



## **Bachelorarbeit**

# **Erstellung datenbewusster Tests am Beispiel eines Anwendungsfalls 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 ...

## **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, 28. Juni 2014

---

(Frank Blechschmidt)

## Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
<b>2. Terminologie und Hintergrund</b>	<b>2</b>
2.1. Geschäftsanwendung . . . . .	2
2.2. Integrated Development Environment (engl. IDE) . . . . .	2
2.3. Relationale Datenbank . . . . .	2
2.4. SQL . . . . .	3
2.5. Testen von Anwendungen . . . . .	3
<b>3. Eine IDE für daten- und performancebewusste Entwicklung</b>	<b>3</b>
<b>4. Verknüpfung von SQL mit dem Anwendungskontext</b>	<b>6</b>
4.1. Dynamische Erstellung von SQL-Statements . . . . .	6
4.2. Verknüpfung von SQL-Parametern und Quelltext-Variablen . . . . .	8
<b>5. Generierung von Vorschlägen für Variablen-Belegungen</b>	<b>10</b>
5.1. Vorschläge auf Basis von Daten-Charakteristiken . . . . .	11
5.2. Adaptive Vorschlagsgenerierung durch Laufzeit-Analysen . . . . .	12
5.3. Vorschläge auf Basis von Query-Plan-Analysen . . . . .	13
5.4. Integration in die Entwicklungsumgebung . . . . .	14
<b>6. Verwaltung von Test-Daten und Test-Systemen</b>	<b>14</b>
6.1. Datenschema für Test-Daten-Sets . . . . .	14
6.2. Integration mehrerer Test-Systeme . . . . .	15
<b>7. Fallbeispiel: Der Zahllauf</b>	<b>15</b>
7.1. Der Einfluss der Eingabe auf die resultierende Query . . . . .	16
7.2. Passende Vorschläge für das Testen des Zahllauf-Programms . . . . .	16
<b>8. Verwandte Forschungsarbeiten</b>	<b>17</b>
8.1. Eingabewerte zum Testen von Datenbank-Anwendungen . . . . .	17
8.2. Live-Analyse von Datenbank-Anwendungen . . . . .	18
8.3. Performance-Test-Frameworks . . . . .	18

<b>9. Zusammenfassung und Ausblick</b>	<b>18</b>
<b>Literatur</b>	<b>19</b>
<b>Anhang A. BSEG-Erläuterung</b>	<b>22</b>
<b>Anhang B. Algorithmus für Testwerte anhand von Daten-Charakteristiken</b>	<b>22</b>



## Abbildungsverzeichnis

1.	Übersicht der Web IDE . . . . .	4
2.	Visualisierung des Kontrollflusses . . . . .	5
3.	Verteilung distinkter Werten einer Auswahl von Spalten aus BSEG	11
4.	ER-Diagramm für die Administration von Test-Sets und -Systemen	14
5.	Eingabemaske des Zahllaufs . . . . .	15



## Abkürzungsverzeichnis

API	Application Programming Interface
ATOM	Atom Syndication Format
BI	<i>BlogIntelligence</i>
BI-Impact	<i>BlogIntelligence</i> -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 und 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, das sie in ihrem Denkprozess unterbrochen werden [Nah04]. Ein zweiter wichtiger Aspekt ist die Entwicklung anhand von Echt-Daten. Sie wird als beste Vorgehensweise 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 sollte von Anfang an in den Entwicklungsprozess einfließen.

Um Informationen aus der Datenbank in der Anwendung zu nutzen, erfolgt der Zugriff durch das Einbetten der, vorwiegend in SQL geschrieben, Anfragen. Die eingebetteten Datenbankanfragen haben allerdings zumeist variable Bestandteile und Parameter, die für Performance-Messungen und -Analysen mit passenden Testwerten gefüllt werden müssen. Häufig wird dies jedoch durch riesige Datenmengen in unverständlich benannten Tabellen und Attributen zusätzlich erschwert. Um dies zu vereinfachen werden in dieser Bachelorarbeit verschiedene Ansätze diskutiert, die das Auswählen relevanter Testwerte durch sinnvolle Vorschläge anhand von Eigenschaften der Datenbank-Informationen unterstützen.

Die 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 zu erörtern. Im Kapitel 6 wird ergänzend die administrative Architektur für Test-Daten und -Systeme betrachtet. Abschließend werden die vorgestellten Ansätze im Rahmen der Umsetzung einer realistischen 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 und -prozesse zu realisieren und zum Teil zu automatisieren. 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 (was soll die Geschäftsanwendung tun?) auch die Nicht-funktionalen (z.B. Leistung, Zuverlässigkeit und Skalierbarkeit) erfüllt werden sollen. Auf Letzteres konzentrierte sich die Arbeit des Bachelorprojektes.

### 2.2. Integrated Development Environment (engl. IDE)

Eine IDE, zu deutsch in etwa "Integrierte Entwicklungsumgebung" ist eine Software-Anwendung und umfasst eine Reihe von Werkzeugen, z.B. einen Text-Editor, die zum Erstellen von Software genutzt werden können. In den letzten Jahren setzt zunehmend der Trend ein die IDE nicht mehr als Desktop-Applikation zu installieren, sondern als Software-as-a-Service [?] im Web zu nutzen. Beispiel dafür sind Codenvy<sup>1</sup>, Cloud9<sup>2</sup> und Koding<sup>3</sup>. Aus diesem Grund ist auch der Prototyp namens »ERIC«, in dem diese Bachelorarbeit eingebettet ist, als Web-Anwendung konzipiert.

### 2.3. Relationale Datenbank

Relationalen Datenbanken liegt das relationale Modell [?] zugrunde, das eine Datenbank als Sammlung gruppierter Daten-Tupel betrachtet, die die Relationen bilden. Die Verwaltung und den Zugriff auf die Daten werden dabei durch

---

<sup>1</sup><https://codenvy.com/>

<sup>2</sup><https://c9.io/>

<sup>3</sup><https://koding.com/>

ein relationales Datenbankmanagementsystem (kurz RDBMS) gewährleistet. Die im Kapitel 3 vorgestellte Web-IDE und in dieser Arbeit beschriebenen Algorithmen nutzen das RDBMS SAP Hana [?].

## **2.4. SQL**

SQL (engl. Structured Query Language) [?] ist eine standardisierte<sup>4</sup>, deklarative Sprache zum Definieren, Abfragen und Bearbeiten von Daten innerhalb eines relationalen 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 innerhalb der Datenbank 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 (eventuell leeren) Datensatz zurückgibt.

## **2.5. Testen von Anwendungen**

# **3. Eine IDE für daten- und performancebewusste Entwicklung**

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 wer-

---

<sup>4</sup>ISO/IEC 9075: [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_tc\\_browse.htm?commid=45342](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45342)

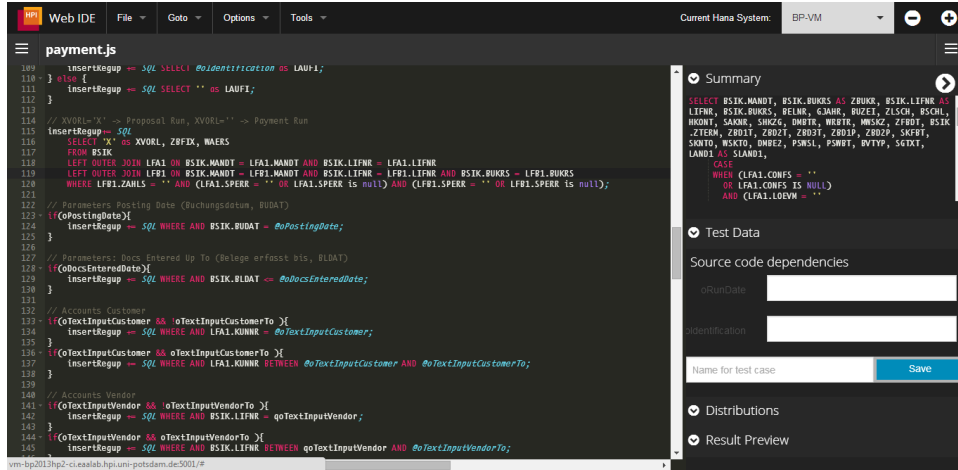


Abbildung 1: Übersicht der Web IDE

den. Grundlage dafür bietet die Einbettung von SQL in die Programmiersprache der Geschäftsanwendung [Hor14] (mehr Details dazu in Kapitel 4). Durch das Parsen des Quelltextes 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 genutzt werden.

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;
```

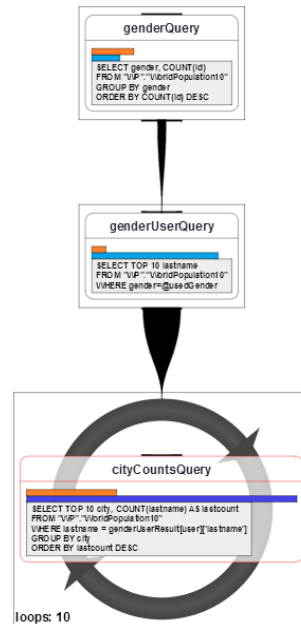


Abbildung 2: Visualisierung des Kontrollflusses

```

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 Session-Daten, Formularen oder Anfrage-Parametern mit unterschiedlichen Werten (`firstname`). Dadurch ist es für den Entwickler schwer abzuschätzen, welche 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 Anwen-



derung 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, 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.

### 4.1. 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 Beziehung 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 von der Anwendung (Code-Beispiel 2). Sie können für

häufige Anfragen, die jederzeit dieselben Informationen aus der Datenbank auslesen (zum Beispiel das Auflisten aller Kunden) genutzt werden. Vor allem für Analysen sind diese Datenbankabfragen leicht aus dem Quelltext heraus zu parsen und müssen nicht verändert oder 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 konstanten Wert (23342341) für LFA1.KUNNR ein. Häufig kommen diesen Informationen aber aus der Anfrage vom Nutzer oder Session-Daten und sind damit nicht 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  var customerTo = request.body.customerTo;
3  var 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 Verzahnung des Kontrollflusses und der resultierenden SQL-Statements, 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 abrufen (Zeile 5 bis 7) oder für ein Intervall von Kundennummern (Zeile 8 bis 11). In beiden Fällen gibt es einen konstanten Part der Anfrage (Zeile 1 bis 4) und fließen die Anfrage-Parameter auch in das SQL-Statement ein (`customer`, `customerTo`). Im folgenden Abschnitt werden diese Beziehungen auf Basis von Variablen im Kontext der Anwendung tiefer 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 die Anwendung eingebunden. Durch diese Umwandlung verlieren sie ihre Komfortfunktionen (zum Beispiel Autovervollständigung, Syntax-Highlighting und -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 verschiedener Zeichenketten abzuschaffen, kann der Quelltext in die Syntax nach [Hor14] übertragen werden (vgl. Code-Beispiel 5). Durch diese Einbettung ähnlich dem Konzept der Language Boxes [DT13] werden die Konzepte von SQL und der umgebenden Sprache miteinander kombiniert und es entstehen Synergien, die die Entwicklung der Anwendung vereinfachen und den Quellcode leichter verständ-

lich machen, zum Beispiel durch die enge Verknüpfung von Quelltext-Variablen mit SQL-Statements und -Parametern.

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

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 1 wird zusätzlich das Schema `CUSTOMER` für das Statement `stmt` festgelegt. Markant dabei ist die Verwendung von `@`. Durch diese Anweisung wird der aktuelle Wert zum der Variable `customer` zum Zeitpunkt der Ausführung fest in das SQL-Statement gesetzt.

Darüber hinaus besteht die Möglichkeit die SQL-Blöcke ähnlich zu Funktionen zu formulieren, um eine Wiederverwendung zu ermöglichen.

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

Code-Beispiel 6: SQL-Blöcke mit Parametern ermöglichen Wiederverwendung.

Neben dem `@` kann nun zusätzlich `:` verwendet werden. Im Code-Beispiels 6 kann `getFunctionParameters` einem existieren SQL-Statement, zum Beispiel mittels `+=` angefügt werden. Dabei wird der Wert von dem Parameter `xskr1` erst beim Ausführen der Datenbankabfrage bestimmt, die Werte von `toleranceDays` und `percentageRate` jedoch schon beim Evaluieren des Statements.

TODO: Skizzieren von AST-Referenz für Zusammenhang von COLUMN und Variable.

Vor allem die Code-Beispiele 4 und 6 zeigen auf, wie Variablen ein SQL-Statement verändern und in dieses an verschiedenen Stellen wieder einfließen. Schon in diesen einfachen Beispielen ist es für Entwickler schwer passende Testwerte zu finden um Analysen auf dem resultierenden SQL-Statement zu ermöglichen ohne die Inhalte der Relationen der Datenbank zu kennen. Zudem können die Relationen kryptische Werte in unverständlich benannten Spalten enthalten, beispielsweise bedeutet der Eintrag `R` in der Spalte `BSIK.ZLSCH` in einem SAP ERP-System, dass eine Rechnung mittels Euroüberweisung beglichen wurde. Im folgenden werden deshalb Algorithmen besprochen, 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, wie anfangs erwähnt, ein Datenbanksystem mit den enthaltenen Echt-Daten. In den folgenden Beispielen werden die realen Unternehmensdaten aus einer SAP-Infrastruktur einer Aktiengesellschaft genutzt. Die Integration solcher Systeme und die Administration von den genutzten Test-Daten werden im Kapitel 6 näher erläutert.

Neben der Betrachtung der Charakteristiken von Daten innerhalb der Datenbank, ist vor allem die Verknüpfung mit Analyse-Ergebnissen, vorrangig den Laufzeit-Messungen, ein Kriterium für die Generierung der Vorschläge. Mittels der 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.

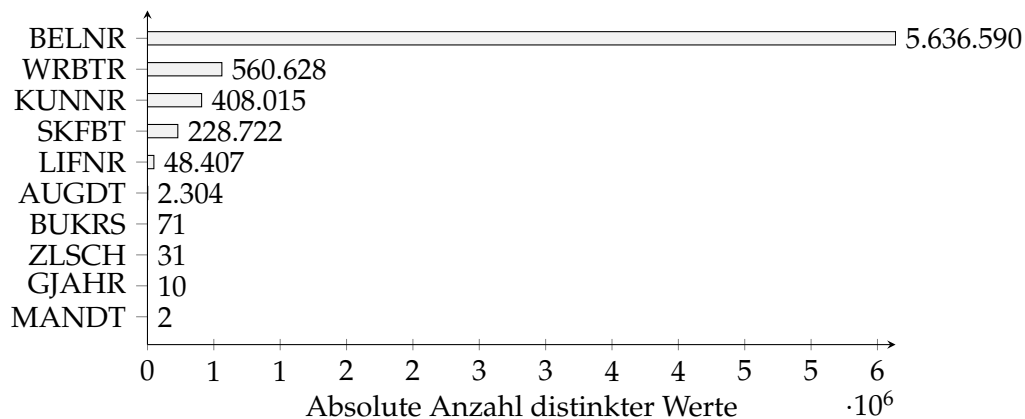


Abbildung 3: Verteilung distinkter Werten einer Auswahl von Spalten aus BSEG

### 5.1. Vorschläge auf Basis von Daten-Charakteristiken

Der erste Anhaltspunkt für das Vorschlagen relevanter Testwerte ist die Charakteristik der Datenbankinhalte. Primär spielen dabei die Verteilung der Daten und die Anzahl unterschiedlicher Ausprägungen eine Rolle.

Die Abbildung 3 zeigt eine Auswahl der 326 Spalten der BSEG-Tabelle aus einem SAP-System. Darin enthalten sind alle einzelnen Belegpositionen zu Buchungsbegleiten 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 in 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 von 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 erschwert. Für diesen Fall werden Äquivalenzklassen anhand der Vorkommen von Werten erzeugt, die sich unterteilen in: die drei häufigste Werte, die drei seltensten Werte, drei Werte um den Median und der Rest.

Für die häufigsten Werte wird eine aufsteigende Sortierung der Anzahl des Vor-

kommens eines Wertes durchgeführt und die ersten drei selektiert. Sollten mehrere Werte dieselbe Anzahl an Vorkommens haben, werden sie zusätzlich anhand ihrer Werte aufsteigend sortiert. Im Unterschied dazu wird bei den seltensten Werten eine 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 Werte. Die SQL-Statements dazu befinden sich im Anhang als Code-Beispiel 8.

Dieser Ansatz ist zum Vorschlagen einzelner Testwerte nützlich, stößt jedoch 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 Rechnungen, 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-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 Messwerten der Analyse-Ergebnisse eine sinnvolle Erweiterung um die Genauigkeit zu steigern.

### 5.2. Adaptive Vorschlagsgenerierung durch Laufzeit-Analysen

Laufzeit-Analysen ([Exn14], [Mue14]) ermöglichen das Verhalten von Datenbankzugriffen in Geschäftsanwendungen nachzuvollziehen. 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 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 großen

Relationen mit vielen 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 dazugehörigen Testdaten-Konstellationen als Testdaten-Sets. Sobald mehrere dieser Sets vorhanden sind, werden dem Entwickler wiederum drei Vorauswahl-Optionen gegeben: das Testdaten-Set mit der höchsten Laufzeit, mit der geringsten Laufzeit und mit einer durchschnittlichen Laufzeit. Die Auswahl einer dieser Optionen füllt die Eingabefelder für die Testdaten automatisch mit den gespeicherten Werten. Um für diese Methode eine Datengrundlage zu schaffen, werden die in Kapitel 5.1 ermittelten Werte genutzt um initiale Kombinationen zu bilden und ihre Laufzeiten zu berechnen. Durch Eingabe weiterer Werte durch den Entwickler kann anschließend das Datenmodell erweitert und zunehmend verbessert werden, zu sehen im Code-Beispiel 7.

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

Code-Beispiel 7: Eingaben von Testwert-Konstellationen erweitern gegebenenfalls das Datenmodell

TODO: Algorithmus beschreiben.

### 5.3. Vorschläge auf Basis von Query-Plan-Analysen

Um den Einfluss von bestimmten Parametern auf Abfrage-Ausführungsplan zu ermitteln, können die Bordmittel des Datenbanksystems 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 gibt somit einen Auskunft über deren Gewichtung. Für eine Kostenanalyse inklusive Ausführung kann die SAP Hana interne Prozedur `PLANVIZ_ACTION` genutzt werden. Die Betrachtung einer solchen



Analyse ist in dieser Arbeit nicht mit erfolgt, kann aber in einer späteren Erweiterung die Präzision von Vorschläge von Testwerten erhöhen.

#### 5.4. Integration in die Entwicklungsumgebung

- Screenshot von der Eingabemaske in der IDE

### 6. Verwaltung von Test-Daten und Test-Systemen

- zur motivation dazu? Verwaltung - Motivation: Mehrere Test-Systeme, gleiche Tasks; mehrere produktive Test-Systeme, unterschiedliche Charakteristiken

#### 6.1. Datenschema für Test-Daten-Sets

Referenz zu Variablen (matching) lines vs 3. variable in funktion

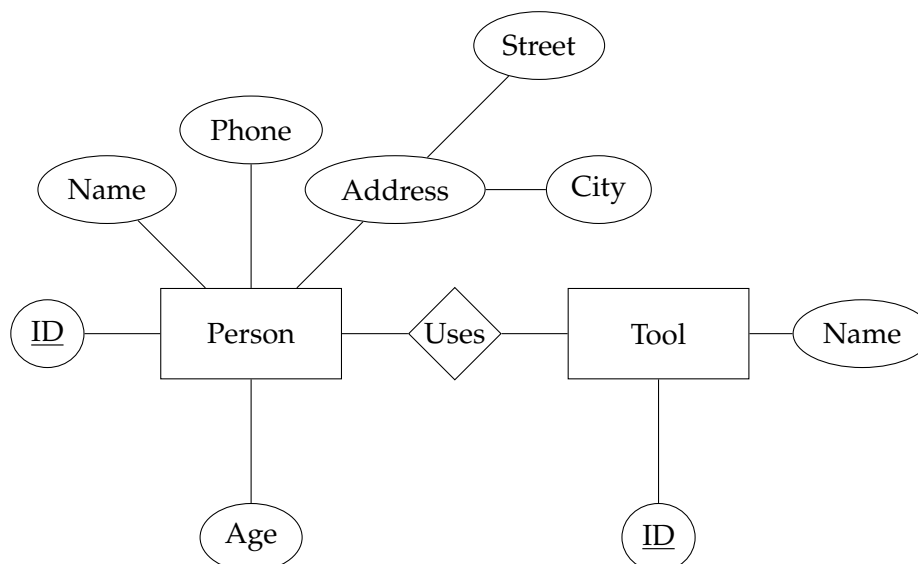
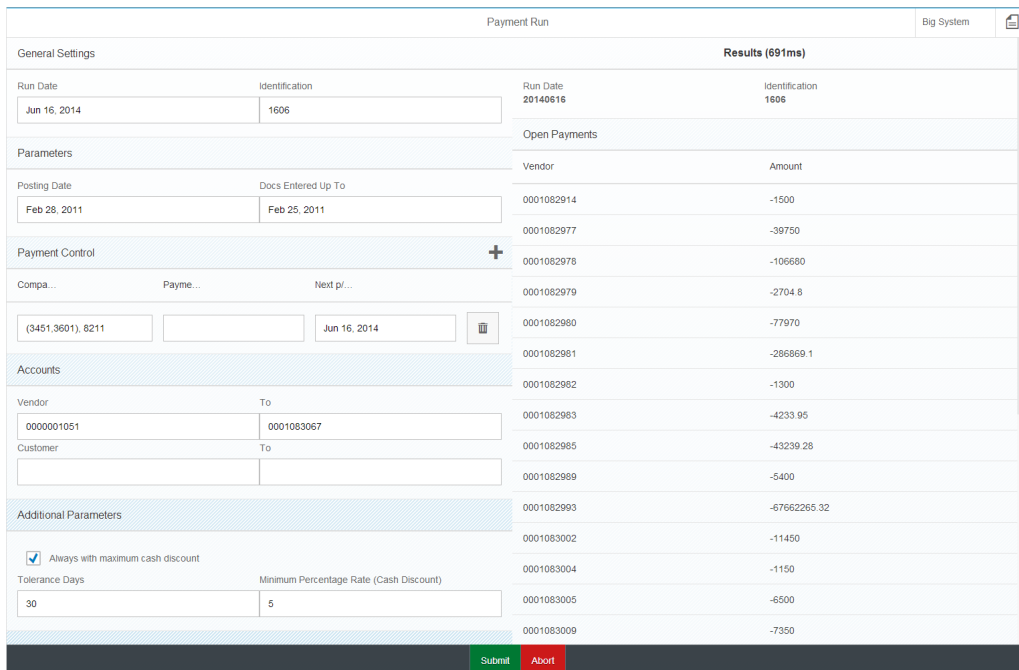


Abbildung 4: ER-Diagramm für die Administration von Test-Sets und -Systemen

## 6.2. Integration mehrerer Test-Systeme

## 7. Fallbeispiel: Der Zahllauf

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 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 den Anwender resultieren.



The screenshot displays the 'Payment Run' (Zahllauf) configuration in SAP. It includes several sections for setting parameters and a table of open payments.

General Settings		Results (691ms)	
Run Date	Identification	Run Date	Identification
Jun 16, 2014	1606	20140616	1606

Parameters		Open Payments	
Posting Date	Docs Entered Up To	Vendor	Amount
Feb 28, 2011	Feb 25, 2011	0001082914	-1500
		0001082977	-39750
		0001082978	-106680
		0001082979	-2704.8
		0001082980	-77970
		0001082981	-286869.1
		0001082982	-1300
		0001082983	-4233.95
		0001082985	-43239.28
		0001082989	-5400
		0001082993	-67662265.32
		0001083002	-11450
		0001083004	-1150
		0001083005	-6500
		0001083009	-7350

Additional Parameters	
Tolerance Days	Minimum Percentage Rate (Cash Discount)
30	5

Abbildung 5: Eingabemaske des Zahllaufs

Dementsprechend gibt es auch 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 aufdecken.

### 7.1. Der Einfluss der Eingabe auf die resultierende Query

Die Eingabemaske 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 (Buchungskreise, Zahlmethoden, nächstes Ausführungsdatum) eingeben um Zahlungen herauszufiltern. Die einzelnen Zellen einer Reihe bilden eine Konjunktion, wobei die Reihen gegenseitig disjunkt sind. Die Komplexität entsteht durch die verschiedenen Möglichkeiten der Eingabe. So können z.B. Buchungskreise kommasepariert, mittels Klammers als Bereich oder kombiniert angegeben 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 Bereichsanfragen möglich sind, wie man im unteren (bereits im Abschnitt 4 gezeigten) Code-Beispiel 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 }
```

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

## 8. Verwandte Forschungsarbeiten

Relevante Test-Daten sind in verschiedenen Schritten im Entwicklungsprozess eine Anwendung von Wichtigkeit. Um Geschäftsanwendungen zu testen, gibt es neben dem Mittel die Datenbank-Anbindung zu mocken<sup>5</sup> auch die Möglichkeit sie mit einzubeziehen. In diesem Kontext werden die Eingabewerte mit dem Zustand der Datenbank in Verbindung gebracht, wobei es dabei verschiedene Vorgehensweise gibt.

### 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 Projekten beschäftigt. Der Fokus der 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<sup>+</sup>00, Cha04, CDF<sup>+</sup>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<sup>+</sup>06, GKS05] die Variablen, die in SQL-Statements einfließen und ihren Änderungen nachverfolgt um daraus passende Programm-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

---

<sup>5</sup>[http://en.wikipedia.org/wiki/Mock\\_object](http://en.wikipedia.org/wiki/Mock_object)

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<sup>6</sup> 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 Parameters 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, sodass die Engpässe verhindert werden. Eine Erweiterung der vorgestellten Algorithmen, Parameter aus den Ausführungsdaten zu extrahieren, würde beide Ideen kombinieren.

## 8.3. Performance-Test-Frameworks

# 9. Zusammenfassung und Ausblick

Zusammenfassung und Ausblick

Betrachtung von SQL-Variablen in Hinblick auf Source-Code-Dependencies (4.2?)

---

<sup>6</sup><http://newrelic.com/>

## Literatur

- [AOH03] Paul Ammann, A. Jefferson Offutt, and Hong Huang.  
Coverage criteria for logical expressions.  
In *ISSRE*, pages 99–107, 2003.
- [CDF<sup>+</sup>00] David Chays, Saikat Dan, Phyllis G. Frankl, Filippas 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<sup>+</sup>04] David Chays, Yuetang Deng, Phyllis G. Frankl, Saikat Dan, Filip-  
pos I. Vokolos, and Elaine J. Weyuker.  
An agenda for testing relational database applications: Research ar-  
ticles.  
*Softw. Test. Verif. Reliab.*, 14(1):17–44, March 2004.
- [CGP<sup>+</sup>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 Commu-  
nications 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.
- [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 Sys-  
tems, 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 background.  
2000.
- [Exn14] Moritz Exner.  
Hier kommt der titel.  
Bachelorarbeit, Hasso-Plattner-Institut, 2014.
- [FCP<sup>+</sup>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 Frahn.  
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.
- [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 8: Bestimmung von vorzuschlagenden Testwerten