aus

Oliver J. Bott, Peter Fricke, Uta Priss, Michael Striewe (Hrsg.)

Automatisierte Bewertung in der Programmierausbildung

Digitale Medien in der Hochschullehre Band 6
2017, 420 Seiten, br., 42,90 €, ISBN 978-3-8309-3606-0

Waxmann Verlag GmbH
www.waxmann.com info@waxmann.com

12 Der Grader aSQLg

Felix Heine und Carsten Kleiner

Zusammenfassung
In diesem Kapitel beschreiben wir das Konzept sowie Implementierungsdetails des Graders aSQLg, der zur Bewertung von SQL-Aufgaben verwendet wird. Zielgruppe der Software sind Studierende in typischen Datenbankvorlesungenen, die im Rahmen der Veranstaltung SQL lernen. Das aSQLg-System ermöglicht es zum einen dem Dozenten bzw. der Dozentin, Aufwand bei der Korrektur von Übungsarbeiten einzusparen, zum anderen unterstützt das System die Studierenden beim Erlernen von SQL, da es möglich ist, die gleiche Aufgabe mehrfach abzugeben und dabei mit Hilfe der Rückmeldungen des Systems die Antwort kontinuierlich zu verbessern.

12.1 Einleitung und Motivation

Das Beherrschen der Sprache SQL ist, neben der Modellierung, ein wichtiges Ziel von Einführungsvorlesungen im Bereich Datenbanken. Die automatische Bewertung von Modellierungsaufgaben erscheint schwierig, wohingegen dies für SQL-Aufgaben gut von Gradern übernommen werden kann. In Einführungsveranstaltungen ist dies besonders wünschenswert, da hier normalerweise große Gruppen unterrichtet werden und die Heterogenität besonders ausgeprägt ist.

Der Grader aSQLg verfolgt daher zwei Hauptziele: zum einen soll er die Studierenden im Lernprozess unterstützen, indem eine Aufgabe ggf. mehrfach und mit Hilfe von Hinweisen des Systems bearbeitet werden kann, und zum anderen soll das Lehrpersonal bei der Korrektur großer Mengen von Aufgabenzetteln entlastet werden. Zusätzlich können die Studierenden zeit- und raumunabhängig die Aufgaben bearbeiten.

Der Einsatz von aSQLg beschränkt sich aber nicht auf Einführungsvorlesungen. Auch in späteren Veranstaltungen kann das System gut verwendet werden, wenn bestimmte SQL-Features unterrichtet werden oder zur Wiederholung in Veranstaltungen, die SQL-Kenntnisse bei den Studierenden voraussetzen.

Da häufig in Kursen bereits ein LMS wie z. B. Moodle im Einsatz ist, beinhaltet aSQLg kein eigenes Interface. Der Grader ist als Bibliothek konzipiert, die in unterschiedlichen Systemen zum Einsatz kommen kann. Es existiert eine Schnittstelle zu Grappa (siehe Kapitel 23), über die aSQLg z. B. in Moodle verwendet wird. Dadurch müssen sich die Studierenden nicht mit einem weiteren System auseinandersetzen, sondern können aSQLg im gewohnten Kontext verwenden.

aSQLg wurde bereits in zahlreichen Kursen unterschiedlicher Niveaus eingesetzt. Evaluationen zeugen von einer generellen Zufriedenheit bei den Studierenden (vgl. [KTH13]). Aufgezeigte Schwächen wurden im Rahmen des Entwicklungsprozesses aufgegriffen und führten u. a. zur Detaillierung der Rückmeldungen.

Nach einem Überblick über verwandte Arbeiten beschreiben wir das Konzept und die Bewertungsschritte des Graders. Darauf folgt eine exemplarische Interaktion eines Studierenden mit dem Grader. Vor der Zusammenfassung beschreiben wir noch aktuelle Entwicklungen und Planungen für den Grader.

12.2 Verwandte Arbeiten

Andere Arbeiten mit Bezug zu aSQLg können grob in zwei Bereiche eingeteilt werden. Zum einen Systeme, die das Lernen von SQL unterstützen und zum anderen solche, die automatisch Lösungen prüfen und bewerten.

Wir betrachten zunächst Publikationen, die sich mit der Unterstützung im Lernprozess beschäftigen. In [Mit03] wird ein solches System beschrieben. Die Studierenden bekommen sofortiges Feedback zu den eingeschickten SQL-Statements. Während der Bearbeitung werden die Studierenden unterstützt, bis sie eine korrekte Lösung gefunden haben. Ähnlich arbeitet das in [KP05b] beschriebene System. In diesem System werden auch über SQL-Abfragen hinausgehende Aspekte berücksichtigt. Beide Systeme haben ein eigenes webbasiertes Frontend. Im Gegensatz zu diesen Systemen setzt aSQLg auf eine Integration in die bekannte LMS-Umgebung.

In den Veröffentlichungen [Pri03; PL04] beschreiben die Autoren ein Bewertungsschema für SQL-Abfragen. Ziel ist eine Unterstützung des Lernprozesses unter der Vermutung, dass die Art der Bewertung das Lernverhalten beeinflusst. Wie auch bei aSQLg steht die interaktive Verbesserung der Antwort im Zentrum, wobei unser Ansatz detaillierter ist. In [DRL07] wird das Werkzeug SQLify beschrieben, welches eine teilweise automatisierte Bewertung von SQL-Abfragen unterstützt. Hier ist allerdings der Schwerpunkt auf Peer-Reviews und Interaktion zur Unterstützung des Lernprozesses, so dass keine vollständige Automatisierung

angestrebt wird. In [Abe+08] wird eine Architektur für SQL-Lernunterstützung und Bewertung beschrieben. Die Bewertungskomponente ist allerdings nicht detailliert erläutert.

In [CP09] ist eine Anwendung basierend auf dem GNU SQL Tutor (siehe [Gnu]) beschrieben. Die Anwendung beinhaltet sowohl einen Teil zum Bewerten als auch zur Lernunterstützung. Der Bewertungsteil ist allerdings nicht detailliert beschrieben. Die Bewertung für einzelne Aufgaben scheint binär zu sein.

Die Generierung von unterschiedlichen Aufgaben aus Vorlagen wird in SQL-KnoT [Bru+10] vorgestellt. Wir halten dies für ein sehr sinnvolles Feature und arbeiten an der Integration einer vergleichbaren Funktionalität.

Im Werkzeug ÜPS, das unter anderem in [Ifl+14b] präsentiert wird, liegt der Fokus auf der Benutzerfreundlichkeit des Systems für den Lehrenden. Es werden auch hier syntaktische Prüfungen sowie Korrektheitsprüfungen des Ergebnisses durchgeführt. Ferner wird durch eine Strukturanalyse versucht, dem Lernenden Unterstützung bei der Verbesserung seiner Lösung zu geben. Das System ist eher als eine eigenständige Anwendung konzipiert und nicht primär für die Integration in ein LMS gedacht. Ferner ist das System eher auf kleine Übungsdatenbanken ausgelegt, da der Lehrende ein Skript zum Anlegen der Testdatenbank mit übergeben muss, das für jeden Lernenden ein eigenes Datenbankschema erzeugt. Dieses Verfahren wäre für sehr große Datenbestände aus Performance-Gründen nicht geeignet. Große Datenbestände ermöglichen es, Aufgaben zu in der Praxis typischen Aspekten von Datenbanksystemen wie bspw. Laufzeitverhalten von Anfragen bei großen Datenbeständen zu stellen. Das System bietet einen Übungs- und einen Abgabebereich, eignet sich also sowohl für den Übungsbetrieb wie auch für Prüfungen/Tests.

Die meisten Tools setzen wie aSQLg u. a. auf einen Ergebnisvergleich um die Korrektheit einer Lösung zu prüfen. Eine alternative Vorgehensweise wird im System SQLator vorgeschlagen [Sad+04]. Hier werden Heuristiken eingesetzt, um die Äquivalenz zwischen einer studentischen Lösung und der Musterlösung festzustellen. Der Nachteil ist, dass möglicherweise korrekte Lösungen nicht akzeptiert werden. Eine Mittelweg wird in [Dol10] vorgestellt. Hier dient der Ergebnisvergleich zur Korrektheitsbestimmung. Ein semantischer Vergleich wird genutzt, um Vorschläge zur Verbesserung der Lösung zu erzeugen.

12.3 Konzept zur automatisierten Bewertung von SQL-Aufgaben

In diesem Abschnitt beschreiben wir unser Konzept zur Bewertung von SQL-Aufgaben. Das Kapitel konzentriert sich dabei auf die Bewertung von Abfragen (Select-Statements). Auf andere Typen von SQL-Kommandos gehen wir im Ausblick ein. Das Konzept ist in Abbildung 12.1 als Flussdiagramm dargestellt. Die Schritte zur Bewertung eines SQL-Statements sind im Einzelnen:

1. Prüfung des Statements auf untersagte oder geforderte Elemente (optional),
2. Prüfung auf syntaktische Korrektheit,
3. Prüfung auf Kosten für die Ausführung,
4. Prüfung der Korrektheit des Ergebnisses,
5. Prüfung des Stils,
6. Ermittlung der Punkte und Erstellung des Berichts.

Als Eingabe für die Prüfung dient zum einen die studentische Abgabe und zum anderen eine vorab vom Dozenten erstellte Musterlösung in Form eines SQL-Statements, welches das in der Aufgabe erwünschte Ergebnis liefert.

Im ersten Schritt wird das Statement auf untersagte oder geforderte Elemente überprüft. Hiermit kann eine bestimmte Lösung für eine Aufgabe erzwungen werden, falls die Aufgabe z. B. ein bestimmtes Konstrukt üben soll. Z. B. kann ein Aufgabentext den Zusatz „Lösen Sie die Aufgabe ohne Verwendung von EXISTS“ enthalten, wenn der Dozent eine Lösung mit Hilfe eines Verbunds wünscht. Die Aufgabe kann dann so konfiguriert werden, dass „EXISTS“ als nicht zulässiges Schlüsselwort aufgeführt wird. Ein Statement mit „EXISTS“ würde dann in diesem Schritt ohne weitere Prüfung abgewiesen. Zur Durchführung dieser Prüfung ist eine syntaktische Analyse des SQL-Statements nötig. Hierzu greift aSQLg auf den JSqlParser [Jsq] zurück. Der Parser erzeugt einen Parse-Baum, welcher von aSQLg durchlaufen wird, um erlaubte oder verbotene Elemente zu identifizieren.

Im nächsten Schritt wird die syntaktische Korrektheit des Statements überprüft. Dies geschieht mit Hilfe der Zieldatenbank, da es ggf. Unterschiede im SQL-Dialekt zwischen JSqlParser und der Zieldatenbank (z. B. Oracle) gibt. Das Statement muss in jedem Fall eine passende Syntax für die Zieldatenbank aufweisen; die Kompatibilität mit dem JSqlParser-Dialekt ist nur erforderlich, falls

der Dozent die optionalen Prüfschritte verwendet. Je nach Konfiguration muss der Dozent die Studierenden über den geforderten Dialekt informieren.

Der genaue technische Ablauf der Syntaxprüfung ist abhängig von der Zieldatenbank. Im Falle von Oracle wird die Syntaxprüfung und die Kostenprüfung in einem Schritt durchgeführt, indem der Ausführungsplan angefordert wird. Wenn dieser nicht erstellt werden kann, wird das Statement als syntaktisch fehlerhaft zurückgewiesen. Für andere DBMS-Typen kann diese Prüfung angepasst werden.

Die Kostenprüfung dient im Wesentlichen dazu, ggf. fehlerhafte Statements abzuweisen, die beim Prüfen zu viel Last auf der Datenbank erzeugen würden. Dazu kann in der Konfiguration eine absolute oder eine relative Kostenobergrenze hinterlegt werden. Die relative Obergrenze bezieht sich auf die Kosten der Musterlösung. Wenn die Obergrenze überschritten wird, wird die Korrektheitsprüfung übersprungen, um das Prüfsystem nicht zu überlasten. Für die Kosten können auch Punkte vergeben werden, wenn z. B. mit Hilfe von Optimizer-Hints effizientere Lösungen möglich sind und dies Ziel der Aufgabe ist.

Im nächsten Schritt wird die Korrektheit des Statements ermittelt. Kern dieser Prüfung ist ein Ergebnisvergleich der studentischen Lösung mit der Musterlösung,

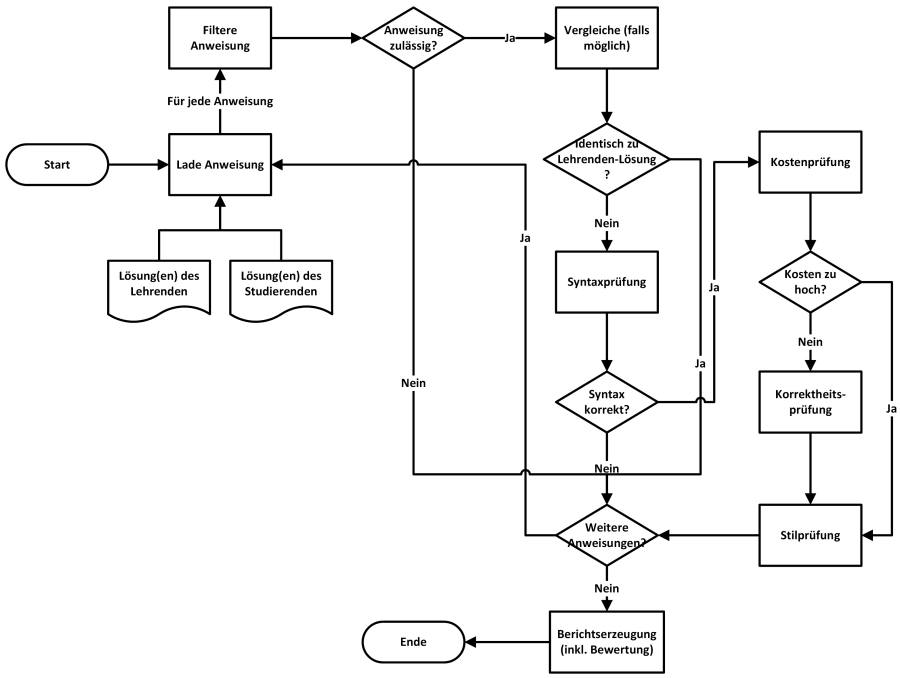

Abbildung 12.1: Flussdiagramm des Grading-Ablaufs

wie in Abbildung 12.2 gezeigt. Um ein genaueres Feedback sowie eine abgestufte Punktevergabe zu ermöglichen, werden allerdings noch weitere Prüfungen durchgeführt.

Zunächst wird geprüft, ob die Ergebnisspalten der beiden Statements in Anzahl, Benennung und Datentyp übereinstimmen. Entsprechende Hinweise können bei der Fehlersuche hilfreich sein. Wenn z. B. eine Spalte mit einem Datum in der Musterlösung den Datentyp „String“ und in der studentischen Lösung den Typ „Date“ hat, kann ggf. geschlossen werden, dass eine passende Formatierung des Datums Teil der Aufgabe ist. Anschließend wird die Zeilenanzahl zwischen den beiden Lösungen verglichen, um Hinweise auf Abweichungen zu geben. So kann z. B. eine zu hohe Zeilenanzahl in der studentischen Lösung ein Hinweis auf einen vergessenen Teil in der „WHERE“-Klausel sein. Zum Abschluss der Korrektheits-

```
( (<student-statement>)
    UNION
  (<reference-statement>))
MINUS
( (<student-statement>)
    INTERSECT
  (<reference-statement>));
```

Abbildung 12.2: Statement zur Korrektheitsprüfung

prüfung wird noch die Reihenfolge der Zeilen geprüft, falls die Musterlösung eine „ORDER-BY“-Klausel enthält.

Der letzte Schritt, in dem Punkte vergeben werden, ist die Stilprüfung. Da es für SQL keine allgemein anerkannten Stilregeln gibt, kann der gewünschte Stil bei aSQLg individuell angepasst werden. Dazu kann der Dozent Stilregeln definieren, deren Einhaltung den Studierenden Punkte einbringt. Es kann z. B. vor bestimmten Teilen ein Zeilenumbruch gefordert werden (z. B. vor der „WHERE“-Klausel), oder es kann gefordert werden, dass Schlüsselwörter in Großschrift geschrieben werden.

Am Ende der Prüfung wird ein Bericht erstellt, der alle Punkte und Feedback sowohl für den Studenten als auch für den Dozenten enthält. Dieser Bericht wird intern als XML-Dokument vorbereitet und über ein Style-Sheet in ein Ausgabeformat wie Text oder HTML umgewandelt. Diese Style-Sheets können angepasst werden, um eine passende Ausgabe zu erreichen. Die fertig formatierten Berichte werden dann z. B. über Grappa an Moodle weitergegeben und dort angezeigt.

Die relative Gewichtung der Punkte aus den einzelnen Schritten lässt sich ebenfalls konfigurieren. So können z. B. bei den ersten Aufgaben bereits 25 % der

Punkte für syntaktische Korrektheit vergeben werden, was ggf. bei fortgeschrittenen Studierenden nicht mehr sinnvoll ist.

Da wir Code der Studierenden auf der Datenbank ausführen, könnten Sicherheitslücken entstehen. Bösartige Studierende könnten versuchen, über diesen Weg die Datenbank anzugreifen. Allerdings ist unser Konzept durch zwei Dinge gut geschützt: Zum einen wird der Code der Studierenden nicht als eigenes Statement ausgeführt, sondern immer nur in Unterabfragen wie in Abbildung 12.2 gezeigt. Zum anderen wird für das Prüfen ein Datenbankzugang verwendet, der nur sehr eingeschränkte Leserechte benötigt. Wenn dieser passend konfiguriert ist, ist ein entsprechender Angriff unmöglich. Bei der geplanten Erweiterung auf DML- und DDL-Kommandos (vgl. Abschnitt 12.6) müsste auch das Sicherheitskonzept angepasst werden, da weder eine Einbettung noch ausschließliche Leserechte möglich sind.

Studierende könnten weiterhin versuchen das System auszunutzen, indem sie z. B. ein Statement „SELECT 5 FROM DUAL“ eingeben, um zumindest Punkte für syntaktische Korrektheit, Kosten und Stil zu bekommen. Als Gegenmaßnahme könnte im Filterschritt z. B. die Verwendung der „DUAL“-Tabelle verboten werden.

Generell ist allerdings anzumerken, dass der Korrektheitstest auf Identität des Ergebnisses beruht, und kein semantischer Vergleich der Lösung mit der Musterlösung stattfindet. Dies bedeutet natürlich, dass Lösungen, die mit völlig anderen Mitteln die korrekte Ergebnisrelation erstellen, als korrekt bewertet werden. Ebenso ist es aktuell nicht vorgesehen, besonders elegante oder ggfs. sogar effizientere Lösungen als die Musterlösung mit besonders positivem Feedback zu versehen. Eine Plagiatserkennung ist derzeit auch nicht vorgesehen. Abhilfe zu diesem Punkt diskutieren wir im Abschnitt 12.6.

12.4 Beispiele

In diesem Abschnitt zeigen wir ein Beispiel aus einer einführenden Datenbankvorlesung. Zu der Veranstaltung gehört eine SQL-Einführung, in deren Rahmen die Studierenden ca. 40 SQL-Aufgaben lösen müssen. Das Beispiel stammt aus dem Beginn der SQL-Einführung. Es wurde ein bekanntes Oracle-Beispielschema verwendet („employees“). Die Aufgabe war es, alle Angestellten mit dem kompletten Namen als ein String sowie dem Einstellungsdatum zu selektieren. Das Ergebnis musste nach Einstellungsjahr und Nachname sortiert werden. Die Musterlösung ist folgende:

```
SELECT first_name || ' ' || last_name name,
TO_NUMBER(TO_CHAR(hire_date, 'YYYY')) hired
FROM hr.employees
ORDER BY hired, last_name;
```

Wir beschreiben jetzt ein vereinfachtes Protokoll eines Studenten, der die Aufgabe lösen wollte. Der erste Versuch war folgende Antwort:

```
Select First_Name||' '||Last_Name,
to_char(Hire_Date, YEAR)
from hr.employees
order by Hire_Date, Last_Name;
```

Die Syntaxprüfung des Graders schlug fehl. Als Teil des Ergebnisberichts wurde auch die Datenbankfehlermeldung von Oracle ausgegeben:

```
\texttt{ORA-00904: "YEAR": invalid identifier.}}
```

Aufgrund der fehlerhaften Syntax wurden alle weiteren Prüfungen übersprungen. Im nächsten Schritt hat der Student den Syntaxfehler korrigiert:

```
... to_char(Hire_Date, 'YYYY') ...
```

Da der Syntaxcheck jetzt erfolgreich war, konnte aSQLg die nächsten Schritte durchführen. Die Kostenprüfung war in diesem Fall unkritisch. Als erster Teil der Korrektheitsprüfung wurden die Spalten vergleichen. Die Spaltenanzahl war zwar wie erwartet zwei, aber die Typprüfung schlug für die Datumsspalte mit folgender Meldung fehl:

```
Datatype of column 2 is wrong.
Expected: NUMBER, your solution: VARCHAR2.
```

Dies veranlasste den Studenten zu folgender Modifikation:

```
... to_number(to_char(Hire_Date, 'YYYY')) ...
```

Die Änderung war erfolgreich, und somit konnte aSQLg die Spaltenprüfung erfolgreich beenden. Zeilenanzahl und Dateninhalte waren ebenfalls korrekt. Allerdings waren die Zeilen noch nicht wie erwünscht sortiert, da die Musterlösung nach Jahr und nicht nach dem gesamten Datum sortiert war. In der letzten Einsendung reagierte der Student auf den entsprechenden Hinweis:

```
Select First_Name||' '||Last_Name,
to_number(to_char(hire_date,'YYYY')) AS datum
from hr.employees
order by datum,Last_Name;
```

Damit konnte die Lösung mit voller Punktzahl akzeptiert werden. Es blieben nur Warnungen bzgl. der Spaltenbenennung, welche aber nicht zu Punktabzug führen. Eine Stilprüfung war in diesem Beispiel nicht konfiguriert.

Insgesamt kann man in diesem Beispiel sehen, wie das detaillierte Feedback Studierende Schritt für Schritt zu richtigen Lösung führen kann.

12.5 Dozentenunterstützung und LMS-Einbettung

Um einen möglichst breiten Einsatz eines Werkzeugs zu erreichen, kommt der Benutzungsfreundlichkeit sowohl für die Lehrenden wie auch für die Lernenden eine große Bedeutung zu. Das Tool aSQLg war ursprünglich als Plugin für das Werkzeug WebCAT ([AEPQ06]) realisiert worden, da dieses automatisierte Programmbewertung für verschiedene Programmiersprachen anbietet. Ein wichtiger Aspekt der Benutzerfreundlichkeit war damit erreicht: ein möglichst breiter Einsatz eines Werkzeugs über mehrere Veranstaltungen. Leider stellte sich heraus, dass die bereitgestellte Oberfläche von WebCAT zum einen weder für Lehrende noch für Lernende besonders benutzerfreundlich war und außerdem auf einer technisch veralteten Basis realisiert worden war. Da WebCAT zudem nicht als vollwertiges LMS konzipiert war, zeigte sich, dass eine Integration von aSQLg in LMS die benutzerfreundlichste Variante für die Lernenden sein würde. Um sich nicht auf ein LMS festlegen zu müssen, wurde das Werkzeug (über eine Zwischenstufe als Standalone-Werkzeug) auf ein Plugin für das Framework Grappa (vgl. Kapitel 23) umgebaut. Damit ist dann eine beliebige LMS-Einbettung möglich, die für die Lernenden maximalen Komfort bei geringem Einarbeitungsaufwand bietet. Die Eingabeschnittstelle für die Lernenden sind Dateien, in denen sie ihre SQL-Kommandos (versehen mit bestimmten minimalen Metainformationen wie Aufgaben-ID) über das LMS bereitstellen. Nach der Bewertung wird das Ergebnis von aSQLg gemäß dem Grappa-Format bereitgestellt und dann im LMS für die Lernenden aufbereitet.

Für die Lehrenden ist der Zugang zum System über ein LMS für den Prozess des Erstellens von Aufgaben zu komplex. Daher wurde basierend auf der o. g. Standalone-Variante für aSQLg eine einfache Java-GUI erstellt, mit der Lehrende Aufgaben und auch Konfigurationsinformationen einfach anlegen und testen können. Die Bedienung von aSQLg über diese Anwendung ist speziell auf die

Bedürfnisse der Lehrenden zugeschnitten und ermöglicht schnelles Erstellen und Testen von Aufgaben. Es können auch Ergebnisse für mögliche studentische Abgaben berechnet und so die Qualität der Unterstützung der Studierenden geprüft werden.

Das Werkzeug aSQLg verwendet intern unterschiedliche Arten von Konfigurationsinformationen, um einen möglichst flexiblen Einsatz zu ermöglichen. Diese Informationen lassen sich grob in die folgenden Bereiche einteilen: Datenbankkonfiguration, Bewertungskonfiguration, Stilprüfungskonfiguration, Sicherheitskonfiguration. Um diese Konfigurationsdateien nicht alle für jede Aufgabe neu anlegen zu müssen (zumal es viele Parameter gibt, die sich nur selten ändern), sieht das Konzept von aSQLg inzwischen vor, dass alle Konfigurationsinformationen selbst in einer Datenbank abgelegt werden und die Inhalte dieser Datenbank dann nur referenziert werden. Die tatsächlich zu übergebende Konfiguration ist damit minimal und kann so im LMS leicht angelegt werden. Bei der Pflege der Detailkonfigurationen in der Datenbank werden Lehrende von der Standalone-GUI unterstützt, so dass auch hier nur minimaler Aufwand anfällt. Abbildung 12.3 illustriert die Lehrendenschnittstelle. Eine Abbildung der aSQLg-Konfigurationen auf

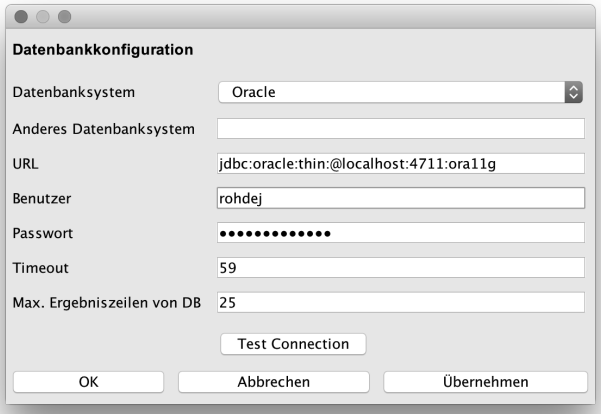

Abbildung 12.3: Screenshot der aSQLg-GUI für Lehrende

das Format zum Austausch von automatisierten Programmieraufgaben (vgl. Kapitel 24) ist ebenfalls erfolgt. Die Standalone-GUI kann die Aufgabenkonfiguration in diesem Format lesen und speichern, um einen Austausch mit anderen Systeme zu erleichtern. Allerdings ist dies bisher nur für die Konfigurationsinformationen und Aufgabentexte realisiert, die Bereitstellung des erforderlichen

Datenbankschemas, auf denen der SQL-Code auszuführen ist, ist bisher nicht implementiert, da das Konzept von aSQLg eine Pflege dieser Daten direkt in der entsprechenden Zieldatenbank vorsieht.

Insgesamt umfasst der Bearbeitungsprozess aus Sicht der Lehrenden die folgenden Aufgaben:

- Anlegen des für die Aufgaben zu verwendenden Datenbankschemas und Import der Daten direkt auf der Ausführungsdatenbank (meist nur einmal pro Veranstaltung).
- Anlegen der häufig zu verwendenden Konfigurationsinformationen in der aSQLg-GUI und Speicherung dieser Informationen in der Datenbank (Grundkonfiguration in wenigen Fällen pro Veranstaltung).
- Anlegen von Aufgabentexten, Musterlösungen und spezifischen Konfigurationen für jede Aufgabe in der aSQLg-GUI; dieser Prozess kann iterativ mit Test der Ausführung aus der GUI heraus erfolgen. Am Ende wird die erstellte Aufgabe gespeichert.
- Anlegen einer Aufgabe im LMS und Hochladen der zuvor gespeicherten Aufgaben- und Konfigurationsdatei (für jede Aufgabe).
- Optional: Nutzung der Auswertungsfunktionalitäten des LMS, um die Ergebnisse der Lernenden zu einer Aufgabe bewerten und berücksichtigen zu können.

Für die Lernenden stellt sich eine aSQLg-Aufgabe im LMS analog zu anderen Aufgaben im LMS dar, so dass eine einfache Bedienung für die Studierenden in der ohnehin gewohnten Umgebung möglich ist.

12.6 Ausblick

In diesem Abschnitt beschreiben wir geplante Erweiterungen des Systems. Dazu gehört das Thema Plagiatsprüfung, die Verfeinerung der Bewertung und des Feedbacks, sowie die Unterstützung von weiteren Typen von SQL-Befehlen.

Wie oben beschrieben, gibt es derzeit in aSQLg keine Plagiatsprüfung. Da das System jedes Statement individuell prüft, ist dies auch im derzeitigen Rahmen nicht lösbar, bzw. würde die persistente Speicherung der bisherigen Lösungen erfordern. Eine weitere Schwierigkeit ergibt sich insbesondere bei einfachen Aufgaben. Diese haben häufig nur eine bzw. wenige sinnvolle Lösungen, so dass eine Plagiatsprüfung zu häufig anschlagen würde.

Wir bevorzugen daher eine Lösung auf Basis der Individualisierung von Aufgaben. Die Grundidee ist dabei, dass Aufgaben mit Platzhaltern variabel gestaltet werden. Ein neues Modul kann dann auf Basis einer solchen Vorlage eine konkrete Aufgabe generieren, wobei z. B. Konstanten verändert werden, Abfragen verändert werden oder sogar andere Spalten genutzt werden. Der Aufgabentext wird entsprechend angepasst. Der eigentliche Bewertungsprozess läuft dann identisch ab, nur kann aufgrund der Veränderungen keine Lösung mehr kopiert werden. Neben der Verhinderung von Plagiaten hat diese Lösung den weiteren Vorteil, dass Studierende zum Üben weitere Variationen einer Aufgabe anfordern können.

Auch wenn der Bewertungsprozess schon verschiedene Teilprüfungen durchläuft, ist eine weitere Verfeinerung denkbar. Zum einen könnten über einen semantischen Vergleich weitere Hinweise zur Verbesserung erzeugt werden. So könnte z. B. erkannt werden, dass das Statement im Prinzip korrekt ist, allerdings in einem Vergleich fälschlicherweise „<“ statt „<=“ verwendet wurde. Diese Analysen könnten neben dem verbesserten Feedback an die Studierenden auch zur weiteren Verbesserung der Abstufung der Bewertungen genutzt werden, mit dem Ziel, dass jede schrittweise Verbesserung auch mehr Punkte erzielt und damit zur Motivation der Studierenden beiträgt.

Aktuell unterstützt aSQLg nur SQL-Abfragen. Wir haben bereits ein Konzept zur Erweiterung auf DML- und DDL-Kommandos erarbeitet. Dies würde es ermöglichen, z. B. „CREATE TABLE“- oder „INSERT“-Anweisungen zu prüfen. Dies erfordert jeweils ein gleiches initiales Schema vor der Bewertung, welches im Anschluss wieder bereinigt werden muss. Ansätze sind bereits prototypisch realisiert worden, diese müssen jedoch noch erweitert und vervollständigt werden. Insbesondere ist das bisher eingesetzte Sicherheitskonzept entsprechend zu erweitern bzw. verbessern.

12.7 Zusammenfassung

In diesem Kapitel haben wir das Konzept von aSQLg vorgestellt. Kern von aSQLg ist ein Bewertungsmodul, welches studentische Lösungen für SQL-Abfragen bewertet und ggf. Hinweise zur Verbesserung der Lösung erstellt. Die Bewertung wird dabei anhand der folgenden Kriterien vorgenommen: syntaktische Korrektheit, Effizienz, Korrektheit des Ergebnisses und Stil des SQL-Statements. Der genaue Ablauf der Prüfung und die Punktevergabe sind konfigurierbar. Optional können über Filter bestimmte Lösungen verboten werden. Die Prüfung läuft so ab, dass auch böswillige Studierende das Prüfsystem nicht kompromittieren können.

Das Ausführen von Statements, welche zu große Last auf das Prüfsystem bringen würden, wird ebenfalls verhindert.

Das System befindet sich seit einigen Jahren im Einsatz. Das Feedback des Systems wurde anhand der im Einsatz gewonnenen Erfahrung (durch Gespräche mit Studierenden und systematische Evaluationen) kontinuierlich verbessert. Ziel ist es, das System so zu gestalten, dass Studierende bestmöglich beim Lernen von SQL unterstützt werden.

Das System kann sowohl zur Lernunterstützung als auch zur Bewertung verwendet werden. Zur reinen Bewertung wird die Wiederholung unterbunden, so dass eine eingeschickte Lösung ohne weitere Verbessungsmöglichkeiten bewertet wird. Das System kann in unterschiedliche LMS eingebunden werden und ermöglicht den Studierenden dadurch, ihre gewohnte Umgebung weiterhin zu nutzen. Das Erlernen der Bedienung einer speziellen Oberfläche ist somit überflüssig.

Durch die automatische Bewertung sind große Effizienzgewinne möglich. Auf der einen Seite wird der Aufwand der Dozenten zur Korrektur minimiert. Auf der anderen Seite müssen Studierende nicht mehr Tage auf die Korrektur ihrer Lösung warten, sondern bekommen sofortige Rückmeldungen, auf die sie direkt mit einer verbesserten Lösung reagieren können. Die schrittweise Erhöhung der Punktzahl wirkt bei manchen Studierenden auch motivierend mit dem Ziel, die volle Punktzahl zu erreichen.

Bereits heute ist das System eine wertvolle Komponente in der SQL-Ausbildung. Durch im Abschnitt 12.6 beschriebene Verbesserungen wird aSQLg in Zukunft noch flexibler einsetzbar sein.

Literatur für dieses Kapitel

[Abe+08] Alberto Abelló u. a. „LEARN-SQL: Automatic Assessment of SQL Based on IMS QTI Specification“. In: *ICALT*. IEEE, 2008, S. 592–593.

[AEPQ06] Rahul Agarwal, Stephen H. Edwards und Manuel A. Pérez-Quiñones. „Designing an adaptive learning module to teach software testing“. In: *Proceedings of the 37th SIGCSE technical symposium on Computer science education*. SIGCSE ’06. ACM, 2006, S. 259–263. DOI: 10.1145/1121341.1121420.

[Bru+10] Peter Brusilovsky u. a. „Learning SQL Programming with Interactive Tools: From Integration to Personalization“. In: *Trans. Comput. Educ.* 9.4 (Jan. 2010), 19:1–19:15. DOI: 10.1145.1656255.1656257.

[CP09] Aleš Cepek und Jan Pytel. „SQLtutor“. In: *Professional Education 2009 – FIG International Workshop Vienna*. FIG Fédération Internationale de Géomètres, 2009.

[Dol10] Robert Dollinger. „SQL Lightweight Tutoring Module – Semantic Analysis of SQL Queries based on XML Representation and LINQ“. In: *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010*. Toronto, Canada: AACE, Juni 2010, S. 3323–3328.

[DRL07] Stijn Dekeyser, Michael de Raadt und Tien Yu Lee. „Computer Assisted Assessment of SQL Query Skills“. In: *ADC*. Hrsg. von James Bailey und Alan Fekete. Bd. 63. CRPIT. Australian Computer Society, 2007, S. 53–62.

[Gnu] *GNU SQL tutor – website*. URL: http://www.gnu.org/software/sqltutor/.

[Ifl+14b] Marianus Ifland u. a. „ÜPS – Ein autorenfreundliches Trainingssystem für SQL-Anfragen“. In: *DeLFI 2014 – Die 12. e-Learning Fachtagung Informatik*. Bd. 233. LNI. GI, 2014, S. 259–264. ISBN: 978-3-88579-627-5.

[Jsq] *JSqlParser – website*. (besucht am 23.07.2011). URL: http://jsqlparser.sourceforge.net/.

[KP05b] Claire Kenny und Claus Pahl. „Automated tutoring for a database skills training environment“. In: *Proceedings of the 36th SIGCSE technical symposium on Computer science education*. SIGCSE '05. ACM, 2005, S. 58–62. DOI: 10.1145/1047344.1047377.

[KTH13] Carsten Kleiner, Christopher Tebbe und Felix Heine. „Automated Grading and Tutoring of SQL Statements to Improve Student Learning“. In: *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*. Koli Calling '13. ACM, 2013, S. 161–168. DOI: 10.1145/2526968.2526986.

[Mit03] Antonija Mitrovic. „An Intelligent SQL Tutor on the Web“. In: *I. J. Artificial Intelligence in Education* 13.2-4 (2003), S. 173–197.

[PL04] Julia Coleman Prior und Raymond Lister. „The backwash effect on SQL skills grading“. In: *ITiCSE*. Hrsg. von Roger D. Boyle, Martyn Clark und Amruth N. Kumar. ACM, 2004, S. 32–36.

[Pri03] Julia Coleman Prior. „Online Assessment of SQL Query Formulation Skills“. In: *ACE*. Hrsg. von Tony Greening und Raymond Lister. Bd. 20. CRPIT. Australian Computer Society, 2003, S. 247–256.

[Sad+04] Shazia Sadiq u. a. „SQLator: an online SQL learning workbench“. In: *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. ITiCSE ’04. ACM, 2004, S. 223–227. DOI: 10.1145/1007996.1008055.