Universität Bremen

Fachbereich 3: Mathematik und Informatik

Bachelor-Thesis

Eine blockbasierte, domänenspezifische Abfragesprache

Marvin Franke

Matrikel-Nr. 422 083 6

02. September 2019

1. Gutachter: Prof. Dr. Sebastian Maneth

2. Gutachter: Prof. Dr. Thomas Schneider

Betreuer: Prof. Dr. Sebastian Maneth

Marvin Franke

Eine blockbasierte, domänenspezifische Abfragesprache

Bachelor-Thesis, Fachbereich 3: Mathematik und Informatik

Universität Bremen, September 2019

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 02. September 2019

Marvin Franke

Zusammenfassung

Der Versuch, Endbenutzern einer Software mithilfe diverser Mittel das Programmieren und auch das Formulieren von Abfragen zu erleichtern, um dieselbe Software selbst anzupassen, ist keine Neuheit. Die Verwendung visueller Mittel ist nur eine Möglichkeit dies zu erreichen. Der Vorteil beim Einsatz visueller Mittel wäre, dass diese Bedeutungen und Strukturen effektiver vermitteln würden.[Nar93]
Im Rahmen dieser Bachelorarbeit soll mithilfe der Bibliothek „Blockly“ von Google eine Datenbankabfragesprache entwickelt werden. Mit dieser sollen Bedingungen definiert werden, mit deren Hilfe die Nutzer einer E-Learning-Lösung gefiltert werden können. Bei der Entwicklung dieser Abfragesprache sollen Ideen weiterverwendet werden, die auch erfolgreich bei anderen visuellen Programmiersprachen und visuellen Abfragesprachen eingesetzt werden.
Die Erlernbarkeit dieser Sprache soll auch mit Nutzertests ermittelt werden, um daraufhin Maßnahmen zu ergreifen, diese zu verbessern.

Inhaltsverzeichnis

Inhaltsverzeichnis i

1 Einleitung **1**

1.1 Ziel und Motivation der Arbeit 1

1.2 Vorgehensweise 2

1.3 Kurzportrait der engram GmbH, des eLMS und QuizCards 3

2 Definitionen und Grundlagen **4**

2.1 End-User Development 4

2.2 Diagrammatische VQLs 5

2.3 Tabellarische VQL am Beispiel von QBE 6

2.4 Google Blockly, Code.org und Scratch 7

2.4.1 Google Blockly 8

2.4.2 Tutorials von Code.org 8

2.4.3 Blöcke wie Legosteine verbinden 10

3 Entwicklung **11**

3.1 Die Datenbank 11

3.1.1 Datenbankstruktur 12

3.1.1.1 Spieler 12

3.1.1.2 Multi-Gruppen-Challenge 13

3.1.1.3 Multi-Gruppen-Challenge Ergebnisse 14

3.2 Die Abfragesprache 16

3.2.1 Verfügbare Blöcke 17

3.2.1.1 Kategorie „Datenbank“ 17

3.2.1.2 Kategorie „Logik“ 20

3.2.1.3 Kategorie „Zahlen“ 22

3.2.1.4 Kategorie „Datum und Uhrzeit“ 25

3.2.1.5 Kategorie „Texte“ 26

3.2.2 Umsetzung mit Blockly 27
3.2.2.1 Implementierung eines Blocks 27
3.2.2.2 Einbinden des Editors 31
3.2.2.3 Beispiel einer Abfrage 32

4 Usability **34**
4.1 Usability Testessen 34
4.2 Maßnahmen nach dem Usability Testessen 35
4.2.1 Benutzerdokumentation 35
4.2.2 Tutorial 36
4.3 Weitere Tests 37
4.3.1 Dokumentation 37
4.3.2 Tutorials 38
4.3.3 Tests nach den Tutorials 38

5 Zusammenfassung und Ausblick **40**
5.1 Ausblick 41

A Appendix **43**
A.1 Abbildungsverzeichnis 43
A.2 Tabellenverzeichnis 44
A.3 Listings 44
A.4 Literatur 44
A.5 Liste der Abkürzungen 46
A.6 Glossar 47
A.7 Inhalte des Datenträgers 49

Kapitel 1

Einleitung

1.1 Ziel und Motivation der Arbeit

Ziel dieser Bachelorarbeit ist es, eine neue Datenbankabfragesprache zu entwickeln und zu evaluieren, wie Nutzer diese möglichst leicht erlernen können.

> „We posit that an increased use of EUD tools in serious games would benefit their efficacy“ [MD17]

Trotz des Potenzials von der Verwendung von End-User Development (EUD) in Serious Games, aber auch Spielen im Allgemeinen, friste EUD dabei aber ein Nischendasein. Einige der Tools, welche es erlauben Spiele zu modifizieren, hätten eine steile Lernkurve und würden schon den Tools, welche von den Entwicklern verwendet werden, ähneln. Wiederum andere seien stark begrenzt in ihren Möglichkeiten, jedoch dafür einfacher zu nutzen. Es geht hierbei also vor allem auch darum, die richtige Balance zu finden zwischen den Fähigkeiten und Anforderungen des Endbenutzers.[MD17]
Mit blockbasierten Programmiersprachen werden die Grundlagen der Programmierung schon Kindern beigebracht.[Cod][MIT] Visuelle Programmiersprachen im Allgemeinen sollen die Erstellung von Skripten oder sogar ganzen Programmen vereinfachen, da Bilder und visuelle Mittel Bedeutungen und Strukturen effektiver vermitteln könnten als Text.[Nar93]
Im Rahmen dieser Arbeit werde ich eine blockbasierte, domänenspezifische Datenbankabfragesprache auf Basis von Google Blockly implementieren und evaluieren, wie diese zu gestalten und zu dokumentieren ist, damit sie für Endbenutzer möglichst einfach zu erlernen ist. Dabei sollen Ansätze und Ideen aus anderen visuellen Sprachen verwendet werden. Sie soll dafür verwendet werden können, Freischaltbedingungen von Erfolgen einer Quiz App zu formulieren. Bearbeitet der Nutzer

in der Quiz App dann Quizzes, werden diese Bedingungen regelmäßig geprüft. Sobald eine Bedingung gilt, wird dem Nutzer der Erfolg eingeblendet und ggf. eine Belohnung, zum Beispiel Bonuspunkte, vergeben. Die Sprache würde es den Anwendern dieser App erlauben, die App für ihre Zwecke selbst anzupassen, ohne Weiterentwicklungen beauftragen zu müssen, welche ggf. teuer und zeitaufwändig werden würden.

1.2 Vorgehensweise

Nach einer kurzen Vorstellung der engram GmbH und des Produkts QuizCards werden die theoretischen Grundlagen erörtert. Es wird damit begonnen zu klären, was EUD ist, wofür es eingesetzt wird und welche Vor- und Nachteile es bietet. Daraufhin werden verschiedene visuelle Sprachen vorgestellt, von denen einige bereits zur Erstellung von Datenbankabfragen dienen. Diese werden als Visual Query Languages (VQLs) bezeichnet. Andere hingegen dienen der Erstellung von Programmen und Skripten und ähneln Programmiersprachen wie JavaScript. Bei der Vorstellung der einzelnen Sprachen wird insbesondere darauf eingegangen, wie sie einen Endbenutzer bei der Bearbeitung von Aufgaben unterstützen.
Im darauf folgenden Kapitel wird die Umsetzung beschrieben: Mit dem Wissen aus dem vorherigen Kapitel soll die im vorherigen Abschnitt genannte Sprache implementiert werden. Für die Auswertung sollen diese Abfragen in die Structured Query Language (SQL) übersetzt werden. Außerdem wird beschrieben, wie die Datenstruktur mithilfe von Views vereinfacht wird.
Im vierten Kapitel steht das Usability Kriterium der Erlernbarkeit im Vordergrund. Mithilfe von Nutzertests sollen zuerst Probleme bei der Verwendung der Sprache ermittelt werden. Mit der Kenntnis dieser werden Maßnahmen entwickelt, die diese beheben sollen. Anschließend wird dies in weiteren Nutzertests überprüft. Mit den Ergebnissen aus den letzten Nutzertests wird dann im letzten Kapitel das Fazit gezogen und auf deren Basis weitere Vorschläge zur Verbesserung der Sprache formuliert.

1.3 Kurzportrait der engram GmbH, des eLMS und QuizCards

Auftraggeber dieses Projektes ist die engram GmbH. Das Bremer Softwareunternehmen besteht seit 1990 und hat sich auf Digital Signage und E-Learning spezialisiert. Zu dem Portfolio der E-Learning Abteilung gehört das engram LernManagementSystem (eLMS). Das eLMS basiert auf der Open Source Lösung Moodle und ermöglicht es engrams Kunden, digitale Aus- und Weiterbildungsprozesse in ihrem Unternehmen anzubieten. Darüber hinaus entwickelt engram auch Lerninhalte in Form von Web Based Trainings (WBTs), welche mithilfe des eLMS von den Lernern bearbeitet werden können.
In das eLMS wurde auch engram QuizCards integriert. Diese Lernapp verwendet spielerische Elemente, um die Lerner zu motivieren: So können zum Beispiel in verschiedenen Spielmodi Punkte gesammelt werden, die wiederum mit höheren Ranglistenplätzen belohnt werden.

Kapitel 2

Definitionen und Grundlagen

2.1 End-User Development

Das Ziel von EUD ist es, den Endbenutzern einer Software zu ermöglichen, ein System nach ihren Bedürfnissen zu entwickeln bzw. anzupassen.[Lie+06] Dies kann, je nach Anwendungsgebiet, sehr unterschiedlich aussehen. So erlauben es zum Beispiel mehrere Tabellenkalkulationen, die Basisfunktionalität von Software mithilfe von Makros zu erweitern[Bai06] oder Mailprogramme E-Mails anhand vom Nutzer festgelegter Regeln automatisch in Ordner zu verschieben.[IBM] Im Rahmen dieser Arbeit werden insbesondere visuelle Sprachen betrachtet. Diese erlauben es dem Benutzer, mit visuellen Mitteln anstelle von Text zu programmieren. Die VQLs erlauben es dagegen, Datenbankabfragen mit visuellen Mitteln zu formulieren.
Ebenso vielseitig wie die Formen sind auch die Vorteile von EUD, sowohl aus der Sicht der Softwareentwickler und der Anwender:

- Features können schneller selbst implementiert werden.[LPW06]
  Eine der am weitesten verbreiteten Vorgehensweisen beim Projektmanagement eines Softwareprojektes ist Scrum. Bei dieser Vorgehensweise wird üblicherweise alle zwei bis vier Wochen eine potentiell auslieferbare Version eines Produktes entwickelt. Wie lange genau ein solcher Sprint dauert, kann sich von Team zu Team unterscheiden. Während eines Sprints sind die Features, welche in diesem Zeitraum entwickelt werden, üblicherweise nicht mehr änderbar. Unter der Annahme, dass das gewünschte Feature zum nächstmöglichen Release verfügbar sein soll, muss also erst einmal der aktive Sprint beendet werden, sodass die Wartezeit auf das Feature vier bis acht Wochen beträgt. Dies wäre aber nur der Idealfall, in dem der Product Owner das Feature hoch genug priorisiert hat.[Cob11]

Mit EUD jedoch verkürzt sich die Wartezeit darauf, wie lange der Endbenutzer bräuchte, um das Feature selbst zu implementieren.

- Der Softwarehersteller implementiert ein Feature nicht.
  Nicht immer werden alle Wünsche eines Kunden durch den Softwarehersteller erfüllt, sei es aufgrund mangelnder Nachfrage bei anderen Kunden oder aufgrund von fehlenden Aufträgen. Falls also der Softwarehersteller sich also gegen die Implementierung eines Features entscheidet, würde EUD es dem Endbenutzers ggf. ermöglichen, dieses selbst nachzurüsten.[LPW06]
- Nicht jeder Endbenutzer würde gleich eine universell einsetzbare Sprache wie Python erlernen.[LPW06]

Natürlich sind auch einige Nachteile zu beachten:

- EUD ist, je nach gewählter Methode, begrenzt:
  Beim Programming by Example beispielsweise ist die Anwendung auf sich wiederholende Aufgaben begrenzt. Und selbst dann könnten sich Schritte nur sehr geringfügig unterscheiden, sodass das Programm diese Abweichung nicht unbedingt erkennen muss.[Hal]
- Auch EUD kann eine steile Lernkurve erfordern. Einige Tools, die Hersteller den Benutzern zur Verfügung stellen, ähneln sogar denen, die Entwickler direkt benutzen und sind dementsprechend nicht leicht verwendbar.[MD17]

2.2 Diagrammatische VQLs

Diagrammatische VQLs visualisieren die Datenbankabfrage häufig als Graph. Häufig stehen Knoten für Objekte und Attribute, Kanten dabei für Beziehungen zwischen diesen. Dabei kann ein Graph entstehen, welcher einem Entity-Relationship-Diagram ähnelt.[CDT08]
Oracle SQL Developer ist beispielsweise ein Tool zum Erstellen von Datenbankabfragen, welches Diagramme verwendet. Will man so alle Nutzer ermitteln, welche ein Quiz mit maximal einem Fehler bearbeitet haben, könnte dies so ähnlich aussehen wie in Abbildung 2.1.[Cor]

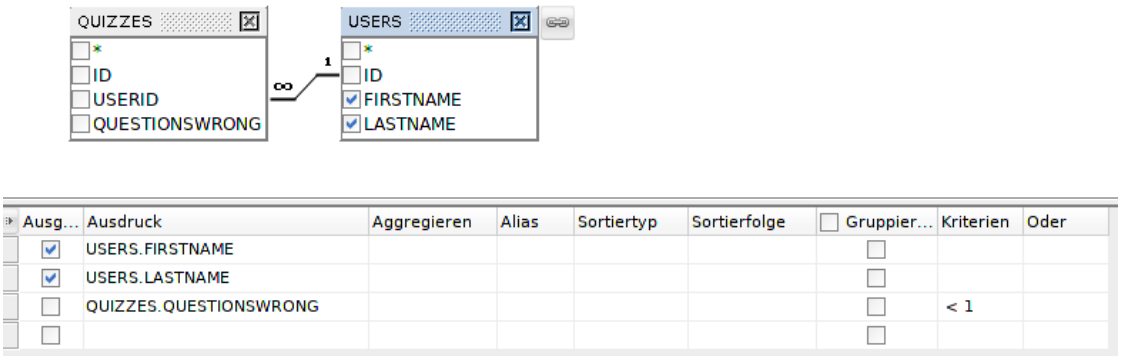


Abbildung 2.1 Beispielabfrage in Oracle SQL Developer

Hierbei sei folgendes Feature hervorgehoben: Das Programm erkennt, wenn zwei Objekte mit Fremdschlüsseln miteinander verbunden werden können und erstellt daraufhin selbstständig ein Join. Sollte dieses Verhalten nicht erwünscht sein, kann dieses Join auch wieder gelöscht werden. Dies funktioniert allerdings nur automatisch, wenn der Fremdschlüssel existiert.
Einige „iconbasierte“ VQLs nutzen dagegen den Ansatz, Icons anstelle von abstrakten Formen zu verwenden. Dies ist auch mit diagrammatischen VQLs kombinierbar.[CDT08]

2.3 Tabellarische VQL am Beispiel von QBE

Tabellarische VQLs können insbesondere dann verwendet werden, wenn die angefragten Objekte alle dieselbe Struktur haben, zum Beispiel wenn das relationale Datenbankmodell verwendet wird. Ein bekannter Vertreter dieser Art ist Query By Example (QBE):[CDT08]
Man stellt eine Abfrage, indem man Tabellen exemplarisch ausfüllt. Dabei kann man „constant elements“ und „example elements“ verwenden. „Example elements“ werden üblicherweise unterstrichen dargestellt. Sie sind mit Variablen vergleichbar und stehen für (fast) beliebige andere Werte. Wird ein „P“ vorangestellt, bedeutet dies, dass diese Variable Teil der Ausgabe ist. „Constant elements“ stellen dagegen Bedingungen dar und werden zum Filtern und Suchen verwendet. Auch Operatoren können unterstützt werden.[Zlo77]

Würde man zum Beispiel die Namen aller Nutzer suchen wollen, die ein Quiz mit maximal einem Fehler beendet haben, könnte dies in etwa wie in Tabelle 2.1 aussehen.

USERS	ID	FirstName	LastName
	userid	P Max	P Mustermann

QUIZZES	ID	UserID	QuestionsWrong
		userid	<1

Tabelle 2.1 Beispielabfrage formuliert mit QBE

> „In other words, since the entries are bound to the table skeleton, the user can only specify admissible queries.“ [Zlo77]

Wie dieses Zitat andeutet, hilft QBE dem Benutzer dabei valide Abfragen zu schreiben, indem es die Benutzer an die Struktur der Tabellen bindet. Dies wäre ein Vorteil gegenüber textbasierten Sprachen wie SQL, bei der es möglich wäre, die Satzstruktur zu verletzen, beispielsweise indem die Reihenfolge der Komponenten „WHERE“ und „ORDER BY“ vertauscht werden. Nach wie vor würde QBE aber nicht syntaktische Korrektheit erzwingen.[Zlo77] Eben dies schlagen Alexander Repenning und Andri Ioannidou in ihren „13 Design Guidelines“ für End-User Development vor, um dem Endbenutzer die Hürde zu nehmen, syntaktische Fehler aufzuspüren und zu beheben:

> „1. Make syntactic errors hard
> 2. Make syntactic errors impossible“ [RI06]

2.4 Google Blockly, Code.org und Scratch

Scratch[MIT] und Code.org[Cod] sind zwei voneinander unabhängige, aber recht ähnliche auf Google Blockly basierende Projekte, die darauf abzielen, Kindern die Grundlagen des Programmierens mithilfe einer blockbasierten Programmiersprache zu vermitteln. Es werden Programme erstellt, welche Cartoon-Figuren durch Labyrinthe steuern oder tanzen lassen, Zeichnungen mittels Turtle-Grafik zeichnen oder kleine Spiele erstellen.

2.4.1 Google Blockly

Blockly ist eine Bibliothek von Google, welche es erlaubt, einen Code-Editor für block basierte Sprachen in Webseiten oder Anwendungen einzubinden. Blockly ist keine visuelle Programmiersprache, sondern erlaubt die Erstellung eigener Blöcke und Sprachen. Auch die Übersetzung der Blöcke in andere Sprachen wird unterstützt.[LLC]

2.4.2 Tutorials von Code.org

Code.org bietet Kurse für verschiedene Altersgruppen an. Diese sind weiter in Level aufgeteilt, welche aus Aufgaben einer der oben genannten Kategorien aufgebaut sind und durch Videos und Quizfragen ergänzt werden.
Die Level und Aufgaben bauen aufeinander auf. So sind zu Beginn beispielsweise nur Blöcke zum Bewegen verfügbar, mit denen nur einfache feste Abfolgen erstellt werden können (vgl. Abb. 2.2). Nach und nach werden weitere Features hinzugefügt (vgl. Abb. 2.3 und Abb. 2.4). Dabei werden einmal eingeführte Blöcke immer weiter zur Verfügung gestellt, auch wenn nicht alle für die Bearbeitung der jeweils nächsten Aufgabe erforderlich wären.

Abbildung 2.2 Aufgabe am Ende eines Levels, mit stark reduzierter Auswahl von Blöcken

Abbildung 2.3 Aufgabe am Ende eines Levels, nachdem weitere Blöcke hinzugefügt worden sind

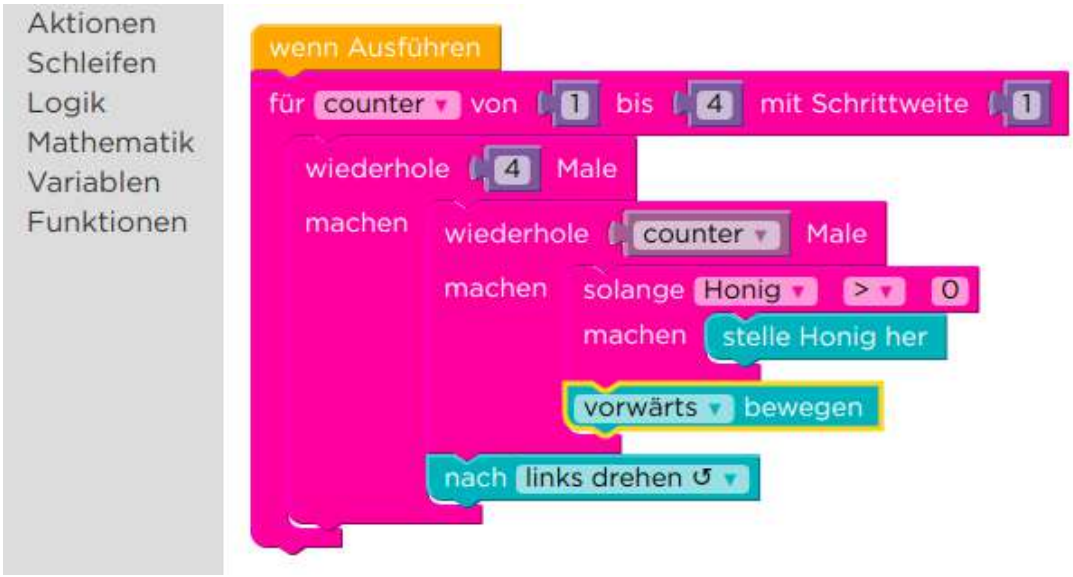


Abbildung 2.4 Aufgabe am Ende eines Kurses mit einer großen Auswahl von Blöcken

Dadurch ergibt sich, dass einige der ersten Aufgaben effizienter, also mit weniger Blöcken, lösbar wären, wenn diese schon für diese zur Auswahl gestanden hätten. In der Aufgabe, die in Abb. 2.5 dargestellt wird, hätten die drei zusammenhängenden Blöcke zum Vorwärtsbewegen mit einem Block weniger gelöst werden können, wenn eine Schleife verwendet worden wäre. Im Rahmen vom Tutorial wird demnach der Funktionsumfang und damit die Ausdrucksstärke reduziert, um dem Endbenutzer die Verwendung zu erleichtern.

Abbildung 2.5 Aufgabe, in der Schleifen noch nicht verfügbar waren

2.4.3 Blöcke wie Legosteine verbinden

> „As with Lego bricks, connectors on the blocks suggest how they should be put together."
>
> [Res+09]

Vor der Entwicklung von Scratch hat die „Lifelong Kindergarten research group" des MIT Media Lab mit Lego an der Entwicklung von Lego Mindstorms zusammengearbeitet.[Res+09] Lego Mindstorms ist ein Roboterbausatz bestehend aus Sensoren, Motoren, Lego-Technik Bausteinen und einem zentralen programmierbaren Block, welcher mit einer visuellen Programmiersprache programmiert werden kann. Hierbei werden Blöcke so verbunden, dass sie eine Art Ablaufdiagramm bilden.[Gro] Bei der Erstellung von Scratch wurde das Ziel verfolgt, beim Programmieren ein ähnliches Gefühl zu erzeugen, wie beim Bauen mit Lego Bausteinen: Die Form der Bausteine, insbesondere der Verbindungsstücke, soll darauf hinweisen, wie sie zu verbinden sind. Dabei sollen Verbindungen durch das Programm verhindert werden, die syntaktisch keinen Sinn ergeben.[Res+09] Dies erfüllt auch die ersten beiden Tipps, welche Alexander Repenning und Andri Ioanning in ihren 13 Guidelines geben: Syntaktische Fehler sollten schwerer gemacht werden oder sogar ganz unmöglich.[RI06] Dazu kommt die Erfüllung ihres dritten Tipps: Es sollten Objekte anstelle von sprachlichen, bzw. textuellen, Elementen verwendet werden. Sie bringen ein Beispiel an und empfehlen Verbindungsstücke, die ähnlich geformt sind wie Puzzleteile, etwa wie in BLOX Pascal oder Lego RCX für den Lego Mindstorms Roboter.[RI06] Auch Code.org und Scratch verwenden teilweise Puzzleteil-ähnliche Verbindungsstücke.[Res+09][Cod] Dazu kommen bei Scratch auch andere Formen, welche zurückgegebene Typen trennen.[Res+09]

Kapitel 3

Entwicklung

In diesem Kapitel werden die wichtigsten Schritte, Entscheidungen und Ergebnisse bei der Entwicklung der Datenbankabfragesprache dokumentiert.

3.1 Die Datenbank

Die Abfragesprache soll verwendet werden, um Freischaltbedingungen für die QuizCards Lösung zu formulieren. Für den Zugriff auf die QuizCards Datenbank sollen Views eingesetzt werden, um die Datenstruktur zu vereinfachen. So sollen zum Beispiel Nullwerte komplett vermieden werden, und stattdessen Standardwerte verwendet werden.
Es soll erst einmal nur ein Teil der Datenbank unterstützt werden. Dies geschieht aus mehreren Gründen: Zum Einen sind die Daten aus einigen Tabellen nicht relevant für Erfolge, da sie keine hierfür interessanten Nutzeraktivitäten und Ergebnisse beinhalten, sondern zum Beispiel nur Zugriffsberechtigungen. Zum Anderen bietet die vollständige Unterstützung keinen Mehrwert im Bezug auf die Beantwortung der Fragen, die diese Bachelorarbeit beantworten soll, da die Abfragesprache auch begrenzt auf einen Teil der Datenbank präsentiert werden könnte. Das wiederum hat auch den Vorteil, dass die Testnutzer nicht erst viel Zeit aufwenden müssen, um sich in eine größere, aber vollständige Datenstruktur einzulesen.
Der Teil, welcher im Rahmen dieser Arbeit unterstützt werden soll, umfasst einen Spielmodus, die Multi-Gruppen-Challenge. In diesem Spielmodus versuchen Spieler im Rahmen von Challenges, aufgeteilt auf mehrere Teams, eine Zielpunktzahl möglichst schnell zu erreichen. Da die Gruppen unterschiedlich groß sind, ist diese Zielpunktzahl von der Größe eines Teams abhängig. Das Team, welches dies zuerst

schafft oder nach Ablauf der Challenge relativ zur Anzahl der Spieler die meisten Punkte erspielen konnte, gewinnt die Challenge.

3.1.1 Datenbankstruktur

3.1.1.1 Spieler

Viele der Datentypen von QuizCards sind mit Spielern verknüpft. Die entsprechenden Fremdschlüssel verweisen auf die Nutzer-Tabelle von Moodle, dessen Nutzerverwaltung vom eLMS und von QuizCards verwendet wird. Über diese Tabelle hinaus verwendet QuizCards allerdings noch eine andere Tabelle, um weitere Nutzerdaten für QuizCards bereitzustellen. Dazu gehört auch ein weiterer Nutzername, der Alias, welcher innerhalb von QuizCards auch anderen Benutzern gezeigt werden darf. Der Benutzername von Moodle darf aus Datenschutzgründen keinem anderen Spieler gezeigt werden. Außerdem wird die Gesamtpunktzahl des Nutzers in dieser zusätzlichen Tabelle abgelegt.
Die anderen Felder beider Tabellen sind für Erfolge nicht relevant, denn dies sind insbesondere persönliche Daten und Einstellungen des Nutzers.
Für die Benutzer der Abfragesprache sollen beide Tabellen, wie in Abbildung 3.1 gezeigt, zusammengeführt werden.

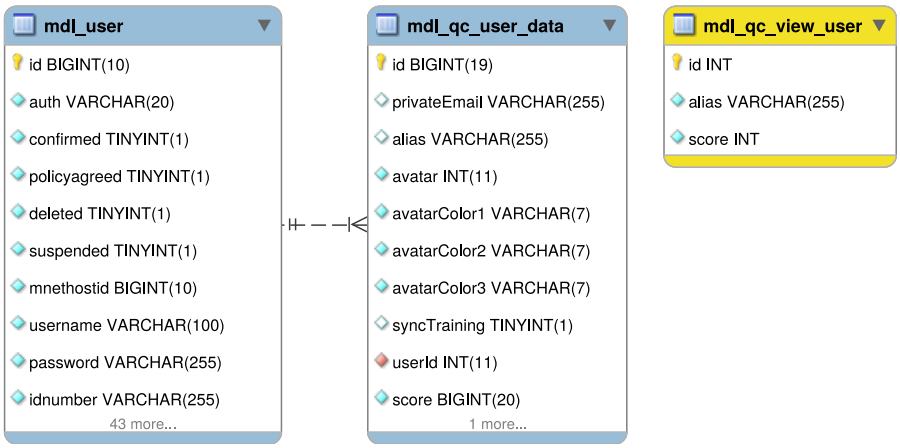


Abbildung 3.1 Die beiden ursprünglichen Benutzer-Tabellen und die View für den Benutzer.

3.1.1.2 Multi-Gruppen-Challenge

Die Multi-Gruppen Challenge besteht aus einer Vielzahl an Tabellen um die Konfiguration der Challenge und Ergebnisse zu repräsentieren.
Kern der Multi-Gruppen-Challenge ist eine Tabelle, die einige Konfigurationen der Multi-Gruppen-Challenge beinhaltet. Viele der anderen Tabellen verweisen per Fremdschlüssel auf eben diese Tabelle. In der Tabelle selbst sind Zeitraum, Name der Challenge und mit wie vielen Punkte eine korrekt beantwortete Frage belohnt werden soll, festgehalten. Die Zielpunktzahl wird in einer anderen Tabelle festgehalten, für eine einfachere Verwendung soll sie aber Teil derselben View werden. Ein weiteres Attribut gibt an, ob die Challenge bereits an die Teilnehmer ausgerollt worden ist. Nur bei ausgerollten und gestarteten Challenges kann Nutzeraktivität vorhanden sein. Daher werden nur diese Challenges Teil der Views.
Eine weitere Tabelle hält fest, aus welchen Fragekategorien die Quizzes der Challenge gebildet werden sollen und wie viele Fragen jeweils entnommen werden. Da von den Fragekategorien nur der Name potentiell für Erfolge relevant ist, wird auch nur dieser aus den Fragekategorien in die View für die ausgewählten Kategorien der Challenge übernommen.
Welche Spieler an Challenges teilnehmen und für welche Gruppen sie antreten wird mithilfe der Quizzes bestimmt. Da die Quizzes erst einmal der Abfragesprache nicht zur Verfügung stehen, werden sie derzeitig nur verwendet, um Views zu erzeugen, die Assoziationen zwischen Nutzer und Challenge, bzw. Nutzer und Gruppe darstellen.
Wie diese Tabellen für den Nutzer zusammengefasst werden, zeigt Abbildung 3.2 noch einmal im Detail.

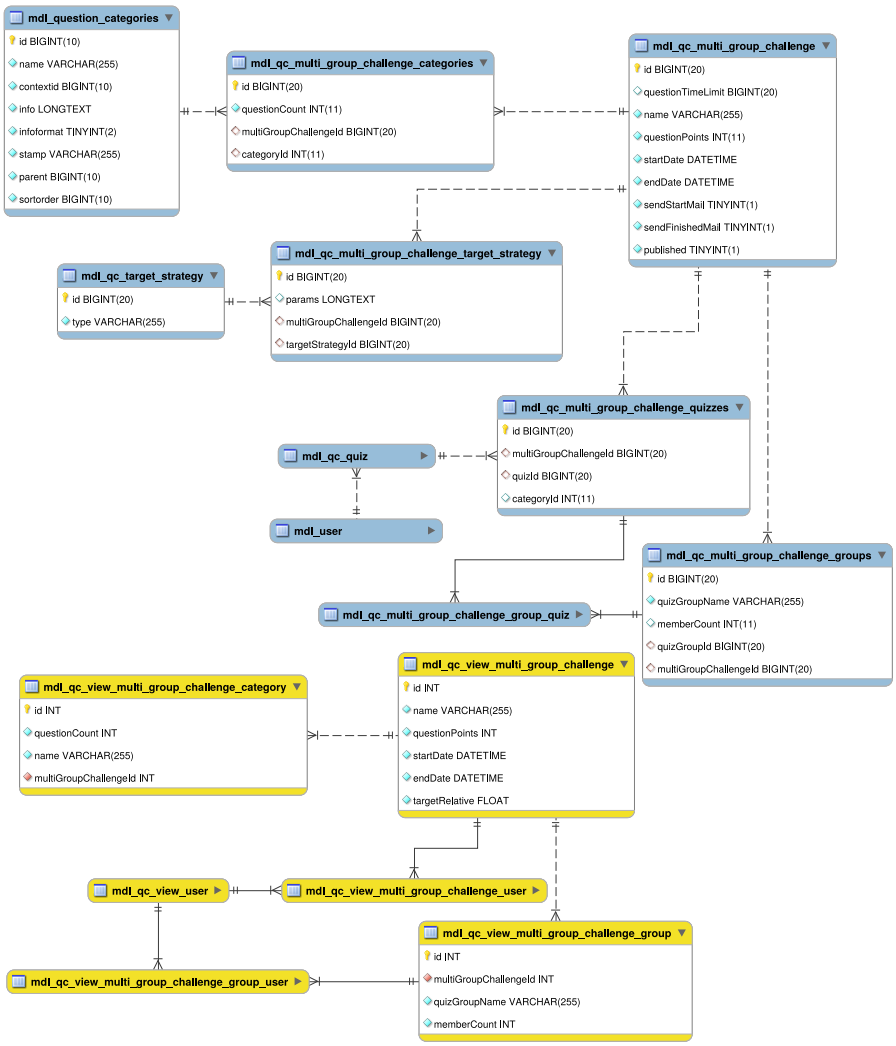


Abbildung 3.2 Die Multi-Group-Challenge mit Gruppen und Nutzern.

3.1.1.3 Multi-Gruppen-Challenge Ergebnisse

Darüber hinaus werden in drei weiteren Tabellen die Ergebnisse der Challenge abgelegt: Die erste Tabelle wird verwendet, um pro Kategorie, Challenge und Nutzer, Ergebnisse abzulegen. Die zweite fasst diese pro Kategorie, Challenge und Gruppe zusammen. In der dritten Tabelle werden letztendlich die Ergebnisse pro Challenge und Gruppe gesammelt.
Die Tabellen sind ähnlich aufgebaut: Alle bestehen aus einer Zielpunktzahl und Zählern für Fragen- und Antwortstatistiken, sowie Fremdschlüssel auf die Chal-

lenge. Dazu kommt, je nach Bezug des Ergebnisses, ein Fremdschlüssel auf Nutzer oder Gruppen, sowie ggf. auf Kategorien.
Einige weitere Attribute sind in einzelnen Tabellen vertreten. Abbildung 3.3 zeigt, um welche Attribute es sich handelt. Vereinigt oder ganz gestrichen werden hier keine Tabellen, da sämtliche Informationen aus den Tabellen dem Anwender bekannt sind, da unter anderem über die Kategorie-Ergebnisse bereits Statistiken erhoben werden und die Endresultate aus der Zusammenfassung gleich an mehreren Stellen angezeigt werden.
Stattdessen wird hier eine neue Tabelle hinzugefügt. Auch eine Zusammenfassung pro Challenge und Nutzer kann hier sinnvoll sein. Diese besteht aus den üblichen Attributen und Verweisen auf den Benutzer und die Challenge.
Abbildung 3.3 zeigt im Detail, aus welchen Informationen die Views der Ergebnisse bestehen.

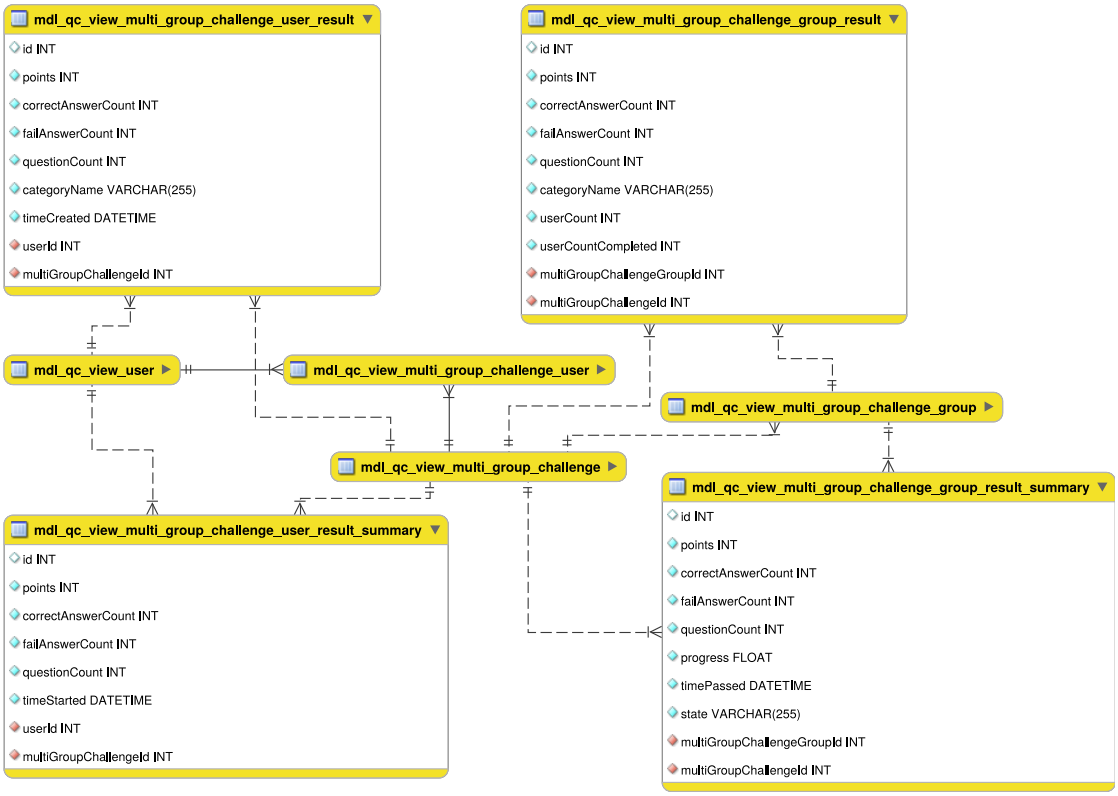


Abbildung 3.3 Nutzersicht auf die Ergebnisse der Multi-Group-Challenges

3.2 Die Abfragesprache

Die Abfragen sollen ähnlich aufgebaut sein wie Formeln der Prädikatenlogik erster Stufe:
Es stehen Quantoren, Junktoren und Prädikate zur Verfügung, um aus (atomaren) Formeln komplexere Formeln zu bilden. Atomare Formeln bestehen aus einem Vergleich zweier Werte: Werte können Zahlen, Texte oder Daten mit Uhrzeiten sein. Darüber hinaus sind Funktionen verfügbar, die Werte als Parameter nehmen und Werte zurückgeben.
Das Universum der Struktur, auf die sich die Formeln beziehen, würde dann die Menge der Tupel der Datenbank und der Werte, egal ob sie in der Datenbank vorkommen oder nicht, umfassen.
Die Variablen werden auf die Tupel der Variablen begrenzt. Um an Werte zu gelangen, müssen daher Funktionen verwendet werden. Einige Funktionen dienen dazu, Attribute aus Tupeln auszulesen.
Da sich die Abfragen auf einen Nutzer der Lernapp beziehen sollen, wird für diesen die freie Variable „Aktiver Nutzer“ verwendet.
Über die Prädikatenlogik erster Stufe hinaus sollen aber auch noch Aggregatfunktionen verwendet werden können.

3.2.1 Verfügbare Blöcke

3.2.1.1 Kategorie „Datenbank“

Abbildung 3.4 Übersicht über die Kategorie „Datenbank“

In dieser Kategorie sind Blöcke untergebracht, die die Funktionen des All- und des Existenzquantor erfüllen. Mit dem Verwenden eines Quantors wird auch direkt die Datenbanktabelle festgelegt, aus welcher das Tupel stammen soll. Der Name der Variable ist frei wählbar. Beiden Quantoren wird eine weitere Bedingung untergeordnet, dort ist dann die Variable, die von dem Quantor gebunden wird, verfügbar. Neben den Blöcken für die Quantoren gibt es noch einen weiteren Block, welcher Variablen bindet. Dieser ist für Aggregat-Funktionen zuständig. Wie bei den Quantoren bindet er eine Variable für eine untergeordnete Bedingung und legt gleich ihren Typ fest. Die untergeordnete Bedingung dient als Filter. Auch wird eine Aggregatfunktion aus den folgenden sechs gewählt:

1. Zählen von Tupeln
2. Zählen von einzigartigen Werten
3. Minimum
4. Maximum
5. Summe
6. Durchschnitt

Mit Ausnahme der ersten Funktion erfordern die Funktionen noch einen Parame-

ter. Für diesen erscheint dann automatisch ein weiteres Verbindungsstück, falls eine von diesen Funktionen gewählt wurde. Es wird je ein Wert für jedes nicht herausgefilterte Tupel generiert und all diese Werte werden von der gewählten Funktion zusammengefasst.
Als letzter Block der Kategorie steht der Vergleich zweier Variablen zur Verfügung, mit dem geprüft werden kann, ob sich zwei Variablen auf dasselbe Tupel beziehen oder nicht.
Bei den ersten drei Blöcken werden Mutatoren von Blockly verwendet, um den Block dynamisch erweitern zu können. Dies wird genutzt, um Variablen miteinander zu verknüpfen. Dies kann auf zwei verschiedene Arten geschehen:

1. Die Variable wird von einer Anderen unterschieden. Damit beschränken sich die Blöcke auf alle Tupel, auf die sich diese andere Variable gerade nicht bezieht. Dafür müssen beide Variablen sich auf Tupel derselben Tabelle beziehen.
2. Die Variable wird auf Tupel beschränkt, die mit dem Tupel der anderen Variable assoziiert werden. Je nachdem aus welchen Tabellen die Tupel stammen, können andere oder vielleicht gar keine Assoziationen zur Auswahl stehen. Diese Funktion entspricht damit dem automatischem Join in Oracle SQL Developer. Siehe hierfür auch Abschnitt 2.2.

Die letzten drei Zeilen der Tabelle 3.1 zeigen, wie diese zusätzlichen Verknüpfungen einzeln übersetzt werden. Mehrere Verknüpfungen werden mit „Und“ verbunden und der untergeordneten Bedingung bei der Übersetzung nach SQL beigefügt. Hierfür wird bei dem Allquantor die Implikation verwendet, für den Existenzquantor und die Aggregatfunktionen die Konjunktion, um die Tupel korrekt einzuschränken.
Abbildung 3.4 zeigt, welche Blöcke existieren und Tabelle 3.1, wie diese übersetzt werden.

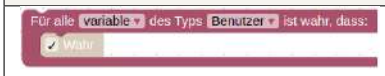
Block	SQL
	NOT EXISTS (SELECT 1 FROM user variable WHERE NOT (1))
	EXISTS (SELECT 1 FROM USER variable WHERE (1))
	SELECT COUNT(1) FROM user variable WHERE (1)
	SELECT SUM(1) FROM user variable WHERE (1)
	activeUser.id=activeUser.id
	A.id!=B.id
	E.userId=A.id
	EXISTS (SELECT 1 FROM challenge_user QueryBuilderAssoc WHERE QueryBuilderAssoc.multiGroupChallengeId=C.id AND QueryBuilderAssoc.userId=A.id)

Tabelle 3.1 Beispielhafte Übersetzung der Blöcke der Kategorie „Datenbank“ nach SQL.

3.2.1.2 Kategorie „Logik“

Abbildung 3.5 Übersicht über die Kategorie „Logik“

Diese Kategorie umfasst Blöcke, welche dieselben Funktionen erfüllen wie Junktoren der Aussagenlogik. Die Blöcke werden in Abbildung 3.5 gezeigt. Auch wenn schon ein Teil der Junktoren aufgrund von funktionaler Vollständigkeit für die gleiche Ausdrucksstärke der Sprache ausreichend gewesen wäre, wurden dennoch alle gezeigten Junktoren für eine einfachere Verwendung implementiert.
Bei den Junktoren „Und“ und „Oder“ wurden, wie auch schon bei den Blöcken aus der vorherigen Kategorie, Mutatoren verwendet. In diesem Fall werden Mutatoren verwendet, um den die Blöcke in Einzelteile aufzuteilen, dies wird in Abbildung 3.6 gezeigt. Es können weitere Einzelteile hinzugefügt werden, um mit demselben Block mehr als zwei Aussagen miteinander zu verknüpfen und nicht mehrere Blöcke verschachteln zu müssen. Es werden immer mindestens zwei Aussagen miteinander verknüpft, da beide Blöcke mit weniger Aussagen sinnlos wären.

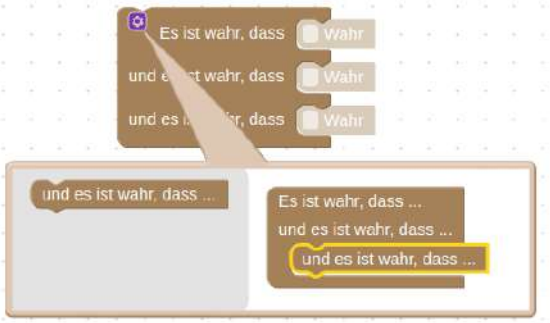


Abbildung 3.6 Mutator zum Hinzufügen von weiteren Aussagen

Tabelle 3.2 zeigt, wie die Blöcke dieser Kategorie übersetzt werden.

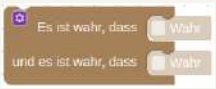
Block	SQL
	1
	(0) AND (0)
	(0) OR (0)
	NOT (0) OR (0)
	(0) = (0)
	NOT (0)

Tabelle 3.2 Beispielhafte Übersetzung der Blöcke der Kategorie „Logik“ nach SQL.

3.2.1.3 Kategorie „Zahlen“

Abbildung 3.7 Übersicht über die Kategorie „Zahlen“

Unter dieser Kategorie finden die Nutzer Blöcke um mit Zahlen zu hantieren. Ganze Zahlen und Fließkommazahlen werden nicht getrennt betrachtet.
Welche Blöcke existieren, wird von 3.7 gezeigt und in den folgenden Absätzen beschrieben.
Der erste Block dient dazu, manuell Werte einzutragen. Blockly verwendet eigentlich einen Dezimalpunkt, da in Deutschland allerdings das Dezimalkomma üblich ist, wurde dies geändert.
Der zweite Block liest Werte aus Objekten aus. Im ersten Feld wird eine Variable ausgewählt, im Zweiten der konkrete Wert. Ist eine Variable an dieser Stelle nicht verfügbar, beispielsweise weil kein übergeordneter Block diese definiert oder aufgrund des Typs keine Zahlen existieren, die ausgelesen werden können, wird die Variable gar nicht erst zur Auswahl zur Verfügung gestellt. Führt dies dazu, dass keine Variable verfügbar ist, färbt sich der Block rot und es wird ein Icon mit einer Fehlermeldung angezeigt. Sobald Variablen und Werte zur Verfügung stehen, wird automatisch der jeweils erste ausgewählt, der Block färbt sich wieder blau und die Fehlermeldung verschwindet.
Mit dem dritten Block werden zwei Zahlen miteinander verglichen. Da das Ergebnis ein Wahrheitswert und keine Zahl ist, unterscheidet sich das Verbindungsstück von den anderen Blöcken. Über das Feld in der Mitte werden verschiedene Vergleiche gewählt. Zur Auswahl stehen:

- Gleichheit
- Ungleichheit
- Kleiner
- Kleiner oder gleich
- Größer
- Größer oder gleich

Es werden die hierfür bekannten Symbole anstatt Texte gewählt.
Der vierte Block bietet dem Nutzer die Grundrechenarten und das Potenzieren.
Einige mathematische Funktionen mit nur einem Parameter bietet der fünfte Block.
Der sechste Block dient zum kaufmännischen Runden ohne Nachkommastelle sowie zum Auf- und Abrunden.
Mit dem letzten Block kann das Ergebnis der Modulo-Division ermittelt werden.
Für die Übersetzung nach SQL werden Literale, Funktionen und Operatoren aus dem SQL-Standard oder von MySQL verwendet, wie in Tabelle 3.3 gezeigt.

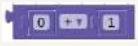
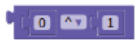
Block	SQL
	123
	variable.score
	(0)<(1)
	(0)+(1)
	POWER(0,1)
	SQRT(1)
	-(1)
	POWER(10,1)
	ROUND(1)
	MOD(7,3)

Tabelle 3.3 Beispielhafte Übersetzung der Blöcke der Kategorie „Zahlen“ nach SQL.

3.2.1.4 Kategorie „Datum und Uhrzeit“

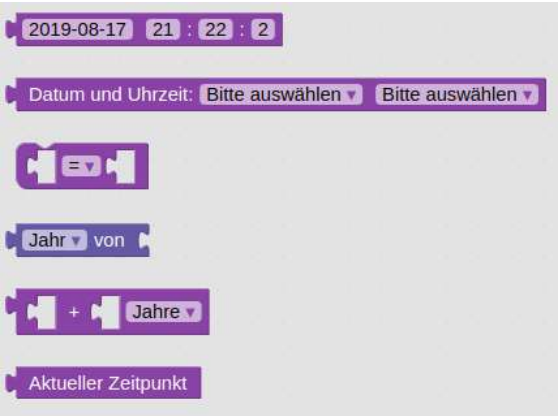


Abbildung 3.8 Übersicht über die Kategorie „Datum und Uhrzeit“

Die ersten drei Blöcke der Kategorie, welche in Abbildung 3.8 gezeigt werden, ähneln in ihrer Funktion denen der Kategorie „Zahlen“:

1. Ein Block zur manuellen Eingabe von Daten. Durch die Einbindung des Kalenders im Feld für das Datum und minimale und maximale Werte bei den Feldern für die Uhrzeit werden ungültige Eingaben vermieden.
2. Ein Block zum Auslesen von Werten.
3. Ein Block zum Vergleichen von zwei Werten. Es sind auch die gleichen Vergleiche wie in der Kategorie „Zahlen“ verfügbar. Dennoch werden verschiedene Blöcke in den Kategorien verwendet, um klarzustellen, welche Art von Werten verglichen werden.

Mithilfe des vierten Blocks werden die einzelnen Bestandteile des Datums oder der Uhrzeit extrahiert. Neben Tag, Monat und Jahr steht für das Datum auch der Wochentag zur Verfügung. Da dieser Block im Gegensatz zu den Anderen aus dieser Kategorie eine Zahl statt eines Datums zurück gibt, ist dieser blau statt violett gefärbt.
Der fünfte Block dient zur Manipulation von Daten. Es können Sekunden, Minuten, Stunden, Tage, Monate oder Jahre hinzugefügt oder abgezogen werden.
Der letzte Block der Kategorie dient dazu, immer den aktuellen Zeitpunkt zurück zu geben, egal wann der Ausdruck ausgewertet wird.

Wie von Pasternak, Fenichel und Marshall empfohlen, werden gleiche Farben verwendet, um Ähnlichkeiten zwischen den Blöcken darzustellen. Bei den Blöcken, die mit Werten arbeiten, stehen Farben für die Art der Werte. Blau steht für Zahlen, Violett für Daten und Cyan für Texte.[PFM17]
Wie die soeben erklärten Blöcke übersetzt werden, wird in Tabelle 3.4 gezeigt.

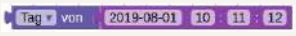
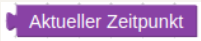
Block	SQL
	'2019-08-01 10:11:12'
	DAY('2019-08-01 10:11:12')
	DATE_ADD(('2019-08-01 10:11:12'), INTERVAL (2) DAY))
	NOW()

Tabelle 3.4 Beispielhafte Übersetzung der Blöcke der Kategorie „Datum und Uhrzeit“ nach SQL.

3.2.1.5 Kategorie „Texte“

Abbildung 3.9 Übersicht über die Kategorie „Texte“

Wie auch schon bei den vorherigen beiden Kategorien gibt es auch in dieser Kategorie, wie sie in Abbildung 3.9 gezeigt wird, wieder drei übliche Blöcke:

1. Ein Block zur manuellen Eingabe von Texten.
2. Ein Block zum Auslesen von Werten.

3. Ein Block zum Vergleichen von zwei Texten. Zwar könnte man z. B. Texte lexikographisch ordnen und auf diese Art miteinander vergleichen, aber dies bietet hier keinen Mehrwert. Stattdessen kann anstelle der Ordnung zwischen zwei Texten relevant sein, ob ein Text in einem Anderen vorkommt.

Neben frei gewählten Namen von Challenges, Gruppen, Kategorien oder Nutzern, kommen in der Datenbank Texte auch in Form des Bearbeitungsstatus von Challenges vor. Hierbei gibt es einige fest definierte Werte. Daher gibt es noch einen vierten Block in der Kategorie, der dazu dient, diese auszuwählen. Aus der Beschreibung der Datenbankstruktur werden diese extrahiert. Gäbe es noch weitere ähnliche Spalten im Datenbankschema, bei denen eine fest definierte Menge an Texten vorkommt, wären diese auch mit extrahiert worden. Zur Trennung der Mengen werden diese nach der Spalte, in der sie vorkommen, gruppiert. Diese Gruppe ist im ersten Auswahlfeld dieses Blockes auswählbar. Im zweiten Auswahlfeld wird dann der Wert selbst gewählt.
Die Übersetzung der Blöcke wird in Tabelle 3.4 demonstriert.

Block	SQL
	'Text'
	0<INSTR('Text','Te')
	0=INSTR('Text','Tes')
	'passed'

Tabelle 3.5 Beispielhafte Übersetzung der Blöcke der Kategorie „Texte“ nach SQL.

3.2.2 Umsetzung mit Blockly

3.2.2.1 Implementierung eines Blocks

Die meisten der Blöcke sind im Rahmen dieser Bachelorarbeit implementiert worden. Einige Ausnahmen befinden sich in der Kategorie „Zahlen“ und existierten

bereits im Vorfeld der Arbeit als Teil von Blockly. Für diese musste nur die Übersetzung nach SQL implementiert werden. Die anderen Blöcke mussten erst definiert werden, wie am Beispiel vom Block „conjunction“ im Listing 3.1 gezeigt wird.

```
Blockly.Blocks['conjunction'] = {
    init: function () {
        this.subCount = 2;
        this.setPreviousStatement(true, 'Boolean');
        this.setColour(30);
        this.setTooltip('');
        this.setHelpUrl('docs/blocks_logic/index.html#und');
        this.setMutator(new Blockly.Mutator(['conjunction_clause']));
        this.updateShape();
    },
    domToMutation: function (container) {
        this.subCount = container.getAttribute('subcount');
        this.updateShape();
    },
    mutationToDom: function () {
        let container = document.createElement('mutation');
        container.setAttribute('subcount', this.subCount);
        return container;
    },
    compose: function (block) {
        let clauseBlock = block.getInputTargetBlock('clauses');
        let counter = 2;
        while (clauseBlock) {
            counter++;
            // Either false or a valid next block
            clauseBlock = clauseBlock.nextConnection && clauseBlock.↩
                nextConnection.targetBlock();
        }
        this.subCount = counter;
        this.updateShape();
    },
    decompose: function (workspace) {
        let topBlock = workspace.newBlock('conjunction_clauses');
        topBlock.initSvg();
        let connection = topBlock.getInput('clauses').connection;
        for (let i = 2; i < this.subCount; i++) {
            let clauseBlock = workspace.newBlock('conjunction_clause');
            clauseBlock.initSvg();
            connection.connect(clauseBlock.previousConnection);
            connection = clauseBlock.nextConnection;

        }
        return topBlock;
    },
    updateShape: function () {
```

```
45          this.render();
46          let i;
47          for (i = 0; i < this.subCount; i++) {
48              if (!this.getInput('sub' + i)) {
49                  let input = this.appendStatementInput('sub' + i)
50                                  .appendField(i === 0 ? 'Es ist wahr, dass' : 'und↩
                                        es ist wahr, dass');
51                  let shadowInput = this.workspace.newBlock('boolConstants');
52                  shadowInput.setShadow(true);
53                  shadowInput.initSvg();
54                  input.connection.connect(shadowInput.previousConnection);
55                  shadowInput.render();
56
57              }
58          }
59          // Remove deleted inputs.
60          while (this.getInput('sub' + i)) {
61              this.removeInput('sub' + i);
62              i++;
63          }
64      }
65  };
66
67  Blockly.Sql['conjunction'] = function (block) {
68      let subClausesAsStrings = [];
69      for (let i = 0; i < this.subCount; i++) {
70          if (this.getInputTargetBlock('sub' + i)) {
71              subClausesAsStrings.push('(' + Blockly.Sql.statementToCode(block, '↩
                    sub' + i) + ')');
72          }
73      }
74      return subClausesAsStrings.join(' AND ');
75  };
76
77  Blockly.Blocks['conjunction_clauses'] = {
78      init: function () {
79          this.appendDummyInput()
80              .appendField('Es ist wahr, dass ...');
81          this.appendDummyInput()
82              .appendField('und es ist wahr, dass ...');
83          this.appendStatementInput('clauses');
84          this.setColour(30);
85          this.setTooltip('');
86          this.setHelpUrl('');
87      }
88  };
89
90  Blockly.Blocks['conjunction_clause'] = {
91      init: function () {
92          this.appendDummyInput()
```

```
93              .appendField('und es ist wahr, dass ...');
94          this.setPreviousStatement(true);
95          this.setNextStatement(true);
96          this.setColour(30);
97          this.setTooltip('');
98          this.setHelpUrl('');
99      }
100 };
```

Listing 3.1 Blocks/logic/conjunction.js

Die Methode *init* wird aufgerufen wenn der Block erstellt werden soll und legt den Aufbau des Blocks fest. Ein Teil davon ist in dem Beispiel in die Methode *updateShape* ausgelagert, da aufgrund des Mutators der Block häufiger neu aufgebaut werden muss.
Die Methode *mutationToDom* erstellt einen Container, in der alle für die Mutation wichtigen Informationen gesammelt werden, hier beschränkt sich dies auf die Anzahl der Konjunkte. Dieser Container kann im XML Format ausgegeben werden. Die Methode *domToMutation* liest die Informationen aus und baut anhand dessen den Block neu auf.
Die Zerlegung des Blocks in Einzelteile geschieht in den Methoden *decompose* und *compose*. Beim Aufrufen des kleineren Editors, der in Abbildung 3.6 gezeigt wird, wird *decompose* verwendet, um den Inhalt des kleineren Editors festzulegen, *compose* baut den Block anhand der Einzelteile des kleineren Editors neu auf. Damit dieser Editor aufrufbar ist, wird in Zeile 8 die Methode *setMutator* der Blockly Bibliothek verwendet. Der Parameter umfasst alle Blöcke, die in der Seitenleiste des kleineren Editors zur Verfügung stehen. Der Einstiegspunkt im Editor muss hier nicht angegeben werden, da er direkt in der Methode *decompose* erstellt wird. Beide Blöcke müssen allerdings selbst, wie jeder andere Block, implementiert werden. Dies geschieht in den Zeilen 77 bis 100. Aufgrund ihres einfachen Aufbaus genügt die Methode *init*.
Wie der Block übersetzt wird, ist in den Zeilen 67 bis 75 implementiert.
Alle weiteren Implementierungen der Blöcke befinden sich auf dem beiliegenden Datenträger im Ordner „Blocks“.

3.2.2.2 Einbinden des Editors

Der Editor kann mit den implementierten Blöcken auf Webseiten eingebunden werden. Welche Blöcke auf der linken Seite, der sogenannten „Toolbox“, verfügbar sind kann mithilfe einer Beschreibung im XML Format festgelegt werden. Auch wenn Blöcke nicht Teil der Toolbox sind, könnten sie durch Skripte dennoch auf dem Arbeitsbereich platziert werden. Dies wird beim Aufruf des Editors mit dem „Einstiegspunkt“ gemacht.
Der Editor kann dann wie in Abbildung 3.10 gezeigt aussehen.

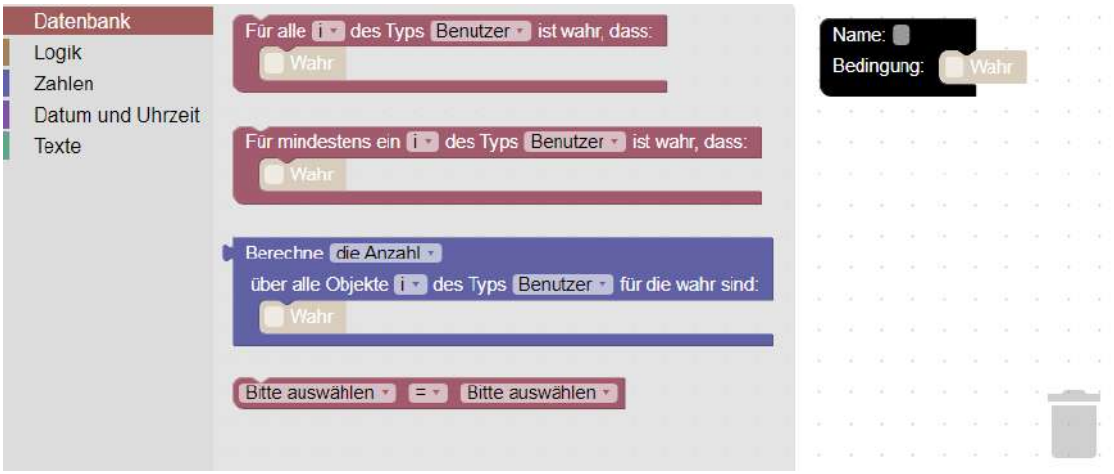


Abbildung 3.10 Der Blockly Editor mit den selbst definierten Kategorien und Blöcken

Dann können Abfragen wie die im Abschnitt 3.2.2.3 Gezeigte vom Nutzer definiert werden.

3.2.2.3 Beispiel einer Abfrage

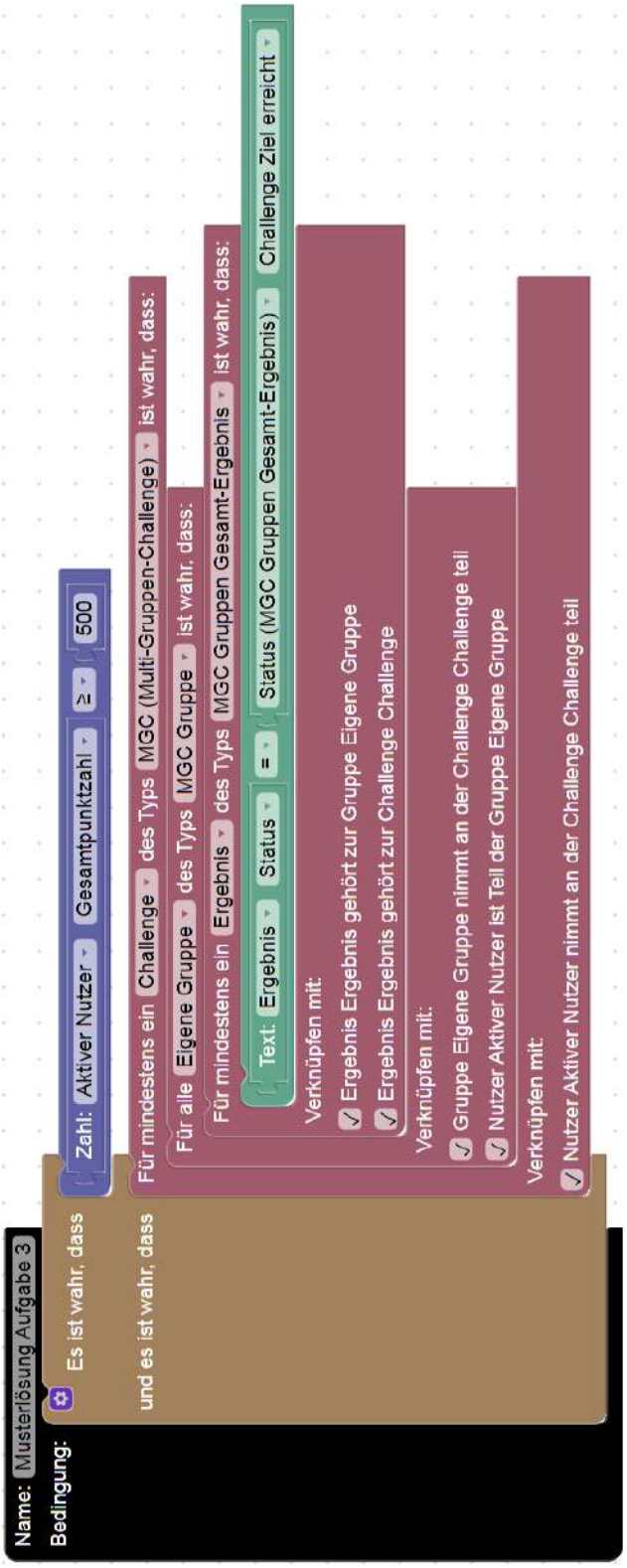


Abbildung 3.11 Ein Beispielerfolg

In der in Abbildung 3.11 gezeigten Abfrage werden zwei voneinander unabhängige Kriterien überprüft:

1. Konnte der Spieler mindestens 500 Punkte erspielen? Hierfür sind die drei Blöcke in blau zuständig.
2. Hat der Spieler an einer Challenge teilgenommen und haben alle Gruppen, für die er angetreten ist, diese bestanden?
   Der äußere rote Block ist dafür zuständig, zu überprüfen, ob es mindestens eine Challenge gibt, die die untergeordnete Bedingung erfüllt. Aufgrund des im unteren Teil des Blocks gesetzten Hakens werden Challenges, an denen der Spieler nicht teilgenommen hat, nicht berücksichtigt.
   Der mittlere rote Block überprüft, ob für alle Gruppen, für die gilt, dass sie an der Challenge teilgenommen haben und der Spieler Mitglied der Gruppe ist, auch die dem Block untergeordnete Bedingung gilt.
   In dem inneren roten Block wird auf das Ergebnis eben dieser Gruppe zurückgegriffen. Ob an dieser Stelle „Für alle“ oder „Für ein“ verwendet wird, spielt keine Rolle, denn es existiert pro Gruppe und Challenge immer genau ein Ergebnis. Aber auch hier wird eine untergeordnete Bedingung geprüft.
   Die Bedingung, ob diese Gruppen bestanden haben, wird mithilfe der Blöcke in Cyan realisiert: Es wird anhand des Feldes „Status“, das sich im Ergebnis finden lässt, überprüft, ob die Gruppe bestanden hat.

Mithilfe des braunen Blocks für die Konjunktion wird ausgedrückt, dass beide Bedingungen erfüllt sein müssen, damit der Spieler den Erfolg freischaltet.
Die Bedingung wird unter dem schwarzen Einstiegspunkt schwarz platziert. Alles, was nicht diesem untergeordnet wird, wird auch nicht berücksichtigt. Außerdem ist unter diesem Block die freie Variable „Aktiver Nutzer“, welcher den Spieler repräsentiert und für den die Bedingung ausgewertet wird, zur Auswahl verfügbar. Ohne den Einstiegspunkt würde nichts funktionieren, daher kann er nicht gelöscht werden.

Kapitel 4

Usability

4.1 Usability Testessen

Um einen ersten Überblick über die Usability der Abfragesprache zu gewinnen, wurde sie beim Usability Testessen vorgestellt. Dies ist eine Veranstaltung, bei der Tester und Unternehmen in entspannter Atmosphäre zusammenkommen, um die Usability von Software zu testen.[glü]
Die Abfragesprache wurde hier von sechs Testern ausprobiert. Dabei sollten diese Abfragen für bis zu fünf Beispielerfolge formulieren:

1. Prüfen, ob der Benutzer mindestens 500 Punkte hat.
2. Prüfen, ob der Benutzer mindestens 500 Punkte hat und an mindestens einer Challenge teilgenommen hat.
3. Prüfen, ob der Benutzer mindestens 500 Punkte hat und alle Gruppen, für die er an einer Challenge teilgenommen hat, diese Challenge auch bestanden haben.
4. Prüfen, ob der Benutzer mindestens 500 Punkte hat und eine Gruppe, für die er an einer Challenge teilgenommen hat, diese Challenge innerhalb eines Tages nach Start bestanden hat.
5. Prüfen, ob der Benutzer mindestens 500 Punkte hat und in einer Challenge in beliebig vielen Gruppen mindestens drei Mitstreiter existieren.

Hierbei sind folgende Probleme deutlich geworden:

- Die Funktion der einzelnen Blöcke wird allein durch die Beschriftung nicht deutlich. Hier wünschten sich die Nutzer mehr Informationen.

- Werden wie in der zweiten Aufgabe weitere Variablen benötigt, wird direkt in den Kategorien nach Stichworten auf den Blöcken gesucht. So wurde auf den Blöcken in der Kategorie „Datenbank“ nicht versucht den Typ im entsprechenden Auswahlfeld zu ändern: Der gesuchte Typ, in Aufgabe 2 die Challenge, wurde nicht direkt in den Kategorien gesehen und der richtige Block daher nicht in Betracht gezogen.
- Es wird versucht, Blöcke mit unterschiedlichen Verbindungsstücken miteinander zu verbinden. Auch wird versucht, den falschen Vergleich für die richtige Art Wert zu verwenden.
- Es wird nicht verstanden, dass wenn auf eine Variable zugegriffen werden soll, es unter dem Block geschehen muss, der diese definiert.
- Es wird nicht verstanden, dass die Variable „Aktiver Nutzer“ unterhalb des Einstiegspunktes verfügbar ist: So wird bei Aufgabe 1 versucht, einen weitere Variable des Typs „Benutzer“ zu definieren, und dessen Punktzahl zu verwenden.

4.2 Maßnahmen nach dem Usability Testessen

4.2.1 Benutzerdokumentation

Die ersten beiden beim Usability Testessen gefundenen Probleme könnte man so zusammenfassen, dass den Benutzern Informationen fehlten. Daher wurde eine Benutzerdokumentation angelegt, welche die Funktion der einzelnen Blöcke beschreibt und auch Beispiele für deren Verwendung gibt. Die Dokumentation von jedem Block kann auch über ein Kontextmenü erreicht werden.
Ebenso wird eine Beschreibung der verfügbaren Typen zusammengestellt, welche auch die Werte aufzählt, aus denen sie bestehen, und mit welchen anderen Typen sie assoziiert werden können. Als Zusammenfassung der Dokumentation der Datenstruktur wurde ein ER-Diagramm eingefügt, welches die Datentypen und ihre Assoziationen zeigt, aber nicht die Attribute, um es klein und übersichtlich zu halten.
Die Dokumentation wurde mit mkdocs[Tea] erstellt. Dieses Tool umfasst ebenfalls eine Suchmaschine, um die Dokumentation nach Stichworten zu durchsuchen, was

ggf. das Problem mit der manuellen Suche, wie es in Problem 2 beschrieben wurde, beschleunigen könnte.

4.2.2 Tutorial

Die anderen drei Probleme betrafen grundlegendere Verständnisprobleme. Um diese zu vermitteln, wurde ein Tutorial aus vier Lektionen erstellt, welche ein paar der zentralen Features der Sprache näherbringen sollten:

1. Verbinden
   Hierbei sollen die Nutzer erst einmal alle auf der Arbeitsfläche befindlichen Blöcke verbinden. Dabei werden beide Arten von Verbindungsstücken gewählt, sowohl der für Formeln, als auch der Puzzleteilförmige für Werte. Bei Letzterem werden über verschiedene Farben die verschiedenen Arten von Werten getrennt. Daher gibt es für jede Art Wert in diesem Tutorial einen Vergleichsblock, der mit konstanten Werten mit der passenden Farbe verbunden werden muss. Bei den Verbindungsstücken für Formeln spielen die Farben keine Rolle.
2. Variablen und Werte
   In diesem Tutorial sollen die Nutzer lernen, wie mit den Blöcken aus der Kategorie „Datenbank“ Variablen definiert werden und wie aus diesen Werte gelesen werden. Es soll für jede Art Wert aus einer beliebigen Variable ein Wert gelesen werden. Dies soll insbesondere Problem vier lösen, da hier der Benutzer dazu gebracht wird, erst die Variable zu definieren und dann den Block zum Auslesen unter diesem zu platzieren. Aber auch Problem zwei wird hier wiederholt: Bei den Blöcken aus der Kategorie „Datenbank“ ist erst der Typ Benutzer ausgewählt. Dieser enthält jedoch kein Datum mit Uhrzeit, sondern nur den Benutzernamen und seine Gesamtpunktzahl, also ein Text und eine Zahl. Damit werden Nutzer hier gezwungen, den Typ zu ändern.
3. Variablen voneinander abgrenzen
   Dies betrifft hauptsächlich Problem 5. In der vorherigen Lektion können die Nutzer zwar auch die Variable „Aktiver Nutzer“ verwenden, aber auch stattdessen nur selbst Definierte. Dies soll hier nachgeholt werden. Hier müssen die Nutzer eine weitere Variable vom Typ Nutzer definieren und die dann von

„Aktiver Nutzer“ unterscheiden.

4. Variablen assoziieren
   Diese Lektion ist der Vorherigen recht ähnlich: Hier sollen die Nutzer eine Variable des Typs „Multi-Group-Challenge“ definieren und diese mit „Aktiver Nutzer“ assoziieren, sodass die untergeordnete Formel sich nur auf Multi-Group-Challenges bezieht, an denen der aktuelle Nutzer teilnimmt.

Um den Nutzern die Dokumentation näher zu bringen, verweisen die Lektionen auf diese.

4.3 Weitere Tests

Um die getroffenen Maßnahmen zu testen und ob sie die Probleme lösen konnten, wurden die Tests mit zwei Testern im Unternehmen wiederholt.

4.3.1 Dokumentation

Während der Tutorials wurde die Dokumentation von beiden Testnutzern während des Lesens der Aufgaben einmal aufgerufen. Als dann aber Probleme auftraten, wurde sie nicht wieder konsultiert.
Nach den Tutorials riefen beide Testnutzer die Dokumentation nicht mehr selbstständig auf. Selbst als Probleme auftraten und wie auch schon in den ersten Tests, wurde in Aufgabe zwei nach der „Multi-Group-Challenge“ gesucht, ohne dass zum Beispiel die Suchfunktion in der Dokumentation verwendet wurde, stattdessen wurden auch hier wieder die Kategorien der Blöcke manuell durchsucht. Auch bei Aufgabe drei wurde gleich vorgegangen. Hierbei wurde zwar nach dem Block gegriffen, welcher die Stringkonstanten und damit auch für die möglichen Status der Challenges bietet, aber es wurde weiterhin nicht die Dokumentation genutzt, um den tatsächlichen Bearbeitungsstatus der Challenges zu finden. Auch wurde nicht die Dokumentation des Blocks mit den Stringkonstanten aufgerufen, obwohl beiden Testern nicht ganz klar war, wofür dieser Block zuständig ist.
Die Dokumentation alleine löst also die Probleme nicht.

4.3.2 Tutorials

Die erste Aufgabe wurde von beiden Testnutzern recht schnell gelöst.
Die zweite Aufgabe hingegen führte zu mehr Problemen: Zunächst gab es ein Verständnisproblem bei der Aufgabenstellung. Nachdem dies geklärt wurde, wurden Variablen definiert und versucht, die Blöcke direkt dem Block unterzuordnen, obwohl die Verbindungsstücke zueinander passten. Obwohl die erste Lektion also selbst recht schnell bearbeitet wurde, wurde der Inhalt offenbar nicht richtig verstanden. Nach ein wenig ausprobieren wurde den Testnutzern jedoch klar, dass ein Vergleich dazwischen gehört. Allerdings wurde zum Teil der falsche Vergleich für den falschen Typen verwendet. Auch war bei dem Block, welcher ein Datum mit Uhrzeit ausliest, nicht ganz klar, wieso eine Variable nicht zur Auswahl stand, obwohl der Block, der diese definiert, diesem übergeordnet war. Die Variable war vom falschen Typ, welcher solch einen Wert nicht bietet. Hier wünschte sich eine Testnutzerin einen Hinweis oder eine Fehlermeldung, wieso diese nicht verfügbar war.
Auch bei den beiden letzten Lektionen gab es Unklarheiten. Wie Variablen verknüpft werden können, wurde den Nutzern beim Ausprobieren mit dem Erscheinen der Checkboxen zwar klar, aber welche Variablen Gegenstand der Aufgabe waren, wussten beide nicht. Statt die Variable „Aktiver Nutzer“ mit einer selbst definierten Variable zu verknüpfen, wurden einfach zwei selbst definiert. Erst nach einem Hinweis wurde ein Block, der eine Variable definiert, unter den Einstiegspunkt verschoben und so eine Verknüpfung mit der Variable „Aktiver Nutzer“ hergestellt.

4.3.3 Tests nach den Tutorials

Nachdem die Tutorials bearbeitet worden sind, wurden die Testnutzer vor dieselben Aufgaben gestellt, wie die Tester beim Usability Testessen.
Die in dem Tutorial vermittelten Kenntnisse sind hierbei leider nicht immer angewendet worden: Zum einen wurde versucht, Blöcke miteinander zu verbinden, bei denen dies nicht möglich ist. Außerdem wurde zum Beispiel nicht die Variable „Aktiver Nutzer“ verwendet, sondern eine weitere Variable desselben Typs definiert. Erst nach einem Hinweis darauf, wurde die Abfrage unter dem Einstiegspunkt de-

finiert und die richtige Variable verwendet.
Nach der Bitte an einen der Tester, in eigenen Worten wiederzugeben, wie seine Abfrage funktioniert, verwendete dieser die erste Person, anstatt von einem „aktiven Nutzer“ o. Ä. zu sprechen.

Kapitel 5

Zusammenfassung und Ausblick

Ziel der Bachelorarbeit war es, eine blockbasierte, domänenspezifische Datenbankabfragesprache zu entwickeln und zu evaluieren, ob diese von Nutzern einfach zu verwenden ist.
Dafür wurde zunächst untersucht, wie andere visuelle Sprachen und visuelle Abfragesprachen versuchen, den Nutzern die Arbeit beim Formulieren von Abfragen und Schreiben von Programmen zu erleichtern und beizubringen. So zeigte sich, dass diese zum Beispiel automatisch mögliche Verbindungen der Objekte erkennen, syntaktische Fehler durch das Tool verhindert oder wenigstens erschwert werden, oder durch Formen und Farben die Funktion sprachlicher Elemente untermalt werden.
Bei der Vorbereitung der Datenbank galt es dann, diese durch das Verbergen nicht relevanter Informationen und teilweises Neustrukturieren der Daten, eine möglichst einfach zu verstehende Datenstruktur zu erschaffen. Dafür wurde auch auf Nullwerte verzichtet und stattdessen lieber Standardwerte verwendet.
Bei der Implementierung der Sprache wurden die oben genannten Ideen, wie diese Tools dem Benutzer die Bedienung erleichtern, auf das Konzept der Datenbankabfragesprache zu übertragen und alle Blöcke erstellt und implementiert. Damit diese daraufhin getestet werden kann, wurde auch die Übersetzung nach SQL implementiert.
Mithilfe von Usability Tests wurde die Sprache das erste Mal von Testnutzern verwendet. Für die hierbei gefundenen Probleme wurde ein Tutorial entwickelt und eine Dokumentation geschrieben, deren Nutzen wurde daraufhin mit weiteren Tests überprüft. Dabei stellte sich heraus, dass diese die Probleme nicht lösen konnten.

5.1 Ausblick

Die derzeitig schwierige Verwendung der Datenbankabfragesprache muss nicht zwangsweise das Aus für diese Idee bedeuten: Weitere Maßnahmen zur Verbesserung der Erlernbarkeit könnten umgesetzt werden.
Da die Dokumentation von den Nutzern allerdings nur selten verwendet wurde, kann ihr Inhalt hierbei kaum etwas verbessern. Aber auch auf andere Arten kann das Tool dem Nutzer während der Benutzung Informationen bieten: So können zum Beispiel Tooltips verwendet werden, welche die Funktionsweise von Blöcken kurz und knapp beschreiben, oder noch spezieller die Ursache von aktuellen Zuständen. Dies wäre insbesondere beim fehlerhaften Zustand der Blöcke interessant, welche Werte aus Variablen lesen: So könnte ermittelt und angezeigt werden, wieso Variablen nicht zur Auswahl stehen und Tipps, wie der Zustand zu korrigieren ist.
Das Tutorial erzielte zwar auch nicht den gewünschten Effekt, aber es kann noch ausgebaut werden. So können mehrere Beispiele gezeigt werden, die im nächsten Schritt selbst nachgestellt werden. Durch mehrere Wiederholungen ähnlicher Aufgaben könnten die Features besser trainiert werden, um so die Chance zu erhöhen, dass diese vom Nutzer verinnerlicht worden sind. Wenn die Lektionen schon in mehrere Schritte aufgeteilt werden, kann auch der Inhalt auf mehrere verteilt werden: Aktuell ist zum Beispiel Lektion zwei dafür zuständig, Variablen und Werte einzuführen. Alle drei verschiedenen Arten von Werten könnten auch in eigenen Schritten nacheinander eingeführt werden.
Aber auch bei der Sprache selbst können Verbesserungen vorgenommen werden: Einer der Testnutzer beispielsweise sprach in der ersten Person anstelle von einem „Aktiven Nutzer“, da er es so leichter verstehen würde. Dies könnte direkt von der Sprache gefördert werden, in dem die Variable umbenannt wird und auch im Einstiegspunkt auf diese hingewiesen wird, beispielsweise mit einem Satz wie „*Ich* schalte diesen Erfolg frei wenn:“. Ggf. kann auch der Funktionsumfang der Sprache gekürzt werden, sodass erst einmal nur auf Zahlen zurückgegriffen wird. So würden beispielsweise die Farben bei den Blöcken, die mit Werten arbeiten, keine Rolle mehr spielen, sodass nur die Form des Verbindungsstücks gleich sein muss.
Weitere Tests würden daraufhin folgen.
Neben der Verbesserung der Usability wäre es allerdings auch noch möglich, die

Sprache für andere Anwendungsbereiche zu verwenden: Bisher arbeitet die Abfrage nur mit einem Nutzer und gibt einen Wahrheitswert zurück. Aber die Eingabe, der Typ der Ausgabe und ggf. die zugrunde liegende Datenbank können geändert werden, um so beispielsweise Statistiken zu erstellen. Alternativ könnten Elemente einer Liste werden anhand einer zurückgegebenen Zahl sortiert oder der Bearbeitungsstatus von einem WBT anhand von selbst festgelegten Kriterien ermittelt werden.

Anhang A

Appendix

A.1 Abbildungsverzeichnis

2.1 Beispielabfrage in Oracle SQL Developer 6
2.2 Aufgabe am Ende eines Levels, mit stark reduzierter Auswahl von Blöcken 8
2.3 Aufgabe am Ende eines Levels, nachdem weitere Blöcke hinzugefügt worden sind 8
2.4 Aufgabe am Ende eines Kurses mit einer großen Auswahl von Blöcken 9
2.5 Aufgabe, in der Schleifen noch nicht verfügbar waren 9

3.1 Die beiden ursprünglichen Benutzer-Tabellen und die View für den Benutzer. 12
3.2 Die Multi-Group-Challenge mit Gruppen und Nutzern. 14
3.3 Nutzersicht auf die Ergebnisse der Multi-Group-Challenges 15
3.4 Übersicht über die Kategorie „Datenbank“ 17
3.5 Übersicht über die Kategorie „Logik“ 20
3.6 Mutator zum Hinzufügen von weiteren Aussagen 21
3.7 Übersicht über die Kategorie „Zahlen“ 22
3.8 Übersicht über die Kategorie „Datum und Uhrzeit“ 25
3.9 Übersicht über die Kategorie „Texte“ 26
3.10 Der Blockly Editor mit den selbst definierten Kategorien und Blöcken 31
3.11 Ein Beispielerfolg 32

A.2 Tabellenverzeichnis

2.1 Beispielabfrage formuliert mit QBE 7

3.1 Beispielhafte Übersetzung der Blöcke der Kategorie „Datenbank“ nach SQL. 19
3.2 Beispielhafte Übersetzung der Blöcke der Kategorie „Logik“ nach SQL. 21
3.3 Beispielhafte Übersetzung der Blöcke der Kategorie „Zahlen“ nach SQL. 24
3.4 Beispielhafte Übersetzung der Blöcke der Kategorie „Datum und Uhrzeit“ nach SQL. 26
3.5 Beispielhafte Übersetzung der Blöcke der Kategorie „Texte“ nach SQL. 27

A.3 Listings

3.1 Blocks/logic/conjunction.js 28

A.4 Literatur

[Bai06] Mark Alexander Bain. *Learn OpenOffice.org spreadsheet macro programming : OOoBasic and Calc automation ; a fast and friendly tutorial to writing macros and spreadsheet applications*. From technologies to solutions. Birmingham, U.K: Packt Pub, 2006. ISBN: 9781847190970.

[CDT08] Maria Chiara Caschera, Arianna D'Ulizia und Leonardo Tininini. »Visual Query Languages, Representation Techniques, and Data Models«. In: *Visual Languages for Interactive Computing*. Hrsg. von Fernando Ferri. Hershey, Pa.: Information Science Reference, 2008. Kap. 9, S. 142–157. ISBN: 978-1-59904-534-4.

[Cob11] Charles G. Cobb. *Making Sense of Agile Project Management. Balancing Control and Agility*. Hoboken, New Jersey: John Wiley & Sons, 2011. ISBN: 978-0-470-94336-6.

[Cod] Code.org. *Code.org*. URL: *https://code.org/international/about* (abgerufen am 08. 08. 2019).

[Cor] Oracle Corporation. *Oracle SQL Developer*. URL: *https://www.oracle.com/database/technologies/appdev/sql-developer.html* (abgerufen am 06. 08. 2019).

[glü] quäntchen + glück GmbH & Co. KG. *Über das Testessen*. URL: *https://usability-testessen.org/ueber-das-testessen/* (abgerufen am 21. 08. 2019).

[Gro] LEGO Group. *Mindstorms: Programmieren lernen*. URL: *https://www.lego.com/de-de/themes/mindstorms/learntoprogram* (abgerufen am 15. 08. 2019).

[Hal] Daniel Conrad Halbert. *Programming by Example*. URL: *https://danhalbert.org/pbe-html.htm* (abgerufen am 09. 04. 2019).

[IBM] IBM. *Filtering Mail Using Rules*. URL: *https://www.ibm.com/support/knowledgecenter/en/SSKTWP_8.5.3/com.ibm.notes85.help.doc/mail_rules_t.html* (abgerufen am 09. 04. 2019).

[Lie+06] Henry Lieberman, Fabio Paternò, Markus Klann und Volker Wulf. »End-User Development: An Emerging Paradigm«. In: *End-User Development*. Hrsg. von Henry Lieberman, Fabio Paternò und Volker Wulf. Bd. 9. Human-Computer Interaction Series. Dordrecht: Springer, 2006. Kap. 1, S. 1–8.

[LLC] Google LLC. *Blockly*. URL: *https://developers.google.com/blockly/* (abgerufen am 08. 08. 2019).

[LPW06] Henry Lieberman, Fabio Paternò und Volker Wulf. In: *End-User Development*. Bd. 9. Human-Computer Interaction Series. Dordrecht: Springer, 2006. Kap. Preface, S. vii–xiii. ISBN: 9781402042201.

[MD17] Zeno Menestrina und Antonella De Angeli. »End-User Development for Serious Games«. In: *New Perspectives in End-User Development*. Hrsg. von Fabio Paternò und Volker Wulf. Cham: Springer International Publishing, 2017, S. 359–383. ISBN: 978-3-319-60291-2. DOI: *10.1007/978-*

3-319-60291-2_14. URL: *https://doi.org/10.1007/978-3-319-60291-2_14*.

[MIT] Lifelong-Kindergarten-Gruppe (Media-Lab des MIT). *Scratch*. URL: *https://scratch.mit.edu/about* (abgerufen am 08. 08. 2019).

[Nar93] Bonnie A. Nardi. *A Small Matter of Programming. Perspectives on End User Computing*. Cambridge, Massachusetts: MIT Press, 1993. ISBN: 0-262-14053-5.

[PFM17] E. Pasternak, R. Fenichel und A. N. Marshall. »Tips for creating a block language with blockly«. In: *2017 IEEE Blocks and Beyond Workshop (B B)*. Okt. 2017, S. 21–24. DOI: *10.1109/BLOCKS.2017.8120404*.

[Res+09] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman und Yasmin Kafai. »Scratch: Programming for All«. In: *Commun. ACM* 52.11 (Nov. 2009), S. 60–67. ISSN: 0001-0782. DOI: *10.1145/1592761.1592779*. URL: *http://doi.acm.org/10.1145/1592761.1592779*.

[RI06] Alexander Repenning und Andri Ioannidou. »What makes End-User Development Tick? 13 Design Guidelines«. In: *End-User Development*. Hrsg. von Henry Lieberman, Fabio Paternò und Volker Wulf. Bd. 9. Human-Computer Interaction Series. Dordrecht: Springer, 2006. Kap. 4, S. 51–85.

[Tea] MkDocs Team. *MkDocs*. URL: *https://www.mkdocs.org/* (abgerufen am 21. 08. 2019).

[Zlo77] M. M. Zloof. »Query-by-example: A Data Base Language«. In: *IBM Syst. J.* (1977), S. 324–343. ISSN: 0018-8670.

A.5 Liste der Abkürzungen

eLMS engram LernManagementSystem, S. 3, 12

EUD End-User Development, S. 1, 2, 4, 5

QBE Query By Example, S. 6, 7, 44

SQL Structured Query Language, S. 2

VQL Visual Query Language, S. 2, 4–6

WBT Web Based Training, S. 3, 42

A.6 Glossar

Endbenutzer
Eine Person, welche Software verwendet. Besitzt üblicherweise grundlegende Computerkenntnisse, aber im Regelfall keine Programmiererfahrung.
S. 1, 2, 4, 5, 7, 9

Entity-Relationship-Diagram
Diagramm welches Objekte (Entitäten), ihre Eigenschaften und Beziehungen untereinander visualisiert
S. 5

Google Blockly
Bibliothek für einen visuellen Code-Editor, welche die Erstellung eigener Blöcke und Sprachen erlaubt.
S. 1

Oracle SQL Developer
Programm zum Verwalten von Datenbanken. Bietet ebenfalls einen visuellen Abfrageneditor.
S. 5, 6, 18, 43

Product Owner
Rolle bei der Vorgehensweise Scrum. Der Produktowner pflegt das Product Backlog, eine nach Prioritäten sortierte Liste der Features, welche in Zukunft Teil des Produktes werden können.
S. 4

Programming by Example
Technik des End-User Development, bei der ein Benutzer zuerst eine Reihe von Aktionen manuell durchführt und aufzeichnen lässt. Daraus wird ein Programm generiert, welches diese Aktionen wiederholt.
S. 5

QuizCards

Corporate Learning App der engram GmbH, mit welcher Unternehmen ihren Mitarbeitern Lerninhalte in Form von Quizzes in verschiedenen Spielmodi anbieten können.

S. 2, 11, 12

Scrum

Vorgehensweise im Projektmanagement, welche häufig in Softwareprojekten angewendet wird.

S. 4

Serious Game

Spiele mit dem Ziel, Fähigkeiten zu trainieren oder Wissen zu vermitteln.

S. 1

Sprint

Iteration bei der Vorgehensweise Scrum, bei der in einem Zeitraum von etwa zwei bis vier Wochen ein neues Produkt oder eine neue Produktversion erstellt wird.

S. 4

A.7 Inhalte des Datenträgers

Blocks/
: Quellcodes der implementierten Blöcke

Documentation/
: Das Benutzerhandbuch

Editor/
: Editor mit den implementierten Blöcken[1]

Testdata/
: Nur lesbarer Editor zum Ansehen der aufgezeichneten Nutzertests.[1]

thesis.pdf
: Bachelor-Thesis im PDF Format

[1] Aufgrund der Cross Origin Policy der meisten Browser reicht es nicht, index.html direkt im Browser zu öffnen. Die Dateien sollten mit einem Webserver ausgeliefert werden. Es werden keine speziellen Konfigurationen oder Plugins wie PHP benötigt, daher reichen auch einfache Webserver, wie der SimpleHTTPServer (Python) oder http-server (Node.js/npm), aus.