe des Vergleichswertes, eine Selectbox mit den möglichen Vergleichsoperatoren an. Da diese Operatoren bereits durch den gQD in natürlicher Sprache vordefiniert werden, wird keine Eingabe von datenbankspezifischen Schlüsselwörtern durch den Benutzer gefordert.

- **Selectbox-Widget:** Bei diesem Widget kann der Vergleichswert für das Kriterium nicht, wie beim Textbox-Widget, durch den Benutzer eingegeben, sondern aus einer Selectbox, welche bereits vordefinierte Werte beinhaltet, ausgewählt werden. Diese Werte können beim Definieren des Widgets in der gQD-Konfiguration entweder flexibel angegeben oder aus einer Tabelle aus der Datenbank gelesen werden.

  Zum Beispiel ist es durch diese Art von Widgets möglich, die Selectbox-

Werte von Lookup-Tabellen (Artikel zu Lookup-Tabellen unter [24]) aus der Datebank zu beziehen und für die Filterfunktion des Widgets als Vergleichswert zu verwenden.

- **Slider-Widget:** Dieser Widgettyp ist für Spalten gedacht, die einen nummerischen Datentyp besitzen. Dabei kann der Benutzer beim Definieren eines Filterkriteriums für die, mit dem Widget verknüpfte, Spalte den Vergleichswert über einen Slider auswählen, welcher einen bestimmten Wertebereich besitzt.

  Der Wertebereich muss beim Definieren des Widgets in der Konfiguration angegeben werden und lässt eine Einschränkung der möglichen Vergleichswerte für das Filterkriterium zu. Durch dieses Widget kann zum Beispiel vermieden werden, dass der Benutzer als Vergleichswerte für das Alter einer Person negative Zahlen auswählt bzw. eingibt.
- **Range-Widget:** Dieses Widget ist ähnlich zum Slider-Widget, jedoch kann beim Range-Widget ein Wertebereich durch zwei Slider ausgewählt werden. Liegen die Daten der verknüpften Tabellenspalte innerhalb dieses ausgewählten Wertebereichs, so gilt das Filterkriterium als erfüllt und die Daten der Spalte werden in den Ergebnisdaten berücksichtigt.
- **Date-Widget:** Die Eingabe eines Datums zum Vergleichen mit anderen Daten kann Probleme mit sich bringen, da es unterschiedliche Eingabeformate bei Datumsangaben gibt. Das Date-Widget bietet für die Angabe eines Datums drei Selectboxen für den Tag, das Monat und die Jahreszahl an. Das eingegebene Datum dient als Vergleichswert für das Filterkriterium und wird durch den gQD in das datenbankspezifische Datumsformat konvertiert.

Im Allgemeinen sollen die unterschiedlichen Widgettypen das Definieren von Filterkriterien für den Benutzer wesentlich erleichtern. Die Verwendung von Auswahlboxen für Werte und Operatoren, sowie Slidern mit Wertebereichen in den Widgets, erfordert keine direkte Eingabe durch den Benutzer. Dadurch sinkt die Fehleranfälligkeit bei der Eingabe und ein Einschränken des Benutzers beim Definieren von Kriterien ist möglich.

4.3.3 Container

Grundsätzlich kann im gQD pro Widget immer nur ein Filterkriterium definiert werden. Jedoch ist es bei Abfragen auf die Datenbank durchaus möglich, dass mehrere Widgets mit aktiver Filterfunktion vorkommen.

Durch Container wird dem Benutzer beim Erstellen der Abfrage die Möglichkeit geboten, die einzelnen Filterkriterien durch logische Verknüpfungen zu einem Gesamtkriterium zu verbinden, wobei das Container-Konzept auch eine beliebige Verschachtelung erlaubt.

Zusätzlich ermöglichen Container das Anwenden von einfachen Funktio-

Abbildung 4.5: Aufbau eines Containers.

nen auf die Daten in der Abfrage. Das Funktionsergebnis ist in der Ergebnistabelle der Abfrage vorhanden.

Aufbau

Im Gegensatz zu den Widgets sind Container wesentlich einfacher aufgebaut (siehe in Abbildung 4.5). Diese setzen sich aus nur zwei Elementen zusammen:

- **Bezeichner:** Container im gQD können unterschiedliche Funktionalitäten besitzen. Durch den Bezeichner des Containers, soll der Benutzer erkennen können, welche Auswirkung die Verwendung des Containers in der Abfrage hat. Beispielsweise kann der Bezeichner auf die verwendete Verknüpfungsart für Filterkriterien schließen lassen oder der Name einer Funktion sein, wobei die Namensgebung benutzerfreundlich und ohne datenbankspezifischen Schlüsselwörter erfolgt.

  Wie schon bei den Widgets lassen sich Container bei der Abfrageerstellung über die Bezeichner-Elemente per Drag & Drop verschieben.
- **Box:** Das Box-Element eines Containers kann im gQD bei der Erstellung der Abfrage Widgets enthalten, deren Filterkriterien verknüpft werden oder auf die eine Funktion ausgeführt wird. Je nach Verwendungsart des Containers kann die Box auch weitere Container beinhalten, um Widgets zu verschachteln.

  Die Größe der Box kann durch den Benutzer je nach Bedarf bei der Abfrageerstellung verändert werden.

Logische Verknüpfungen

Wird der Container für logische Verknüpfungen von Filterkriterien verwendet, so stehen dem Benutzer drei verschiedene Typen von Containern im gQD zur Verfügung:

- **UND:** Alle Filterkriterien der Widgets, die in der Box des UND-Containers enthalten sind, werden so verknüpft, dass das Gesamtkriterium der Abfrage nur dann erfüllt ist, wenn auch alle einzelnen Kriterien übereinstimmen. Enthält die Box weitere Container, so werden auch diese Verknüpfungen abhängig vom verwendeten Containertyp

durch den gQD aufgelöst und zum Gesamtkriterium hinzugefügt. Befinden sich mehrere Widgets mit Filterkriterien in der Querybox und es wird kein Container zur Verknüpfung verwendet, so werden diese automatisch durch den gQD mit UND-Verknüpfungen zu einem Gesamtkriterium verbunden.

- **ODER:** Im Gegensatz zum UND-Container, ist bei diesem Containertyp das Gesamtkriterium bereits dann erfüllt, wenn mindestens eines der einzelnen Kriterien übereinstimmt.
- **NICHT:** Der NICHT-Container funktioniert gegensätzlich zum UND-Container. Die Erfüllung des Gesamtkriteriums erfolgt nur dann, wenn kein einziges Filterkriterium der Widgets auf die Daten bei der Abfrage zutrifft.

Einfache Funktionen

SQL unterscheidet zwei verschiedene Arten von Funktionen. Während Skalarfunktionen auf einzelne Daten einer Spalte angewendet werden, lassen sich Aggregatfunktionen auf die gesamten Spaltendaten ausführen, wobei diese nur einen Wert als Ergebnis zurückgeben. Zusätzlich kann bei der Verwendung von Aggregatfunktionen noch eine Gruppierung hinzugefügt werden, um das Ergebnis der Funktion pro Teilmenge zu erhalten. Weiterführende Informationen zu Aggregatfunktionen unter [10, S. 118–119].

Die Funktionen im gQD werden auf alle, in der Box des Containers enthaltenen, Widgets ausgeführt, wobei abhängig von der verwendeten Funktion das Verschachteln von Containern erlaubt ist. Beispielsweise könnten folgende Skalarfunktionen durch Container im gQD auf Spalten von Tabellen angewendet werden:

- **Rechenfunktionen:** Diese Container werden hauptsächlich für Widgets benötigt, die auf Tabellenspalten mit nummerischen Datentypen verweisen. Der gQD stellt Container für die Addition und Subtraktion von Daten zweier Tabellenspalten zur Verfügung. Komplizierte Rechenausdrücke kann der Benutzer durch Verschachtelung dieser Container erstellen. Die Ergebnisse der Rechenausdrücke werden in einer Spalte in der Ergebnistabelle der Abfrage angezeigt.
- **Zeichenkettenfunktionen:** Der gQD bietet einen Container an, der die Konkatenation[1] von Zeichenketten aus unterschiedlichen Tabellenspalten ermöglicht. Die verknüpften Zeichenketten werden dann in einer einzigen Spalte in der Ergebnistabelle der Abfrage dargestellt. Zum Beispiel könnten dadurch die Daten der beiden Spalten *Vorname* und *Nachname* bei der Abfrage zu einer Spalte *Name* in der Ergebnistabelle zusammengeführt werden.

[1] Laut Duden bedeutet Konkatenation die Verkettung von Zeichen oder Zeichenketten.

Ein weiterer Container im gQD ermöglicht die Berechnung der Zeichenanzahl von Daten in der Tabellenspalte, die durch das Widget in der Box des Containers angegeben wird. Diese Funktion kann nicht auf mehrere Widgets ausgeführt werden.

Folgende Container könnten für die Verwendung von Aggregatfunktionen im gQD umgesetzt werden:

- **Anzahl:** Der Container ermöglicht alle vorhandenen Daten in der Tabellenspalte, die vom Widget repräsentiert wird, zu zählen. Beispielsweise könnte dieser Container die Personenanzahl, durch Zählen der Einträge in der Tabelle *Personen*, ermitteln und dem Ergebnis der Abfrage hinzufügen. Dabei darf nur ein Widget in der Box des Containers enthalten sein.
- **Minimum, Maximum und Durchschnitt:** Diese drei Container können nur für Widgets verwendet werden, dessen verknüpfte Tabellenspalten nummerische Datentypen besitzen. Das Abfrageergebnis liefert je nach verwendeten Container den minimalen, maximalen oder durschnittlichen Wert aus den Spaltendaten.
- **Summe:** Auch die Summe aus den Daten der Tabellenspalte kann durch einen Container im gQD berechnet werden, wobei für die Berechnung nummerische Datenwerte vorliegen müssen.

4.3.4 Konfiguration

Bevor der Benutzer durch den gQD Abfragen auf Datenbanken erstellen und ausführen kann, muss eine Konfiguration für den gQD vorliegen. Diese enthält u. a. die zu verwendende Datenbank sowie alle notwendigen Parameter, um die Verbindung durch den gQD zur Datenbank aufzubauen. Das Erstellen der Konfiguration sollte durch Personen mit datenbankspezifischen Kenntnissen erfolgen, wobei dies in Unternehmen durch Systemadministratoren durchgeführt werden könnte.

Wie schon bereits in Abschnitt 4.3.2 erwähnt, müssen für alle Tabellenspalten, die für die Abfragenerstellung im gQD zur Verfügung stehen sollen, Widgets in der Konfiguration definiert werden. Dabei gilt es, abhängig vom Datentyp der jeweiligen Tabellenspalte, den richtigen Widgettyp auszuwählen und die vom Typ abhängigen Einstellungen wie Wertebereiche oder Auswahlmöglichkeiten für Selectboxen zu definieren. Der Vorteil dieser Konfiguration besteht darin, dass dem Benutzer nur jene Widgets zur Verfügung gestellt werden können, die für ihn bei der Erstellung der Abfrage relevant sind. Somit ist der gQD für jeden Benutzer individuell anpassbar.

Außerdem kann eine logische Gruppierung für die Widgets in der Konfiguration definiert werden, welche eine übersichtliche Darstellung der Widgets in der Widgetbox des gQD-Interfaces zur Folge hat. Die Gruppierung muss nicht zwingend nach der Tabellenstruktur in der Datenbank erfolgen und

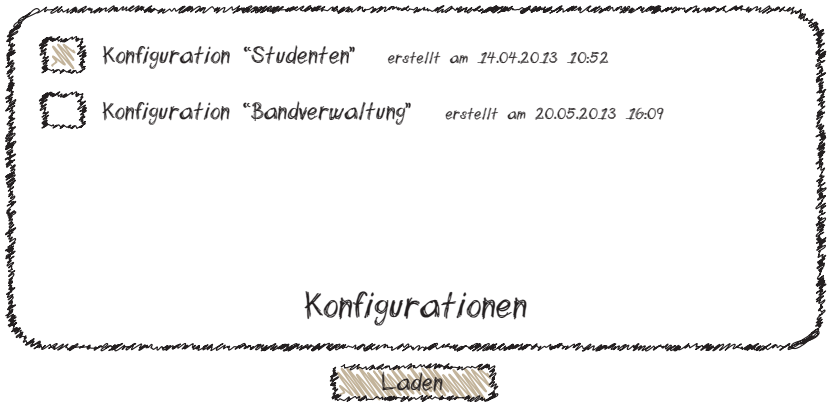

Abbildung 4.6: Interface für die Konfigurationsauswahl im gQD.

der Zusammenhang zwischen den unterschiedlichen Widgets kann für den Benutzer deutlich gemacht werden.

Auswahl der Konfiguration durch den Benutzer

Die vorhandenen Konfigurationen können durch den Benutzer ausgewählt und dynamisch geladen werden. Dadurch ist das einfache Austauschen der Datenbank für technisch unversierte Anwender möglich, wenn die entsprechende Konfiguration vorliegt.

Die Benutzeroberfläche des gQDs für die Auswahl der Konfiguration wird in Abbildung 4.6 skizziert dargestellt. Beim Starten der Anwendung werden alle verfügbaren Konfigurationen geladen und in der entsprechenden Box im Interface angezeigt. Der Benutzer muss die gewünschte Konfiguration über die Benutzeroberfläche auswählen. Durch Drücken des Buttons „Laden“ wird das Interface für die Erstellung von Abfragen auf Grund der ausgewählten Konfiguration aufgebaut und dem Benutzer angezeigt.

4.3.5 Erstellen der Abfrage

Wurde die Konfiguration durch den Benutzer ausgewählt und das Interface mit den verfügbaren Widgets und Containern geladen, so kann das Erstellen der Abfrage nach dem Baukastenprinzip im gQD erfolgen. Dabei bilden die Widgets und die Container die Einzelbausteine, die durch den Benutzer zu einer Abfrage in der Querybox kombiniert werden können.

Verwenden von Widgets in der Querybox

Wie schon bei der Auswahl der Konfiguration, wird auch beim Hinzufügen von Widgets zur Querybox das Bedienkonzept von Drag & Drop verwendet,

wobei die Mehrfachverwendung von gleichen Widgets bei der Abfrageerstellung durch den gQD erlaubt ist. Nach dem Hinzufügen des Widgets zur Querybox, ist der Button „Anzeigen“ standardmäßig aktiviert. Dies bedeutet, dass die Daten der Tabellenspalte, die das Widget repräsentiert, in der Ergebnistabelle der Abfrage angezeigt werden. Möchte der Benutzer die Filterfunktion des Widgets verwenden, so kann der Benutzer den Button „Filter“ aktivieren und die, vom Widgettyp abhängigen, Definitionen für das Kriterium vornehmen. Soll ein Widget nur zur Einschränkung der Daten in der Ergebnistabelle dienen, die verbundene Tabellenspalte jedoch nicht im Abfrageergebnis aufscheinen, so kann der Button „Anzeigen“ im Widget auch deaktiviert werden.

Das Löschen von Widgets aus der Querybox erfolgt durch Platzieren des zu löschenden Widgets über der Infobox. Dabei wird während dem Verschieben des Widgets innerhalb der Querybox, die Darstellung der Infobox verändert, um die Löschfunktion für den Benutzer über das Interface deutlich zu machen.

Verwenden von Containern in der Querybox

Das Platzieren von Containern in der Querybox erfolgt im gQD ebenfalls durch die Methode Drag & Drop. Die Box jedes Containers ist in ihrer Größe veränderbar und ermöglicht die Aufnahme von Widgets oder Containern, wobei das Hinzufügen dieser Elemente durch Platzieren über der Box des Containers funktioniert. Die Funktionalität des verwendeten Containers wird auf die Elemente der Box ausgeführt.

Container können analog zu den Widgets aus der Querybox gelöscht werden, wobei beim Entfernen eines Containers alle vorhandenen Elemente innerhalb des Containers mitgelöscht werden.

Beispielabfrage

Den Datenbestand für das Beispiel bildet eine Tabelle, die Einträge zu Personen mit dem Namen, dem Alter sowie dem Wohnort verwaltet. Die ausgewählte Beispielabfrage soll alle Personen mit dem Namen und dem Wohnort als Ergebnis anzeigen, die genau 20 Jahre alt sind und nicht in Linz wohnen.

Für die Formulierung dieser Beispielabfrage durch den Benutzer im gQD sind folgende Schritte notwendig:

1. Das NAME-Widget, welches auf die Tabellenspalte *Name* verweist, muss der Querybox hinzugefügt werden. Der Button „Anzeigen“ wird automatisch durch den gQD aktiviert, somit wird die Spalte *Name* in der Ergebnistabelle angezeigt.
2. Um die beiden Kriterien für das Alter und den Wohnort zu verknüpfen, wird ein UND-Container in der Querybox benötigt. Die Boxgröße des Containers kann bei Bedarf verändert werden.

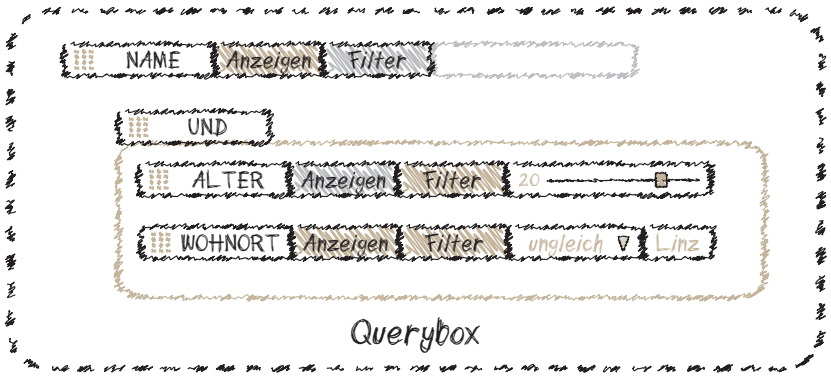

Abbildung 4.7: Skizzierte Querybox für Beispielabfrage im gQD.

3. Der Box wird das ALTER-Widget hinzugefügt. Da die Ergebnistabelle der Abfrage die Spalte *Alter* nicht beinhalten soll, ist das Deaktivieren des Buttons „Anzeigen“ im Widget erforderlich.
4. Die Filterfunktion des ALTER-Widgets muss aktiviert werden und anschließend ist der Wert „20“ beim Slider einzustellen.
5. Das WOHNORT-Widget wird zum UND-Container hinzugefügt. Das Anzeigen der Spalte *Wohnort* im Abfrageergebnis wurde bereits automatisch durch den gQD aktiviert und erfordert daher keine zusätzliche Interaktion durch den Benutzer.
6. Der Button „Filter“ im WOHNORT-Widget muss aktiviert werden. Somit kann in der Selectbox der Vergleichsoperator „ungleich“ ausgewählt und als Vergleichswert der Ort „Linz“ eingegeben werden.

Die Querybox des gQDs, nach der Ausführung aller zuvor beschriebenen Schritte, wird in Abbildung 4.7 skizziert dargestellt.

Feedback und Hilfe

Die Infobox im Interface des gQDs ermöglicht die Kommunikation zwischen dem gQD und dem Benutzer in natürlicher Sprache. Das Konzept sieht vor, den Anwender mit Feedback und weiterführenden Informationen zu versorgen und ihn wesentlich bei der Abfrageerstellung zu unterstützen.

Nach jeder, vom Benutzer durchgeführten, Aktualisierung in der Querybox, wird die Abfrage vom gQD neu interpretiert und in natürlicher Sprache in der Infobox dargestellt. Beispielsweise können alle Filterkriterien, welche in der Abfrage vorkommen, übersichtlich in der Infobox dargestellt werden. Durch dieses Feedback lässt sich einfacher und schneller feststellen, ob die formulierte Abfrage in der Querybox das gewünschte Ergebnis liefert. Mögliche Fehler können schneller erkannt und beseitigt werden.

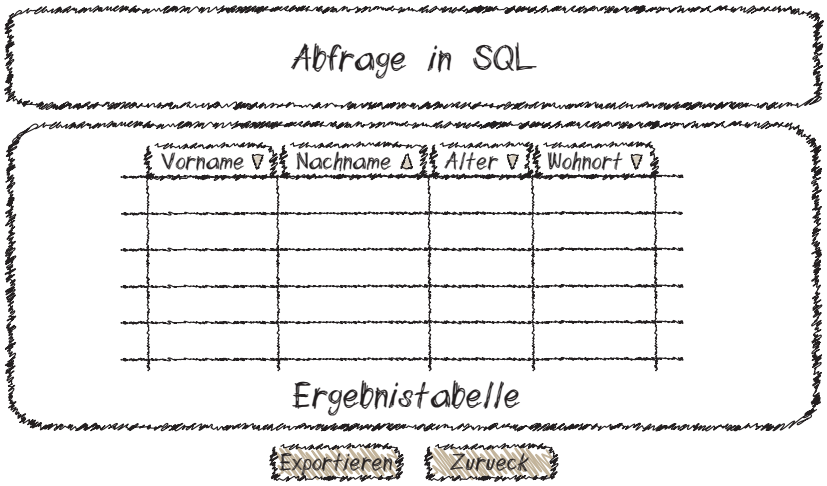

Abbildung 4.8: Interface für das Ausführen der Abfrage im gQD.

Während mit den Widgets und Containern interagiert wird, zeigt der gQD Tipps und Informationen zum verwendeten Element in der Infobox an. Zum Beispiel kann dem Benutzer während dem Verschieben eines Containers von der Containerbox in die Querybox, die Interaktion zum Vergrößern der Box des Containers erklärt werden. Jedoch besteht für den Benutzer die Möglichkeit, diese zusätzlichen Erklärungen im gQD zu deaktivieren, da die Verwendung der Widgets und Container nach einer bestimmten Einarbeitungszeit ohnehin klar sein sollte.

Der gQD bietet zusätzlich Erklärungen zu den Widgets und Containern im Interface an, wobei nur jene Elemente erklärt werden, die dem Benutzer in der verwendeten Konfiguration zur Verfügung stehen. Somit wird das Interface für Hilfestellungen zu den Elementen dynamisch auf Grund der Konfiguration des gQDs aufgebaut.

Speichern von Abfragen

Die in der Querybox erstellte Abfrage kann durch den gQD persistent gespeichert werden, wobei der Benutzer einen Speichernamen für die Abfrage über das Interface vergeben muss. Die gespeicherten Abfragen können nur dann im gQD wieder geladen werden, wenn der gQD die selbe Konfiguration, die beim Speichern der Abfrage verwendet wurde, benutzt.

Das Laden der Abfragen erfolgt analog zum Laden von Konfigurationen über das entsprechende Interface im gQD. Dabei werden alle, für die verwendete Konfiguration verfügbaren, Abfragen angezeigt und die zu ladende Abfrage kann durch den Benutzer ausgewählt werden.

4.3.6 Ausführen der Abfrage

Wurde die Abfrage in der Querybox erstellt, so kann die Abfrage auf die Datenbank erfolgen. Anschließend stellt der gQD das Ergebnis der Datenabfrage in einer Ergebnistabelle im Interface dar. Der Aufbau dieses Interfaces wird als Skizze in Abbildung 4.8 dargestellt.

Dem Benutzer des gQDs wird die Möglichkeit geboten, die Daten in der Ergebnistabelle aufsteigend bzw. absteigend zu sortieren. Diese Sortierung erfolgt durch Klicken auf die entsprechende Spalte in der Tabelle. Zusätzlich zur Ergebnistabelle wird in diesem Interface die Abfrage in SQL-Syntax angezeigt. Dadurch kann der Benutzer mit der Datenbanksprache vertraut gemacht werden.

Durch den Button „Exportieren“ im Interface, kann der Benutzer die angezeigten Ergebnisdaten entweder in bereits aufbereiteter Form, beispielsweise als PDF-Dokument, oder in Form von Rohdaten für die Weiterverwendung in anderen Anwendungen exportieren lassen. Beispielsweise könnten die Ergebnisdaten durch den gQD in XML abgebildet werden und als XML-Dokument für den Export bereitstehen.

Kapitel 5

Umsetzung des Prototyps

Basierend auf dem vorgestellten Konzept wurde im Zuge dieser Arbeit ein Prototyp entwickelt, der das Erstellen von Abfragen über das Interface sowie das anschließende Ausführen der Abfrage auf die Datenbank ermöglicht.

Der Prototyp soll in weiterer Folge die Grundlage für die Durchführung einer Evaluierung darstellen. Dabei gilt herauszufinden, ob das umgesetzte Konzept des gQDs die Bedienung durch technisch unversierte Benutzer ermöglicht und wo eventuelle Probleme bei der Verwendung auftreten können.

Dieses Kapitel beschreibt anfänglich den Aufbau des Prototyps. Dabei wird auf den technischen Ablauf bei der Benützung des Prototyps aus der Sicht des Client-Server-Modells sowie auf die Architektur eingegangen. Anschließend werden die Kernpunkte bei der Implementierung des gQD-Prototyps näher betrachtet. Diese sollen Aufschluss darüber geben, wie der gQD umgesetzt wurde.

5.1 Verwendete Webtechnologien

Da es sich bei dem entwickelten gQD lediglich um einen Prototyp handelt, wurden bei der Auswahl der Technologien für die webbasierte Anwendung keine speziellen Evaluierungen vorgenommen.

Serverseitig kommt bei dem Prototyp die plattformunabhängige Scriptsprache PHP[1] zum Einsatz. Die Scriptsprache ermöglicht die objektorientierte Programmierung auf der Serverseite und bietet u. a. Funktionen für die Ausführung der Abfrage auf die Datenbank sowie für das Einlesen der gQD-Konfiguration im XML-Format an.

Das Konzept des Prototypen sieht eine desktopartige Bedienung durch den Benutzer vor. Um dies zu ermöglichen, wird JavaScript auf der Clientseite verwendet. Auf Grund ihrer einfachen Anwendung werden die beiden

[1] PHP: http://php.net/

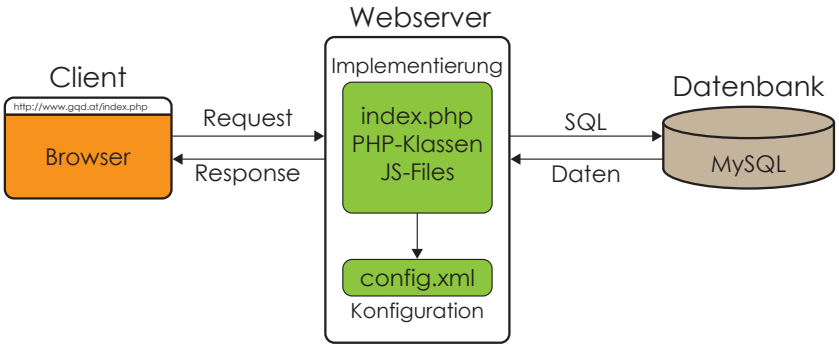

Abbildung 5.1: Prototyp im Client-Server-Modell.

Frameworks *jQuery*[2] und *jQuery UI*[3] eingesetzt. *jQuery* stellt u. a. Funktionen zur DOM-Manipulation und zum Reagieren auf Benutzerinteraktionen (Events) zur Verfügung. Weiters unterstützt *jQuery* das Konzept von Ajax (Erklärung in [25]) und enthält Funktionen für das Absetzen von Ajax-Requests an den Server. *jQuery UI* bietet zusätzliche Lösungen für Bedienelemente, wie beispielsweise einen Slider, oder Bedienkonzepte, wie z. B. die Verwendung von Drag & Drop, an.

Der Prototyp unterstützt die Abfragenerstellung für Datenbanken des Datenbankmanagementsystems MySQL[4].

5.2 Aufbau

Der Prototyp des gQDs wurde als webbasierte Anwendung entwickelt, welche nach dem Client-Server-Modell aufgebaut ist. Die gesamte Implementierung des gQDs liegt dabei zentral am Webserver und beinhaltet alle objektorientierten PHP-Klassen, den benötigten JavaScript-Code sowie das PHP-File *index.php*, welches den Einstiegspunkt für den Benutzer bildet.

Zusätzlich werden auch die Konfigurationsfiles für den gQD am Server gespeichert. Der Zugriff auf den Prototyp erfolgt plattformunabhängig über das Internet oder Intranet mit einem Browser, der am Rechner des Clients installiert ist.

In der Abbildung 5.1 wird der Prototyp im Client-Server-Modell skizziert, wobei sich die dargestellte Datenbank entweder am Webserver oder auf einem externen Datenbankserver befinden kann.

[2] jQuery: http://jquery.com/
[3] jQuery UI: http://jqueryui.com/
[4] MySQL: http://www.mysql.com/

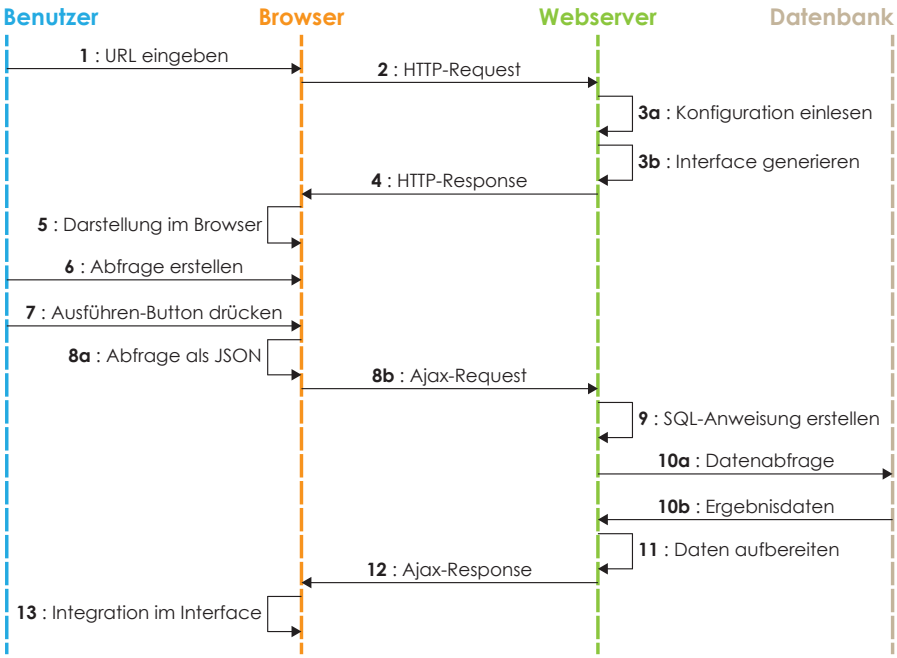

Abbildung 5.2: Sequenzdiagramm für die Ablaufschritte im Client-Server-Modell.

5.2.1 Ablauf im Client-Server-Modell

Die Funktionsweise des Prototyps, bei der Erstellung von Abfragen durch den Benutzer, kann durch folgende Ablaufschritte beschrieben werden:

1. Der Benutzer (Client) greift auf den gQD zu, indem er die URL des Webservers im Browser eingibt. Als Einstiegspunkt dient die Seite *index.php*, welche sich auf dem Webserver befindet.
2. Dadurch wird eine Anfrage über das Übertragungsprotokoll HTTP (HTTP-Request) an den Webserver gesendet.
3. Der Webserver nimmt die Anfrage des Clients zur Verarbeitung entgegen. Da die Konfiguration des gQDs den Aufbau des Interfaces mitbestimmt, muss das entsprechende XML-File eingelesen und interpretiert werden. Anschließend erfolgt die Generierung des HTML- und JavaScript-Codes für das Interface, welches am Client dargestellt werden soll.
4. In einem HTTP-Response wird der dynamisch erstellte Code an den Client zurückgesendet.
5. Die Darstellung des Interfaces erfolgt im Browser am Client. Dieses bietet dem Benutzer weitere Interaktionsmöglichkeiten an.

6. Die gewünschte Abfrage kann über das, im Browser angezeigte, Interface erstellt werden, wobei während der Erstellung keine Kommunikation mit dem Server stattfindet, da diese durch das clientseitige JavaScript ermöglicht wird.
7. Möchte der Benutzer die, im Prototyp formulierte, Abfrage auf die Datenbank ausführen, so drückt dieser den entsprechenden Button.
8. Die Abfrage wird am Client interpretiert und im JSON-Format für den Datenaustausch abgebildet. Ein Ajax-Request an den Webserver erfolgt und die Abfrage wird im JSON-Format als Parameter beigefügt.
9. Die Logik des Prototyps am Server interpretiert die durch den Request erhaltene Abfrage im JSON-Format und erzeugt die SQL-Anweisung für die Datenbankabfrage.
10. Der Webserver führt die generierte SQL-Abfrage auf die Datenbank aus und die Ergebnisdaten werden von der Datenbank zurückgegeben.
11. Da es sich bei den Ergebnisdaten um Rohdaten aus der Datenbank handelt, werden diese am Server in HTML-Code integriert und so für die Darstellung im Browser des Benutzers aufbereitet.
12. Der Webserver sendet den HTML-Code als Ajax-Response an den Client zurück.
13. Die Ergebnisdaten werden durch JavaScript in das bereits dargestellte Interface im Browser eingefügt. Durch die Verwendung des Ajax-Konzepts ist kein Neuladen der Seite erforderlich und die erstellte Abfrage im Interface bleibt für weitere Veränderungen oder ein neuerliches Ausführen auf die Datenbank bestehen.

In der Abbildung 5.2 werden die beschriebenen Ablaufschritte in einem Sequenzdiagramm zusammengefasst.

Beim verwendeten Client-Server-Modell macht es für den Benutzer keinen Unterschied, ob die beiden serverseitigen Komponenten, der Webserver und die Datenbank, über das Internet oder innerhalb eines Intranets erreichbar sind. Der Zugriff auf die Anwendung über den Browser kann durch mehrere Benutzer gleichzeitig erfolgen. Der Webserver arbeitet die jeweiligen Anfragen der Clients sequenziell ab.

5.2.2 Clientseitige Architektur

Bei einer klassischen Webanwendung befindet sich die gesamte Applikationslogik am Server. Der Browser dient nur zur Darstellung der Daten und um Benutzerinteraktionen mit dem Server zu ermöglichen. Die Architektur des gQD-Prototyps weicht von der klassischen Architektur ab, da neben der Logik am Server auch clientseitige Logik zum Einsatz kommt. Diese ermöglicht das Formulieren von Abfragen in der Querybox über Widgets und Container, wobei Container wiederum Widgets und weitere Container beinhalten können. Für diesen Ansatz bietet sich bei der Umsetzung der clientseitigen Logik

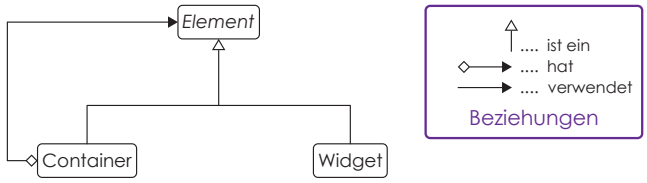

Abbildung 5.3: Klassendiagramm unter Anwendung des *Composite Patterns* für die clientseitige Architektur.

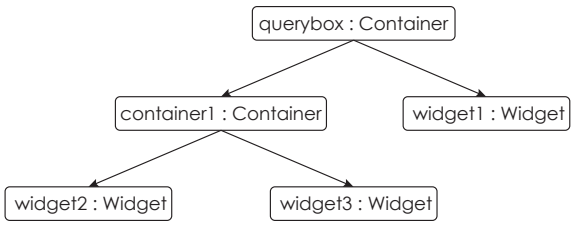

Abbildung 5.4: Objektdiagramm für die Elemente einer erstellten Abfrage im Prototyp.

das Prinzip des *Composite Patterns* an, wodurch das in Abbildung 5.3 dargestellte Klassendiagramm für die clientseitige Architektur zustande kommt. Die Klasse `Element` bildet die Basisklasse für `Container` und `Widget`. Weiters kann das `Container`-Objekt beliebig viele Instanzen vom Typ `Element` beinhalten. Allgemeine Erklärungen zum Pattern sind in [4] zu finden.

Bei der Erstellung der Abfrage durch den Benutzer entsteht durch die Möglichkeit die Elemente zu verschachteln eine baumartige Struktur, in der die Querybox den Root-Knoten des Baumes darstellt (siehe Objektdiagramm in Abbildung 5.4).

Die im Klassendiagramm dargestellten Klassen werden zwar nicht objektorientiert durch JavaScript im Prototyp implementiert, jedoch wird das Prinzip des *Composite Patterns* durch die Kombination aus CSS und JavaScript umgesetzt.

5.2.3 Serverseitige Architektur

Serverseitig wurde der gQD-Prototyp im objektorientierten Stil entworfen und beinhaltet sämtliche Klassen, welche Abhängigkeiten untereinander aufweisen. Durch diese Klassen wird die eingelesene Konfiguration abgebildet und die erstellte Abfrage auf die Datenbank ausgeführt. Abbildung 5.5 stellt das zugrunde liegende Klassendiagramm für den Prototyp auf der Server-

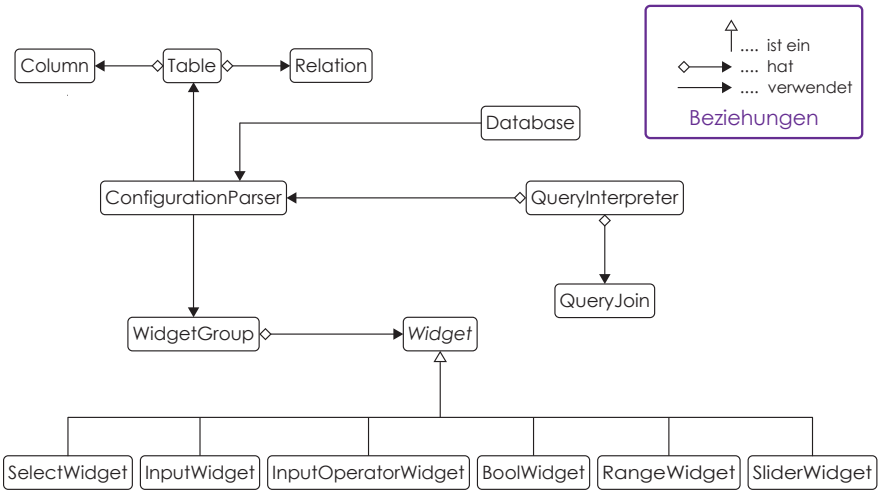

Abbildung 5.5: Serverseitige Architektur des Prototyps mit den verwendeten Klassen und deren Abhängigkeiten.

seite skizziert dar. Die Aufgaben der darin enthaltenen Klassen werden im folgenden Abschnitt beschrieben:

Widget-Klassen

Diese Klassen bilden die Widgets ab, die in der Konfiguration des gQDs definiert sind. Dabei stellt die abstrakte Klasse `Widget` die Basisklasse für alle Widgets dar. Sie enthält gemeinsame Eigenschaften und schreibt zu implementierende Funktionen für alle Unterklassen vor.

Für jeden Widgettyp, der im gQD vorhanden ist, wird eine Klasse abgeleitet. Diese wird, auf Grund der unterschiedlichen Filtermöglichkeiten in den Widgets, um typspezifische Eigenschaften erweitert. Jeder Widget-Klasse wird durch die Basisklasse vorgeschrieben, Funktionen zu implementieren, die auf Grund der Eigenschaften den HTML-Code für die Darstellung und den JavaScript-Code für die Funktionalität des Widgets generieren.

ConfigurationParser

In der Klasse `ConfigurationParser` wird die Konfiguration als XML-Dokument, welches sich am Webserver befindet, eingelesen und interpretiert. Das Schema der Tabellen und die Beziehungen zueinander erhält der Parser aus der angegebenen Konfiguration. Anschließend erfolgt die Abbildung der definierten Tabellen durch Instanzen der Klasse `Table`, welche selbst durch Objekte der Klassen `Column` und `Relation` beschrieben werden.

Die `ConfigurationParser`-Klasse erzeugt für jedes in der Konfiguration definierte Widget eine Instanz vom entsprechenden Widgettyp. Die angegebenen Daten, wie beispielsweise der Name oder der Wertebereich des Widgets, werden im instanzierten Objekt gespeichert. Weist die Konfiguration Gruppierungen von Widgets für die Darstellung im Interface auf, so können diese durch die Klasse `WidgetGroup` abgebildet werden. Diese Klasse fasst mehrere Instanzen von Widgets zu einer Gruppe zusammen.

Weiters bietet der Parser Funktionen an, um die Parameter für die Herstellung der Verbindung zur Datenbank aus der Konfiguration zu lesen.

QueryInterpreter

Bei der Ausführung einer erstellten Abfrage auf die Datenbank erhält der Server die Abfrage vom Client im Datenaustauschformat JSON und gibt diese dem Objekt vom Typ `QueryInterpreter` bei der Instanzierung als Parameter mit. Die Klasse interpretiert die Abfrage im JSON-Format und generiert daraus die SQL-Anweisung oder den HTML-Code für die Darstellung der Abfrage in natürlicher Sprache.

Die Hilfsklasse `QueryJoin` repräsentiert Verknüpfungen zwischen Tabellen, die beim Generieren der SQL-Anweisung berücksichtigt werden müssen.

Database

Das Datenbank-Objekt der Klasse `Database` erhält die Parameter aus der Konfiguration über den `ConfigurationParser` und baut dadurch die Verbindung zur angegebenen Datenbank auf. In weiterer Folge ermöglicht die erzeugte Instanz der `Database`-Klasse, vom `QueryInterpreter` generierte SQL-Anweisungen auf die Datenbank auszuführen und die Ergebnisdaten zurückzuliefern.

5.3 Implementierung

Dieser Abschnitt soll Aufschluss darüber geben, wie die Kernpunkte des Prototyps umgesetzt wurden.

5.3.1 Mapping des Datenbankschemas

Wie schon bereits erwähnt, erfolgt die Konfiguration des gQDs über ein datenzentriertes XML-Dokument (mehr dazu auch in Abschnitt 2.2.1), welches am Webserver gespeichert ist. Die Datenbankparameter werden innerhalb der XML-Elemente `<Host>`, `<Name>`, `<User>` und `<Password>` angegeben und werden für den Verbindungsaufbau zur Datenbank benötigt.

Das Schema aller Tabellen in der ausgewählten Datenbank muss beim Erstellen der Konfiguration für den gQD definiert werden. Tabellen, die für

die Abfrage durch den Benutzer nicht zur Verfügung stehen sollen, benötigen keine Schemaangabe in der Konfiguration. Außerdem gilt es, die Beziehungen zwischen den verfügbaren Tabellen anzugeben, um diese beim Generieren der SQL-Anweisung aus der erstellten Abfrage zu berücksichtigen.

Die Angabe aller Schemata für die Tabellen erfolgt innerhalb des Tags `<Tables>` im XML-Dokument. Zum Beispiel wird das Schema für die Tabelle *Band* wie folgt in der Konfiguration definiert:

```
<Table name="Band">
  <Columns>
    <Column name="Id" key="PRIMARY" type="INT"/>
    <Column name="Name" type="VARCHAR"/>
    <Column name="Gründungsjahr" type="VARCHAR"/>
    <Column name="Bandmitglieder" type="INT"/>
  </Columns>
  <Relations>
    <Relation column="Id" reltable="Album" relcolumn="BandId"/>
  </Relations>
</Table>
```

Der Name der Tabelle wird über das Attribut `name` im XML-Element `<Table>` definiert und die Angabe der Tabellenspalten erfolgt über das XML-Element `<Column>`. Dieses Element gibt durch Hinzufügen von Attributen den Namen und den Datentyp der Spalte an. Tritt die Spalte als Primärschlüssel auf, so kann dies durch ein zusätzliches Attribut `key` in der Konfiguration festgelegt werden.

Weiters erfordert die Implementierung die Angabe von Relationen zu anderen Tabellen. Dabei muss die Definition, der in Beziehung stehenden, Tabelle (Attribut `reltable`) sowie der beiden Spalten, die für die Tabellenverknüpfung verwendet werden sollen (Attribut `column` und `relcolumn`), über das XML-Element `<Relation>` erfolgen.

Wie schon oberhalb erwähnt, wird das Datenbankschema nach dem Parsen der Konfiguration durch Klassen abgebildet und beim Generieren der SQL-Anweisung verwendet.

Datenbankschema aus Metadaten

Das Mapping des Datenbankschemas über die Konfiguration ist eine Implementierungsmöglichkeit, die zwar für den Prototyp ausreichend ist jedoch müssen Veränderungen an der Datenbankstruktur auch ständig in der Konfiguration aktualisiert werden.

Das MySQL-Datenbankmanagementsystem stellt alle Schemainformationen zu den vorhandenen Datenbanken in einer gemeinsamen Systemdatenbank *INFORMATION_SCHEMA* zur Verfügung. Daher könnte der gQD die benötigten Schemata zu den Tabellen direkt aus der Systemdatenbank abfragen, interpretieren und für die Verwendung in der Anwendung durch Klassen abbilden.

5.3.2 Widgets

Jedes Widgets repräsentiert eine Spalte in einer Tabelle. Die Konfiguration ermöglicht das Definieren von Widgets, die in der Widgetbox im gQD-Interface angezeigt werden sollen. Diese Widgets kann der Benutzer für die Erstellung der Abfrage verwenden.

Konfiguration

Die Implementierung sieht vor, dass alle Widgets innerhalb des Elements `<Widgets>` im XML-Dokument angegeben werden. Zusätzlich können die Widgets durch das `<WidgetGroup>`-Tag beliebig in Gruppen unterteilt werden. Das Attribut `title` gibt den Gruppennamen an. Widgetgruppen bekommen für die Darstellung im Interface durch die Anwendung eine Farbe zugewiesen und die zugehörigen Widgets werden ebenfalls in der jeweiligen Gruppenfarbe gekennzeichnet.

Die Auswahl des Widgettyps beim Definieren von Widgets in der Konfiguration erfolgt durch die Verwendung des jeweiligen XML-Elements. Zum Beispiel sieht die Definition eines Range-Widgets im XML-Dokument folgendermaßen aus:

```
1 <RangeWidget id="gruendungsjahr" title="GRÜNDUNGSJAHR">
2   <DataMapping table="Band" column="Gründungsjahr"/>
3   <RangeData fromValue="1950" toValue="2013" slideStep="1"/>
4 </RangeWidget>
```

Das `<RangeWidget>`-Element benötigt das Attribut `id` für die eindeutige Erkennung sowie das Attribut `title`, das den Widgetnamen für die Darstellung im gQD-Interface definiert. Durch das Tag `<DataMapping>` wird über die beiden Attribute die Tabellenspalte angegeben, die das Widget repräsentiert.

Je nach verwendetem Widgettyp erfordert die Implementierung noch weitere Einstellungen für den Slider, der das Definieren von Filterkriterien ermöglicht. Daher wird im `<RangeData>`-Element der Wertebereich (Attribut `fromValue` und `toValue`) sowie die Schrittweite (Attribut `slideStep`) des Sliders konfiguriert.

Verwenden Widgettypen für das Definieren von Filterkriterien Auswahlboxen, so können die Auswahloptionen entweder direkt in der Konfiguration angegeben oder aus einer Tabellenspalte ausgelesen werden. Die Spalte für die Optionen wird über ein entsprechendes XML-Element in der Konfiguration definiert.

Darstellung

Der Parser (Klasse `ConfigurationParser`) liest das XML-Dokument ein und bildet die definierte Widgetstruktur in Klassen ab. Wie bereits erwähnt, verfügen alle Klassen, die von der abstrakten Basisklasse `Widget` ableiten über

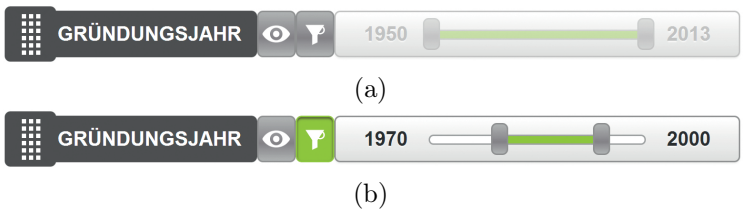

Abbildung 5.6: Darstellung eines Range-Widgets im Interface des gQDs. Das Range-Widget in (a) ist deaktiviert, während beim Range-Widget in (b) das Filterkriterium aktiv ist und der Slider durch den Benutzer verwendet werden kann.

die zwei Funktionen `generateHTML()` und `generateScript()`. Während die erstgenannte Funktion den HTML-Code aus den konfigurierten Widgetdaten für die Darstellung im Interface generiert, liefert die zweite Funktion den JavaScript-Code, der für das Widget benötigt wird.

Wie nachfolgender generierter HTML-Code für das Range-Widget aus der Beispielkonfiguration oberhalb zeigt, setzen sich Widgets aus den HTML-Elementen `<div>` und `<button>` zusammen. Anhand der CSS-Klassen, die den unterschiedlichen HTML-Elementen zugeteilt sind, lassen sich Widgets später clientseitig interpretieren.

```
1 <div id="widgetgruendungsjahr" title="gruendungsjahr" class="widget
      RangeWidget widgetDraggable">
2   <div class="dragWidget dragHandle"></div>
3   <div class="labelWidget dragHandle">GRÜNDUNGSJAHR</div>
4   <button class="buttonView buttonInactive" title="Anzeigen" />
5   <button class="buttonFilter buttonInactive" title="Filterkriterium" />
6   <div class="rangeFieldWidget">
7     <div class="sliderValueWidget">1950</div>
8     <!-- Element für den Slider !-->
9     <div id="slidergruendungsjahr" class="sliderWidget"></div>
10    <div class="sliderValueWidget">2013</div>
11  </div>
12 </div>
```

Den Slider für das Range-Widget erzeugt der generierte JavaScript-Code. Dabei wird die Funktion `slider()` auf ein bestimmtes `<div>`-Element im Widget ausgeführt.

Für den Aufbau des Interfaces werden alle abgebildeten Widgets durchlaufen und die bereitgestellten Funktionen zum Generieren des Codes aufgerufen. Der gesamte Code wird an den Client gesendet und dort angezeigt. Die Darstellung des Range-Widgets ist in Abbildung 5.6 ersichtlich. Ist der Button für das Verwenden der Filterfunktion im Range-Widget aktiviert, so können die Sliderwerte durch den Benutzer verändert werden.

Abbildung 5.7: Darstellung eines ODER-Containers im Interface des gQDs.

5.3.3 Container

Im Gegensatz zu den Widgets, sind die für die Abfrageerstellung verfügbaren Container im Prototyp fix definiert und müssen nicht konfiguriert werden. Dies hat zur Folge, dass die serverseitige Abbildung von Containern in Klassen nicht erforderlich ist. Jedoch könnte die Umsetzung auch so erfolgen, dass die Container konfigurierbar sind und so der Funktionsumfang des gQDs für den Benutzer individuell angepasst werden kann.

Container sind im Prototyp so implementiert, dass sie Widgets und weitere Container in ihrer Box beinhalten können. Neben dem logischen Verknüpfen von Filterkriterien der Widgets können durch Container auch Funktionen auf Widgets ausgeführt werden. Der statische HTML-Code eines ODER-Containers sieht wie folgt aus:

```
1 <div id="containerOR" class="container ContainerOr containerDraggable
      draggable">
2   <div class="dragContainer dragHandle"></div>
3   <div class="labelContainer dragHandle">ODER</div>
4   <!-- Containerbox für Elemente !-->
5   <div id="contentContainerOR" class="contentContainer"></div>
6 </div>
```

Ähnlich zu den Widgets werden auch Container durch `<div>`-Elemente erzeugt. Durch die Verwendung von CSS-Klassen lässt sich der Typ des Containers bei der Interpretation der Abfrage feststellen. Auf die Box des Containers wird die Eigenschaft `resizable()` aus *jquery UI* angewendet und ermöglicht das Verändern der Boxgröße durch Maus-Interaktionen. Die Darstellung des ODER-Containers im Interface des Prototyps ist in Abbildung 5.7 ersichtlich.

5.3.4 Erstellen der Abfrage

Das Interface des Prototyps zum Erstellen der Abfrage ist in Abbildung 5.8 ersichtlich. Im Prototyp werden Abfragen durch Verschieben von Elementen in die Querybox formuliert. Durch die Verknüpfung von JavaScript-Funktionalität mit CSS-Klassen, können die Funktionalitäten oder Eigenschaften eines HTML-Elements über die Vergabe von CSS-Klassen geregelt werden.

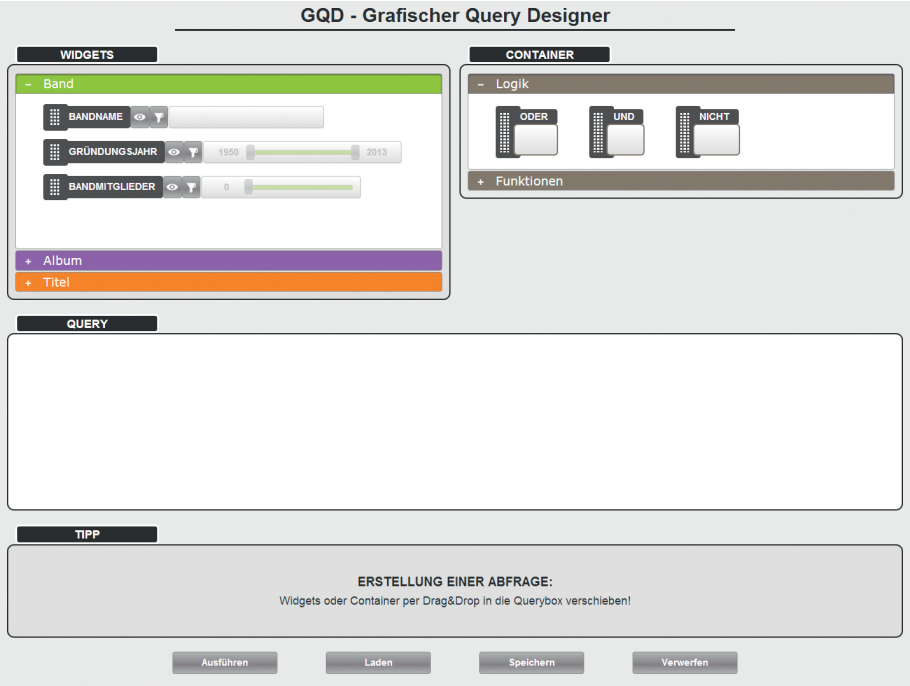

Abbildung 5.8: Interface für das Erstellen der Abfrage im Prototyp.

Draggable und Droppable

Alle verfügbaren Widgets und Container erhalten bereits bei der Erstellung die CSS-Klassen `widgetDraggable` bzw. `containerDraggable`. Die Zuweisung der Eigenschaft `draggable()` erfolgt über die beiden CSS-Klassen. Dadurch werden Widgets und Container im Interface verschiebbar gemacht.

Nach dem gleichen Prinzip wurden auch die Boxen für die Aufnahme von Elementen implementiert. Die CSS-Klasse `queryDroppable` wird sowohl der Querybox als auch den Boxen der unterschiedlichen Container hinzugefügt. Die Eigenschaft `droppable()`, welche mit der CSS-Klasse verknüpft wird, ermöglicht die Platzierung von Widgets oder Containern in der Box. Das Prinzip des vorher beschriebenen *Composite Patterns* wird durch diese Implementierung umgesetzt, da alle Boxen mit der CSS-Klasse `queryDroppable` Widgets oder Container aufnehmen können und so die Erstellung von hierarchischen Strukturen ermöglicht wird.

Die beiden verwendeten Eigenschaften `draggable()` und `droppable()` aus dem *jquery UI*-Framework bieten zusätzliche Events an. Dadurch kann beispielsweise während dem Verschieben eines Elements auf die Interaktion reagiert werden.

Hinzufügen von Elementen zur Box

Gleiche Widgets oder Container können in der erstellten Abfrage mehrfach vorkommen. Aus diesem Grund wird das Element beim Hinzufügen zur Abfrage durch die Funktion `clone()` dupliziert und über die `append()`-Funktion zum jeweiligen HTML-Element der Box hinzugefügt. Somit bleibt das Element auch in der Widgetbox bzw. Querybox für die erneute Verwendung in der Abfrage bestehen.

Beim Verschieben eines Containers innerhalb der formulierten Abfrage von einer Box in eine andere Box wird der Container der neuen Box hinzugefügt. Alle Elemente, die der Container in seiner Box beinhaltet, werden mitverschoben.

Platziert der Benutzer Widgets oder Container in nicht zulässigen Bereichen des Interfaces, so hat dies ein Zurücksetzen des Elements an die Ausgangsposition zur Folge. Das Löschen eines Elements aus der Abfrage wird durch Platzieren über der Infobox vollzogen. Dabei bekommt das HTML-Element der Infobox während dem Verschieben des Elements eine CSS-Klasse zugewiesen, die zum Aufnehmen von Elementen befugt. Nach Platzieren des Widgets oder Containers über der Infobox, wird das Element durch die `remove()`-Funktion gelöscht.

5.3.5 Interpretation der Abfrage am Client

Die durch den Benutzer erstellte Abfrage in der Querybox wird clientseitig durch die JavaScript-Funktion `buildJsonOfQuery()` in einen JSON-String konvertiert. Die Interpretation der einzelnen Elemente in der Abfrage erfolgt über die zugewiesenen CSS-Klassen. Da durch die Verwendung von Containern in der Abfrage eine hierarchische Struktur entsteht, bot sich eine rekursive Implementierung der Funktion an. Das Grundgerüst der Funktion sieht wie folgt aus:

```
1 function buildJsonOfQuery(box) {
2   $(root).children().filter(".draggable").each(function () {
3     if ($(this).hasClass("widget") && (isViewEnabled(this) ||
      isFilterEnabled(this))) {
4       /* Widget im JSON-Format zum String hinzufügen */
5     } else if ($(this).hasClass("container")) {
6       /* Rekursiver Aufruf der Funktion mit der Box des Containers */
7     }
8   });
9 }
```

Beim ersten Funktionsaufruf wird die Querybox als Parameter übergeben. In der Funktion wird über alle verschiebbaren Elemente in der Querybox iteriert (Zeile 2). Widgets, die nicht deaktiviert sind, werden mit allen Einstellungen im JSON-Format abgebildet und zum String hinzugefügt. Liegt ein Container als aktuelles Element bei der Iteration vor, so erfolgt ein rekursiver Aufruf der Funktion mit der Box des Containers.

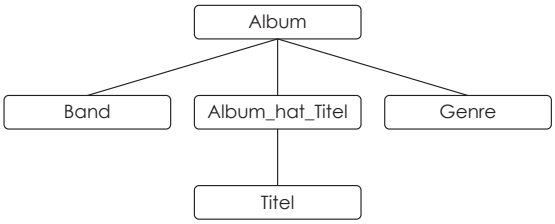

Abbildung 5.9: Baumstruktur für Beispieldatenbank, die Beziehungen zwischen den Tabellen abbildet.

Das Range-Widget, welches als Beispiel in den vorherigen Abschnitten diente, wird im JSON-Format wie folgt dargestellt:

```
{"id": "gruendungsjahr","type": "RangeWidget","view": "true","filter": "
    true", "name": "GRÜNDUNGSJAHR", "fromValue": "1950", "toValue": "
    2013"}
```

Die Abfrage als JSON-String wird über einen Ajax-Request dem Server übermittelt und dient als Grundlage für das Erstellen der SQL-Anweisung durch das `QueryInterpreter`-Objekt am Server.

5.3.6 Abfragen über mehrere Tabellen

Die erstellte Abfrage kann durchaus Widgets enthalten, die Spalten aus unterschiedlichen Tabellen repräsentieren. Daher gilt es, alle von der Abfrage betroffenen Tabellen über die *JOIN*-Klausel in der SQL-Anweisung zu verknüpfen.

Die erste Abfragetabelle stellt die Basistabelle dar und wird über die *FROM*-Klausel angegeben. Ausgehend von der Basistabelle (Root-Knoten) wird durch die Interpreter-Funktion `getRelationTree()` eine Baumstruktur generiert, welche die Beziehungen zu allen anderen Tabellen in der Datenbank abbildet. Die Funktion `getRelationPath()` ermittelt, ausgehend vom Root-Knoten (Basistabelle), alle Pfade zu den weiteren Abfragetabellen im Baum. Diese Pfade beinhalten dann jene Tabellen (Knoten), die in der *JOIN*-Klausel der SQL-Anweisung definiert werden müssen. Beinhalten die Pfade gleiche Tabellen, so werden diese nur einmal in der Klausel berücksichtigt.

Beispiel

Als Ausgangspunkt dient eine Abfrage auf eine Datenbank, welche Bands und ihre Alben abbildet. Die Datenbankabfrage beinhaltet Widgets, die Spalten aus den Tabellen *Album* und *Titel* repräsentieren. Somit stellen die beiden die Abfragetabellen dar. Die erste Abfragetabelle, also die Tabelle *Album*,

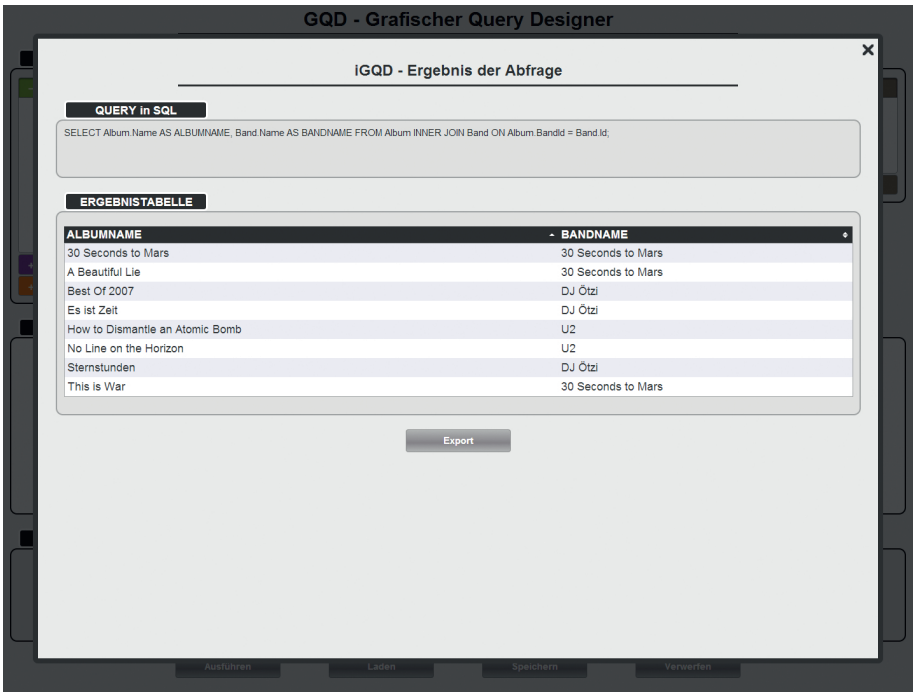

Abbildung 5.10: Dialogbox beim Anzeigen der Ergebnistabelle im Prototyp.

dient als Basistabelle und wird in der *FROM*-Klausel angegeben. Anschließend wird die in Abbildung 5.9 dargestellte Baumstruktur generiert, wobei die Basistabelle *Album* als Root-Knoten fungiert. In der Abbildung werden die Tabellen durch Knoten und die Beziehungen durch Kanten repräsentiert. Der Baum bietet die Grundlage, um den Pfad von der Basistabelle *Album* zur zweiten Abfragetabelle *Titel* zu ermitteln. Dieser beinhaltet in diesem Beispiel die beiden Tabellen *Album_ hat_ Titel* und *Titel*, die für das Generieren der *JOIN*-Klausel benötigt werden. Die *FROM*- und die *JOIN*-Klausel in SQL-Syntax sieht dann wie folgt aus:

```
FROM Album
INNER JOIN Album_hat_Titel ON Album.Id = Album_hat_Titel.AlbumId
INNER JOIN Titel ON Album_hat_Titel.TitelId = Titel.Id;
```

5.3.7 Ausführen der Abfrage

Nachdem die *SELECT*-Klausel und die *WHERE*-Klausel über eine rekursive Funktion generiert wurden, lassen sich alle im `QueryInterpreter`-Objekt generierten Klauseln zu einer SQL-Anweisung zusammenfügen. Anschließend wird die Verbindung zur Datenbank aufgebaut und die Abfrage durch eine Instanz der Klasse `Database` auf die Datenbank ausgeführt. Serverseitig

erfolgt die Strukturierung der Ergebnisdaten durch HTML-Elemente. Per Ajax-Response werden die bereits aufbereiteten Ergebnisdaten an den Client gesendet.

Das Abfrageergebnis wird dem Benutzer im Prototyp über eine Dialogbox präsentiert. Zusätzlich ermöglicht die Verwendung des JavaScript-Plugins *Tablesorter* das Sortieren der einzelnen Spalten in der Ergebnistabelle. Abbildung 5.10 zeigt die Dialogbox beim Anzeigen der Ergebnistabelle im Prototyp.

Kapitel 6

Evaluierung

In diesem Kapitel wird auf die Evaluierung des Prototyps eingegangen. Dabei sollen Probleme, welche bei der Verwendung des Prototyps auftreten können, mithilfe von Evaluierungsmethoden erkannt werden, die sowohl Experten als auch mögliche Endbenutzer als Probanden erfordern.

Zuerst werden die verwendeten Evaluierungsmethoden im Detail vorgestellt und der Ablauf der Evaluierung beschrieben. Anschließend erfolgt die Auflistung aller Probleme, die im Zuge der Evaluierung mit dem Prototyp von den Probanden erkannt wurden.

6.1 Methoden

In der Human-Computer-Interaction (HCI) gibt es verschiedene Methoden um die Gebrauchstauglichkeit (Usability) von Systemen zu überprüfen. Dabei sind sowohl Methoden bekannt, bei denen Endbenutzer zum Einsatz kommen, als auch solche, die den Endbenutzer nicht in die Evaluierung miteinbeziehen. Die Auswahl der besten Methode für die Evaluierung ist stark von der herrschenden Situation im Entwicklungsprozess der Software abhängig [9, S. 72].

Für die Evaluierung des Prototyps wurden zwei Methoden ausgewählt. Die Heuristische Evaluierung und die Think Aloud Methode, welche nachstehend vorgestellt werden.

6.1.1 Heuristische Evaluierung

Bei dieser Methode prüft der Evaluierende, ob bestimmte Usability Prinzipien im System verletzt sind. Diese Prinzipien werden als Heuristiken bezeichnet [9, S. 72].

Jakob Nielsen und Rolf Molich entwickelten zehn Heuristiken zur Evaluierung von User Interfaces, die den Evaluierenden beim Feststellen von Usability Problemen helfen sollen [14, 15]. Die Durchführung der Methode benötigt

drei bis fünf Experten. Dabei unterscheidet Nielsen zwischen *Regular Experts* und *Double Experts*. *Regular Experts* sind Experten mit Fachkenntnissen im Bereich der Usability. *Double Experts* weisen zusätzliche Erfahrung im Fachgebiet des zu evaluierenden Systems auf. Auch Laien ohne Kenntnisse in den beiden Fachgebieten können bei der Evaluierung zum Einsatz kommen, jedoch hat dies negative Auswirkungen auf die Fehlererkennungsrate [13].

Besonders wichtig bei der Heuristischen Evaluierung ist, dass die Evaluierenden untereinander nicht kommunizieren und sich beeinflussen können. Jedes gefundene Usability Problem erhält einen Grad, der die schwere des Problems angibt. Zusätzlich notiert der Evaluierende, welche Heuristik verletzt wurde. Haben alle Testpersonen ihre Evaluierungen abgeschlossen, so erfolgt ein Zusammenfügen aller Ergebnisse zu einer gesammelten Problemliste. Dabei ist es möglich darüber zu diskutieren, ob das Problem in die Liste aufgenommen wird oder nicht [9, S. 72], [13, 15].

6.1.2 Think Aloud

Die Think Aloud Methode (TA) ist eine weitverbreitete Methode zum Testen von Usability und setzt, im Gegensatz zur Heuristischen Evaluierung, mögliche Endbenutzer des Systems als Probanden ein [9, S. 73]. Die Testpersonen erhalten bei der Think Aloud Methode bestimmte Aufgaben, die mithilfe des zu testenden Systems gelöst werden sollen. Während der Durchführung der Aufgaben wird von den Probanden verlangt, dass sich diese über ihre Gedanken und ihr Handeln äußern. Experten, welche Fachkenntnisse im Bereich Usability aufweisen, beobachten die Probanden bei der Verwendung des Systems und protokollieren den Vorgang. Diese Aufzeichnungen lassen die Experten auf die Wahrnehmung des Systems durch die Testpersonen sowie auf mögliche Probleme im Umgang mit dem System schließen [5, S. 31], [19, S. 322].

Die Think Aloud Methode kann in verschiedenen Varianten angewendet werden [7, S. 1154]:

- **Concurrent Think Aloud (CTA):** Die Testperson formuliert ihre Gedanken während dem Abarbeiten der Aufgaben.
- **Retrospective Think Aloud (RTA):** Der Proband erfüllt seine Aufgaben und der Testvorgang wird auf Video aufgezeichnet. Anschließend beschreibt dieser seine Gedanken, auf Basis des aufgezeichneten Videomaterials.

Ein Nachteil der CTA ist, dass sich der Proband durch das gleichzeitige Beschreiben seiner Gedanken nicht voll auf das Erfüllen der Aufgaben konzentrieren kann. Dadurch könnte die Testperson bei der Verwendung des Systems fehleranfälliger sein und in weiterer Folge das Testergebnis der CTA verfälscht werden [8, S. 349].

6.2 Durchführung

Bereits in [15, S. 254] wird durch die Autoren Nielsen und Molich empfohlen, die Ergebnisse der Heuristischen Evaluierung durch Erkenntnisse der Think Aloud Methode zu ergänzen. Im Zuge dieser Arbeit wurde der Prototyp des gQDs mittels diesen beiden Methoden evaluiert. Die Kombination der Heuristischen Evaluierung und der Think Aloud Methode bringt den Vorteil mit sich, dass eine Evaluierung sowohl mit Experten als auch mit Endbenutzern durchgeführt wird.

Im Folgenden wird beschrieben, wie die Evaluierungen des Prototyps durchgeführt wurden und welche Vorbereitungsarbeiten dafür notwendig waren.

6.2.1 Heuristische Evaluierung

Als erste Methode wurde die Heuristische Evaluierung verwendet, die Experten als Probanden zulässt.

Experten

An der Evaluierung des Prototyps nahmen vier weibliche Studentinnen im Alter zwischen 23 und 26 aus dem Masterstudiengang *Interactive Media* der *University of Applied Sciences Upper Austria* in Hagenberg teil. Alle Probanden weisen mehrjährige Erfahrung im Bereich Webdesign auf. Zusätzlich wurden im Zuge des Studiums datenbankspezifische Fachkenntnisse erworben, die bei der Verwendung des Prototyps von Vorteil sein könnten. Da jedoch keine der vier Studentinnen Fachkenntnisse auf dem Gebiet der Usability hatte, stellten diese bei der Heuristischen Evaluierung *Single Experts* dar. Allen Probanden war die Verwendung von Heuristiken zur Evaluierung von Systemen unbekannt.

Vorbereitung

Der Prototyp des gQDs wurde als webbasierte Anwendung implementiert und erforderte daher keine Installation im Vorfeld der Evaluierung. Die Probanden stellten ihre privaten Notebooks mit Browser und Internetzugang zur Verfügung, um den zu evaluierenden Prototyp verwenden zu können. Damit Beispielabfragen mit dem Prototyp formuliert werden konnten, musste eine Datenbank mit Beispieldaten angelegt und die Konfiguration für den gQD erstellt werden.

Für die Evaluierung wurden die Heuristiken von Nielsen und Molich [14, 15] verwendet. Diese konnten aus dem Artikel in [26] mit Erklärungen entnommen und für die Probanden ausgedruckt werden. Tabelle A.1 in Anhang A beinhaltet die Auflistung dieser Heuristiken. Zusätzlich wurde eine Evaluierungstabelle für das Eintragen der gefundenen Usability Probleme erstellt

und für alle Evaluierenden ausgedruckt. Diese wird in Tabelle A.2 im Anhang A dargestellt.

Durchführung

Nachdem alle Probanden für die Evaluierung anwesend waren, erfolgte eine kurze Erklärung des geplanten Ablaufs. Anschließend wurde auf allen Notebooks der zu evaluierende Prototyp gestartet und die Evaluierungstabelle an die Probanden ausgeteilt. Um ein gegenseitiges Beeinflussen der Evaluierenden zu vermeiden, wurden alle nochmal darauf hingewiesen, dass die Evaluierung alleine durchzuführen ist und kein Absprechen zwischen den Probanden erfolgen sollte. Somit konnte mit der Evaluierung begonnen werden. Anfangs wurden den Probanden drei Aufgaben gestellt, um die Abfragemöglichkeiten des Systems aufzuzeigen. Mit fortlaufender Dauer der Evaluierung definierten die Experten jedoch selbständig ihre Abfragewünsche und versuchten diese im Prototyp abzubilden. Auffallend während der Evaluierung war, dass bei den Probanden Unsicherheiten beim Zuweisen des festgestellten Problems zu einer bestimmten Heuristik auftraten. Zur Beruhigung wurde den Experten mitgeteilt, dass die Zuteilung nach der Evaluierung gemeinsam diskutiert und überarbeitet werden kann.

Die Evaluierung dauerte ca. eine Stunde und es erfolgte anschließend eine gemeinsame Diskussion der Ergebnisse. Dabei wurden alle festgestellten Probleme genannt und in einer gemeinsamen Problemliste gesammelt. Trat ein Problem nur bei einem Probanden auf, so diskutierten die Evaluierenden, ob das Problem in die Liste aufgenommen wird oder nicht. Zusätzlich vergaben die Experten für jedes Problem in der Liste einen Schweregrad.

6.2.2 Think Aloud

Die zweite Evaluierung des Prototyps wurde mittels der CTA, einer Variante der Think Aloud Methode, durchgeführt. Die Methode verwendet mögliche Endbenutzer als Probanden und diese werden aufgefordert, ihre Gedanken während dem Abarbeiten von Aufgaben laut auszusprechen.

Probanden

Als Probanden konnten drei Frauen und drei Männer im Alter zwischen 19 und 34 gefunden werden. Beim Auswählen der Testpersonen wurde darauf Wert gelegt, dass diese keine Erfahrung mit Datenbanken oder mit der Datenbanksprache SQL hatten. Ein Proband wies zwar Fachkenntnisse auf dem Gebiet der Usability auf, jedoch konnten alle Testpersonen als mögliche Endbenutzer für die Think Aloud Methode gewertet werden.

Vorbereitung

Die Beispieldatenbank für den Prototyp wurde bereits für die erste Evaluierung erstellt. Somit mussten nur mehr Aufgaben formuliert werden, die der Proband im Zuge der Evaluierung mit dem gQD-Prototyp lösen sollte. Die elf Aufgaben stellten Abfragen an die Beispieldatenbank dar, die bestimmte Ergebnisdaten zurückliefern sollten. Außerdem wurden die Aufgaben so entwickelt, dass der Schwierigkeitsgrad langsam anstieg und die Testpersonen von Aufgabe zu Aufgabe neue Funktionalitäten im Prototyp verwenden mussten. Die verwendeten Aufgaben stehen im Abschnitt A.3 im Anhang A zur Verfügung.

Durchführung

Gegensätzlich zur Heuristischen Evaluierung wurde die Evaluierung bei der Think Aloud Methode mit allen Probanden einzeln durchgeführt. Dabei bekam jede Testperson einen Termin für die Evaluierung zugewiesen.

Zu Beginn des Tests wurde den Probanden die Vorgehensweise bei einer Think Aloud Evaluierung kurz erklärt. Besonders wichtig war es, die Testperson darauf hinzuweisen, dass sie während dem Erledigen der Aufgaben ihr Handeln und ihre Gedanken laut kommentieren sollte. Die Evaluierung wurde von zwei Experten überwacht. Während ein Experte der Reihe nach die Aufgaben dem Probanden vorstellte, protokollierte der Zweite das Verhalten und die Gedanken der Testperson beim Abarbeiten der Aufgaben mit.

Bemerkbar war, dass für einige Probanden die Vorgehensweise, während dem Test beobachtet zu werden, eine unangenehme Situation darstellte. Weiters mussten die Probanden mehrmals während der Evaluierung daran erinnert werden, dass sie ihre Gedanken den Beobachtern mitteilen sollen. Die Evaluierung dauerte pro Proband zwischen 15 und 30 Minuten. Nach einer kurzen abschließenden Besprechung zwischen den Experten und dem Probanden konnte der Testvorgang als beendet erklärt werden.

6.3 Ergebnisse

Im nachfolgenden Abschnitt werden die erkannten Probleme aus den Evaluierungen aufgelistet und mögliche Lösungsvorschläge zur Verbesserung der Bedienbarkeit des gQDs erwähnt.

6.3.1 Heuristische Evaluierung

Folgende Probleme wurden von den vier Experten bei der Heuristischen Evaluierung festgestellt.

Darstellung der Infobox

Die Experten bemängelten die Darstellung von Informationen in der dafür vorgesehenen Infobox im Interface. Die Probanden übersahen laufend Tipps, die während der Abfrageerstellung im Prototyp angezeigt wurden. Dadurch waren einige Interaktionsmöglichkeiten im Interface für die Experten schwer erkennbar.

Da die Infobox zum Großteil in Grautönen angezeigt wird, schlugen die Probanden vor, eine wesentlich kontrastreichere Darstellung für die Informationen und Tipps im Interface zu wählen. Auch eine Umpositionierung der Infobox könnte angedacht werden. Die dadurch erhaltene Dominanz der Infobox im Interface soll den Benutzer zum Lesen der zur Verfügung gestellten Informationen bewegen.

Bezeichnungen im Interface

Ein weiteres Problem wurde durch die Probanden bei der Namensgebung für einzelne Boxen im Interface festgestellt. Da die Zielgruppe des gQDs hauptsächlich technisch unversierte Benutzer sind, könnten diese durch die englischen und sehr technischen Bezeichnungen der Boxen „Widgets“, „Container“ und „Query“ verunsichert werden. Die Experten merkten an, dass dadurch eine Vermischung von Fach- und Laienbegriffen im Interface erfolge.

Als Lösung für dieses Problem gilt es, die Bezeichnungen von Boxen im Interface zu überarbeiten und sprechende Namen zu verwenden.

Größe des Containers

Wird ein Container der Querybox hinzugefügt, so ist die Boxgröße anfangs so klein, dass nur ein Widget darin platziert werden kann. Somit musste der Container in den meisten Fällen vor dem Platzieren von Elementen durch die Experten vergrößert werden. Aus diesem Grund schlugen alle Probanden vor, die Standardgröße des Containers so anzupassen, dass zu mindestens zwei Elemente dem Container hinzugefügt werden können ohne vorher die Boxgröße durch eine zusätzliche Interaktion verändern zu müssen.

Verändern der Containergröße

Bei den Containern wurde noch ein zusätzliches Problem festgestellt. Zwei der Probanden konnten über das Interface nicht erkennen, dass die Boxgröße des Containers verändert werden kann. Das am rechten, unteren Rand platzierte Symbol in der Box, welches auf die Interaktion zum Verändern der Größe aufmerksam machen soll, wurde nicht wahrgenommen.

Die Probanden brachten den Vorschlag ein, dass auf die Interaktion zum Vergrößern der Box im gQD verzichtet wird und der gQD die Containergröße automatisch beim Hinzufügen eines Elements zum Container anpasst.

Funktionen

Container können im gQD auch für das Ausführen von Funktionen auf Widgets verwendet werden. Diese Container sind unter der Kategorie „Funktionen“ in der Querybox zu finden. Die unterschiedlichen Kategorien wurden so im Prototyp implementiert, dass sie durch den Benutzer ein- und ausgeklappt werden können. Da die Kategorie „Funktionen“ standardmäßig eingeklappt ist, konnten die Experten den benötigten Container nicht sofort finden. Auch die Interaktion zum Öffnen der Kategorie wurde übersehen, da alle Kategorien in der gleichen Farbe dargestellt werden.

Vermutlich könnte die Verwendung von unterschiedlichen Farbcodierungen für die einzelnen Kategorien dieses Problem beseitigen.

Tipps bei Interaktionen

Die Probanden bemängelten, dass Tipps zu bestimmten Interaktionen nur solange über das Interface gegeben werden, bis die Interaktion durch den Benutzer das erste Mal ausgeführt wurde. Dadurch könnten dem Benutzer bei irrtümlich durchgeführten Interaktionen wichtige Tipps verborgen bleiben, da diese nachher nicht mehr angezeigt werden.

Grundsätzlich befürworteten die Experten ein dauerhaftes Anzeigen von Tipps für die Interaktionen, wobei eine Möglichkeit zum Ausblenden der Hilfestellungen für fortgeschrittene Benutzer angeboten werden könnte.

Buttons für Abfrageoptionen

Die Buttons, die das Ausführen, Verwerfen, Laden und Speichern von Abfragen ermöglichen, wurden laut den Experten nicht eindeutig bezeichnet. Daher wussten diese nicht, ob beispielsweise der Button für das Laden nur die Abfrage oder die gesamte gQD-Konfiguration betrifft. Zusätzlich wurde durch die Probanden angemerkt, dass die Buttons durch das verwendete Design deaktiviert wirken und den Benutzer dadurch nicht auf mögliche Interaktionen aufmerksam machen.

Diese, bei der Evaluierung aufgetretenen, Unklarheiten könnten durch die Verwendung von anderen Bezeichnern und einer anderen Darstellung für die Buttons vermieden werden. Durch Platzieren der Buttons direkt unter der Querybox wäre eine Verbindung zwischen den Aktionen der Buttons und der Abfrage für den Benutzer besser erkennbar.

Platzieren von Elementen in Containern

Die Experten forderten Feedback vom System, wenn ein Element über einen Container in der Querybox gezogen wird. Dadurch sollte dem Benutzer deutlich gemacht werden, dass ein Hinzufügen des Elements zum darunterliegenden Container möglich ist. Für das gewünschte Feedback wurde von den Eva-

luierenden ein Verändern der Rahmenfarbe des Containers oder die strichlierte Darstellung des Containerrahmens vorgeschlagen.

Rückgängig machen von Aktionen

Alle vier Experten probierten während der Erstellung der Abfragen im Prototyp die bekannten Shortcuts für das Rückgängigmachen bzw. Wiederherstellen von Aktionen aus. Da diese Shortcuts jedoch nicht im Prototyp umgesetzt waren, empfahlen die Probanden diese Funktionalität in weiteren Versionen des Prototyps zu unterstützen.

Darstellung der Filtereinstellungen

Die Filtereinstellungen von Widgets mit deaktivierter Filterfunktion werden im Interface des gQDs in Grautönen dargestellt. Die Evaluierenden merkten jedoch an, dass durch Ausblenden der Filtereinstellungen bei deaktivierter Filterfunktion die Darstellung der Abfrage in der Querybox wesentlich übersichtlicher wäre.

Außerdem könnte laut den Evaluierenden eine stärkere Gruppierung zwischen dem Button zum Aktivieren bzw. Deaktivieren der Filterfunktion und den Filtereinstellungen erfolgen, um die bestehende Verbindung für den Benutzer deutlicher zu machen.

UND-Verknüpfung innerhalb der Querybox

Alle Filterkriterien innerhalb der Querybox werden standardmäßig mit dem UND-Operator verknüpft. Diese Festlegung war für die Experten nicht von Anfang an klar und daher wurden UND-Container bei der Abfrageerstellung eingesetzt, die nicht notwendig gewesen wären. Um dem Benutzer die Verknüpfungsart der Querybox über das Interface zu übermitteln, könnte ein weiterer Tipp in der Infobox zum Einsatz kommen.

6.3.2 Think Aloud

Bei den durchgeführten Evaluierungen mit der Think Aloud Methode traten zum Teil die gleichen Probleme wie bei der Heuristischen Evaluierung auf. Weitere durch die Evaluierung mit Endbenutzern festgestellten Erkenntnisse werden nachfolgend angeführt.

Lesen von Tipps

Nach der Evaluierung durch die Experten wurden Änderungen bei der Darstellung der Infobox vorgenommen. Dadurch sollte gewährleistet werden, dass die Probanden bei den Evaluierungen die Tipps und Informationen besser wahrnehmen können.

Beim Beobachten der Probanden fiel jedoch auf, dass die, in der Infobox bereitgestellten, Tipps weitgehend nicht gelesen wurden. Einige Evaluierende gaben zum Beispiel an, dass sie beim Verwenden der Widgets, welche ganz oben im Interface platziert sind, die Tipps in der Infobox am unteren Rand des Bildschirms nicht erkennen konnten. Daraus lässt sich ableiten, dass Tipps und Informationen zu Elementen immer in der Nähe dieser Elemente platziert werden sollten, da der Fokus des Benutzers auf diese gerichtet ist.

Container

Da die Tipps nicht beachtet wurden, kam es in weiterer Folge zu anfänglichen Problemen bei der Verwendung von Containern. Dabei war den Probanden zum Teil nicht klar, dass Container weitere Elemente enthalten können und dadurch eine beliebige Verschachtelung von Filterkriterien ermöglichen. Einige Probanden versuchten beim logischen Verknüpfen von zwei Filterkriterien den Container als Verbindungsglied zwischen den beiden Widgets einzusetzen. Nach einer kurzen Hilfestellung durch den Beobachter konnten die Probanden die geforderte Aufgabe trotzdem noch lösen.

Die Container wirkten auf die Endbenutzer noch zu technisch und sollten für die Verwendung durch technisch unversierte Benutzer noch überarbeitet werden.

Löschen von Elementen

Das Löschen von Elementen aus der Querybox wurde im gQD so umgesetzt, dass während dem Verschieben des zu löschenden Elements die Infobox als Box mit dem Symbol eines Mülleimers dargestellt wird. Ein Platzieren des Elements über dieser Box löscht das Element aus der Querybox.

Die Probanden versuchten die Elemente beim Löschen wieder zurück in die Widget- bzw. Containerbox oder einfach aus der Querybox zu verschieben. Nachdem dies nicht funktionierte, wurden die Evaluierenden auf die Funktion zum Löschen über die Infobox aufmerksam. Einige Probanden fanden die Funktionsänderung der Infobox ungewohnt. Diese Unklarheiten könnten durch das dauerhafte Darstellen einer Box zum Löschen von Elementen im Interface beseitigt werden.

Gewohnte Funktionen

Auffallend bei allen sechs Probanden während den Evaluierungen war, dass diese beim Benutzen des Prototyps gewohnte Funktionen aus dem alltäglichen Umgang mit Computern verwenden wollten. So versuchte beispielsweise ein Proband mehrere Elemente in der Querybox mit der Maus zu markieren, um die Elemente gemeinsam über die ENTF-Taste auf der Tastatur zu

löschen. Weiters wurden nicht gefundene Funktionen mehrmals in Kontextmenüs gesucht und das Kopieren von Elementen in der Querybox probiert. Diese beobachteten Vorgehensweisen lassen darauf schließen, dass vor allem Endbenutzern die Unterschiede zwischen Desktop- und Webanwendungen nicht klar sind und daher die gewohnten Funktionen aus desktopbasierten Anwendungen im gQD gewünscht werden.

6.4 Fazit

Die Evaluierung zeigte, dass das Konzept des gQDs weitgehend funktioniert und auch ohne Erklärungen im Vorfeld der Evaluierung von den Probanden verstanden wurde. Sowohl bei der Heuristischen Evaluierung als auch bei der Evaluierung mit der Think Aloud Methode konnte festgestellt werden, dass das Interpretieren der erstellten Abfrage und Anzeigen in natürlicher Sprache ein wichtiges Feedback im gQD darstellt. Die Probanden konnten dadurch Fehler in der erstellten Abfrage schneller erkennen. Zusätzlich fiel auf, dass die Evaluierenden unklare Funktionen oder Einstellungen in der Abfrage einfach ausprobierten und die daraus resultierenden Auswirkungen auf die Abfrage über das Feedback in natürlicher Sprache erkennen konnten.

Die bei der Evaluierung festgestellten Probleme mit den Containern entstanden einerseits durch das zu technisch gehaltene Konzept für diese Elemente und andererseits durch die fehlende Wahrnehmung der gegebenen Tipps über das Interface. Daher sollte das Verknüpfen und Verschachteln von Filterkriterien, sowie das Anwenden von Funktionen für den technisch unversierten Benutzer durch den Einsatz von natürlicher Sprache verbessert werden.

Der Prototyp des gQDs konnte bei allen Evaluierenden Anklang finden und überzeugte die Probanden durch das Abstrahieren der Datenbankebene und die Möglichkeit, die Formulierung der Datenbankabfrage grafisch vorzunehmen.

Kapitel 7

Zusammenfassung

Ziel dieser Arbeit war es, eine Möglichkeit für technisch unversierte Benutzer zu schaffen, um Abfragen auf Datenbestände auszuführen, ohne dabei auf komplizierte Abfragesprachen zurückgreifen zu müssen.

Bereits beim Analysieren von bestehenden User Interfaces zur Datenabfrage wurde festgestellt, dass die Abfragen in mehreren Anwendungen sehr technisch und datenbankspezifisch formuliert werden müssen. So forderten beispielsweise manche Benutzerschnittstellen die Eingabe von Vergleichsoperatoren in der Syntax der speziellen Abfragesprache. Zusätzlich wurde bei den Anwendungen auf Feedback in der Sprache des Benutzers nicht viel Wert gelegt.

Das, in dieser Arbeit entwickelte, Konzept für einen grafischen Query Designer ermöglicht das Formulieren von Abfragen nach dem Baukastenprinzip. Dabei bilden Elemente wie Widgets oder Container die verfügbaren Einzelbausteine und können durch Kombinieren und Verschachteln zu einer Abfrage geformt werden. Das Konzept schreibt eine Konfiguration für den gQD vor, um Flexibilität bei der Datenabfrage zu gewährleisten. Ein entscheidender Punkt dabei ist, dass das Interface der Anwendung in der Sprache des Benutzers gehalten wird und das System entsprechendes Feedback bei der Erstellung der Abfrage bietet. So wäre wünschenswert, dass das Grundkonzept des Systems ohne Erklärungen im Vorfeld verstanden wird und die Möglichkeiten des gQDs während der Benutzung erkannt werden.

Dieses Konzept wurde im Zuge dieser Arbeit als Prototyp in einer webbasierten Anwendung umgesetzt. Nach Anlegen der Konfiguration für einen bestimmten Datenbestand, kann dadurch der Query Designer plattformunabhängig und ohne Installation über den Browser verwendet werden.

Die Evaluierung des Prototyps wurde mit zwei verschiedenen Methoden vorgenommen. Bei der Auswahl dieser Methoden war besonders wichtig, dass sowohl Experten mit datenbankspezifischen Fachkenntnissen, als auch mögliche Endbenutzer des gQDs zum Einsatz kamen und auf vorhandene Probleme im Konzept aufmerksam machten.

Die Evaluierung zeigte, dass das erstellte Konzept weitgehend funktioniert und von den Probanden verstanden wurde.

Auffallend bei allen Probanden war, dass das Anzeigen der erstellten Abfrage in natürlicher Sprache ein wichtiges Feedback im gQD darstellt. Dadurch konnten Fehler in der grafisch formulierten Abfrage selbst erkannt und behoben werden. Diese Erkenntnis bestätigt die Anlehnung an die natürliche Sprache beim Erstellen einer benutzerfreundlichen Schnittstelle als ein funktionierendes Konzept.

Das momentan eher an die Mathematik bzw. Aussagenlogik angelehnte Konzept der Container wurde als problematisch empfunden und sollte durch Elemente ersetzt werden, die sich vom Prinzip her am Satzbau der natürlichen Sprache orientieren. So könnten UND-Elemente verwendet werden, um Konjunktionen zwischen einzelnen Widgets herzustellen.

Der gQD könnte meiner Meinung nach durchaus Anwendung in der Praxis finden. So wäre es beispielsweise vorstellbar, dass für verschiedene Aufgabenbereiche in einem Unternehmen, die jeweiligen Konfigurationen einmalig durch den Systemadministrator erstellt werden und die Benutzer, durch Laden der Konfiguration im gQD, Abfragen auf den jeweiligen Datenbestand ausführen können.

Als weitere Schritte müssten für die, durch die Evaluierung erkannten Probleme, Lösungen gefunden und in das Konzept eingearbeitet werden. Anschließend wird eine erneute Evaluierung empfohlen, um zu kontrollieren, ob durch das Abändern des Konzepts die aufgetretenen Probleme beseitigt werden konnten. Außerdem wäre es interessant, die Think Aloud Methode einerseits mit einer bereits bestehenden Benutzerschnittstelle zur Datenabfrage und andererseits mit dem gQD-Prototyp durchzuführen. Dabei wären bei beiden Evaluierungen den Probanden die gleichen Aufgaben zu stellen, um anschließend die Unterschiede in der Bedienbarkeit zu erkennen.

Das Konzept des gQDs ist in jedem Fall erweiterbar. So kann dieses um neue Widgettypen oder Funktionen ergänzt werden, die dann in weiterer Folge dem Benutzer bei der Abfrageerstellung zur Verfügung stehen.

Zusammenfassend lässt sich festhalten, dass das Prinzip der grafischen Abfrageerstellung für relationale Datenbanken von Benutzern gut angenommen wird. Durch eine Erweiterung des Interpreters um die Funktionalität, grafisch erstellte Abfragen in weitere Sprachen zu übersetzen, wäre es möglich auch auf nicht relationale Datenbanksysteme zuzugreifen.

Anhang A

Evaluierung des Prototyps

A.1 Auflistung der Heuristiken

Table A.1: Heuristiken für User Interfaces nach Nielsen und Molich, entnommen aus [26].

01 - Visibility of system status
The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
02 - Match between system and the real world
The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
03 - User control and freedom
Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
04 - Consistency and standards
Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
05 - Error prevention
Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
06 - Recognition rather than recall

Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
07 - Flexibility and efficiency of use
Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
08 - Aesthetic and minimalist design
Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
09 - Help users recognize, diagnose, and recover from errors
Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
10 - Help and documentation
Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

A.2 Evaluierungstabelle

Tabelle A.2: Evaluierungstabelle für die Experten,
N. = Fortlaufende Nummer, Grad (0 = leicht, 4 = schwer), H. = Heuristik.

Name des Experten:				
Nr.	Problem	Grad (0 - 4)	H.	Lösung

A.3 Aufgaben für die Think Aloud Evaluierung

Für die Evaluierung wurde eine Beispieldatenbank angelegt und die entsprechende gQD-Konfiguration erstellt. Die Datenbank beinhaltet fünf Tabellen und besitzt folgende Einschränkungen:

- Bands werden mit dem Namen, dem Gründungsjahr und der Anzahl der Bandmitglieder abgebildet.
- Alben enthalten den Albumnamen, das Label, das Erscheinungsjahr sowie das Genre des Albums.
- Jedes Album wird von genau einer Band produziert.
- Jedes Album enthält mindestens einen Titel.
- Titel werden mit dem Namen und der Titellänge abgebildet.
- Ein Titel ist auf mindestens einem Album vorhanden, der gleiche Titel kann jedoch auch auf mehreren Alben erscheinen.

Folgende Aufgaben wurden für die Evaluierung mit der Think Aloud Methode vorbereitet:

1. Zeigen Sie alle Bands in der Datenbank mit dem Bandnamen und dem Gründungsjahr an.
2. Löschen Sie die Anzeige des Gründungsjahres aus der bestehenden Abfrage und zeigen Sie stattdessen die Bandmitgliederanzahl an.
3. Zeigen Sie alle Bands in der Datenbank mit dem Namen an, die nur ein Bandmitglied besitzen.
4. Zeigen Sie das Gründungsjahr und die Bandmitgliederanzahl der Band „U2“ an.
5. Zeigen Sie alle Alben in der Datenbank mit dem Albumnamen, dem Label und dem dazugehörigen Bandnamen an.
6. Zeigen Sie alle Alben in der Datenbank mit dem Albumnamen, dem Erscheinungsjahr und dem dazugehörigen Bandnamen an, die im Jahre „2009“ erschienen sind und das Genre „Rock“ zugewiesen haben.
7. Zeigen Sie alle Titel in der Datenbank mit dem Titelnamen an. Zusätzlich soll der Name des Albums, auf dem der Titel vertreten ist und der Name der Band, die das Album produziert hat, angezeigt werden.
8. Zeigen Sie alle Titel in der Datenbank mit dem Titelnamen an, die auf dem Album mit dem Namen „This is War“ enthalten sind.
9. Zeigen Sie alle Bands in der Datenbank mit dem Bandnamen an, die entweder „3“ oder „4“ Bandmitglieder besitzen.
10. Ermitteln Sie für alle Alben in der Datenbank, den Albumnamen und die Anzahl der Titel, die auf dem Album enthalten sind.
11. Zeigen Sie alle Bands in der Datenbank mit dem Bandnamen an, die entweder „3“ oder „4“ Bandmitglieder besitzen oder zwischen den Jahren „1999“ und „2013“ gegründet worden sind.

Anhang B

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

B.1 Masterarbeit

Pfad: /

Peintner_Thomas_2013.pdf Masterarbeit (Gesamtdokument)

B.2 Onlinequellen

Pfad: /Onlinequellen

10 Heuristics for User Interface Design - Article by Jakob Nielsen.pdf
Ajax - A New Approach to Web Applications - Adaptive Path.pdf
Extensible Markup Language - Wikipedia.pdf
Look up Tables in SQL - Simple Talk.pdf
Microsoft Access - Wikipedia.pdf
Microsoft Excel - Wikipedia.pdf

B.3 Abbildungen

Pfad: /Abbildungen

gqdAusfuehren.pdf . . .	Skizze für Interface „Ausführen der Abfrage“
gqdBeispiel.pdf	Beispiel für Abfrage in der Querybox
gqdContainer.pdf	Skizze für Container
gqdDateWidget.pdf . .	Skizze für Date-Widget
gqdErstellen.pdf	Skizze für Interface „Erstellen der Abfrage“
gqdKonfiguration.pdf .	Skizze für Interface „Laden der Konfiguration“

gqdKonzept.pdf	Skizze für Aufbau des gQDs
gqdOperatorWidget.pdf	Skizze für Operator-Widget
gqdRangeWidget.pdf . .	Skizze für Range-Widget
gqdSelectboxWidget.pdf	Skizze für Selectbox-Widget
gqdSliderWidget.pdf . .	Skizze für Slider-Widget
gqdTextboxWidget.pdf	Skizze für Textbox-Widget
gqdWidgets.pdf	Skizze für den allgemeinen Aufbau eines Widgets
implAblaufSequenz.pdf	Sequenzdiagramm für die Ablaufschritte
implArchitektur.pdf . .	Serverseitige Architektur
implAufbau.pdf	Allgemeiner Aufbau des gQDs
implAusfuehren.pdf . .	Screenshot Interface „Ausführen der Abfrage“
implClientArch.pdf . . .	Klassendiagramm für Clientseite
implClientArchTree.pdf	Objektdiagramm für Elemente
implContainer.pdf . . .	Container
implErstellen.pdf	Screenshot Interface „Erstellen der Abfrage“
implRangeWidget.pdf .	Range-Widget, deaktivierte Filterfunktion
implRangeWidgetA.pdf	Range-Widget, aktivierte Filterfunktion
implRelationTree.pdf . .	Beispiel für Beziehungen zwischen Tabellen
rdfGraph.pdf	Graph mit RDF-Beispiel
screenFilterExcel.pdf . .	Screenshot Microsoft Excel mit Filterfunktion
screenQueryAccess.pdf .	Screenshot Microsoft Access bei Abfrage
screenQueryEQ.pdf . .	Screenshot Easy Query bei Abfrage
screenQueryExcel.pdf .	Screenshot Microsoft Query bei Abfrage
screenQueryPMA.pdf .	Screenshot phpMyAdmin bei Abfrage
screenViewDrupal.pdf .	Screenshot Drupal bei Erstellung einer View

B.4 Prototyp

B.4.1 Code

Die gesamte Implementierung des Prototyps befindet sich unter /Prototyp/Code auf der CD-ROM.

B.4.2 Beispieldatenbank

Pfad: /Prototyp/Beispieldatenbank

datenbank.sql	SQL für Erstellung der Datenbank
datenmodell.pdf	Datenmodell für Beispieldatenbank

B.4.3 Dokumentation

Pfad: /Prototyp/Dokumentation

installationsAnleitung.pdf	Installationsanleitung für den Prototyp

B.4.4 Screenshots

Pfad: /Prototyp/Screenshots

screenCreateQuery.png	Erstellen der Abfrage
screenExampleQuery.png	Beispiel für erstellte Abfrage
screenLoadQuery.png	Laden der Abfrage
screenResultData.png	Ausführen der Abfrage (Ergebnistabelle)
screenSaveQuery.png	Speichern der Abfrage

Quellenverzeichnis

Literatur

[1] Andreas Dengel. *Semantische Technologien – Grundlagen, Konzepte, Anwendungen*. Spektrum Akademischer Verlag, 2012.

[2] Stefan Edlich u. a. *NoSQL – Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. 2. Aufl. München: Carl Hanser Verlag, 2011.

[3] Ju Fan, Guoliang Li und Lizhu Zhou. „Interactive SQL Query Suggestion: Making Databases User-Friendly“. In: *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*. (Hannover). Washington: IEEE Computer Society, Apr. 2011, S. 351–362.

[4] Erich Gamma u. a. *Design Patterns – Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley, 1995.

[5] Amy M. Gill und Blair Nonnecke. „Think Aloud: Effects and Validity“. In: *Proceedings of the 30th ACM International Conference on Design of Communication*. (Seattle). New York: ACM, Okt. 2012, S. 31–36.

[6] Hagen Graf. *Drupal*. München: Addison-Wesley, 2006.

[7] Maaike J. Van den Haak, Menno D. T. De Jong und Peter Jan Schellens. „Employing think-aloud protocols and constructive interaction to test the usability of online library catalogues: a methodological comparison“. In: *Interacting with Computers* 16.6 (2004), S. 1153–1170.

[8] Maaike J. Van den Haak, Menno D. T. De Jong und Peter Jan Schellens. „Retrospective vs. concurrent think-aloud protocols: testing the usability of an online library catalogue“. In: *Behaviour & Information Technology* 22.5 (2003), S. 339–351.

[9] Andreas Holzinger. „Usability engineering methods for software developers“. In: *Communications of the ACM* 48.1 (Jan. 2005), S. 71–74.

[10] Alfons Kemper und André Eickler. *Datenbanksysteme – Eine Einführung*. 8. Aufl. München: Oldenbourg Verlag, 2011.

[11] Microsoft Corp. *Microsoft Access 2010 Produkthandbuch*. 2010. URL: http://download.microsoft.com/download/8/3/7/83728F9F-B2F7-486B-85FD-AA0C18687DDC/Microsoft%20Access%202010%20Product%20Guide.pdf.

[12] Microsoft Corp. *Microsoft Excel 2010 Produkthandbuch*. 2010. URL: http://download.microsoft.com/download/8/3/7/83728F9F-B2F7-486B-85FD-AA0C18687DDC/Microsoft%20Excel%202010%20Product%20Guide.pdf.

[13] Jakob Nielsen. „Finding usability problems through heuristic evaluation“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. (Monterey). New York: ACM, Mai 1992, S. 373–380.

[14] Jakob Nielsen. „Heuristic Evaluation“. In: *Usability Inspection Methods*. Hrsg. von Jakob Nielsen und Robert L. Mack. New York: John Wiley & Sons, Inc., 1994. Kap. 2, S. 25–62.

[15] Jakob Nielsen und Rolf Molich. „Heuristic evaluation of user interfaces“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. (Seattle). New York: ACM, Apr. 1990, S. 249–256.

[16] Julia Coleman Prior. „Online Assessment of SQL Query Formulation Skills“. In: *Proceedings of the fifth Australasian Conference on Computing Education - Volume 20*. (Adelaide). Darlinghurst: Australian Computer Society, Inc., Feb. 2003, S. 247–256.

[17] Gunter Saake, Kai-Uwe Sattler und Andreas Heuer. *Datenbanken – Konzepte und Sprachen*. 4. Aufl. Heidelberg: mitp, 2010.

[18] Nicolai Schwarz. *Drupal 7 – Das Praxisbuch für Ein- und Umsteiger*. Bonn: Galileo Press, 2011.

[19] Torsten Stapelkamp. *Interaction- und Interfacedesign: Web-, Game-, Produkt- und Servicedesign, Usability und Interface als Corporate Identity*. Heidelberg: Springer, 2010.

[20] Helmut Vonhoegen. *Einstieg in XML – Grundlagen, Praxis, Referenzen*. 4. Aufl. Bonn: Galileo Press, 2007.

Online-Quellen

[21] URL: http://de.wikipedia.org/wiki/Xml (besucht am 10.04.2013).

[22] URL: http://de.wikipedia.org/wiki/Microsoft_Excel (besucht am 15.04.2013).

[23] URL: http://de.wikipedia.org/wiki/Microsoft_Access (besucht am 17.04.2013).

[24] URL: https://www.simple-talk.com/sql/t-sql-programming/look-up-tables-in-sql-/ (besucht am 18.05.2013).

[25] James. J. Garret. *Ajax: A New Approach to Web Applications*. Feb. 2005. URL: http://www.adaptivepath.com/ideas/essays/archives/000385.php (besucht am 02.06.2013).

[26] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. Jan. 1995. URL: http://www.nngroup.com/articles/ten-usability-heuristics/ (besucht am 04.05.2013).