

# Something, something, SQL

Masterthesis

Abgegeben am: 29. Januar 2016

Autor: Marcus Riemer, B.Sc.  
Matr.-Nr.: 100478  
E-Mail: [inf100478@fh-wedel.de](mailto:inf100478@fh-wedel.de)

Betreuer: PD. Dr. Frank Huch  
E-Mail: [fh@informatik.uni-kiel.de](mailto:fh@informatik.uni-kiel.de)

Prof. Dr. Ulrich Hoffmann  
[uh@fh-wedel.de](mailto:uh@fh-wedel.de)

## Unfertige Arbeit!

An dieser Arbeit ist noch nichts fertig! Ich habe sie online gestellt, um einigen hilfreichen Personen ein einfaches “mitlesen” zu ermöglichen.

Momentan lassen sich die folgenden Kapitel einigermaßen sinnvoll lesen. Das heißt nicht, dass sie in irgendeiner Form “fertig” sind. Aber sie enthalten dann zumindest einige Kommentare an Stellen, wo der Inhalt noch sehr wenig endgültig ist. Sofern nicht anders erwähnt sind alle Unterkapitel zu einem Überkapitel ebenfalls mit eingeschlossen.

- Schon einigermaßen endgültig und vollständig sind die [3.1 Grundprinzipien](#). Auch endgültig, aber weniger vollständig ist der Ausschlußkatalog: [3.2. “Out-of-Scope”](#)
- [3.6 Editor für SQL](#) steckt zwar voller TODO-Hinweise, lässt sich aber schon ganz gut lesen.

Deutsch?

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Eine spezielle Entwicklungsumgebung . . . . .	1
<b>2</b>	<b>Vergleichbare Arbeiten</b>	<b>2</b>
2.1	Software: Scratch . . . . .	2
2.2	Software: Visual Studio Lightswitch . . . . .	2
2.3	Buch: Die Macht der Abstraktion . . . . .	2
2.4	Software: Microsoft Access . . . . .	2
2.5	Software: MySQL Workbench . . . . .	2
<b>3</b>	<b>Anforderungsanalyse</b>	<b>3</b>
3.1	Grundprinzipien . . . . .	3
3.2	“Out-of-Scope” . . . . .	4
3.3	Datenbanksystem . . . . .	5
3.4	Zwei Modi: Entwickeln und Anschauen . . . . .	5
3.5	Einfaches anlegen und kopieren von Projekten . . . . .	5
3.6	Editor für SQL . . . . .	6
3.6.1	Visuelle Gestaltung . . . . .	6
3.6.2	Grundsätzlicher Aufbau des Editors . . . . .	7
3.6.3	Deaktivierbare Komponenten . . . . .	9
3.6.4	Änderung der Reihenfolge . . . . .	9
3.6.5	GROUP BY . . . . .	9
3.6.6	Funktionen . . . . .	9
3.6.7	Unterabfragen . . . . .	10
3.6.8	Abfragen mit Parametern . . . . .	10
3.6.9	Manipulation von Daten . . . . .	10

Deutsch?

3.7	Beispielhafte Abfragen . . . . .	10
3.8	Oberflächen für Endbenutzer . . . . .	11
3.8.1	Navigation . . . . .	11
3.8.2	Erfassen von Beziehungen . . . . .	11
<b>4</b>	<b>Umsetzungsanalyse</b>	<b>11</b>
4.1	Systemübersicht . . . . .	11
4.2	Verwaltung von Projekten . . . . .	12
4.3	Verwendung von Domänenspezifischen Sprachen . . . . .	12
4.4	Roadmap . . . . .	12

# 1 Einleitung

Diese Arbeit beschäftigt sich mit der Konzeption und Implementierung einer einsteigerfreundlichen Entwicklungsumgebung für den anwendungsorientierten Umgang mit SQL-Datenbanken. Zielgruppen dieser Software sind Schülerinnen und Schüler ab der Mittelstufe sowie deren Lehrkräfte.

Die Anwender der Software sollen in die Lage versetzt werden, zu einem gegebenen Datenmodell sowohl inhaltliche Fragen mit Bezug zu einem konkreten Datenbestand zu beantworten als auch neue Daten in das Modell einzupflegen. Darüber hinaus soll es Ihnen möglich sein, die Datenbank auch einer begrenzten Öffentlichkeit zugänglich zu machen.

## 1.1 Eine spezielle Entwicklungsumgebung

Die wesentliche Anforderung an die im Rahmen dieser Arbeit zu erstellende Software ergibt sich also schon aus dem Titel der Arbeit: Es geht vorrangig um die Vermittlung von praktischen Kenntnissen zur Abfrage- und Manipulation von komplexen Datenbeständen in Anlehnung an die Projektideen der Lehrpläne [1] bzw. Fachanforderungen [2] für Informatik des Landes Schleswig-Holstein<sup>1</sup>. Auch wenn sich aktuell eine zunehmende Pluralität von Paradigmen zur Datenspeicherung abzeichnet, die das relationale Modell in Frage stellen oder ergänzen, behandelt diese Arbeit explizit die Vermittlung von SQL-Kenntnissen.

Da der Inhalt dieser Arbeit die Konzeption und Umsetzung einer Software ist, deren Zweck wiederum die Entwicklung anderer Software ist, bedarf es zunächst eines gemeinsamen Verständnisses für die Bezeichnungen der beteiligten Systeme.

### **(Schüler-)Entwicklungsumgebung**

Bezeichnet die von den Lernenden zu nutzende Software, die im Rahmen dieser Arbeit erstellt wurde. Sofern Verwechslungen mit anderen Entwicklungsumgebungen (engl. “integrated development environment”, IDE) auftreten könnten, wird explizit das Präfix “Schüler” genutzt. Letztere Unterscheidung wird insbesondere bei Vergleichen relevant sein.

### **(Schüler-)Projekt**

Bezeichnet die von den Lernenden, unter Nutzung der Schülerentwicklungsumgebung, erstellte Software. Dazu gehört unter anderem das Datenbankschema, die verschiedenen Abfragen und die gestaltete Benutzeroberfläche.

---

<sup>1</sup>Die exakte Verortung oder auch die Konzeption von konkreten Schulstunden ist hingegen nicht Teil dieser Arbeit.

## 2 Vergleichbare Arbeiten

Andere Entwicklungsumgebungen für Datenbanken und auch Generatoren für Abfragemasken gibt es zuhauf. Dieses Kapitel stellt einige der verfügbaren Programme vor, insbesondere im Hinblick auf für diese Arbeit formulierten Prinzipien und unter Betrachtung der avisierten Zielgruppe.

**Unsure:** Momentan eher eine sehr lose Sammlung denn eine tatsächliche Recherche.

### 2.1 Software: Scratch

**Info:** Paradebeispiel für eine schülerorientierte Entwicklungsumgebung, insbesondere auch eine Betrachtung der visuellen Programmierung.

### 2.2 Software: Visual Studio Lightswitch

**Info:** Generator für Datengetriebene Geschäftsanwendung mit überschaubarer Applikationslogik.

### 2.3 Buch: Die Macht der Abstraktion

**Info:** Buch zum Einstieg in die Programmierung mit einem interessanten, sehr Datentypgetriebenen Ansatz.

### 2.4 Software: Microsoft Access

**Info:** Eher eine Datenbank als eine sinnvolle Eingabemaske für unversierte Benutzer.

### 2.5 Software: MySQL Workbench

**Info:** Wirklich ein Datenbanktool, keinerlei Benutzerschnittstelle für “normale” Benutzer.

## 3 Anforderungsanalyse

Dieses Kapitel beschreibt die generellen Anforderungen an das Projekt, ohne auf Details wie die technische Machbarkeit ausführlich zu analysieren oder konkrete Umsetzungsstrategien zu planen. Diese softwaretechnischen Aspekte werden im nächsten Kapitel ([4 Umsetzungsanalyse](#)) besprochen.

**Info:** Der Verzicht auf die technischen Hintergründe in diesem Kapitel soll es auch für “normale” Informatiklehrer verständlich machen, ohne Sie gleich mit den Hintergründen der technischen Realisierung zu erschlagen.

### 3.1 Grundprinzipien

Nach der Betrachtung der zu bedienenden Zielgruppe und der Beschäftigung mit bereits existierenden Alternativen, ist es nun an der Zeit ein paar allgemeine Grundprinzipien zu formulieren. Diese Prinzipien bilden die Philosophie hinter der Schülerentwicklungsumgebung ab und dienen als Leitfaden.

Praktisch erlaubt das vor allem eine relativ akkurate Abschätzung, ob sich die Implementierung einer bestimmten Idee lohnt und wie sie zu priorisieren ist. Daher sind auch diese Prinzipien ihrer Bedeutung für diese Arbeit nach absteigend sortiert. Die sich dabei ergebende Auswahl Reihenfolge berücksichtigt auch den Zeitrahmen der Masterthesis, ist also nicht nach rein didaktischen Gesichtspunkten zu bewerten.

#### Semantik vor Syntax

Den Lernenden sollen kontextsensitiv sinnvolle Operationen angeboten werden, optimalerweise mit einer kurzen Erläuterung, warum gerade nur diese Teilmenge an Operationen möglich ist. Die eigentliche Programmierung der Abfragen erfolgt dann durch die Kombination von Bausteinen, ähnlich wie bei der Lernsoftware “Scratch”. Durch kontinuierliches Feedback der Entwicklungsumgebung sollen die Lernenden in die Lage versetzt werden, auch ohne eine ständige Rückversicherung bei der Lehrkraft die eigenen Ansätze zu erproben.

#### Motivation durch praktisch vorzeigbare Ergebnissen

Typischerweise ist der Einstieg in die Programmierung von relativ langweiligen Programmen geprägt, häufig textbasierten Konsolenanwendungen. Im Sonderfall der Vermittlung von SQL Kenntnissen ist das Ergebnis der Arbeit sogar überhaupt nicht sinnvoll zu demonstrieren, weil die erstellten Abfragen isoliert für sich stehen und häufig auch nur in der Entwicklungsumgebung der jeweiligen Datenbank ausführbar sind. Mit der im Rahmen dieser Arbeit zu erstellenden Software sollen sich hingegen praktisch relevante, allerdings sehr datenorientierte Programme umsetzen lassen. Diese verfügen über von den Lernenden zusammengestellte Eingabemasken um Daten einzufügen oder zu manipulieren und verschiedene Ausgabesichten um den Datenbestand sinnvoll zu präsentieren.

### **Einfache Inbetriebnahme**

Eine initiale Hürde jeder (Lern-)Software ist deren Installation, insbesondere bei Programmen aus dem Datenbankumfeld. Die Inbetriebnahme der für Server konzipierten Programme auf privaten, "normalen" Rechnern führt immer wieder zu Problemen aufgrund von nicht aufgelösten Abhängigkeiten oder fehlenden Rechten beim Starten von Systemdiensten oder bei Dateizugriffen. Die zunehmende Heterogenität an Betriebssystemen, insbesondere die zunehmende Verwendung MacOS, tut ein Übriges um die Verteilung von Software zu erschweren. Damit der eigentliche Lernprozess nicht schon vor dem Start der Entwicklungsumgebung behindert wird, hat eine möglichst einfache Inbetriebnahme dementsprechend Priorität. Prinzipiell stehen Informatiklehrkräfte bei dem Betrieb der jeweiligen Programme vor ähnlichen Problemen wie ihre Schüler, nur dass Sie auf die Konfiguration des Rechnerpools ihrer Schule oftmals nur einen eingeschränkten Einfluss haben. Die im Vergleich zu privaten Rechnern wesentlich restriktiver gehandhabte Rechte eines Schul-PCs verkomplizieren diesen Umstand zusätzlich.

### **Schrittweise komplexere Benutzeroberfläche**

Konventionelle Entwicklungsumgebungen sind Programme von Profis für Profis und bieten einen dementsprechend ausgerichteten Funktionsumfang. Gerade wenn man aber dabei ist etwas Neues zu lernen kann es sinnvoll sein, die Menge der möglichen Optionen zu beschränken. In diesem Sinne sollte auch die Lehrkraft die Möglichkeit haben den Funktionsumfang der Entwicklungsumgebung für Schüler gezielt zu reduzieren, falls einzelne Konzepte sich als noch zu fortgeschritten erweisen sollten.

### **Fortführung der entwickelten Projekte**

Viele Lernumgebungen sind in sich geschlossene Systeme, deren Arbeitsergebnisse nur schwer in anderen Programmen oder Kontexten von Nutzen sind. Sobald der Lernende dann die Grenzen der verwendeten Lernsoftware erreicht hat, steckt er in einer Sackgasse fest. Die Arbeitsergebnisse dieser zu entwickelnden Software sollen daher zumindest einfach einsehbar sein, optimalerweise nach einem Export sogar einfach mit gängigen externen Programmen erweiterbar.

## **3.2 "Out-of-Scope"**

Deutsch?

Umgekehrt ist es auch wichtig, den Umfang des Projekts für die Thesis zu begrenzen. Die folgenden Ideen wären mehr oder minder naheliegende Ergänzungen, welche den Rahmen dieser Thesis aber sprengen würden.

### **Datenmodellierung**

Der Schwerpunkt dieser Arbeit liegt zunächst auf der Vermittlung von Kenntnissen zur Abfrage und Manipulation von Daten in einem bestehenden Schema. Änderungen an diesem Schema sind nicht vorgesehen, demzufolge ist auch der Neuentwurf eines Schemas mit externen Mitteln zu bewerkstelligen. Völlig außerhalb

des Umfangs dieser Arbeit ist die Überführung von konzeptionellen Modellen (z.B. ER-Schemata) in physikalische Modelle.

### **Aufwändiges Design von Benutzerschnittstellen**

Auch wenn die Konzeption der Benutzerschnittstelle für die verschiedenen Masken in den eben aufgezählten Prinzipien auftaucht, ist es wichtig den engen Rahmen dieses Aspektes zu verstehen. Es geht um die Schaffung von einfachen, datenzentrierten Eingabemöglichkeiten, nicht um die Umsetzung besonders kreativer Bedienkonzepte. Dementsprechend ist z.B. die Erweiterung der zur Verfügung stehenden Eingabeelemente durch die Lernenden außerhalb des Rahmens dieser Arbeit.

## **3.3 Datenbanksystem**

Eine der naheliegendsten zu treffenden Entscheidungen hat zwar einen technischen Hintergrund, ist aber trotzdem unbedingt Bestandteil dieser Anforderungsanalyse: Es geht um die Wahl des zu lehrenden Datenbanksystems. Dieser Aspekt hat einen unmittelbaren Einfluss auf die Inbetriebnahme der Entwicklungsumgebung und auch auf die Fortführung der Projekte mit externen Programmen.

Im Hinblick auf die einfache Installation und Verwendung bietet sich eine eingebettete Datenbank an, da die Skalierung der von den Lernenden entwickelten Applikation vernachlässigt werden kann. Optimalerweise ist ein Backup des gesamten Datenbestandes mit einer einfachen Kopie einer Datei zu erledigen.

**Unsure:** Theoretisch ist die Menge an denkbaren Systemen fast unüberschaubar groß, praktisch sticht SQLite aus der Masse an Optionen heraus. Wie ausführlich muss ich das begründen?

## **3.4 Zwei Modi: Entwickeln und Anschauen**

Grundsätzlich unterschieden werden muss bei der Schülerentwicklungsumgebung, ähnlich wie bei Scratch, zwischen zwei Betriebsmodi für ein Projekt: Zunächst wird ein Projekt im Entwicklermodus betrieben, in diesem Fall stehen dem Benutzer alle Entwicklungstools zur Verfügung. Wenn es dann später einmal fertig ist und zum Beispiel an Bekannte weitergegeben wird, sollen diese natürlich nur die normale Benutzeroberfläche sehen. Der Wechsel zwischen diesen beiden Modi sollte dabei zu jedem Zeitpunkt möglich sein.

## **3.5 Einfaches anlegen und kopieren von Projekten**

Im Sinne einer möglichst niedrigen Einstiegshürde soll es den Anwendern leicht gemacht werden, bestehende Projekte zu übernehmen. Im Informatikunterricht müssen häufiger



zu Beginn bestimmte Schritte einfach ausgeführt werden, “weil das nun mal so” (also eine ausführliche Erklärung zu diesem Zeitpunkt zu weit gehen würde). Im Falle des Sprachumfangs von SQL ist das zwar im Vergleich zu z.B. Java (“Herr Lehrer, was macht eigentlich dieses public static void?”) nicht ganz so dramatisch, aber nach Möglichkeit zu vermeiden.

Dementsprechend sollte möglichst jeder Schritt beim Anlegen eines neuen Projektes für die Schüler nachvollziehbar sein. Wenn dann doch einmal eine Serie von “mechanisch auszuführenden” Anweisungen erforderlich sein sollte, wäre es aber dennoch praktisch das durch eine Kopie eines schon bestehenden Projektes abzukürzen. Die Lehrkraft würde in diesem Fall also ein Projekt für ihre Schüler vorbereiten, welches diese dann optimalerweise mit einem einzigen Klick importieren können.

### 3.6 Editor für SQL

Der grafische Editor soll grundsätzlich ähnlich zu den aus Scratch bekannten Bedienelementen funktionieren. Es kommen also distinkte Bedienelemente für die verschiedenen Komponenten einer SQL Abfrage zum Einsatz, kein reiner Texteditor. Der komponentenorientierte Editor soll dabei nicht unbedingt die Konzeption von beliebigen Abfragen ermöglichen, wohl aber die in Kapitel [3.7 Beispielhafte Abfragen](#) beschriebenen exemplarischen Arbeitsabläufe unterstützen.

Wenn ein Anwender der Entwicklungsumgebung an die Grenzen des unterstützten Editors stößt, sollte er die Möglichkeit haben einmalig und für diese konkrete Abfrage eine Umwandlung in eine textbasierte Darstellung vornehmen zu können. Der umgekehrte Weg, also der Import von beliebigen SQL-Abfragen in den grafischen Editor, ist jedoch explizit ausgeschlossen.

Grundsätzlich bedürfen einige Komponenten der Abfrage besonderer Aufmerksamkeit, weil sie große Auswirkungen auf das Verhalten der anderen Komponenten haben. Vorrangig ist hier die **GROUP-BY** Komponente zu nennen. Sobald die Query mit einer **GROUP BY** Komponente ausgestattet wird ist der Zugriff auf die konkreten Spalten einzelner Zeilen z.B. im allgemeinen Fall nicht mehr möglich. Das hinzufügen (oder entfernen) dieser Komponente hat also große Auswirkungen auf die Korrektheit der gesamten Abfrage.

#### 3.6.1 Visuelle Gestaltung

Für technische Details irrelevant, aber unbedingt ebenfalls im Voraus zu klären ist die Frage, ob es sinnvoll wäre, das sehr bunte, blockige Design von Scratch zu imitieren. [Abbildung 1](#) zeigt einen Vergleich eines frühen Prototypen in einem an Syntax-Highlighting angelehnten Design und in einem wesentlich bunteren, blockigen Design.

**Todo:** Quelle mit Beleg für “Kinderfreundliches” buntes Design suchen. Das wird man bei Scratch schon aus gutem Grund gemacht haben. Relevante Frage: Zielgruppe (in Jahren) von Scratch vs. Zielgruppe dieses Projektes?

Aber auch wenn man den sehr bunten Stil gegen einen etwas nüchtereren, aber immer noch “blockartigen” Stil austauschen würde, ergeben sich dadurch nur vergleichsweise wenige Vorteile. In Scratch zeigen z.B. die Konnektoren des Blocks für eine Endlosschleife sehr deutlich, wie sich der Kontrollfluss durch dieses Element verändern wird. Für eine vollständige Programmiersprache ist das mit Sicherheit eine gute Wahl, aber für den sehr linearen Ablauf einer SQL-Abfrage ist dieser Umstand nicht von Bedeutung.

Ein weiterer Vorteil eines etwas nüchtereren, IDE-ähnlichen Designs wäre zudem die größere Nähe zu “normalen” Entwicklerprogrammen. Diese wirken möglicherweise weniger einschüchternd, wenn man sich schon an den Anblick von recht viel Text mit Syntax-Highlighting gewöhnt hat.

Letztendlich überwiegt aber ein sehr viel profanerer Fakt: Der Autor dieser Arbeit ist kein Grafikdesigner und würde vermutlich kein ansprechendes und zugleich buntes Farbkonzept auf die Beine stellen können.

```
SELECT p.id, p.name, p.geb_dat, e.id,
       e.bezeichnung, e.geb_dat
FROM personen p
INNER JOIN ereignisse e ON p.geb_dat = e.dat
WHERE p.id ≥ 5
AND p.id ≤ 10
OR p.name = "Hans"
```

(a) Starke Farbakzente, ähnlich zu Scratch

```
SELECT p.id, p.name, p.geb_dat, e.id,
       e.bezeichnung, e.geb_dat
FROM person p
INNER JOIN ereignisse e ON p.geb_dat = e.dat
WHERE p.id ≥ 5
AND p.id ≤ 10
OR p.name = "Hans"
```

(b) Syntax-Highlighting, ähnlich einer IDE

Abbildung 1: Vergleich unterschiedlicher Gestaltungsansätze

### 3.6.2 Grundsätzlicher Aufbau des Editors

Ein grundsätzlicher Nachteil dieses Komponentenorientierten Bedienkonzeptes ist der nötige Platz für die Unterbringung aller verwendbaren Komponenten. Es muss sorgfältig geplant werden, wie diese anzuordnen sind und welche Optionen in welchem Kontext gerade sichtbar sein müssen. Anders als bei Scratch muss dabei nicht unbedingt Drag & Drop das vorherrschende Bedienparadigma sein. Da der Benutzer immer nur eine Abfrage zur Zeit bearbeiten können soll, ergibt sich der einzig mögliche Platz für viele Blöcke automatisch. Darüber hinaus ist die Reihenfolge eines Großteils der Komponenten einer Abfrage sehr strikt festgelegt, so kann eine GROUP BY Komponente nicht an beliebigen Stellen verwendet werden, sondern nur nach der FROM oder der WHERE Anweisung. Andere Komponenten wie z.B. HAVING oder logische Verknüpfungen mit AND oder OR sind nicht

nur von der Reihenfolge, sondern auch von der Existenz anderer Bestandteile abhängig. Die Verwendung eines separaten Bereichs mit allen Bestandteilen einer SQL-Abfrage (bei Scratch “Toolbox” genannt) wäre also zwar denkbar, sollte aber zumindest keinesfalls Drag & Drop erzwingen, sofern das hinzufügen der Komponente nur an einer einzigen Stelle möglich ist.

Die Alternative dazu wäre eine Anzeige von “Platzhaltern” für die entsprechenden Komponenten unterhalb der eigenen Abfrage. Ein Klick auf den Platzhalter wandelt diesen dann in einen konkreten Block um und fordert ggfs. zur Angabe der benötigten Parameter auf. Im Rahmen der entwickelten Prototypen hat sich herausgestellt, dass eine permanente Anzeige aller Editierungsmöglichkeiten mit einem sehr überladen wirkenden Benutzerinterface einher geht, insbesondere was die Einblendung von Platzhaltern angeht. Abbildung 2 zeigt einen Screenshot des Prototypen, bei dem nahezu alle denkbaren Editierungsoptionen gleichzeitig verfügbar sind.

Dementsprechend wird ein anderer Ansatz verfolgt werden müssen: Es lässt sich immer ein Block zur Zeit in einen “editieren” Modus versetzen. Nur innerhalb dieses Blocks erlauben die Bedienelemente dann eine Änderung der Komponente. Für einige Komponenten, speziell FROM, WHERE und HAVING lassen sich neu anzuhängende Komponenten auch sehr gut im Rahmen dieses Modus anbieten. Es stellt sich dann natürlich die Frage, wie man Blöcke ohne einen eindeutig passenden Kontext, z.B. ein ORDER BY, der Abfrage hinzufügen kann. Eine Antwort darauf wird sich vermutlich erst durch reale Tests mit der Oberfläche finden lassen, die folgenden Ansätze sind aber denkbar:

- Die SELECT-Komponente hat einen losen Bezug zu allen anderen Komponenten. Schwierig einzuordnende Komponenten wie WHERE, GROUP BY, ORDER BY LIMIT oder auch UNION werden immer dann angeboten, wenn die SELECT-Komponente gerade editiert wird.
- Eine Leiste mit möglichen Komponenten ohne eindeutigen Kontext erlaubt deren Ergänzung zu jedem Zeitpunkt.

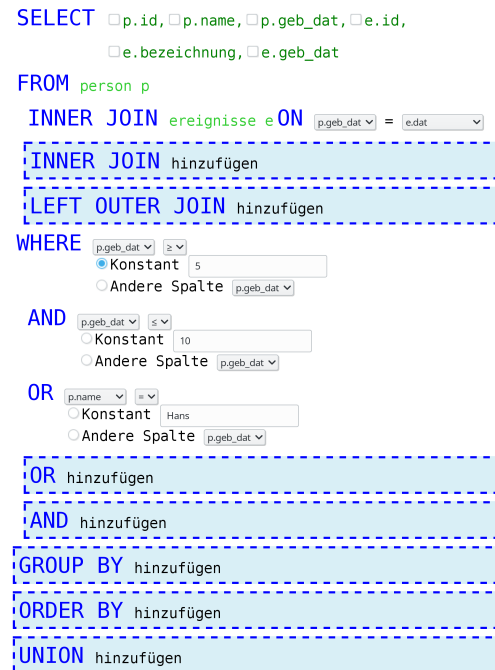


Abbildung 2: Simultane Anzeige (fast) aller Möglichkeiten

**Info:** Im Editor immer zu sehen, aber aktuell nicht beschrieben: Eine Tabelle mit den Daten, welche die Abfrage im aktuellen Zustand liefern würde.

### 3.6.3 Deaktivierbare Komponenten

Unmittelbar nach dem Hinzufügen einer neuen Komponente sind möglicherweise noch nicht alle nötigen Informationen verfügbar. Daher werden Komponenten immer in einem deaktivierten Zustand hinzugefügt, sie lassen sich erst aktivieren wenn alle relevanten Informationen eingetragen worden sind. Genau so werden Komponenten mit inhaltlichen Fehlern deaktiviert, bis der Fehler behoben worden ist. Und natürlich kann es auch Gründe geben eine Komponente freiwillig zu deaktivieren, auch das soll selbstverständlich möglich sein.

Speziell für die **WHERE** und **HAVING** Komponenten ist zu klären, welche Auswirkung eine Deaktivierung haben soll. Sollen alle Kinder ebenfalls deaktiviert werden oder wird die nächste folgende Komponente zum **WHERE** bzw. **HAVING**? Auch hier sind beide Möglichkeiten denkbar, die bessere Alternative muss sich durch Tests mit realen Benutzern zeigen.

### 3.6.4 Änderung der Reihenfolge

Wiederum für die **WHERE** und **HAVING**, aber auch für die **JOIN** Komponenten wäre es sehr hilfreich, die Reihenfolge durch eine Schaltfläche oder auch Drag & Drop variieren zu können.

### 3.6.5 GROUP BY

Sobald in einer Abfrage eine **GROUP BY** Komponente auftaucht, hat dies Auswirkungen auf die Möglichkeiten innerhalb der **SELECT** Anweisung. Da keine Auswahl von ungruppierten und nicht-aggregierten Spalten möglich sein darf, müssen diese entfernt werden. Insbesondere wenn der Benutzer vorher schon mit komplizierten Ausdrücken im **SELECT** gearbeitet haben sollte, ist also zumindest eine Warnung nötig.

### 3.6.6 Funktionen

An allen Stellen bei denen in SQL Werte verglichen oder ausgegeben werden ist es möglich, diese durch Funktionen zu verändern oder durch Ausdrücke zu berechnen. Es wird kaum möglich sein den vollen Umfang aller denkbaren Funktionen bereitzustellen und auch Ausdrücke können sehr schnell enorm kompliziert werden.

**Unsure:** Ich bin mir noch nicht sicher, wie ich das gerne handhaben möchte. Möglicherweise ein separater Editor für Ausdrücke in einer Leiste “unterhalb” der eigentlichen Abfrage? Auf jeden Fall muss ich hier den Umfang sehr deutlich vorher abstecken.

### 3.6.7 Unterabfragen

Noch komplizierter wird der Umgang mit Ausdrücken, wenn man in diesen auch Unterabfragen zulassen möchte.

**Unsure:** Hier das gleiche Problem wie mit Ausdrücke, ich tendiere momentan zu “mach ich nicht, sehe ich wenn überhaupt konzeptuell vor”.

### 3.6.8 Abfragen mit Parametern

**Info:** Neben festgelegten Vergleichen mit Beziehungen oder konstanten Werten soll es möglich sein einzelne Werte zur Laufzeit vom Benutzer zu erfragen. Dieser Aspekt der Beschreibung hat also viel Bedeutung für den Editor der Oberflächen.

### 3.6.9 Manipulation von Daten

**Info:** Bisher viel Query Language beschrieben, INSERT, UPDATE und DELETE sollten sich aber vergleichsweise einfach implementieren lassen.

## 3.7 Beispielhafte Abfragen

In den folgenden Beispielen steht jeder nummerierte Punkt für einen vom Lernenden abgeschlossenen Schritt, nach welchem die entstandene Query syntaktisch korrekt sein sollte und daher unmittelbar ausgeführt wird. So erhält der Benutzer ein unmittelbares Feedback über die Auswirkung seiner Änderung.

Wir beginnen zunächst mit einer einfachen Abfrage aus nur einer Tabelle, deren Daten relativ einfach eingeschränkt und projiziert werden sollen.

1. Zu diesem Zeitpunkt steht dem Benutzer nur die Auswahl einer Tabelle für die FROM-Komponente offen. Da eine Abfrage ohne Angabe einer SELECT-Komponente ungültig wäre, wird diese automatisch ergänzt. Eine Einschränkung der Spalten findet dabei nicht statt, es handelt sich also konkret um einen `SELECT * FROM <table>` Ausdruck.

2. Nun soll die Ergebnismenge mit einer `WHERE`-Komponente eingeschränkt werden. Zur Verfügung steht dem Benutzer der unscharfe Vergleich von Strings mit `LIKE` und die typischen Vergleichsoperatoren `=`, `<>`, `>`, `<`, `>=` und `<=`.
3. Die Angabe von weiteren Ausdrücken erfordert nun die Angabe eines logischen Operators. Da in SQLite das logische `AND` Vorrang vor dem logischen `OR`, wäre eine entsprechende Hervorhebung vermutlich hilfreich.

## 3.8 Oberflächen für Endbenutzer

**Info:** Binden von Werten aus der Abfrage an einfache Eingabelemente wie Textfelder oder Komboboxen.

### 3.8.1 Navigation

### 3.8.2 Erfassen von Beziehungen

**Info:** Das vermutlich komplizierteste Bedienelement.

## 4 Umsetzungsanalyse

**Info:** Das ist jetzt der Abschnitt für Software-Ingenieure.

Um die einfachste Inbetriebnahme der Software für Lernende zu gewährleisten, wird die Anwendung für Webbrowser entwickelt. Um erste Schritte mit SQL zu machen reicht dann ein beliebiger aktueller Browser. Diese Entscheidung bedeutet praktisch vor allem eine Verschiebung der Probleme mit der Inbetriebnahme auf z.B. eine Lehrperson.

Ebenfalls aus Gründen der einfacheren Zugänglichkeit sollte eine einzelne dazugehörige Serverinstanz in der Lage sein, mehrere Projekte simultan zu bedienen. Die Lehrperson kann also mit einem einzigen Serverprozess eine ganze Klasse versorgen.

### 4.1 Systemübersicht

#### Server: Ruby mit Sinatra

Die Aufgaben des Servers sollen sich konzeptionell möglichst auf die Auslieferung und Speicherung von Daten beschränken. Die Interaktion findet dabei primär über eine REST-artige JSON Schnittstelle statt.

### **Client: Typescript mit Angular 2**

Aufgrund des hohen Grades an Interaktivität bietet sich eine rein clientseitige Visualisierung an, die weitestgehend auf Roundtrips zum Server verzichtet.

## **4.2 Verwaltung von Projekten**

Um den Betrieb für Schüler und Lehrer zu vereinfachen, wird eine Instanz des Servers also in der Lage sein mehrere Schülerprojekte gleichzeitig bereitzustellen. Dafür ist es aber zunächst einmal notwendig zu definieren, wie ein solches Projekt überhaupt strukturiert ist.

## **4.3 Verwendung von Domänenspezifischen Sprachen**

Da die Verarbeitung von beliebig komplexen SQL-Abfragen ebenfalls nicht Bestandteil dieser Arbeit ist, erfolgt die Speicherung der Abfrage in einer eigenen, maschinenlesbaren Notation. Gleiches gilt für die Beschreibung der Benutzeroberfläche.

Aus praktischen Gründen setzen beide dieser Sprachen auf XML auf, ganz konkret sogar auf HTML.

## **4.4 Roadmap**

Diese Arbeitsschritte stellen eine eher technische Roadmap der technischen Umsetzung dar.

### **Abstrakte Repräsentation von SQL(ite) Schemata**

Als unmittelbare Eingabe für die Entwicklungsumgebung sollen einigermaßen einfache, aber grundsätzlich beliebige SQLite Datenbanken dienen. Diese Datenbanken stellen den Ausgangspunkt für die Schülerprojekte dar und wurden von einem externen Tool oder der Lehrkraft erzeugt.

### **Visualisierung des Schemas**

Auch wenn die Projekte der Lernenden im Normalfall auf sehr überschaubare Datenbanken aufbauen werden, ist eine vernünftige Visualisierung der beteiligten Tabellen und deren Beziehung untereinander essentiell.

### **Visualisierung der Daten**

In diesem Schritt werden keine Experimente vorgenommen: Es geht um die einfache tabellarische Auflistung der Daten.

### **Ausführung von beliebigen SELECT-Abfragen**

Dieser Schritt dient der Vorbereitung des im nächsten Schritt zu implementierenden grafischen Editors und beinhaltet insbesondere die Konzeption von begleitenden Bedienelementen.

## Grafischer Editor für SELECT-Abfragen








Als alternative zu dem Freitexteditor soll nun ein grafischer Editor implementiert werden. Dieser benutzt eine abstrakten interne Darstellung um aufwändige Parsingvorgänge von SQL-Strings zu vermeiden.

## Optional: Limitierung der verfügbaren Möglichkeiten

## Notes

■ Deutsch? . . . . .	II
■ Deutsch? . . . . .	II
■ <b>Unsure:</b> Momentan eher eine sehr lose Sammlung denn eine tatsächliche Recherche. . . . .	2
■ <b>Info:</b> Paradebeispiel für eine schülerorientierte Entwicklungsumgebung, insbesondere auch eine Betrachtung der visuellen Programmierung. . . . .	2
■ <b>Info:</b> Generator für Datengetriebene Geschäftsanwendung mit überschaubarer Applikationslogik. . . . .	2
■ <b>Info:</b> Buch zum Einstieg in die Programmierung mit einem interessanten, sehr Datentypgetriebenen Ansatz. . . . .	2
■ <b>Info:</b> Eher eine Datenbank als eine sinnvolle Eingabemaske für unversierte Benutzer. . . . .	2
■ <b>Info:</b> Wirklich ein Datenbanktool, keinerlei Benutzerschnittstelle für “normale” Benutzer. . . . .	2
■ <b>Info:</b> Der Verzicht auf die technischen Hintergründe in diesem Kapitel soll es auch für “normale” Informatiklehrer verständlich machen, ohne Sie gleich mit den Hintergründen der technischen Realisierung zu erschlagen. . . . .	3
■ Deutsch? . . . . .	4
■ <b>Unsure:</b> Theoretisch ist die Menge an denkbaren Systemen fast unüberschaubar groß, praktisch sticht SQLite aus der Masse an Optionen heraus. Wie ausführlich muss ich das begründen? . . . . .	5
■ <b>Todo:</b> Quelle mit Beleg für “Kinderfreundliches” buntes Design suchen. Das wird man bei Scratch schon aus gutem Grund gemacht haben. Relevante Frage: Zielgruppe (in Jahren) von Scratch vs. Zielgruppe dieses Projektes? . .	6
■ <b>Info:</b> Im Editor immer zu sehen, aber aktuell nicht beschrieben: Eine Tabelle mit den Daten, welche die Abfrage im aktuellen Zustand liefern würde. . . .	8



	<b>Unsure:</b> Ich bin mir noch nicht sicher, wie ich das gerne handhaben möchte. Möglicherweise ein separater Editor für Ausdrücke in einer Leiste “unterhalb” der eigentlichen Abfrage? Auf jeden Fall muss ich hier den Umfang sehr deutlich vorher abstecken. . . . .	9
	<b>Unsure:</b> Hier das gleiche Problem wie mit Ausdrücke, ich tendiere momentan zu “mach ich nicht, sehe ich wenn überhaupt konzeptuell vor”. . . . .	10
	<b>Info:</b> Neben festgelegten Vergleichen mit Beziehungen oder konstanten Werten soll es möglich sein einzelne Werte zur Laufzeit vom Benutzer zu erfragen. Dieser Aspekt der Beschreibung hat also viel Bedeutung für den Editor der Oberflächen. . . . .	10
	<b>Info:</b> Bisher viel Query Language beschrieben, INSERT, UPDATE und DELETE sollten sich aber vergleichsweise einfach implementieren lassen. . . . .	10
	<b>Info:</b> Binden von Werten aus der Abfrage an einfache Eingabelemente wie Textfelder oder Komboboxen. . . . .	11
	<b>Info:</b> Das vermutlich komplizierteste Bedienelement. . . . .	11
	<b>Info:</b> Das ist jetzt der Abschnitt für Software-Ingenieure. . . . .	11

## Literatur

- [1] *Lehrplan für Angewandte Informatik in der Sekundarstufe I.* 2010. URL: <http://lehrplan.lernnetz.de/index.php?wahl=138>.
- [2] *Lehrplan für Angewandte Informatik in der Sekundarstufe II.* 2002. URL: <http://lehrplan.lernnetz.de/index.php?DownloadID=73>.