Bachelorarbeit

Erstellung datenbewusster Tests an einem Fallbeispiel aus dem Unternehmenssektor

Guided Creation of Data-Aware Test Cases Based on a Real World Business Use Case

von

Frank Blechschmidt

Potsdam, June 2014

Betreuer

Prof. Dr. Hasso Plattner

Dr. Matthias Uflacker, Thomas Kowark, Keven Richly,

Ralf Teusner, Arian Treffer

Enterprise Platform and Integration Concepts

Kurzfassung

Abstract auf Deutsch

Abstract

Abstract auf English

Danksagung

Ich danke ...

HPI Hasso Plattner Institut

Eigenständigkeitserklärung

Hiermit versichere ich, dass diese Arbeit selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, 5. Juli 2014

_____

(Frank Blechschmidt)

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungverzeichnis

API	Application Programming Interface
ATOM	Atom Syndication Format
BI	*BlogIntelligence*
BI-Impact	*BlogIntelligence*-Impact-Score
Blog	Weblog
HDFS	Hadoop Distributed File System
HITS	Hyperlink-Induced Topic Search
HTTP	Hypertext Transfer Protocol
IR	Information Retrieval
RAM	Random-Access Memory
RPC	Remote Procedure Call
RSS	Rich Site Summary
Splog	Spam Blog
SQL	Structured Query Language
tf*idf	Term Frequency-Inverse Document Frequency
URI	Uniform Resource Identifier
WWW	World Wide Web
XML	Extensible Markup Language

1. Einleitung

In vielen Bereichen, vor allem bei Geschäftsanwendungen, dienen relationale Datenbanken als essentielle Persistenz-Schicht. Bei der Entwicklung solcher Anwendungen spielt neben der Validität auch die Performance eine wichtige Rolle. Wichtigster Indikator dafür ist die tolerierbare Wartezeit, die sich laut psychologischen Studien schon nach 2 Sekunden negativ auf die Aufmerksamkeit der Nutzer auswirkt, sodass sie in ihrem Denkprozess unterbrochen werden [Nah04]. Ein zweiter wichtiger Aspekt ist die Entwicklung anhand von Echt-Daten. Sie wird als Best Practice betrachtet [Pla13, S. 212], da sie die Charakteristiken der realen Welt als Maßstab nutzt. Das frühzeitige Betrachten der Performance einer Anwendung auf Echt-Daten liegt so in der Verantwortung des Entwicklers und soll von Anfang an in den Entwicklungsprozess einfließen.

Um Informationen aus der Datenbank in der Anwendung zu nutzen, erfolgt der Zugriff durch das Einbetten von in SQL geschrieben Anfragen. Die eingebetteten Datenbankanfragen haben zumeist variable Bestandteile und Parameter, für die für Performance-Messungen und -Analysen passende Testwerte ausgewählt werden müssen. Dies wird jedoch häufig durch riesige Datenmengen in unverständlich benannten Tabellen und Attributen zusätzlich erschwert. Mit dem Ziel der Vereinfachung werden in dieser Bachelorarbeit verschiedene Ansätze diskutiert, die das Auswählen relevanter Testwerte durch sinnvolle Vorschläge anhand von Eigenschaften der Datenbank-Informationen unterstützen.

Die vorgestellten Ansätze bilden einen essentiellen Teil in einer Reihe von Konzepten für Entwicklungsumgebungen, die im 3. Kapitel zusammengetragen werden und bei der Entwicklung von Geschäftsanwendungen assistieren. Anschließend wird die Einbettung von SQL in die Programmiersprache der Anwendungen untersucht um auf deren Basis die im Kapitel 5 vorgestellten Algorithmen zur Vorschlagsgenerierung von Test-Daten darzulegen. Im Kapitel 6 wird ergänzend die administrative Architektur für Test-Daten und -Systeme betrachtet. Abschließend werden die vorgestellten Ansätze im Rahmen der Umsetzung einer praxisnahen Geschäftsanwendung evaluiert.

2. Terminologie und Hintergrund

2.1. Geschäftsanwendung

Eine Geschäftsanwendung ist eine komplexe Software, die innerhalb von Unternehmen bzw. Organisationen eingesetzt wird, um Unternehmensfunktionen bzw. -prozesse zu realisieren. Sie ist in den meisten Fällen angebunden an eine oder mehrere Datenbanken und kann durch Interaktionen mit Nutzern und / oder anderen Systemen gesteuert werden. Die Entwicklung solcher Anwendungen ist ein aufwendiger Prozess, bei dem am Ende neben den funktionalen Anforderungen auch die Nicht-funktionalen (z.B. Leistung, Zuverlässigkeit und Skalierbarkeit) erfüllt werden sollen.

2.2. IDE (engl. Integrated Development Environment)

Eine IDE (deutsch: “Integrierte Entwicklungsumgebung”), ist eine Anwendung, die eine Reihe von Werkzeugen, z.B. einen Text-Editor, zum Erstellen von Software beinhaltet. In den letzten Jahren setzt zunehmend der Trend ein die IDE nicht mehr als Desktop-Applikation, sondern als Software-as-a-Service [ea00] im Web zu nutzen. Beispiel dafür sind Cloud9[1] und Koding[2]. Aus diesem Grund ist der Prototyp des Bachelorprojektes namens »ERIC«, in dem diese Ergebnisse dieser Arbeit eingebettet sind, als Web-Anwendung konzipiert.

2.3. Relationale Datenbank

Relationalen Datenbanken liegt das relationale Modell [Cod70] zugrunde, das eine Datenbank als Sammlung gruppierter Datentupel betrachtet, die die Relationen bilden. Zur Beschreibung der Struktur der Daten und deren Beziehungen innerhalb der Datenbank dient das Datenbank-Schema. Es legt die Namen der Relationen und ihre Reihenfolgen von Attributen mit Datentypen fest. Zu dessen Veranschaulichung können ER-Diagramme [Che76] genutzt werden. Die

[1] `https://c9.io/`
[2] `https://koding.com/`

Verwaltung und der Zugriff auf die Daten werden dabei durch ein relationales Datenbankmanagementsystem (kurz RDBMS) gewährleistet. Die im Kapitel 3 vorgestellte Web-IDE und die Algorithmen dieser Arbeit nutzen das RDBMS SAP Hana [FCP+11].

2.4. SQL

SQL (engl. Structured Query Language) [DD97] ist eine standardisierte[3], deklarative Sprache zum Definieren, Abfragen und Bearbeiten von Daten innerhalb eines relationales Datenbanksystems. Sie unterteilt sich in die Data Manipulation Language (DML) zum Einfügen, Verändern und Löschen von Daten, die Data Definition Language (DDL) zum Erzeugen, Verändern, Kürzen und Löschen der Relationen und die Data Control Language (DCL) für die Rechtekontrolle. Der Fokus dieser Bachelorarbeit liegt auf der Data Manipulation Language, insbesondere dem Teilbereich der Datenabfrage. In diesem Kontext werden zwei Begriffe unterschieden: ein SQL-Statement ist ein beliebiger, valider SQL-Ausdruck, wohingegen eine SQL-Query eine Unterkategorie davon darstellt, die zur Abfrage von Daten dient und damit einen Datensatz zurückgibt.

2.5. Testen von Software

Einer der wichtigsten Schritte im Entwicklungsprozess von Software ist das Testen. Dabei kann zwischen funktionalen und nicht-funktionalen Tests unterscheiden werden, die sich aus den oben genannten Anforderungen ableiten. Ziel ist es neben der Verifikation der Anwendung (sind alle Funktionen der Anwendung implementiert und korrekt?) auch die nicht-funktionalen Aspekte zu testen. Auf Letzteres, insbesondere die Teilbereiche Leistung und Skalierbarkeit, konzentriert sich die Arbeit des Bachelorprojektes, da sie einen wichtigen Anspruch bei der Entwicklung von Geschäftsanwendungen darstellen.

[3] ISO/IEC 9075: `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45342`

3. Eine IDE für daten- und performancebewusste Entwicklung

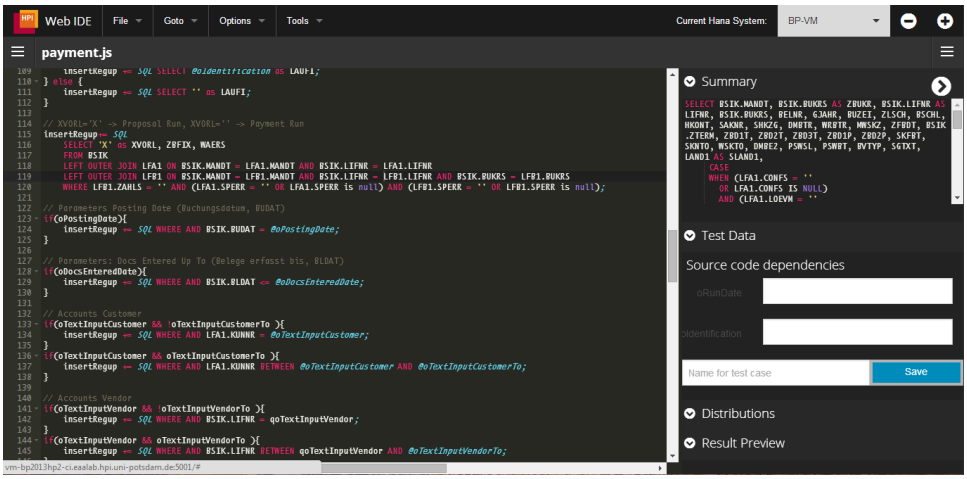

Abbildung 1: Übersicht der Web-IDE

Die im Rahmen des Bachelorprojektes "Modern Computer-Aided Software Engineering" entwickelte Web-IDE (Abbildung 1) vereint eine Reihe von Konzepten zur Entwicklung von Geschäftsanwendungen mit dem Fokus auf der besseren Integration von Informationen aus Datenbanken. Besonders die Schärfung des Bewusstseins für Daten und Datenmengen sowie das vorausschauende Entwickeln in Hinblick auf die Skalierung der Anwendung soll gefördert werden. Grundlage dafür bietet die Einbettung von SQL in die Programmiersprache der Geschäftsanwendung [Hor14] (mehr Details dazu in Kapitel 4). Durch das Parsen des Quelltextes [Hor14] und der darin enthaltenen SQL-Statements [Sch14] werden die Voraussetzungen geschaffen, Analysen und Visualisierungen der Datenabfragen durchzuführen. Unter anderem kann anschließend eine Abschätzung über die Laufzeiten und Ergebnisgrößen von SQL-Statements gegeben werden, sowie eine Vorschau der Ergebnisse und die Verteilung der Daten in den angefragten Spalten. Für die Berechnung der abgeschätzten Laufzeiten und Ergebnisgrößen stehen zwei Verfahren zur Auswahl: auf Basis von Sampling [Exn14] werden mithilfe von Teilmengen der Relationen ungefähre Größen hochgerechnet und durch den Ansatz des Machine Learnings [Mue14] können Ergebnisse vergangener Anfragen als Grundlage der Berechnung ge-

nutzt werden.

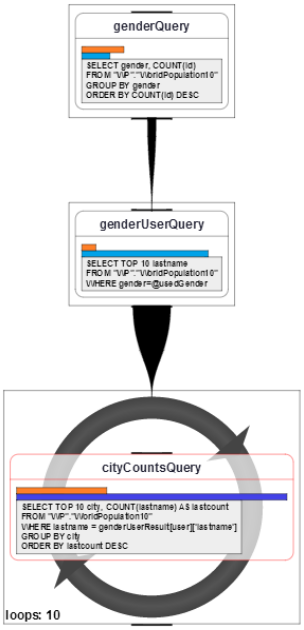

Abbildung 2: Visualisierung des Kontrollflusses

Zusätzlich ist es möglich den Verlauf des Kontrollflusses in Zusammenhang mit den Datenanfragen als Visualisierung zu betrachten [Fra14] (zu sehen in Abbildung 2) um die Ursachen von Performance-Problemen ausfindig zu machen.

Die betrachteten Features für die Analysen von SQL-Statements haben dabei zwei Voraussetzungen: das SQL-Statement muss komplett erfasst und dessen variable Parameter müssen testweise mit Werten belegt sein.

```
1  var firstname = request.body.firstname;
2  var country = request.body.country;
3  var stmt = ''SELECT * FROM WorldPopulation WHERE country = '';
4  if(country == ''D''){
5    stmt += ''Germany'';
6  } else {
7    stmt += ''China'';
8  }
9  if(firstname){
10   stmt += '' AND firstname = '' firstname;
```

```
11  }
```

Code-Beispiel 1: Variablen nehmen Einfluss auf die SQL-Query und -Parameter

Schon einfache Algorithmen, wie im Code-Beispiel 1, lassen das SQL-Statement auf Basis von Programmvariablen variieren (`country`) und belegen die SQL-Parameter in Abhängigkeit von zum Beispiel Sitzungs-Daten, Formularen oder Anfrage-Parametern mit unterschiedlichen Werten (`firstname`). Dadurch ist es für den Entwickler schwer abzuschätzen, welche testweisen Werte repräsentativ sind oder sogar Randfälle darstellen und die Antwortzeit der Anwendung in die Höhe treiben. Deshalb ist es wichtig sinnvolle Testwerte und Kombinationen von Testwerten zu nutzen, die die verschiedenen Szenarien innerhalb der Anwendung abdecken. Mit der Theorie und möglichen Algorithmen zum Vorschlagen dieser Daten beschäftigt sich diese Bachelorarbeit und diskutiert sie in den folgenden Kapiteln.

4. Verknüpfung von SQL mit dem Anwendungskontext

Mit der Einbettung von SQL-Anfragen in andere Programmiersprachen, zum Beispiel JavaScript[4], treffen zwei unterschiedliche Konzepte aufeinander: die imperative Programmiersprache der Anwendung und die deklarative Abfragesprache der Datenbank.

Häufig fließen dabei Informationen aus dem Kontext der Anwendung in das SQL-Statement ein, zum Beispiel als Parameter für Filterbedingungen, oder wirken sich durch den Kontrollfluss auf das Erstellen von SQL-Statements aus. Im Folgenden werden die verschiedenen Varianten von SQL-Statements, deren Erstellung sowie Zusammenhänge mit Quelltext-Variablen untersucht.

[4] ECMAScript Language Specification: `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`

4.1. Dynamische Erstellung von SQL-Statements

SQL-Statements können auf verschiedenste Weisen in den Quelltext einer Anwendung integriert werden, die sich vor Allem durch die Stärke der Bindung von SQL-Statements und dem umliegenden Anwendungskontext unterscheiden. Grundlegend kann man drei Arten differenzieren:

Statische SQL-Statements

```
1    stmt.execute("SELECT *
2    FROM CUSTOMER.BSIK LEFT OUTER JOIN CUSTOMER.LFA1
3    ON BSIK.MANDT = LFA1.MANDT AND BSIK.LIFNR = LFA1.LIFNR");
```

Code-Beispiel 2: Statisches Statement eingebettet im Quelltext

Statische SQL-Statements bleiben über den Kontrollfluss hinweg unverändert und sind unabhängig vom Kontext der Anwendung (Code-Beispiel 2). Sie können für Anfragen genutzt werden, die jederzeit dieselben Informationen aus der Datenbank auslesen (zum Beispiel das Auflisten aller Kunden). Vor allem für Analysen sind diese Datenbankanfragen leicht aus dem Quelltext herauszuparsen und müssen nicht verändert oder mit Testwerten ergänzt werden.

Prepared SQL-Statements

```
1    var stmt = con.prepareStatement("
2    SELECT *
3    FROM CUSTOMER.BSIK LEFT OUTER JOIN CUSTOMER.LFA1
4    ON BSIK.MANDT = LFA1.MANDT AND BSIK.LIFNR = LFA1.LIFNR
5    WHERE LFA1.KUNNR = ?");
6    stmt.setInt(1, 23342341);
7    stmt.execute();
```

Code-Beispiel 3: Prepared Statements eingebettet im Quelltext

Der Anwendungsfall von Prepared Statements ist das mehrfache Ausführen derselben Anfrage mit verschiedenen Parametern. Dabei werden die variablen Stellen mit Fragezeichen versehen und vor der Datenabfrage explizit gesetzt. Die im Code-Beispiel 3 gezeigte Variante setzt dabei in Zeile 6 einen konstan-

ten Wert (23342341) für `LFA1.KUNNR` ein. Häufig kommen diesen Informationen aus der Anfrage vom Nutzer oder Sitzungs-Daten und sind damit nicht als Konstanten im Quelltext erfasst. Das resultierende Statement ist somit abhängig von zu setzenden Parametern, wird jedoch nicht strukturell verändert.

Dynamische SQL-Statements

```
1   var customer = request.body.customer,
2       customerTo = request.body.customerTo,
3       stmt = "SELECT *
4         FROM CUSTOMER.BSIK LEFT OUTER JOIN CUSTOMER.LFA1
5         ON BSIK.MANDT = LFA1.MANDT AND BSIK.LIFNR = LFA1.LIFNR
6         WHERE ";
7   if(customer && !customerTo){
8     stmt += "LFA1.KUNNR = '" + customer + "'";
9   }
10  if(customer && customerTo){
11    stmt += "LFA1.KUNNR BETWEEN '" + customer +
12            "' AND '" + customerTo + "'" ;
13  }
```

Code-Beispiel 4: Dynamische SQL-Statements können verschiedene Ausprägungen annehmen.

Dynamische SQL-Statements werden erst zur Laufzeit in Abhängigkeit vom Kontrollfluss des Programms erstellt. So ist es möglich, Variablen aus der Anwendung sowohl zur Anpassung des SQL-Statements zu nutzen, als auch als Parameter für die Abfrage. Es entsteht eine enge Bindung des Kontrollflusses an das resultierende SQL-Statement, wodurch die Variabilität steigt, aber auch mit zunehmender Komplexität das Lesen und Verstehen der Anwendung erschwert wird. Im Code-Beispiel 4 verändert sich das SQL-Statement durch das Setzen bzw. Nicht-Setzen von Anfrage-Parameter durch den Nutzer. Dabei kann der Nutzer entscheiden, ob er die Informationen für eine konkrete Kundennummer (Zeile 5 bis 7) oder für ein Intervall von Kundennummern (Zeile 8 bis 11) abruft. In beiden Fällen gibt es einen konstanten Part der Anfrage (Zeile 1 bis 4) und es fließen die Anfrage-Parameter in das SQL-Statement ein (`customer`, `customerTo`).

Im folgenden Abschnitt werden diese Beziehungen auf Basis von Variablen aus dem Kontext der Anwendung ausführlicher untersucht.

4.2. Verknüpfung von SQL-Parametern und Quelltext-Variablen

Aus dem Code-Beispiel 4 geht bereits deutlich hervor, dass Kontext-Variablen einen Einfluss auf das SQL-Statement und vorrangig dessen Parameter haben. Allerdings wurden in den vorherigen Beispielen SQL-Teile stets nur als Zeichenketten in der Anwendung genutzt. Durch diese Umwandlung verlieren sie ihre Komfortfunktionen (z.B. Autovervollständigung und Syntax-Überprüfung) und bergen zeitgleich das Risiko von Fehlern, zum Beispiel durch vergessene Leerzeichen, ohne die das finale SQL-Statement nicht valide wäre. Um die Nachteile dieser Konkatenation von Zeichenketten abzuschaffen, kann der Quelltext in die Syntax nach [Hor14] übertragen werden (vgl. Code-Beispiel 5). Durch die Einbettung der Datenbankanfragen nach dem Prinzip von Language Boxes [DT13] werden die Konzepte von SQL und der umgebenen Sprache miteinander kombiniert und es entstehen Synergien, die die Entwicklung der Anwendung vereinfachen und den Quellcode leichter verständlich machen, zum Beispiel durch die explizite Verknüpfung von Quelltext-Variablen mit SQL-Statements und -Parametern.

```
1   var customer = request.body.customer,
2       customerTo = request.body.customerTo,
3       stmt = SQL[CUSTOMER]
4         SELECT *
5         FROM BSIK LEFT OUTER JOIN CUSTOMER.LFA1
6         ON BSIK.MANDT = LFA1.MANDT AND BSIK.LIFNR = LFA1.LIFNR;
7   if(customer && !customerTo){
8     stmt += SQL WHERE LFA1.KUNNR = @customer;
9   }
10  if(customer && customerTo){
11    stmt += SQL WHERE LFA1.KUNNR BETWEEN @customer AND @customerTo;
12  }
```

Code-Beispiel 5: Übertragung des Code-Beispiels 4 in die Syntax nach [Hor14]

Die einzelnen SQL-Blöcke im Code-Beispiel 5 beginnen mit einer `SQL`-Anweisung woraufhin der eigentliche SQL-Text folgt und mit einem Semikolon abgeschlossen wird. In Zeile 3 wird zusätzlich das Schema `CUSTOMER` für das

Statement `stmt` festgelegt. Markant dabei ist die Verwendung von `@`. Durch diese Anweisung wird der zum Zeitpunkt der Definition des SQL-Statements aktuelle Wert der Variable `customer` fest im SQL-Statement gesetzt.

Um eine Wiederverwendung von SQL-Blöcken zu ermöglichen, können sie ähnlich zu Funktionen als SQL-Templates [Hor14] formuliert werden.

```
1  var toleranceDays = request.body.toleranceDays,
2      percentageRate = request.body.percentageRate,
3      getFunctionParameters = SQL[CUSTOMER](xskr1)
4        REGUP.BUDAT, REGUP.WSKTO, REGUP.BLDAT,
5        :xskr1, @toleranceDays, @percentageRate;
6  getFunctionParameters(11).execute();
```

Code-Beispiel 6: SQL-Templates ermöglichen Wiederverwendung

Neben dem `@` kann so zusätzlich `:` zur Referenzierung dienen. Der Unterschied liegt darin, dass der Wert des Parameters (`xskr1`) erst beim Aufruf des SQL-Templates festgesetzt wird, die Werte von `toleranceDays` und `percentageRate` hingegen zum Zeitpunkt der Definition des SQL-Statements. Das SQL-Template `getFunctionParameters` kann anschließend mittels der Anweisung `+=` einem existieren SQL-Statement angefügt werden.

Diese Code-Beispiele machen deutlich, dass Variablen durchaus ein SQL-Statement verändern und in dieses an verschiedenen Stellen wieder einfließen. Im Folgenden wird deshalb eine Unterscheidung anhand der Verbindung zwischen dem eingebetteten SQL und dem umliegende Quelltext vorgenommen.

4.3. Differenzierung von Kontrollfluss- und SQL-Statement-Abhängigkeiten

Neben dem direkten Einfließen von Variablen in ein SQL-Statement (vgl. Kapitel 4.2), können diese auch (nur) als Abhängigkeit im Kontrollfluss auftreten.

```
1    var noDunning = request.body.noDunning,
2        customer = request.body.customer,
3        selection = request.body.selection;
4    var stmt = SQL[CUSTOMER]
5      SELECT @selection
```

```
6       FROM BSIK LEFT OUTER JOIN CUSTOMER.LFA1
7       ON BSIK.MANDT = LFA1.MANDT AND BSIK.LIFNR = LFA1.LIFNR;
8     if(noDunning){
9       stmt += SQL WHERE BSIK.MANST = 0;
10    }
11    if(customer){
12      stmt += SQL WHERE LFA1.KUNNR = @customer;
13    }
```

Code-Beispiel 7: Verschiedene Arten von Abhängigkeiten der Variablen

Im Code-Beispiel 7 sind alle drei Möglichkeiten der Einflussnahme auf SQL-Statements durch Variablen dargestellt. Sie können direkt in das SQL-Statement eingebunden werden (`selection`), das SQL-Statement in der Struktur manipulieren `noDunning`) oder beides zugleich (`customer`).

Schon bei diesen einfachen Algorithmen ist es für Entwickler schwer relevante Testwerte zu finden um Analysen auf dem resultierenden SQL-Statement zu ermöglichen ohne zum Beispiel die Inhalte der Relationen der zugrundeliegenden Datenbank zu kennen. Zudem können die Relationen kryptische Werte in unverständlich benannten Spalten enthalten, beispielsweise bedeutet in einem SAP ERP-System der Eintrag `R` in der Spalte `BSIK.ZLSCH`, dass eine Rechnung mittels Euroüberweisung beglichen wurde.

Auf der anderen Seite stellen die Variationen an Verknüpfungen von Variablen und SQL-Statements eine Herausforderung für das Vorschlagen passender Testdaten dar, denn nicht immer können Informationen aus der Datenbank genutzt werden. Aus diesem Grund werden im Kapitel 5 verschiedene Lösungsstrategien und Ansätze erörtert, die den Entwickler unterstützen repräsentative Testwerte durch passende Vorschläge zu finden um aussagekräftige Analysen auf SQL-Statements zu ermöglichen.

5. Generierung von Vorschlägen für Variablen-Belegungen

Um dem Entwickler repräsentative Testwerte, die die Variationspunkte eines SQL-Statements beeinflussen, vorzuschlagen, gibt es verschiedene Strategien.

Grundlage dafür bietet neben dem Quelltext, ein Datenbanksystem mit den enthaltenen Echt-Daten. In den folgenden Beispielen werden Unternehmensdaten aus einer SAP-Infrastruktur einer Aktiengesellschaft genutzt. Die Integration solcher Datenbanksysteme und die Administration von den genutzten Test-Daten werden im Kapitel 6 ausführlicher behandelt.

Neben der Betrachtung der Charakteristiken von Daten innerhalb der Datenbank und dem Kontext aus dem Quelltext, ist vor allem die Verknüpfung mit Analyse-Ergebnissen, vorrangig den Laufzeit-Messungen, ein Kriterium für die Generierung der Vorschläge. Mittels Auswahl unterschiedlicher, vorgeschlagener Testwerte ist es dem Entwickler möglich konkrete Ausprägungen von SQL-Statements nachzuvollziehen und durch die Auswertung der Messungen gegebenenfalls Optimierungen durchzuführen bis das gewünschte Performance-Verhalten erreicht ist. Doch zuerst müssen die Variablen ermittelt werden, die einen Einfluss auf das SQL-Statement haben.

5.1. Bestimmung der zu betrachtenden Variablen

Kapitel 4.3 zeigte bereits auf, dass Variablen in unterschiedlichen Weisen auf ein SQL-Statement einwirken können. Dies schlägt sich auch auf die Vorschlagsgenerierung nieder. Um die Variablen, die auf ein SQL-Statement Einfluss nehmen, zu bestimmen, dient die Schnittstelle `javascriptParser.getSqlQueryAtPosition()` des Quelltext-Parsers von [Hor14]. Sie liefert ein Objekt zurück mit zwei wichtigen Attributen: `dependencies` und `variables`. Dabei enthält `dependencies` alle Variablen, von den das SQL-Statement im Kontrollfluss abhängig ist, wohingegen `variables` alle Variablen umfasst, die innerhalb des SQL-Statements auftreten. Darüber hinaus können Variablen auch in beiden Listen vorkommen. Die Zuordnung erfolgt in diesem Fall anhand des Attributs `uniqueName`. Die Elemente in `variables` enthalten zusätzlich Kontext-Informationen, die eine Zuordnung zu Spalten innerhalb einer Relation ermöglichen, und so die Grundlage für die Generierung von Testdaten-Vorschläge schaffen. In Abbildung 3 ist eine solche Datenstruktur beispielhaft für das Code-Beispiel 7 dargestellt.

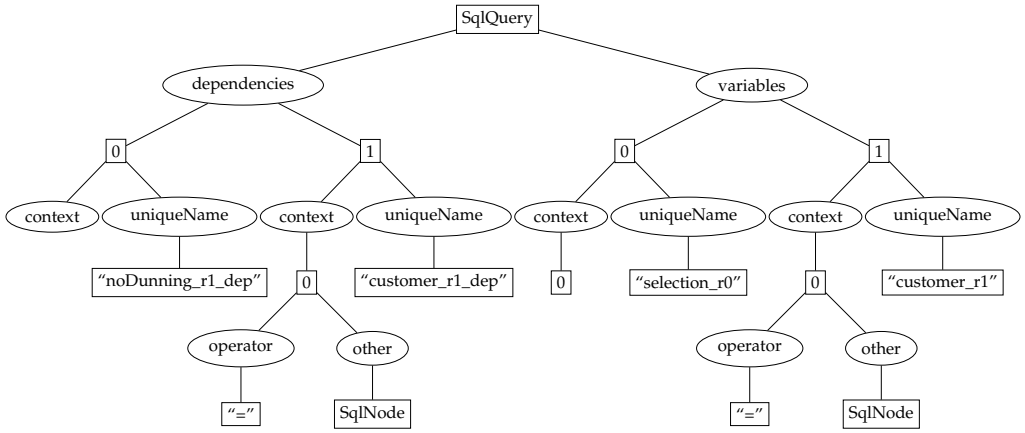

Abbildung 3: Verteilung distinkter Werten einer Auswahl von Spalten aus BSEG

5.2. Vorschläge auf Basis von Daten-Charakteristiken

Sobald Variablen nicht nur die Gestalt eines SQL-Statements variieren, sondern auch darin einfließen, ist der erste Anhaltspunkt für das Vorschlagen relevanter Testwerte die Charakteristik der Datenbankinhalte. Für Selektionsfilter spielt dabei primär die Verteilung der Daten innerhalb der Relationen eine Rolle, sowie die Anzahl ihrer unterschiedlichen Werte-Ausprägungen. Im Gegensatz dazu werden für Projektion, Sortierung und Gruppierung Meta-Informationen zu den Relationen der Datenbank berücksichtigt. Zweiteres wird im Kapitel 5.3 näher betrachtet.

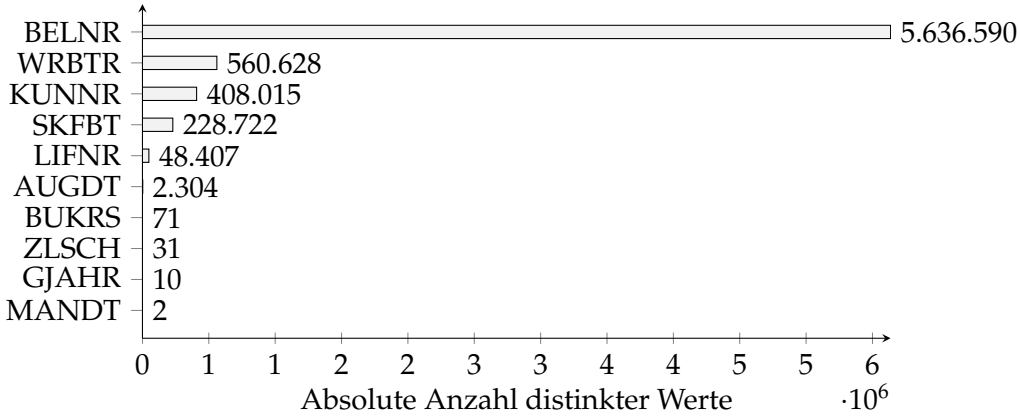

Abbildung 4: Verteilung distinkter Werten einer Auswahl von Spalten aus BSEG

Die Abbildung 4 zeigt eine Auswahl der 326 Spalten der BSEG-Tabelle aus einem SAP-System. Die Tabelle enthält alle einzelnen Belegpositionen zu den Buchungsbelegen des Unternehmens. Eine Erläuterung zu der Bedeutung der einzelnen Spalten befindet sich im Anhang (Tabelle 1).

Sollte die Anzahl der distinkten Werte einstellig sein (beispielsweise bei der Spalte MANDT), können dem Entwickler alle möglichen Ausprägungen in einem Auswahl-Menü zur Verfügung gestellt werden. Damit wird gleichzeitig auch sichergestellt, dass nur Werte eingegeben werden können, die beim testweisen Ausführen des SQL-Statements ein Ergebnis zurückgeben. Auf der anderen Seite können sich Spalten jedoch über eine große Menge von verschiedenen Datenausprägungen erstrecken, wie zum Beispiel bei Belegnummern (BELNR), was eine einfache Auswahl passender Testdaten kompliziert gestaltet. Für diesen Fall werden Äquivalenzklassen anhand der Vorkommen der Werte erzeugt, die sich unterteilen in: die drei häufigsten Werte, die drei seltensten Werte, drei Werte um den Median und der Rest.

Für die häufigsten Werte wird eine aufsteigende Sortierung der Anzahl des Vorkommens eines Wertes vorgenommen und die ersten drei selektiert. Sollten mehrere Werte dieselbe Anzahl an Vorkommen vorweisen, werden sie zusätzlich anhand ihrer Werte aufsteigend sortiert. Im Unterschied dazu wird bei der Klasse der seltensten Werten initiale ein absteigende Sortierung vorgenommen. Für die Werte um den Median muss zuerst die Anzahl der verschiedene Werte ermittelt werden. Anschließend dient die Halbierung des Ergebnisses als Offset für die Bestimmung der drei vorzuschlagenden Werte. Die SQL-Statements dazu befinden sich im Anhang als Code-Beispiel 11.

5.3. Vorschläge anhand von Meta-Informationen

Für Projektion, Sortierung oder Gruppierung sind weniger die Spalte-Inhalte wichtig, als vielmehr die Spalten an sich. Um Vorschläge für diese Art SQL-Statement-Variablen zu erzeugen, werden dazu anhand der Meta-Informationen über Spalten der betrachteten Relation aus der SAP Hana-internen System-Tabelle `SYS.TABLE_COLUMNS` die Spalten-Namen bestimmt (vgl. Code-Beispiel8).

```
1 SELECT COLUMN_NAME
2 FROM SYS.TABLE_COLUMNS
3 WHERE SCHEMA_NAME = '<schema>'
4 AND TABLE_NAME = '<table>'
```

Code-Beispiel 8: Systemtabellen von SAP Hana liefern Meta-Informationen zu Relationen

Die notwendigen Informationen (Schema und Tabelle) werden dazu dem SQL-Statement entnommen.

Nutzung von Meta-Informationen für UI-Elemente

Die Meta-Informationen zu Spalten umfassen neben den Namen auch noch weitere nützliche Informationen, die die Eingabe von Testdaten unterstützen können. Beispielsweise können der Datentyp von Spalten (`DATA_TYPE_NAME`), die Erlaubnis der Eingabe von NULL-Werten (`IS_NULLABLE`) oder auch die Länge (`LENGTH`) für die Erstellung der Eingabefelder genutzt werden kann. So kann die Eingabe, die eine Spalte mit Datumsangaben referenziert, durch einen Kalender vereinfacht werden. Auch die Einschränkung auf Datentypen, z.B. Ganzzahlen, kann fehlerhafte Eingabe durch den Entwickler verhindern. Eine weitere Unterstützung ist die Autovervollständigung von teilweise eingegeben Werten. Mittels des SQL-Operators `LIKE` kann dazu nach Daten gesucht werden, die dem Muster der bisherigen Eingabe entsprechen. Diese Zusatzinformationen erhöhen die Komfortabilität der Eingabe und können fehlerhafte Eingaben reduzieren.

5.4. Adaptive Vorschlagsgenerierung durch Laufzeit-Analysen

Der in Kapitel 5.2 vorgestelle Ansatz zum Vorschlage einzelner Testwerte stößt an seine Grenzen, sobald mehrere Parameter genutzt werden und diese voneinander abhängig sind. Im Code-Beispiel 4 aus dem Kapitel 4 werden beispielsweise offene Rechnungen und deren Einzelposten gesucht. Die Variable `customer` gibt dabei die Kundennummer an, mit der die Rechnungen assoziiert sind. Sucht man nun nach der Kundennummer mit den meisten Rech-

nungen, bedeutet dies nicht zwangsläufig, dass man auch die mit den meisten Einzelposten oder höchsten Gesamtsummen findet. Diese, noch recht einfache, Abhängigkeit kann beliebig erweitert werden. Damit entstehen komplexe SQL- und Programm-Strukturen, die durch das einfache Vorschlagen anhand von Charakteristiken einzelner Spalten nicht zwangsläufig die Randfälle aufzeigen, die der Entwickler sucht. Aus diesem Grund ist die Betrachtung von Messegebnissen aus Laufzeit-Analysen ([Exn14], [Mue14]) eine sinnvolle Erweiterung um die Genauigkeit der Vorschläge zu steigern. Die variablen Stellen von SQL-Statements werden dabei durch die vom Entwickler ausgewählten Werte gefüllt. Im nächsten Schritt werden nun diese atomaren Vorschläge kombiniert und mit dem dazugehörigen Ergebnis aus der Laufzeit-Analyse verknüpft. Dies ermöglicht Vergleichbarkeit verschiedener Konstellationen. Für die Erstellung der initialen Daten können zwei verschiedene Ansätze verfolgt werden.

Mithilfe der Brute-Force-Methode können alle Kombinationen durchprobiert und gemessen werden. Der enorme Aufwand, gerade bei besonders Relationen mit vielen Einträgen und distinkten Werten in den Spalten, stellt jedoch aufgrund der enormen Berechnungszeit ein großes Hindernis dar.

Dem gegenüber steht der adaptive Ansatz, bei dem der Testdaten-Bestand kontinuierlich erweitert wird. Diese Variante speichert die Ergebnisse der Laufzeit-Analysen mit den dazugehören Testdaten-Konstellationen als Testdaten-Sets. Sobald mehrere dieser Sets vorhanden sind, können dem Entwickler drei Vorauswahl-Optionen gegeben: das Testdaten-Set mit der höchsten Laufzeit, mit der geringsten Laufzeit und mittlerer Laufzeit.

Um für diese Methode eine Datengrundlage zu schaffen, werden die in Kapitel 5.2 ermittelten Werte genutzt um initiale Kombinationen zu bilden und ihre Laufzeiten zu berechnen. Durch Eingabe weiterer Werte durch den Entwickler kann anschließend das Datenmodell kontinuierlich erweitert und zunehmend verbessert werden, zu sehen im Code-Beispiel 9.

```
1 hier
2 kommt
3 der
4 algorithmus
```

```
5   rein
```

Code-Beispiel 9: Eingaben von Testwert-Konstellationen erweitern gegebenenfalls das Datenmodell

TODO: Algorithmus beschreiben.

5.5. Betrachtung von Zeitbereichsanfragen

Ein typischer Bestandteil von Datenbankanfragen in Geschäftsanwendungen sind Bereichsfilter mittels `column BETWEEN x AND y`, insbesondere für Zeiträume. Der in Kapitel 5.2 vorgestellte Algorithmus generiert Vorschläge basierend auf binären Operationen (zumeist Vergleiche), wohingegen ein Bereichsfilter ternär ist. Deshalb ist eine besondere Betrachtung für passende Vorschläge sinnvoll.

Sollte eine Bereichsanfrage erkannt werden, die eine Spalte vom Datentyp `DATE` referenziert, so kann der Entwickler eine vorgeschlagene Zeitraumgröße (z.B. eine Woche, ein Monat oder ein Jahr) auswählen oder selbst festlegen. Anschließend wird die referenzierte Spalte mithilfe der angegebenen Zeitraumgröße gescannt, ähnlich dem Prinzip des Sliding Window[5]. Auf dieser Basis werden, ähnlich dem Algorithmus aus Kapitel 5.2, bis zu 3 Vorschläge für die größte, kleinste und mittlere Anzahl an Resultaten erzeugt. Alternativ steht es dem Entwickler offen, Start- und Enddatum eigenständig festzulegen.

5.6. Vorschläge für Variablenwerte ohne Datenbankbezug

TODO

5.7. Integration in die IDE

Die in Kapitel 5.2 und 5.4 erzeugten Vorschläge werden in der zur Web-IDE gehörenden Datenbank gespeichert und in einer Sidebar im Frontend eingebun-

[5] `http://en.wikipedia.org/wiki/Sliding_window_protocol`

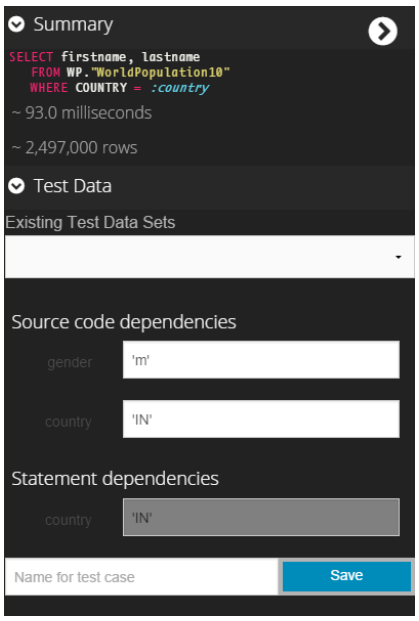

Abbildung 5: Einbindung der Vorschläge in die Sidebar der Web-IDE

den (vgl. Abbildung 5). Durch das Auswählen des Entwicklers einer Quelltext-Zeile mit SQL-Inhalt, werden die Informationen zu dem dazugehörigen SQL-Statement aggregiert und aufbereitet (vgl. Kapitel 3) und schließlich in der Sidebar angezeigt.

Aus der im Kapitel 5.4 ermittelten Menge an Testdaten-Sets werden zum einen die drei mit der höchsten, geringsten und durchschnittlichen Laufzeit angeboten, zum anderen ein Auswahl-Menü mit allen gespeicherten Sets. Die Auswahl einer dieser Optionen füllt die Eingabefelder für die Testdaten automatisch mit den gespeicherten Werten und löst eine Analyse aus. Möchte der Entwickler weitere Testdaten-Sets hinzufügen, so kann er diese benennen und direkt abspeichern. Die Verwaltung dieser gespeicherten Daten wird im folgenden Kapitel behandelt.

6. Verwaltung von Test-Daten und Test-Systemen

An das Backend der Web-IDE ist eine eigene SAP Hana-Instanz angebunden. Darin werden die Testdaten-Sets und Zugangsdaten für Test-Systeme hinterlegt. Im folgenden geht es um das zugrundeliegende Schema, die Integration verschiedener Test-Systeme und das Pflegen der Test-Daten, insbesondere in Hinblick auf deren Gültigkeit.

6.1. Datenschema für Testdaten-Sets

Die Herausforderung beim Speichern der Testdaten-Sets liegt in der Wiederzuordnung zu den Variablen in der vom Entwickler geöffneten Quelltext-Datei. Der in der Web-IDE genutzte Quelltext-Parser [Hor14] nutzt symbolische Ausführung [Kin76] zur Bestimmung von Variablen und deren eventuell bereits festgelegten Werten. Die auf diesem Wege gefundenen Variablen bekommen eine eindeutige Kennzeichnung, die zum Identifizieren genutzt werden kann. Sollten sie im Laufe des Programmflusses in mindestens ein SQL-Statement einfließen, werden sie als Test-Variablen betrachtet. Eine Menge von Test-Variablen bildet zusammen mit dem Dateipfad und einem vom Entwickler angegebenen Namen ein Test-Set. Diese Test-Sets können wiederum in Beziehung mit dem Test-System gebracht werden, auf dem die Performance-Analysen erfolgten, um die Abhängigkeit der System-Auswahl auf die Laufzeiten der Test-Sets zu berücksichtigen. In Abbildung 6 ist das dazugehörige Datenschema dargestellt.

6.2. Caching und Gültigkeit von Test-Daten

Um nicht für jede Vorschlagsanfrage einen Datenbankzugriff durchzuführen, werden die Testdaten und Testdaten-Sets sowohl im Frontend als auch im Backend gecached. Die Caches unterscheiden sich in der Hinsicht, dass es einen Frontend-Cache für jede Sitzung eines Entwicklers gibt, wohingegen der Backend-Cache global agiert. Eine Anfrage für Testdaten-Vorschläge zu einer Spalte einer Relation wird dabei erst versucht durch den Frontend-Cache zu beantworten. Sollten dort keine passenden Daten vorliegen, wird eine Anfrage

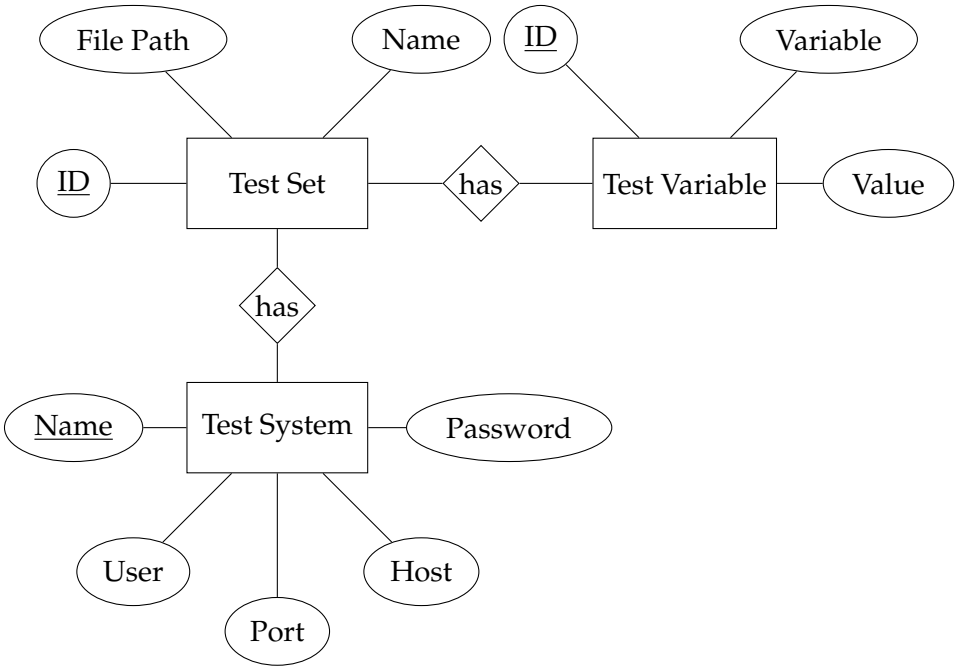

Abbildung 6: ER-Diagram für die Administration von Test-Sets und -Systemen

an das Backend ausgelöst. Dort versucht der Backend-Cache als erste Instanz diese Anfrage zu beantworten. Sollten auch dort keine Daten vorliegen, werden die Vorschläge mithilfe der in Kapitel 5 vorgestellten Algorithmen bestimmt und in den Caches hinterlegt.

Eine Herausforderung stellt das Überprüfen der Gültigkeit der Vorschläge und Testdaten-Sets dar. Diese kann durch zwei Fälle beeinflusst werden: die Daten in der genutzten Datenbank werden verändert (erweitert, aktualisiert oder gelöscht) oder der Quelltext der Anwendung wird in einer Weise abgeändert, die die Variablen aus den SQL-Statements beeinflusst.

Für Ersteres gibt es Ansätze [HSN97], die kontinuierlich verändernde SQL-Statements nachverfolgen und dementsprechend ihren Algorithmus durch manuelles Hoch- bzw. Runterzählen der Anzahl von Einträgen in der betrachteten Relation anpassen. Ein solches Vorgehen ist für die in Kapitel 5 beschrieben Algorithmen nicht umsetzbar, da sie fest gekoppelt sind an die Werte in der Datenbank und nicht nur an ihre Anzahl. Nichtsdestotrotz floss die Idee der

Betrachtung von manipulierenden SQL-Statements mit in die Implementierung ein, indem ein Schwellwert für Datenveränderungen festgelegt wird (derzeit 10), ab dem eine Neubestimmung von Vorschlägen und initial berechneten Testdaten-Sets erfolgt.

Die Invalidierung der Daten durch Veränderung des dazugehörigen Quelltextes wurde im Zuge dieser Arbeit nicht betrachtet, ist aber als Weiterentwicklung geplant. Die Schwierigkeit liegt darin, eine Granularität zu finden, auf der Änderungen zur Invalidierung führen, und gegebenenfalls diese Veränderung nachzuvollziehen. Legt man die Granularitätsstufe auf das gesamte Dokument fest, würde dies dazu führen, dass die Test-Daten zu häufig als ungültig gekennzeichnet werden, obwohl das nicht zwangsläufig zutreffen muss. Wählt man eine feine Granularitätsstufe (zum Beispiel alle Änderungen, die nur Variablen betreffen, die in SQL-Statements einfließen), so ist eine komplexe Nachverfolgung der Änderungen am Dokument über dessen verschiedene Revisionen notwendig. Sollte ein Versionsverwaltungssystem für die Quelltext-Dateien genutzt werden, können dessen Informationen in er Nachverfolgung berücksichtig werden.

6.3. Integration mehrerer Test-Systeme

Typischerweise dient nicht nur ein System als Grundlage für Performance-Analysen einer Geschäftsanwendung, sondern ein Auswahl an Systemen mit unterschiedlichen Ausstattungsmerkmalen. Dies kann mehrere Gründe haben. In Hinblick auf die Daten können so Datenschutz-Bestimmungen eingehalten oder auch Datensätze mit verschiedene Charakteristiken getestet werden. Zum anderen kann so auch die Auswirkung verschiedener System-Konfigurationen auf die Performance der Anwendung geprüft werden, wodurch Kosten gespart werden können, indem man eine passendes System für den Produktiveinsatz auswählt. Realisiert wird dies durch ein Menü in der Web-IDE (vgl. Abbildung 7) bei dem das gewünschte Test-System ausgewählt oder aber auch neue hinzufügt oder existierende entfernt werden können. Die Zugangsdaten für die Systeme werden in der Datenbank der Web-IDE hinterlegt (vgl. Abbildung 6) und das derzeit vom Entwickler ausgewählte System in seinen Sitzungsdaten

hinterlegt.

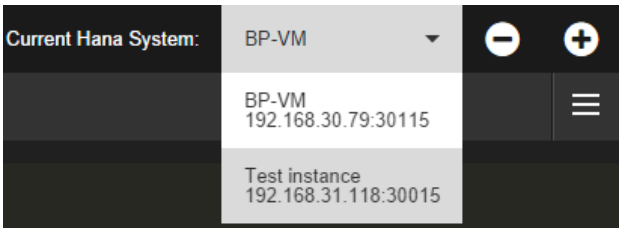

Abbildung 7: Auswahl-Menü für verschiedene Datenbank-Server

7. Fallbeispiel: Der Zahllauf

TODO: Umschreiben zu Thomas' Vorschlag!!!

Im Bereich der Geschäftsanwendungen gibt es eine Reihe hochkomplexer Prozesse, die in der IT abgebildet werden. Einer davon ist der sogenannte Zahllauf, bei dem sich der Anwender eine Liste von offenen Zahlungen erst vorschlagen lässt und anschließend veranlasst. Dabei spielen die Abhängigkeiten von Zahlungen, Lieferanten und Rabattverträgen eine wichtige Rolle, da sie im Idealfall in einer günstigen Konstellation für das Unternehmen resultieren.

Dementsprechend gibt es viele Einflussfaktoren und Variablen in der Umsetzung des dazugehörigen Algorithmus'. Dies macht es vor allem dem Entwickler schwer: große Tabellen mit zum Teil kryptischen Feldern und Werten dienen als Grundlage, Zwischenspeicher und Ausgabe. Da es wichtig ist bereits während der Entwicklung die Performance zu testen, sind sinnvolle Testdaten nötig, die frühzeitig auch die Randfälle der Implementierung aufdecken.

7.1. Der Einfluss der Eingabe auf die resultierende Query

Die Eingabemaske des Zahllaufs besteht aus vielen allgemeinen Parametern und einzelnen Suchfeldern für bestimmte Rechnungen. Neben den Identifikationsmerkmalen für den Lauf, kann ein Zeitbereich festgelegt werden, in dem die Zahlungen liegen. Anschließend werden in Tabellenform Suchkriterien (Bu-

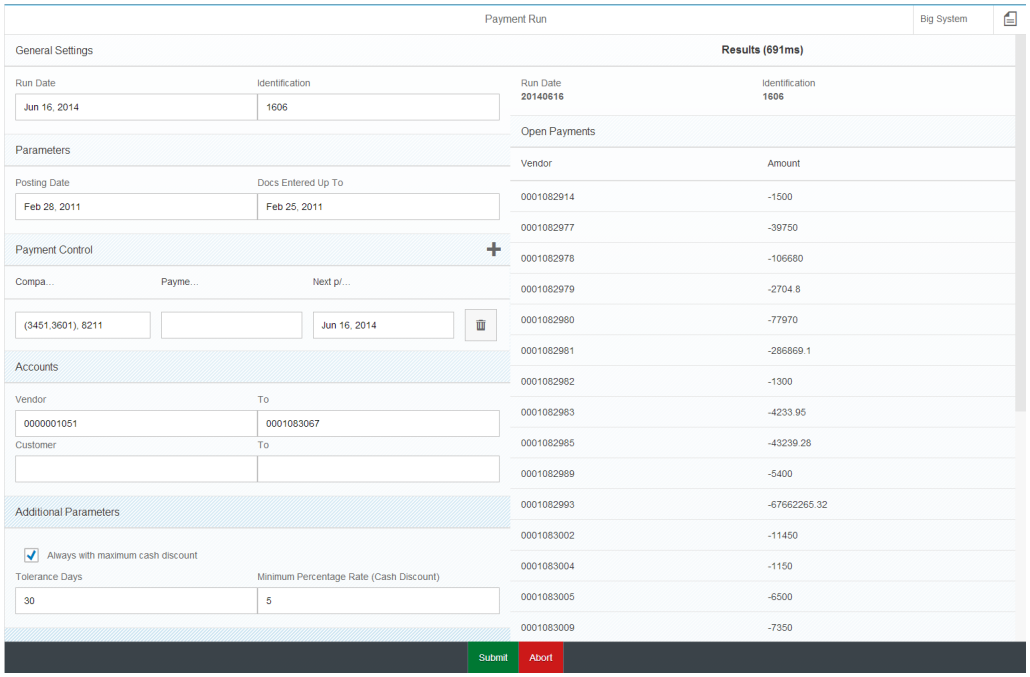

Abbildung 8: Eingabemaske des Zahllaufs

chungskreise, Zahlmethoden, nächstes Ausführungsdatum) eingeben um Zahlungen herauszufiltern. Die einzelnen Zellen einer Reihe bilden eine Konjunktion, wobei die Reihen miteinander disjunkt sind. Komplexität entsteht durch die vielen Möglichkeiten von Eingaben. So können z.B. Buchungskreise kommasepariert, mittels Klammern als Bereich oder beides kombiniert angeben werden. Die Zahlmethoden werden mit Großbuchstaben abgekürzt und aneinander gereiht um mehrere Methoden zuzulassen. Schlussendlich können noch Filter nach Kunden- und Lieferanten-Nummern angegeben werden, wobei hier einzelne oder Bereichsangaben möglich sind, wie man im (bereits im Kapitel 4 gezeigten) Code-Beispiel 10 erkennen kann.

```
1  var stmt = "SELECT BSIK.BUKRS, BSIK.LIFNR, LFA1.CONFS
2  FROM CUSTOMER.BSIK LEFT OUTER JOIN CUSTOMER.LFA1
3  ON BSIK.MANDT = LFA1.MANDT AND BSIK.LIFNR = LFA1.LIFNR
4  WHERE ";
5  if(customer && !customerTo){
6    stmt += "LFA1.KUNNR = '" + customer + "'";
7  }
```

```
8   if(customer && customerTo){
9     stmt += "LFA1.KUNNR BETWEEN '" + customer +
10            "' AND '" + customerTo + "'" ;
11  }
```

Code-Beispiel 10: Mehrere Möglichkeiten des Auswahl anhand von Kundennummern

Diese Variabilität in der Eingabemaske schlägt sich auch auf das resultierende SQL-Statement nieder, dessen Aussehen, Komplexität und Laufzeit durch die optionalen Parameter bestimmt wird, weshalb passende Testdaten für den Entwickler wichtig sind.

7.2. Passende Vorschläge für das Testen des Zahllauf-Programms

Skizzieren welche Vorschläge der Algorithmus für das Programm liefert

- Vergleich von meinem Ansatz mit Brute-Force beim Fallbeispiel - Evaluiering: Zeit-Aufwand vs. bessere Ergebnisse - Betrachtung von Datum-Feldern

8. Verwandte Forschungsarbeiten

Relevante Test-Daten sind in verschiedenen Schritten im Entwicklungsprozess eine Anwendung von Relevanz. Um Geschäftsanwendungen zu testen, gibt es neben dem Mittel die Datenbank-Anbindung zu mocken[6] auch die Möglichkeit sie mit einzubeziehen. In diesem Kontext werden die Eingabewerte mit dem Zustand der Datenbank in Verbindung gebracht. Dabei gibt es verschiedene Vorgehensweise.

8.1. Eingabewerte zum Testen von Datenbank-Anwendungen

Mit dem Erzeugen von relevanter Eingabewerten für Anwendungstests haben sich in den letzten Jahren eine Reihe von Forschungsprojekten beschäftigt. Der

[6] http://en.wikipedia.org/wiki/Mock_object

Fokus der nachfolgend beschriebenen Ansätze liegt dabei, im Kontrast zu dieser Bachelorarbeit, auf der Code- und Branch-Abdeckung von Anwendungstests durch Einbeziehung des Status der Datenbank.

Im Sektor Datenbank-Anwendungs-Tests bietet das AGENDA Framework [CDF+00, Cha04, CDF+04, DFC05, CSF08] eine Palette an Tools für das funktionale Testen. Dafür nutzt es Meta-Informationen aus der Datenbank in Kombination mit Voreinstellungen vom Entwickler um eine Test-Datenbank synthetisch zu erzeugen.

Auf Basis von Microsofts Testing-Framework Pex für die .Net-Plattform [TDH08] entwickelten Pan et al. mehrere Erweiterungen [PWX11b, PWX11a], die die Code-Abdeckung durch Einbeziehung der Datenbank und ihrer Daten erhöhen. Dabei werden mittels Dynamic Symbolic Execution (DSE) [CGP+06, GKS05] die Variablen, die in SQL-Statements einfließen und ihren Änderungen im Programmfluss nachverfolgt um daraus passende Eingaben zu generieren [PWX11b]. Durch die zusätzlichen Anforderungen an Logical Coverage (LC) [AOH03] und Boundary Value Coverage (BVC) [KLPU04] werden die gefunden Variablen zusätzlich noch im Zusammenhang mit Bedingungen innerhalb der Anwendung betrachtet, wodurch gegebenenfalls weitere Eingabe-Werte erzeugt werden.

TODO: Weitere Paper betrachten!

8.2. Live-Analyse von Datenbank-Anwendungen

Eine alternative Möglichkeit die Performance eine Datenbank-Anwendung zu ermitteln, ist das Monitoring. Software-Lösungen wie New Relic[7] betten eigene Komponenten in die laufende Anwendung ein um Metriken aus dem laufenden Betrieb aufzunehmen und zu analysieren. Sie können dann unter anderem die langsamsten SQL-Statements mitsamt ihren Parametern dem Entwickler anzeigen. Allerdings wird eine solche Analyse erst dann ausgeführt, wenn es schon zu spät sein kann im produktiven Einsatz. Die vorgestellte Entwicklungsumgebung soll hingegen Performance-Engpässe schon von vornherein aufdecken,

[7] http://newrelic.com/

sodass die Engpässe verhindert werden. Eine Erweiterung der vorgestellten Algorithmen, Parameter aus den Ausführungsdaten zu extrahieren, würde beide Ideen kombinieren. So können häufig genutzte Werte aus dem Betrieb für Vorschläge von Testdaten zur Weiterentwicklung der Anwendung genutzt.

8.3. Performance-Test-Frameworks

9. Zusammenfassung und Ausblick

TODO: Zusammenfassung und Ausblick

9.1. Vorschläge auf Basis von Query-Plan-Analysen

Um den Einfluss von bestimmten Parametern auf Abfrage-Ausführungsplan zu ermitteln, können die Bordmittel des Datenbanksystems als Ergänzung genutzt werden. Die Analyse des SQL-Statements durch den SQL-Befehl `EXPLAIN PLAN` liefert dafür eine Kostenaufschlüsselung der einzelnen SQL-Operatoren vor der eigentlichen Ausführung. Die Zuordnung von den Kosten zu den Parametern mit den zuvor ermittelten Testwerten würde eine Auskunft über deren Gewichtung geben. Für eine Kostenanalyse inklusive Ausführung kann die SAP Hana interne Prozedur `PLANVIZ_ACTION` genutzt werden. Die Betrachtung einer solchen Analyse ist nicht Teil dieser Arbeit, kann aber in einer späteren Erweiterung die Präzision der Vorschläge von Testwerten erhöhen.

9.2. Einbeziehung von Vorwissen über das genutzte System

Sollte ein bestimmtes System genutzt werden, z.B. SAP ERP-Software, so kann Vorwissen über dessen Charakteristiken in den Vorschlägen zu Testdaten berücksichtigt werden. Ein Beispiel dafür sind Standardwerte, die in jeder Instanz des Systems verwendet werden (z.B. feste Benutzerkennungen), oder die Betrachtung besonderer Zeiträume, wie das Jahresende. Somit können Informationen aus dem Kontext des Systems die Vorschläge erweitern um typische Szenarien abzudecken.

Literatur

[AOH03] Paul Ammann, A. Jefferson Offutt, and Hong Huang.
Coverage criteria for logical expressions.
In *ISSRE*, pages 99–107, 2003.

[CDF+00] David Chays, Saikat Dan, Phyllis G. Frankl, Filippos I. Vokolos, and Elaine J. Weber.
A framework for testing database applications.
In *Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '00, pages 147–157, New York, NY, USA, 2000. ACM.

[CDF+04] David Chays, Yuetang Deng, Phyllis G. Frankl, Saikat Dan, Filippos I. Vokolos, and Elaine J. Weyuker.
An agenda for testing relational database applications: Research articles.
Softw. Test. Verif. Reliab., 14(1):17–44, March 2004.

[CGP+06] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler.
Exe: Automatically generating inputs of death.
In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 322–335, New York, NY, USA, 2006. ACM.

[Cha04] D. Chays.
Test Data Generation for Relational Database Applications.
PhD thesis, Brooklyn, NY, USA, 2004.
AAI3115007.

[Che76] Peter Pin-Shan Chen.
The entity-relationship model—toward a unified view of data.
ACM Trans. Database Syst., 1(1):9–36, March 1976.

[Cod70] E. F. Codd.
A relational model of data for large shared data banks.
Commun. ACM, 13(6):377–387, June 1970.

[CSF08] David Chays, John Shahid, and Phyllis G. Frankl.

Query-based test generation for database applications.
In *Proceedings of the 1st International Workshop on Testing Database Systems*, DBTest '08, pages 6:1–6:6, New York, NY, USA, 2008. ACM.

[DD97] C. J. Date and Hugh Darwen.
A Guide to SQL Standard, 4th Edition.
Addison-Wesley, 1997.

[DFC05] Yuetang Deng, Phyllis Frankl, and David Chays.
Testing database transactions with agenda.
In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 78–87, New York, NY, USA, 2005. ACM.

[DT13] Lukas Diekmann and Laurence Tratt.
Parsing composed grammars with language boxes.
In *Workshop on Scalable Language Specifications*, 2013.

[ea00] Fred Hoch et al.
Software as a service: Strategic backgrounder.
2000.

[Exn14] Moritz Exner.
Hier kommt der titel.
Bachelorarbeit, Hasso-Plattner-Institut, 2014.

[FCP+11] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner.
Sap hana database: data management for modern business applications.
pages 45–51, 2011.

[Fra14] Clemens Frahnow.
Hier kommt der titel.
Bachelorarbeit, Hasso-Plattner-Institut, 2014.

[GKS05] Patrice Godefroid, Nils Klarlund, and Koushik Sen.
Dart: Directed automated random testing.
In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 213–223, New York, NY, USA, 2005. ACM.

[Hor14] Friedrich Horschig.
Hier kommt der titel.

Bachelorarbeit, Hasso-Plattner-Institut, 2014.

[HSN97] Banchong Harangsri, John Shepherd, and Anne H. H. Ngu.
Query size estimation using machine learning.
In *DASFAA*, pages 97–106, 1997.

[Kin76] James C. King.
Symbolic execution and program testing.
Commun. ACM, 19(7):385–394, 1976.

[KLPU04] Nikolai Kosmatov, Bruno Legeard, Fabien Peureux, and Mark Utting.
Boundary coverage criteria for test generation from formal models.
In *ISSRE*, pages 139–150, 2004.

[Mue14] Malte Mues.
Hier kommt der titel.
Bachelorarbeit, Hasso-Plattner-Institut, 2014.

[Nah04] Fiona Fui-Hoon Nah.
A study on tolerable waiting time: how long are web users willing to wait?
Behaviour & IT, 23(3):153–163, 2004.

[Pla13] Hasso Plattner.
A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases.
Springer Publishing Company, Incorporated, 2013.

[PWX11a] Kai Pan, Xintao Wu, and Tao Xie.
Database state generation via dynamic symbolic execution for coverage criteria.
In *Proceedings of the Fourth International Workshop on Testing Database Systems*, DBTest '11, pages 4:1–4:6, New York, NY, USA, 2011. ACM.

[PWX11b] Kai Pan, Xintao Wu, and Tao Xie.
Generating program inputs for database application testing.
In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 73–82, Washington, DC, USA, 2011. IEEE Computer Society.

[Sch14] Jasper Schulz.

Hier kommt der titel.
Bachelorarbeit, Hasso-Plattner-Institut, 2014.

[TDH08] Nikolai Tillmann and Jonathan De Halleux.
Pex: White box test generation for .net.
In *Proceedings of the 2Nd International Conference on Tests and Proofs*, TAP'08, pages 134–153, Berlin, Heidelberg, 2008. Springer-Verlag.

A. BSEG-Erläuterung

BSEG-Spalten	
MANDT	Mandant
GJAHR	Geschäftsjahr
ZLSCH	Zahlweg
BUKRS	Buchungskreis
AUGDT	Datum des Ausgleichs
LIFNR	Kontonummer des Lieferanten bzw. Kreditors
SKFBT	Skontofähiger Betrag in Belegwährung
WRBTR	Betrag in Belegwährung
KUNNR	Debitorennummer
BELNR	Belegnummer eines Buchhaltungsbeleges

Tabelle 1: Erklärung zu der Auswahl an BSEG-Spalten

B. Algorithmus für Testwerte anhand von Daten-Charakteristiken

```
1 -- Häufigste Werte
2 SELECT BELNR, COUNT(<column>) AS OCCURENCES
3 FROM <schema>.<table>
4 GROUP BY <column>
5 ORDER BY OCCURENCES DESC, <column> ASC
6 LIMIT 3;
7
8 -- Seltenste Werte
9 SELECT BELNR, COUNT(<column>) AS OCCURENCES
10 FROM <schema>.<table>
11 GROUP BY <column>
12 ORDER BY OCCURENCES ASC, <column> ASC
13 LIMIT 3;
14
15 -- Bestimmung der Anzahl an distinkten Werten
16 SELECT COUNT(DISTINCT <column>) AS OCCURENCES
17 FROM <schema>.<table>;
```

```
18
19 -- Werte um dem Median
20 SELECT <column>, COUNT(<column>) AS OCCURENCES
21 FROM <schema>.<table>
22 GROUP BY <column>
23 ORDER BY OCCURENCES ASC, <column> ASC
24 LIMIT 3 OFFSET OCCURENCES-1;
```

Code-Beispiel 11: Bestimmung von vorzuschlagenden Testwerten