Technische Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr.-Ing. Dr. h.c. Theo Härder

Unterstützung von datenbankorientierten Software-Produktlinien durch domänenspezifische Spracherweiterungen für SQL

Diplomarbeit

von

Christian Weber

Betreuer:

univ. dipl. inz. Jernej Kovse

März 2004

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur angefertigt habe.

Kaiserslautern, den 16. März 2004

Christian Weber

Inhaltsverzeichnis

Kapitel 1 Einleitung

In diesem Kapitel wird zunächst eine Einführung zum Thema Software-Produktlinien, generativer Programmierung sowie zu domänenspezifischen Sprachen gegeben. Danach werden die Ziele dieser Arbeit vorgestellt. Zum Abschluss dieses Kapitels wird ein Überblick über die einzelnen Kapitel dieser Arbeit gegeben.

1.1 Einführung

Die Bedeutung von Software in unserer Gesellschaft ist in den letzten Jahrzehnten stetig gewachsen. Dabei hat Software in vielen Bereichen, die täglich von uns genutzt werden, Einzug gehalten. Der Telekommunikationssektor und die Automobilindustrie sind dabei nur zwei Beispiele. Die zunehmende Integration von Software in unseren täglichen Lebensablauf hat die Gesellschaft abhängig von Softwaresystemen gemacht. Aus diesem Grund wird die Verlässlichkeit der eingesetzten Softwaresysteme immer wichtiger, um Katastrophen zu vermeiden. Dadurch wird auch die Verantwortung der Softwareindustrie immer größer.

Die Forschung im Bereich des Software-Engineering beschäftigt sich unter anderem mit der Entwicklung von Methoden und Techniken, um eine möglichst fehlerfreie Software effizient planen und implementieren zu können. Die Effizienz der Implementierung lässt sich durch eine Wiederverwendung von Softwarekomponenten erhöhen. Dafür existieren in der Praxis jedoch noch keine allgemein gültigen Vorgehensweisen. In der Praxis werden Anwendungen von Einzelfall zu Einzelfall komplett neu und individuell auf das Anwendungsproblem zugeschnitten entwickelt. Dabei kommt es nicht zu einer systematischen Wiederverwendung von bereits entwickelter Software. Teilweise werden entwickelte Komponenten wiederverwendet, die jedoch noch an den neuen Anwendungszweck angepasst werden müssen. Da die entwickelten Systeme häufig speziell auf einen Anwendungszweck zugeschnitten sind, ist eine Wiederverwendung größerer Teile einer Anwendung nicht möglich, da der Aufwand für die Anpassung an den neuen Einsatzzweck im Vergleich zu einer Neuimplementierung zu groß wäre.

Mit der industriellen Revolution im Bereich der klassischen produzierenden Industrie begann die Massenfertigung gleichartiger Produkte aus standardisierten Bauteilen. Neben der Automatisierung der Produktion, die zu einer größeren Produktionsmenge und der damit verbundenen Kosteneinsparung führte, konnte durch die Massenfertigung aus standardisierten und vielseitig verwendbaren Teilen eine große Menge von ähnlichen Produkten effizient hergestellt werden. Im Vergleich zur industriellen Produktion ist die junge Ingenieurwissenschaft der Softwareentwicklung auf einem Entwicklungsstand vor der industriellen Revolution. Jedoch können Erfahrungen der industriellen Produktion auf die Softwareentwicklung übertragen werden.

Die wichtigsten Faktoren für Erfolge in der Softwaretechnik sind wie bei der produzierenden

Industrie eine schnelle Markteinführung, hohe Produktqualität sowie niedrige Kosten. Durch eine systematische Codewiederverwendung bei der Softwareentwicklung sollen diese Ziele erreicht werden. Dabei kamen zunächst Prozeduren zum Einsatz, die jeweils an die neuen Anforderungen angepasst wurden. Die Wiederverwendung von relativ kleinen Codestücken, die als Bausteine für ein neues System dienen, führte zur komponentenbasierten Softwareentwicklung. Jedoch wurde die Komplexität der Softwarewiederverwendung unterschätzt. Lange Zeit wurde davon ausgegangen, dass man Komponenten wie Lego-Steine zusammensetzen könnte und ein funktionierendes Softwaresystem erhält. Inzwischen wurde jedoch eingesehen, dass die Wiederverwendung von Softwarekomponenten geplant werden muss.

Eine systematische Codewiederverwendung kann durch die Einführung von produktlinienorientierter Softwareentwicklung unterstützt werden. Dabei werden mehrere Softwareprodukte, die aus dem gleichen Anwendungsbereich stammen, auf Grundlage einer gemeinsamen Produktlinienplattform entwickelt. Mitglieder einer Produktlinie sind miteinander verwandte Softwareprodukte, die für eine ähnliche Aufgabe entwickelt wurden. Im Gegensatz zu Mitgliedern einer Systemfamilie können sich Mitglieder einer Produktlinie in ihrer internen Struktur und technischen Realisierung deutlich unterscheiden. Es ist jedoch sinnvoll, dass die Mitglieder einer Produktlinie auch eine Systemfamilie bilden und so auch eine gleiche Architektur besitzen.

Software-Produktlinien sind ein Softwareentwicklungsansatz, bei dem Softwaresysteme, an die ähnliche Anforderungen gestellt werden, für eine bestimmte Anwendungsdomäne gemeinsam entwickelt werden. Dabei werden die Softwaresysteme nicht als einzelne unabhängige Produkte sondern als zusammengehörige Produkte betrachtet. Mit diesem Ansatz ist zunächst ein größerer Aufwand im Vergleich zur Entwicklung eines einzelnen Softwaresystems verbunden. Zunächst muss eine domänenspezifische Anforderungsanalyse durchgeführt werden, um die Anforderungen, die an die Mitglieder der Produktlinie gestellt werden, zu erfassen. Anschließend muss die zu realisierende Funktionalität auf verschiedene Komponenten verteilt werden, die später zu einen Softwaresystem zusammen gesetzt werden. Diese Komponenten entsprechen den standardisierten Teilen aus der produzierenden Industrie und können bei verschiedenen Mitgliedern der Produktlinie eingesetzt werden.

Mit Hilfe der generativen Programmierung können die Konzepte der Produktlinien im Bereich der Softwareentwicklung umgesetzt werden. Ziel der generativen Programmierung ist es, die Auswahl von Komponenten und die Erzeugung des Softwareproduktes zu automatisieren. Lediglich die Eigenschaften des Softwaresystems müssen durch einen Entwickler beschrieben werden, so dass das fertige Produkt mit Hilfe eines Generators aus den zur Verfügung stehenden Komponenten generiert werden kann. Auf diese Weise können die bereits entwickelten Komponenten effizient wiederverwendet werden.

Die Konfiguration eines Mitglieds einer Produktlinie kann durch domänenspezifische Konfigurationssprachen, in denen die typischen Begriffe und Merkmale der Anwendungsdomäne enthalten sind, unterstützt werden. Bei datenbankorientierten Produktlinien, kann auch eine domänenspezifische Sprache für Anfrage- und Datenmanipulationsbefehle entwickelt werden. Durch diese domänenspezifische Anfrage- und Datenmanipulationssprache kann beispielsweise eine Menge von SQL-Anweisungen die Ausführung einer bestimmten Operation, die in der Anwendungsdomäne benötigt wird, in einer domänenspezifischen Anweisung zusammengefasst werden. Ein weiterer Vorteil derartiger domänenspezifischer Anfrage- und Datenmanipulationssprachen liegt darin, dass dem Entwickler einer Anwendung über die auf die datenbankorientierte Anwendung zugegriffen werden soll, das zu Grunde liegende Datenbankschema nicht bekannt sein muss. Die Anweisungen können sehr leicht in der domänenspezifischen Sprache entwickelt werden, und werden von der datenbankorientierten Anwendung unter Berücksichtigung des verwendeten Datenbankschemas in herkömmliche SQL-Anweisungen übersetzt. Auf diese Weise kann eine komfortable domänenspezifische Schnittstelle angeboten werden.

1.2 Ziel dieser Arbeit

Im Rahmen dieser Diplomarbeit werden die Konzepte zur Unterstützung von datenbankorientierten Software-Produktlinien durch domänenspezifische Sprachen am Beispiel von Versionierungssystemen untersucht. Versionierungssysteme sind sehr gut geeignet, um sie im Rahmen einer Produktlinie zu entwickeln, da die Anwendungsdomäne klar abgegrenzt werden kann und viele Variationsmöglichkeiten bietet. Versionierungssysteme zeichnen sich durch die versionierte Speicherung von Informationen aus, so dass auch auf ältere Versionen von Informationen zugegriffen werden kann. In der Praxis werden derartige Systeme unter anderem zur Speicherung von Quellcode in der Softwareentwicklung eingesetzt.

Ziel dieser Arbeit ist es, die zeitlichen Kosten, die durch die Nutzung einer domänenspezifischen Sprache entstehen, zu bestimmen. Dabei werden unterschiedliche Datenbankschemata verwendet, um zu untersuchen, welcher Zusammenhang zwischen der Komplexität des Datenbankschemas und der Übersetzung einer domänenspezifischen Anweisung in eine Reihe von herkömmlichen SQL-Anweisungen besteht. Der Vorgang der Übersetzung einer domänenspezifischen Anweisung in herkömmliche SQL-Anweisungen wird als *Reduktion* bezeichnet. Die Methoden, in denen diese Übersetzungen durchgeführt werden, heißen *Reduktionsmethoden*. Für die unterschiedlichen Datenbankschemata werden für die Reduktion der domänenspezifischen Anweisungen unterschiedliche Reduktionsmethoden benötigt. Es wird untersucht, in wie weit sich die Komplexität der Reduktionsmethoden und der Zeitbedarf für die Reduktion ändern, wenn Veränderungen am Datenbankschema vorgenommen werden.

SQLInteract ist ein Softwaresystem, das für die Konfiguration von Mitgliedern einer Produktlinie entwickelt wurde. Mit Hilfe dieses Konfigurationswerkzeugs kann ein produktspezifisches Modell erzeugt werden, aus dem dann ein auf das Modell zugeschnittenes Datenbankschema generiert wird. Bei diesem Vorgang wird ein domänenspezifisches Datenbankschema auf ein SQL-Datenbankschema reduziert. Das Werkzeug SQLInteract wird im Rahmen dieser Arbeit um vier verschieden Methoden für die Reduktion eines domänenspezifischen Datenbankschemas erweitert.

Für jedes der vier unterschiedlichen SQL-Datenbankschemata, wird ein passender Datenbanktreiber entwickelt, der die Reduktionsmethoden für die Reduktion von Anweisungen der domänenspezifischen Anfrage- und Datenmanipulationssprache für Versionierungssysteme enthält. Nach einer Analyse der Anwendungsdomäne der Versionierungssysteme wird diese Anfrage- und Datenmanipulationssprache speziell für Versionierungssysteme entwickelt.

Um die zeitlichen Kosten für die Reduktion zu bestimmen, werden Leistungsuntersuchungen durchgeführt. Grundlage für diese Leistungsuntersuchungen sind domänenspezifische Anweisungen, die von einem speziell für diesen Zweck entwickelten Generator erzeugt wurden. Diese generierten domänenspezifischen Anweisungen werden mit den unterschiedlichen Datenbanktreibern auf dem passenden Datenbankschema ausgeführt. Bei der Ausführung der domänenspezifischen Anweisungen werden verschiedene Messwerte gesammelt und anschließend ausgewertet.

1.3 Gliederung der Arbeit

Im folgenden Kapitel 2 werden zunächst einige Verfahren und Technologien vorgestellt. Dabei handelt es sich um Ansätze, um die Wiederverwendung von bereits entwickelter Softwareprodukten zu erhöhen. Nach einer Einführung in das Gebiet der Software-Produktlinien werden die Ideen und die Abläufe bei der *generativen Programmierung* vorgestellt. Anschließend werden die Konzepte der *aspektorientierten Programmierung*, der *intentionalen Programmierung* sowie die der *modellgetriebenen Architekturen* beschrieben.

Als Beispiel für eine Software-Produktlinie werden in Kapitel 3 objektorientierte Versionierungssysteme vorgestellt.Dabei wird zunächst die versionierte Speicherung von Informationen beschrieben. Dabei werden besondere Eigenschaften, die die Mitglieder der Produktlinie für Versionierungssysteme besitzen können, vorgestellt. Anschließend werden die Operationen, die auf den im Versionierungssystem gespeicherten Daten ausgeführt werden können, vorgestellt. Abschließend wird ein Beispiel für ein Versionierungssystem eingeführt, das als Grundlage für eine spätere Leistungsuntersuchung dient.

Kapitel 4 beschäftigt sich zunächst mit den Eigenschaften von domänenspezifischen Sprachen. Anschließend werden die Konzepte einer Meta-Produktlinie für die Spezifikation von Mitgliedern verschiedenster datenbankintensiver Produktlinien vorgestellt. Dabei wird auf den domänenspezifischen Datenbankschemaentwurf und die Abläufe in domänenspezifischen Datenbanktreibern eingegangen.

Die im Kapitel 4 vorgestellten Konzepte zur Generierung von Softwareprodukten einer Produktlinie mit Hilfe eines Generators werden in Kapitel 5 am Beispiel von Versionierungssystemen betrachtet. Dazu wird zunächst das Metamodell für die Konfiguration von Versionierungssystemen vorgestellt. Anschließend werden domänenspezifische Sprachen, sowohl für die Konfiguration als auch für die Nutzung von Versionierungssystemen eingeführt. Danach werden vier unterschiedliche Varianten für die Reduktion eines produktspezifischen Modells auf ein Datenbankschema erläutert und die Auswirkungen auf die Reduktion von domänenspezifischen Datenmanipulationsanweisungen betrachtet.

In Kapitel 6 werden die vier entwickelten domänenspezifischen Datenbanktreiber werden nach verschieden Kriterien bewertet. Diese Bewertung untergliedert sich in zwei Teile. Zunächst wird eine Bewertung der Reduktionsmethoden durch Softwaremetriken vorgenommen. Anschließend wird eine Leistungsbewertung der vier verschiedenen Datenbanktreiber und den dazugehörigen Datenbankschemata durchgeführt. Mit Hilfe der Softwaremetriken wird die Komplexität der Reduktionsmethoden der vier unterschiedlichen Implementierungen verglichen. Die Leistungsuntersuchungen dienen zum Vergleich der unterschiedlichen Datenbankschemata und den dazugehörigen domänenspezifischen Datenbanktreibern.

Abschließend werden in Kapitel 7 die wichtigsten Erkenntnisse dieser Arbeit zusammengefasst und ein Ausblick auf weiterführende Forschungsfelder gegeben.

Kapitel 2

Software-Produktlinien und generative Programmierung

Die Softwarekrise Ende der sechziger Jahre entstand auf Grund der Fehlerhäufigkeit der stetig an Größe und Komplexität zunehmenden Software. Auf der NATO Software Engineering Conference 1968 stellte McIlroy [McI68] einen Zusammenhang zwischen industrieller Produktion und der Entwicklung von Software her. Er erkannte, dass eine erfolgreiche industrielle Produktion auf austauschbaren Modulen, den so genannten Komponenten, mit klar definierten Schnittstellen basieren sollte. Erst durch den Einsatz dieser Komponenten ist eine Massenproduktion möglich und nur Systeme, die aus Komponenten zusammengesetzt werden, sind beherrschbar und wartbar.

Durch die Einführung einer ingenieurmäßigen Entwicklung von Software sollten diese Probleme gelöst werden. Auf dem Gebiet des Software Engineering wurden seitdem große Fortschritte erreicht. So gibt es inzwischen eine Reihe von Prozessmodellen, mit deren Hilfe der Softwareentwicklungsprozess standardisiert werden kann. Das Ziel, Software schnell und günstig erstellen zu können, ist jedoch noch nicht erreicht.

Die Bedeutung von Software in unserer Gesellschaft ist in den letzten Jahrzehnten stetig gewachsen. Dabei hat Software in vielen Bereichen, die täglich von uns genutzt werden, Einzug gehalten. Der Telekommunikationssektor und die Automobilindustrie sind dabei nur zwei Beispiele. Die zunehmende Integration von Software in unseren täglichen Lebensablauf hat die Gesellschaft abhängig von Softwaresystemen gemacht. Dadurch steigt auch die Verantwortung der Softwareindustrie. Viele Softwaresysteme, die früher eigenständig waren, werden heute zu mächtigeren Softwarekomponenten zusammengefasst. Dadurch wird nicht nur der Funktionsumfang der Softwaresysteme erweitert, sondern es wird auch immer wichtiger, dass die einzelnen Komponenten fehlerfrei funktionieren. Das bedeutet, dass Softwaresysteme immer komplexer werden und immer höheren Qualitätsansprüchen genügen müssen. Dazu steht jedoch die Forderung, dass Softwaresysteme immer günstiger werden und schneller zur Verfügung stehen sollen, im Widerspruch [Bos00].

In diesem Kapitel werden zunächst einige Verfahren und Technologien vorgestellt. Dabei handelt es sich um Ansätze, um die Wiederverwendung bereits entwickelter Softwareprodukte zu erhöhen. Nach einer Einführung in das Gebiet der Software-Produktlinien werden die Ideen und die Abläufe bei der *generativen Programmierung* vorgestellt. Anschließend werden die Konzepte der *aspektorientierten Programmierung*, der *intentionalen Programmierung* sowie die der *modellgetriebenen Architekturen* beschrieben.

2.1 Software-Produktlinien und Systemfamilien

Die wichtigsten Faktoren für Erfolge in der Softwaretechnik sind eine schnelle Markteinführung, hohe Produktqualität und niedrige Kosten. Das Erzielen dieser scheinbar unvereinbaren Ziele wird durch eine systematische Codewiederverwendung während der Entwicklung von Software ermöglicht. Der Ansatz der Wiederverwendung wird in der Softwaretechnik schon länger verfolgt. So wurde das Konzept der Prozeduren bereits um 1960 verwendet. In den siebziger Jahren stellte man fest, dass sich Prozeduren leichter wiederverwenden lassen, wenn sie zu Modulen zusammengefasst werden. Prozeduren können entweder so genutzt werden, wie sie vorhanden sind, oder der Code des Moduls muss an die neuen Anforderungen angepasst werden. Die objektorientierte Softwareentwicklung, die in den 80er Jahren populär wurde, ermöglicht eine nachträgliche Anpassung von Code durch Vererbung. Die Wiederverwendung von relativ kleinen Codestücken, die als Bausteine für ein neues System dienen, führte zur komponentenbasierten Softwareentwicklung (Component-Based Software Engineering).

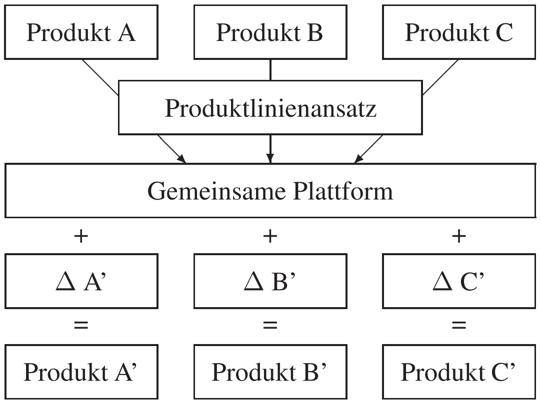
Doch die beschriebenen Ansätze zur Wiederverwendung führten nicht zum erhofften Erfolg, da die Komplexität der Softwarewiederverwendung unterschätzt wurde. Lange Zeit wurde davon ausgegangen, dass man Komponenten wie Lego-Steine zusammensetzen könnte und ein funktionierendes Softwaresystem erhält. Inzwischen wurde jedoch eingesehen, dass die Wiederverwendung von Softwarekomponenten geplant werden muss. Bei der Wiederverwendung von Software unterscheidet man zwei Arten. Zum einen die *ungeplante Wiederverwendung*, bei der Codefragmente aus einem bestehenden Projekt zur Lösung eines aktuellen Problems zusammengesucht werden. Zum anderen die *systematische Wiederverwendung*, bei der die Komponenten gezielt für eine Wiederverwendung entwickelt werden und die notwendige Variabilität besitzen [Bos00].

Die systematische Codewiederverwendung kann auch durch die Einführung von produktlinienorientierter Softwareentwicklung unterstützt werden. Dabei werden mehrere Softwareprodukte, die aus dem gleichen Anwendungsbereich stammen, auf Grundlage einer gemeinsamen Produktlinienplattform entwickelt. Im Gegensatz zu Produktlinien, bei denen die Produkte einen ähnlichen Einsatzzweck besitzen, wird die Ähnlichkeit der Mitglieder einer Systemfamilie durch technische Eigenschaften bestimmt. So weisen die Mitglieder einer Systemfamilie beispielsweise die gleiche Architektur auf, die sich bei den Mitglieder einer Produktlinie unterscheiden kann. Es ist jedoch durchaus ökonomisch sinnvoll, dass die Mitglieder einer Produktlinie auch eine Systemfamilie bilden und somit auch Gemeinsamkeiten in der technischen Realisierung aufweisen. Der Produktlinienansatz ist besonders für Softwarehersteller interessant, die sich auf bestimmte Anwendungsbereiche spezialisieren, in denen sie Erfahrungen akkumulieren und strategisches Wissen über die Entwicklung ihrer Produkte sammeln. Diese Produkte haben eine gemeinsame Struktur und einen gemeinsamen Zweck und können somit als Bestandteile einer Produktfamilie angesehen werden.

Abbildung 2.1 zeigt die Struktur einer Software-Produktlinie. Dabei werden die Gemeinsamkeiten der verschiedenen Produkte A, B und C in der Produktlinienplattform zusammengefasst. Die Produkte A', B' und C', die als Produktlinie realisiert wurden, bestehen aus Komponenten der Produktlinienplattform, die zum Beispiel durch Parametrisierung oder Vererbung angepasst wurden, sowie die produktspezifischen Komponenten Δ A', Δ B' und Δ C'.

Die Architektur einer Produktlinie, die als Systemfamilie realisiert ist, bildet die technische Grundlage der Produktlinie. Durch sie werden, neben der Struktur des Systems und dem Aufbau der einzelnen Komponenten, auch Qualitätsmerkmale wie Zuverlässigkeit und Wartbarkeit

Abbildung 2.1 Struktur einer Software-Produktlinie



festgelegt. Die Entscheidungen, die bei der Wahl einer Produktlinienarchitektur getroffen werden, beeinflussen alle Produkte der Produktlinie. Bei der Auswahl der Produktlinienarchitektur muss der Anwendungsbereich der Produkte berücksichtigt werden, damit die Produktlinie erfolgreich wird. Die Wahl der Produktlinienarchitektur bestimmt den Erfolg und die Lebenszeit der gesamten Produktlinie. Sie muss offen genug sein, um möglichst viele Produkte realisieren zu können, gleichzeitig aber auch stabil genug, um dies über mehrere Produktgenerationen hinaus gewährleisten zu können [Lic01].

Produktlinien werden in der Industrie schon seit Jahren erfolgreich eingesetzt. Auf dem Gebiet der Softwareentwicklung gibt es erst relativ wenige Erfahrungen mit Produktlinien. Jedoch lassen sich einige Erfahrungen aus der Industrie auch auf die Softwareentwicklung übertragen. Zunächst müssen alle Anforderungen des Anwendungsbereichs an die Produkte erfasst werden. Diese Anforderungen können dann für die Entwicklung neuer Produkte übernommen und eventuell erweitert werden. Auf diese Art und Weise kann bei der Anforderungsanalyse für neue Produkte Zeit eingespart werden. Ähnlich verhält es sich bei der Auswahl der Systemarchitektur, durch die wichtige Qualitätsmerkmale festgelegt werden. Durch die Realisierung der Produkte im Rahmen einer Produktlinie müssen kritische Architekturentscheidungen nicht für jedes Produkt neu getroffen werden. Wenn kritische Anforderungen von einem Produkt erfüllt werden, so ist es auch sehr wahrscheinlich, dass diese Anforderungen auch von späteren Produkten erfüllt werden. Gerade die frühen Phasen des Softwareentwicklungsprozesses sind bei Software-Produktlinien besonders wichtig, da durch sie Kosten eingespart und Planungssicherheit gewonnen werden kann.

2.2 Generative Programmierung

Software wird in vielen Bereichen eingesetzt, um Prozesse zu vereinfachen oder auch komplett zu automatisieren. Die Prozesse in der Softwareentwicklung sind im Vergleich dazu wenig automatisiert. Eine komplette Automatisierung des Softwareentwicklungsprozesses ist auch nicht zu erwarten. Mit Hilfe der *Generativen Programmierung* wird versucht, den Rückstand der Softwareentwicklung gegenüber der produzierenden Industrie zu verringern und Abläufe zu automatisieren. Ziel der generativen Programmierung ist die Generierung von Systemen in einer Produktlinie mit möglichst geringem Aufwand. Durch die Verwendung von Generatoren wird die Softwareentwicklung beschleunigt, die Kosten dafür gesenkt. Außerdem wird eine Qualitätssteigerung durch eine geringere Fehleranfälligkeit erreicht.

Abbildung 2.2 Gegenüberstellung der Meilensteine der industriellen und der Softwareentwicklung

In Abbildung 2.2 werden die Meilensteine der industriellen Entwicklung und der Softwareentwicklung gegenübergestellt. Auf die Bedeutung der aufgeführten Ereignisse in der industriellen Entwicklung muss sicherlich nicht mehr eingegangen werden. Im Bereich der Softwareentwicklung gibt es Technologien, die im Hinblick auf die Produktivitätssteigerungen eine ähnliche Bedeutung haben. Die Abbildung verdeutlicht, welche Erwartungen an den Erfolg der generativen Programmierung gestellt werden.

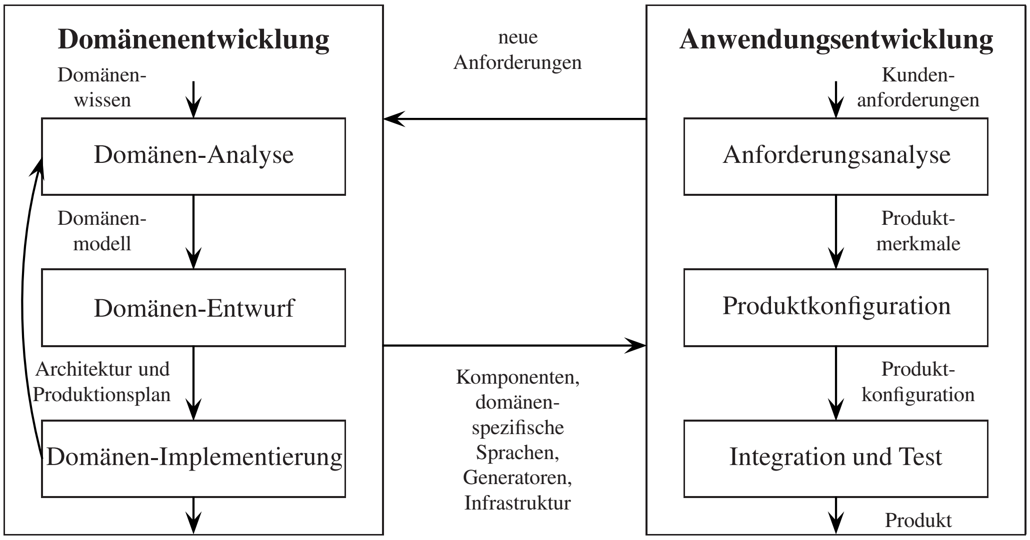
Die generative Programmierung ist ein neues Softwareentwicklungsparadigma, das jedoch nicht mit aktuellen Sichtweisen, wie zum Beispiel der objektorientierten Programmierung, in Konkurrenz steht, sondern diese vielmehr ergänzt. Mit Hilfe der generativen Programmierung werden nicht mehr Einzelsysteme produziert, die bei wechselnden Anforderungen per Copy & Paste kopiert und von Hand angepasst werden, sondern es steht von Beginn an die Entwicklung einer Systemfamilie im Vordergrund. Generative Programmierung unterstützt Softwaresystemfamilien, so dass ausgehend von einer Anforderungsspezifikation mittels Konfigurationswissen aus elementaren, wiederverwendbaren Implementierungskomponenten, ein hoch angepasstes und optimiertes Zwischen- oder Endsystem nach Bedarf automatisch erzeugt werden kann [Cza01].

Bei der generativen Programmierung besteht der Softwareentwicklungsprozess aus zwei Teilen. Zum einen aus den Prozessen der Domänenentwicklung und zum anderen aus den Prozessen der Anwendungsentwicklung. Dabei handelt es sich bei der Domänenentwicklung um eine Entwicklung für die Wiederverwendung, die dann in der Anwendungsentwicklung zum Einsatz kommt.

Um die Mitglieder einer Systemfamilie möglichst automatisiert erzeugen zu können, müssen zunächst die Anforderungen an alle Mitglieder der Systemfamilie im Rahmen der Domänenentwicklung ermittelt und modelliert werden. Darauf baut die Anwendungsentwicklung auf, in der die einzelnen Mitglieder der Systemfamilie nach Bedarf erzeugt und gegebenenfalls angepasst werden.

Abbildung 2.3 zeigt die Domänenentwicklung und die Anwendungsentwicklung als zwei parallele, miteinander verzahnte Prozesse. Dabei baut die Anwendungsentwicklung im Normalfall

Abbildung 2.3 Zusammenhang zwischen der Entwicklung von Domäne und Anwendung



auf der Domänenentwicklung auf. Bevor mit der Domänenentwicklung begonnen wird, sollte der Anwendungsentwicklungsprozess mindestens einmal durchlaufen worden sein, um die Anforderungen an die Softwaresystemfamilie besser verstehen zu können. Werden im Laufe der Zeit neue Anforderungen an die Systemfamilie bekannt, so muss der Prozess der Domänenentwicklung erneut durchlaufen werden, damit die Anwendungsentwicklung komplett auf der Domänenentwicklung aufbaut und sich nicht davon ablöst. Der Prozess der Domänenentwicklung und der der Anwendungsentwicklung müssen daher kontinuierlich aufeinander abgestimmt werden.

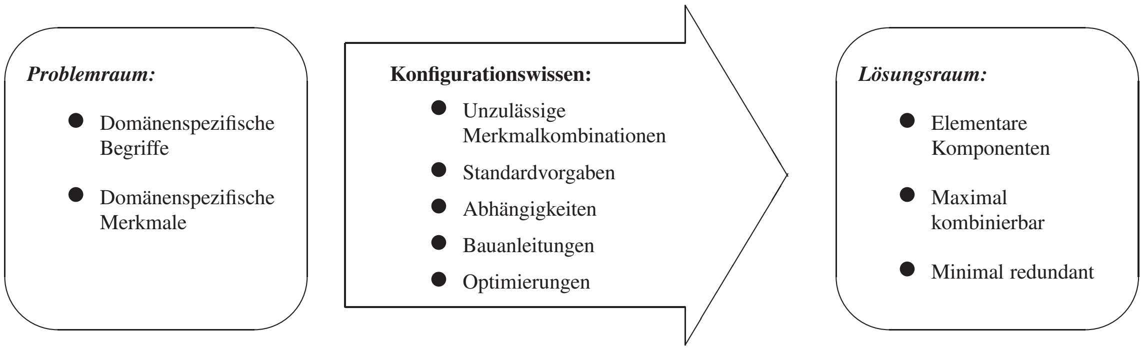
2.2.1 Domänenentwicklung

Domänenentwicklung (Domain-Engineering) ist das Sammeln, Organisieren und Speichern bisheriger Erfahrungen im Erstellen von Systemen oder Systemteilen innerhalb einer bestimmten Domäne, in Form von wiederverwendbaren *Assets* (d.h. wiederverwendbare Anforderungen, Architekturen, Komponenten und sonstige Formen von wiederverwendbarem Entwicklungswissen), sowie das zur Verfügung stellen von ausreichenden Mitteln zur Wiederverwendung dieser *Assets* zum Erstellen neuer Systeme [CE00].

Wie in Abbildung 2.3 gezeigt, umfasst die Domänenentwicklung drei Phasen, die aufeinander aufbauen und bei Bedarf iterativ durchlaufen werden. In der Domänenanalyse werden Gemeinsamkeiten und Variabilitäten innerhalb einer Systemfamilie analysiert. Dabei werden die wiederverwendbaren Anforderungen, Architekturen, Komponenten und das sonstige Entwicklungswissen festgehalten. Als Ergebnis entsteht ein Domänen-Modell für die entsprechende Anwendungsdomäne. In der Phase des Domänenentwurfs werden die Architektur und der Produktionsplan der Softwaresystemfamilie bestimmt. Das Ergebnis der Domänenimplementierung besteht aus der Implementierung der Bestandteile des generativen Domänenmodells (Abbildung 2.4), aus dem die Mitglieder der Softwaresystemfamilien automatisch generiert werden sollen.

Das generative Domänenmodell besteht aus dem Problemraum, dem Konfigurationswissen sowie dem Lösungsraum. Eine Spezifikation aus dem Problemraum soll mit Hilfe des Konfigurationswissens in ein Produkt transformiert werden, das aus Komponenten des Lösungsraums besteht. Der Problemraum besteht aus domänenspezifischen Merkmalen und Begriffen. Durch sie können die Anforderungen an die Produkte spezifiziert werden. Der Lösungsraum besteht aus einer Menge von Komponenten, aus denen die Mitglieder der Systemfamilie generiert werden. Beim Entwurf dieser Komponenten sollte darauf geachtet werden, dass sich möglichst viele Komponenten miteinander kombinieren lassen und so wenig Code wie möglich redundant vorhanden ist. Das Konfigurationswissen beinhaltet Informationen über zulässige und unzulässige Merkmalskombinationen, Abhängigkeiten zwischen einzelnen Komponenten sowie Bauanleitungen und mögliche Optimierungen.

Abbildung 2.4 Generatives Domänenmodell

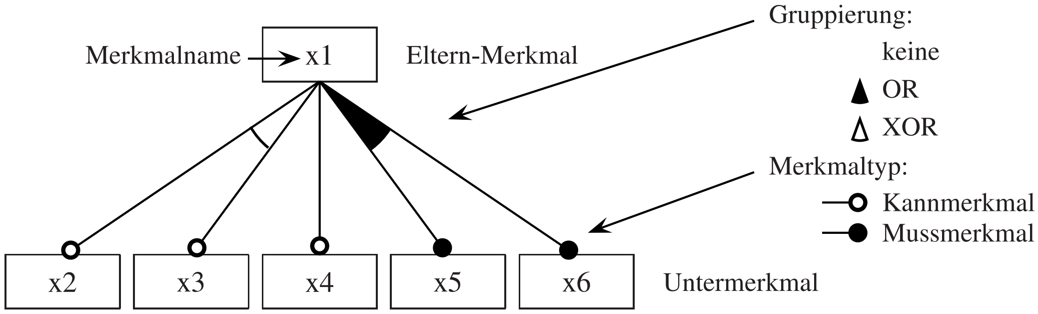


Domänenanalyse

Ein wichtiger Bestandteil der Domänenentwicklung ist die Domänenanalyse. Dabei wird ein Modell der entsprechenden Anwendungsdomäne erstellt. Dazu wird versucht, die gemeinsamen und variablen Merkmale der Softwaresystemfamilie, sowie die Wechselwirkungen zwischen den variablen Merkmalen zu analysieren und zu beschreiben. Diese werden in einem Merkmalmodell dokumentiert, welches aus Merkmaldiagrammen und zusätzlichen Informationen besteht. Dabei wird festgelegt, welche Merkmalkombinationen zulässig sind und welche nicht. Um das Merkmalmodell zu erstellen, ist es wichtig, so viele Informationsquellen wie möglich auszuwerten. Als Informationsquellen kommen Entwickler, Anwender, Experten, Auftraggeber der entsprechenden Anwendungsdomäne als auch Wissensquellen wie Fachartikel, Bücher und Internetseiten in Frage. Dabei sollten auch Marktanalysen und eine Vorschau auf zukünftige Technologien betrachtet werden. Bei der Auswertung der Informationsquellen muss festgelegt werden was zur Domäne gehört und was nicht.

Zu dem Domänenmodell gehört eine genaue Beschreibung der Domäne, ein Lexikon des in der Domäne vorkommenden Vokabulars, konzeptionelle Modelle, in denen typische Abläufe der Domäne beschrieben werden, sowie Merkmaldiagramme. Zur kompakten und übersichtlichen Darstellung gemeinsamer und variabler Merkmale einer Domäne sind Merkmaldiagramme besonders gut geeignet. Diese werden in einer Baumstruktur dargestellt. Es ist möglich, dass ein Merkmal mehrfach referenziert wird. Die innere Struktur eines Merkmaldiagramms ist daher ein gerichteter, zyklenfreier Graph mit einem Wurzelelement. In Abbildung 2.5 sind die unterschiedlichen Elemente eines Merkmaldiagramms dargestellt.

Abbildung 2.5 Elemente eines Merkmaldiagramms



Domänenentwurf

Im Domänenentwurf werden die Bestandteile des generativen Domänenmodells bestimmt. Auf Grundlage des Domänenmodells wird eine flexible Architektur für die Systemfamilie festgelegt. Um die Mitglieder der Systemfamilie beschreiben zu können, wird oft eine Spezifikationssprache (Domain-Specific Language, DSL) entworfen. Nach dem Entwurf dieser domänenspezifischen Spezifikationssprache kann dann ein Generator geplant werden, der anhand eines DSL-Ausdrucks elementare Bausteine gemäß der Systemfamilienarchitektur entsprechend konfiguriert. Domänenspezifische Sprachen sind ein wesentlicher Bestandteil dieser Arbeit. Dabei werden sowohl domänenspezifische Sprachen für die Konfiguration von Mitgliedern einer Produktlinie, als auch für domänenspezifische Anweisungen betrachtet. Diese domänenspezifischen Anweisungen werden von den Mitgliedern der Produktlinie verwendet.

Es werden die Randbedingungen der Implementierung und Wiederverwendungsinfrastruktur identifiziert. Die wesentlichen Ergebnisse des Domänenentwurfs sind die Architektur der Softwaresystemfamilie und der Produktionsplan.

Domänenimplementierung

In der letzten Phase der Domänenentwicklung, der Domänenimplementierung, werden aus den Informationen der vorherigen Phasen die Bestandteile des generativen Domänenmodells implementiert. Dabei handelt es sich um die domänenspezifische Sprache, den Generator, sowie die wiederverwendbaren Komponenten, die vom Generator beim Erstellen einzelner Systeme benötigt werden. In diesem Schritt können auch unterstützende Komponenten für die Spezifikation erstellt werden. Zu diesen Komponenten gehören beispielsweise grafische Benutzeroberflächen, mit denen sich die Anforderungen an ein zu generierendes Produkt der Systemfamilie einfach zusammenstellen lassen.

2.2.2 Anwendungsentwicklung

Anwendungsentwicklung (Application Engineering) ist der Prozess, in dem durch Verwendung der Ergebnisse der Domänenentwicklung Produkte der Systemfamilie aus einer konkreten Spezifikation, die in der domänenspezifischen Sprache vorliegt, mit Hilfe des Generators erzeugt werden. Zur Erstellung dieser Spezifikation können spezielle Werkzeuge bereitgestellt werden. Diese Werkzeuge können bereits überwachen, dass nur Systeme mit gültigen Merkmalskombinationen spezifiziert werden.

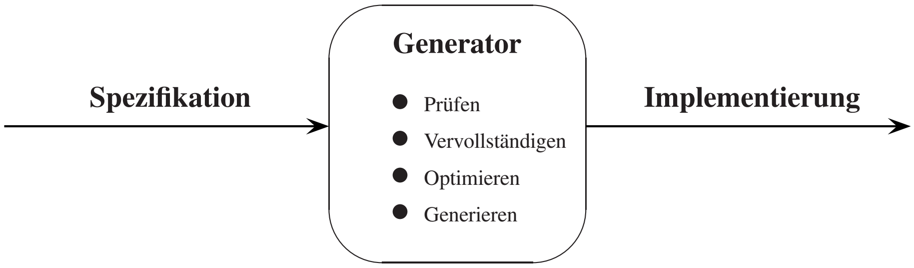
Während der Prozess der Domänenentwicklung im Normalfall nur einmal durchlaufen wird, soll der Prozess der Anwendungsentwicklung mehrfach durchlaufen werden. Dies ist erforderlich, da nicht nur ein Produkt der Systemfamilie generiert werden soll, sondern eine Menge von unterschiedlichen Produkten. Falls jedoch Anforderungen bei der Domänenentwicklung noch nicht berücksichtigt wurden und diese nicht in der domänenspezifischen Sprache spezifiziert werden können, so muss der Prozess der Domänenentwicklung erneut angestoßen werden und die neue Anforderung hinzugefügt werden. Erst dann kann ein Produkt, dass diese neue Anforderung erfüllen soll, generiert werden. Abhängig vom Automatisierungsgrad sind noch manuelle Nacharbeiten notwendig, um das vom Generator erzeugte Produkt einsetzen zu können.

2.3 Generatoren

Der Begriff *Generator* ist sehr allgemein und umfasst viele Technologien, wie zum Beispiel Compiler, Präprozessoren, Metafunktionen, die Klassen und Prozeduren erzeugen, sowie Codegeneratoren und transformierende Komponenten. Compiler nehmen zum Beispiel Programme, die in einer höheren Programmiersprache (z.B. C++ oder Java) geschrieben sind, und generieren daraus Maschinen- oder Bytecode.

Im Allgemeinen werden durch einen Generator die in Abbildung 2.6 aufgeführten Aufgaben durchgeführt. Zunächst muss die Spezifikation, die ein Generator als Eingabe erhält, validiert und gegebenenfalls Warnungen oder auch Fehler gemeldet werden. Dazu muss die Eingabespezifikation, die in den meisten Fällen in Form einer textuellen Beschreibung vorliegt, zunächst in eine interne Repräsentation (z. B. abstrakte Syntaxbäume) überführt werden. In der zweiten Phase wird die Spezifikation vervollständigt. Eigenschaften, die für den Kunden nicht relevant sind, müssen in der Spezifikation nicht berücksichtigt werden, sondern werden vom Generator auf Standardeinstellungen gesetzt. In der dritten Phase werden Optimierungen durchgeführt, um eine optimale Kombination der zur Verfügung stehenden Komponenten zu bestimmen. Abschließend kann dann die Ausgabe generiert werden. Im einfachsten Fall können diese vier Phasen aufeinander folgend und unabhängig voneinander ausgeführt werden. Diese Trennung in die einzelnen Phasen ist jedoch nicht immer möglich, da es erforderlich sein kann, dass einige der vier Phasen parallel oder auch interaktiv durchgeführt werden müssen.

Abbildung 2.6 Aufgaben eines Generators



Generatoren, die komplexe Spezifikationen in komplexe Implementierungen umsetzen, sind häufig sehr komplex, so dass sie modularisiert werden müssen. So werden große Generatoren durch eine Menge von kleineren Generatoren realisiert, die miteinander interagieren. Es gibt viele Möglichkeiten, die Aufgaben eines großen Generators auf mehrere kleinere zu verteilen.

So können zum Beispiel kleinere Generatoren von Größeren aufgerufen werden, um spezifische Komponenten zu generieren, Optimierungen durchzuführen oder auch temporäre Repräsentationen für andere Generatoren zu erstellen.

Es gibt im Wesentlichen drei Möglichkeiten, um einen Generator zu erstellen:

- *Komplette Eigenentwicklung des Generators*
  Eine komplette Eigenentwicklung eines Generators ist die aufwändigste Variante, um einen Generator zu erzeugen, da sowohl die interne Repräsentation der Eingabe, die Validierung, die Optimierung, als auch die eigentliche Generierung der Ausgabe komplett von Hand implementiert werden muss. Nur für das Lexikalisieren und Parsen können existierende Werkzeuge, wie zum Beispiel *yacc*[1] oder *lex* [2], genutzt werden. Neben dem sehr großen Entwicklungsaufwand, den diese Möglichkeit mit sich bringt, gibt es bei dieser Variante zusätzlich den Nachteil, dass eine Integration von unterschiedlichen Notationen und Entwicklungswerkzeugen erschwert wird, da sie unterschiedliche interne Repräsentationen des Quellcode verwenden. Als Ergebnis entsteht eine Menge von nicht kompatiblen domänenspezifischen Sprachen.

- *Generatorentwicklung durch Metaprogrammierung in Programmiersprachen*
  Einige Programmiersprachen bieten eingebaute Möglichkeiten zur Manipulation des Quellcodes zur Übersetzungszeit oder sogar zur Laufzeit. Ein Beispiel dafür ist die Templateprogrammierung in C++. Dabei wird zur Übersetzungszeit der Code durch das Zusammenführen von Funktionen und Klassentemplates generiert. Der Vorteil dieser Generatoren liegt darin, dass sie, genauso wie bei der herkömmlichen Programmierung, auf verschiedene Komponenten aufgeteilt werden können. Die integrierten Möglichkeiten zur Metaprogrammierung sind für die entsprechende Sprache optimiert und einfacher zu benutzen als allgemeine interne Coderepräsentationen, wie zum Beispiel abstrakte Syntaxbäume. Die Erstellung von Generatoren durch Nutzung der vorhandenen Hilfsmittel zur Metaprogrammierung ist wesentlich leichter als eine komplette Eigenentwicklung. Dennoch gibt es bei der Verwendung der integrierten Metaprogrammierungshilfen einige Einschränkungen. So gibt es zum Beispiel Sprachen, in denen die Spezifikationsnotation durch die Syntax der jeweiligen Sprache eingeschränkt ist und nicht erweitert werden kann. Ein weiterer Nachteil ist die schlechte Unterstützung der Debugmöglichkeiten des Metacodes.

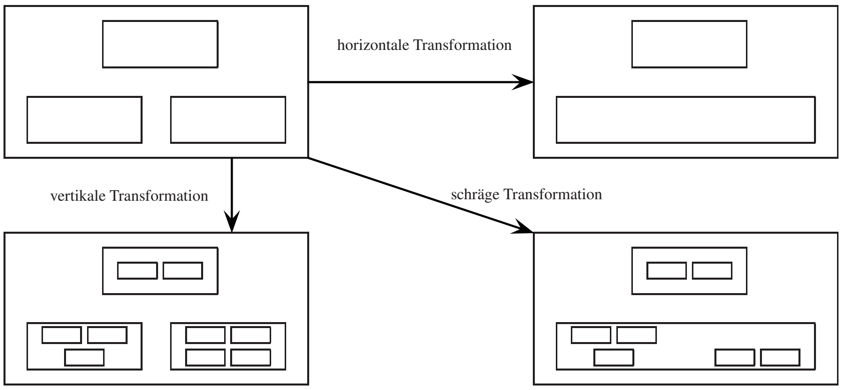
- *Entwicklung des Generators durch Nutzung einer Generatorinfrastruktur*
  Die angemessenste Variante, um Generatoren zu erstellen, ist die Nutzung einer gebräuchlichen Infrastruktur, die alle einfachen Hilfsmittel bereitstellt, die zum Erzeugen eines Generators benötigt werden. Diese Infrastruktur beinhaltet ein allgemeines Format für die interne Repräsentation und verschiedene Möglichkeiten für die Transformation des Codes. Des weiteren soll eine derartige Infrastruktur die Erzeugung von Eingabe- und Ausgabehilfen unterstützen und Möglichkeiten zum Debuggen bereitstellen.

Es existieren unterschiedliche Möglichkeiten, wie der Generator die Ausgabe generieren kann. Die Generatoren können in drei unterschiedliche Klassen eingeteilt werden. Die erste Klasse ist die Klasse der Generatoren, die eine vertikale Transformation durchführen. Generatoren die zu dieser Klasse gehören, führen eine so genannte Vorwärtsverfeinerung (*forward refinement*) durch. Dabei wird eine Repräsentation von einer hohen Abstraktionsebene auf eine Repräsentation auf einer niedrigeren Abstraktionsebene unter Beibehaltung der vorhandenen Modularität

[1] Yet Another Compiler-Compiler
[2] A Lexical Analyzer Generator

der höheren Abstraktionsebene transformiert. Unter Abstraktion wird die Auslassung irrelevanter Details verstanden. Eine Abstraktionsebene wird als höhere Abstraktionsebene bezeichnet wenn mehr Details ausgelassen werden. Dementsprechend werden auf einer niedrigeren Abstraktionsebene weniger Details ausgelassen. Vorwärtsverfeinerung realisiert eine hierarchische Dekomposition. Daher werden Generatoren, die eine Vorwärtsverfeinerung durchführen, auch zusammensetzende Generatoren genannt, da sie die Systeme durch Komposition von kleineren Komponenten zu größeren Komponenten auf hierarchische Art und Weise durchführen. Bei zusammensetzenden Generatoren wird die Modulstruktur der höheren Abstraktionsebene bei der Transformation nicht verändert. Ein Beispiel für eine Transformation, die die Modulstruktur neu definiert, ist in Abbildung 2.7 zu sehen. Bei der horizontalen Transformation kann die Modulstruktur auf der gleichen Abstraktionsebene verändert werden. Dies kann durch das Löschen, Zusammenführen oder Hinzufügen von Modulen geschehen. Die aspektorientierte Programmierung, die im Kapitel 2.4 genauer beschrieben wird, baut auf Generatoren dieser Klasse auf. Wie in Abbildung 2.7 ebenfalls zu sehen ist, kann die vertikale und horizontale Transformation auch zur schrägen Transformation kombiniert werden. Dabei wird sowohl die Repräsentation der Abstraktionsebene transformiert, als auch die Modulstruktur verändert.

Abbildung 2.7 Vertikale, horizontale und schräge Transformation



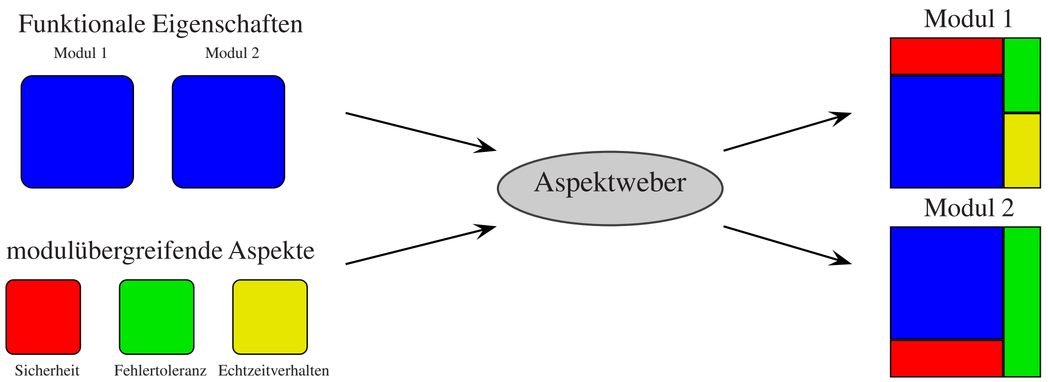
2.4 Aspektorientierte Programmierung

Bei der Entwicklung von Softwaresystemen wird die Gesamtaufgabe in mehrere Teile zerlegt, die unabhängig voneinander implementiert werden können, um das System übersichtlich zu gestalten. Beim objektorientierten Programmieren wird dazu eine Unterteilung des Systems in funktionale Komponenten vorgenommen. Eine gute Strukturierung des Quellcodes erleichtert das Verständnis der Implementierung sowie eine mögliche Wiederverwendung des Codes. Es gibt jedoch technische Anforderungen an Systeme, wie zum Beispiel Fehlererkennung und Fehlerbehandlung, Optimierung oder Persistenz, die sich nicht auf eine funktionale Einheit beschränken lassen. Sie durchdringen die gesamte Implementierung und lassen sich daher nicht kapseln. Die Codefragmente, die durch diese Anforderungen entstehen und in der gesamten Implementierung vorkommen können, lassen sich nicht immer direkt zuordnen und erschweren somit die Lesbarkeit und Verständlichkeit des Quelltextes. Die Wiederverwendbarkeit von Komponenten, die Codefragmente von verschiedenen technischen Anforderungen enthalten, ist daher meist nicht unverändert möglich.

Diese Erkenntnis führte zur Entwicklung des aspektorientierten Programmierens (AOP, Aspect-

Oriented Programming) [KLM+97]. Durch das aspektorientierte Programmieren sollen Techniken und Methoden bereitgestellt werden, um Problemstellungen in funktionale Komponenten und Aspekte, die die Grenzen der funktionalen Komponenten überschreiten, aufteilen zu können. Dadurch können diese Aspekte gekapselt werden und die Verständlichkeit des Codes wird erhöht. Die Komposition von funktionalen Komponenten und Aspekten zu einem Gesamtsystem sollte automatisch erfolgen. Daher erfolgt die Implementierung der Aspekte meistens in einer speziellen Aspektsprache. Durch einen Aspektweber werden die Aspekte dann mit dem restlichen System kombiniert, wie in Abbildung 2.8 veranschaulicht wird.

Abbildung 2.8 Kombination von funktionalen Komponenten und Aspekten



Der Aspektweber muss natürlich wissen, wo der Aspektcode im System eingefügt werden soll. Deshalb können in den Aspektsprachen so genannte Webepunkte (Join Points) definiert werden. Typische Webepunkte sind zum Beispiel Methodenaufrufe oder der Zugriff auf Variablen. Aspekte können statisch oder dynamisch gewebt werden. Beim statischen Weben wird der Aspektcode zur Übersetzungszeit fest in den Code integriert. Die Alternative zum statischen Weben ist das dynamische Weben, bei dem das eigentliche Weben erst zur Laufzeit durchgeführt wird. Statisches Weben bringt Vorteile bei der Optimierung zur Übersetzungszeit, während beim dynamischen Weben während der Laufzeit auf Veränderungen reagiert und somit das Programmverhalten angepasst werden kann.

Statisches Weben kann durch Präprozessoren erledigt werden, die ein Aspektprogramm in eine traditionelle Programmiersprache umformen. Vorteil dieser Technik ist die relativ einfache Implementierung. Die meist vorhandene Infrastruktur zum Erzeugen von Maschinencode oder zum Zusammenfügen von kompilierten Modulen kann ohne Änderungen weiter benutzt werden. Nachteil dieser Lösung ist die häufig fehlende Integration in die Programmiersprache. Sämtliche Informationen über die Aspekte im Quelltext sind nach dem Weben verloren und können nicht für Fehlermeldungen und zum Debugging benutzt werden, was den Entwicklungskomfort beeinträchtigt. Die zweite Alternative ist das Verwenden einer eigenen Sprache oder von Spracherweiterungen zu existierenden Programmiersprachen. Der Vorteil ist die direkte Integration der Aspekte in die Programmiersprache. Fehlermeldungen und Debugging können nun speziell auf die Gegebenheiten der Aspekte zugeschnitten werden.

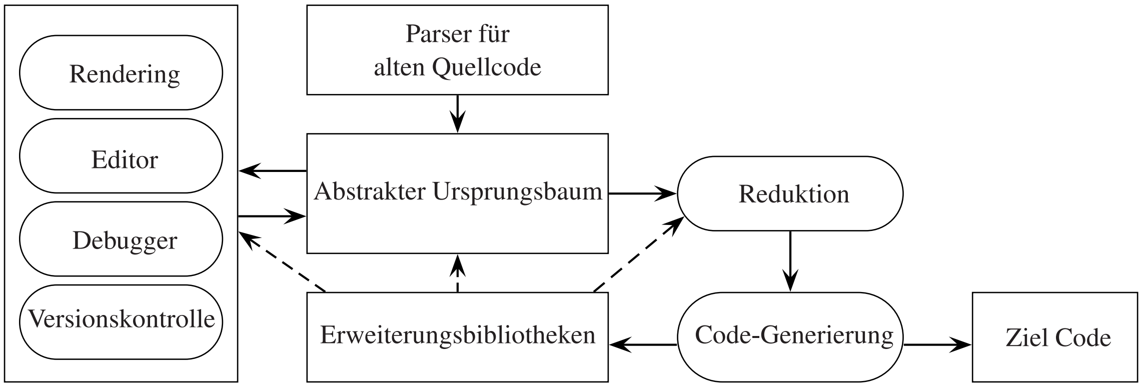
Das dynamische Weben ermöglicht eine flexible Auswahl der Aspekte zur Laufzeit. So kann beispielsweise aus mehreren gleichartigen Aspekten mit verschiedenem Laufzeitverhalten, der für die aktuellen Daten am besten passende Aspekt ausgewählt werden. Der Preis für dynamisches Weben ist erhöhter Speicherverbrauch und ein schlechteres Laufzeitverhalten, da zusätzliche Funktionen ausgeführt werden müssen, die nicht wie beim statischen Weben bereits in die vorhandenen Funktionen integriert sind.

Die momentan bekannteste Aspektsprache ist sicherlich AspectJ [Asp03]. AspectJ wurde ursprünglich am Xerox Palo Alto Research Center entwickelt und ist inzwischen ein Bestandteil des Open-Source Projekt *eclipse.org*. AspectJ verwendet statisches Weben. Es handelt sich hierbei um eine eigenständige Sprache mit eigenem Compiler, der allerdings Java-Bytecode produziert. Somit kann die bestehende Infrastruktur der Java-Programmiersprache genutzt werden.

2.5 Intentionale Programmierung

Die intentionale Programmierung (IP, Intentional Programming) wurde von Charles Simonyi bei Microsoft Research entwickelt [Sim95]. Dabei handelt es sich um einen völlig neuen Ansatz der Softwareentwicklung, da er sich nicht auf eine neue Art von Sprachkonstrukten beschränkt, sondern den gesamten Entwicklungsprozess umfasst. Der Grundgedanke des intentionalen Programmierens ist der Versuch, den essentiellen Inhalt (d.h. die Intention) eines Programms weitgehend sprachunabhängig zu repräsentieren. Die Entwickler erzeugen und manipulieren beim intentionalen Programmieren direkt abstrakte Syntaxbäume. Zur Anzeige der Syntaxbäume und ihrer Knoten können beliebige Methoden geschrieben werden, ebenso zur Überführung der Syntaxbäume in eine höhere Programmiersprache oder direkt in Maschinencode. Intentionales Programmieren bietet eine integrierte Umgebung für Programmierung und Metaprogrammierung, in der der Entwickler seinen Systementwurf von der ersten Modellierung bis zum ausführbaren Code durchführen kann. Entscheidendes Merkmal der Umgebung ist die Erweiterbarkeit. Über Erweiterungsbibliotheken kann die gesamte Umgebung mit Editoren, Compilern, Debuggern und sonstigen Hilfsmitteln an die Bedürfnisse des Entwicklers angepasst werden. In Abbildung 2.9 sind die Hauptbestandteile der integrierten Entwicklungsumgebung dargestellt [Sim96].

Abbildung 2.9 Entwicklungsumgebung der intentionalen Programmierung

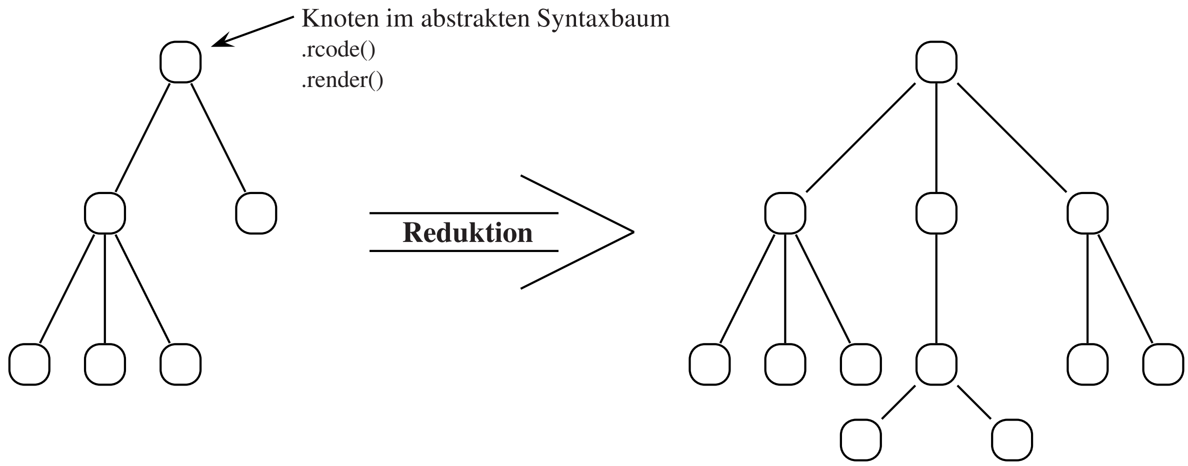


Eine der Hauptideen des intentionalen Programmierens ist, den Quellcode nicht als einfachen Text zu präsentieren, sondern als so genannten aktiven Quellcode (*active source*). Dabei handelt es sich um Code, der seine eigenen Bearbeitungs-, Kompilierungs- und Debuggingmöglichkeiten bereitstellt. Die Darstellung und Eingabe von Konstrukten im Editor wird durch *Rendering-* und *Type-In Methods* automatisch geregelt. Des weiteren existieren *Refactoring Methods* für die Optimierung des Syntaxbaums, um weniger Code genieren zu müssen. Die Optimierung kann recht einfach durchgeführt werden, da der Code bereits in der Form eines abstrakten Syntaxbaums vorliegt und nicht erst in eine derartige Form überführt werden muss. Intentionale Programmierung ist dadurch eine ideale Umgebung für die Verwendung von domänenspezifischen Sprachen.

Mit Hilfe von *Intentions* und einer entsprechenden Extension Library können Sprachabstraktionen beliebiger Sprachen dargestellt werden. Auf diese Art und Weise können auch domänenspezifische Sprachen genutzt werden, durch die bestimmte Eigenschaften leicht verständlich dargestellt werden können. Dabei wird die angezeigte Notation von der internen Darstellung abgekoppelt, ohne Abstriche bei der Flexibilität oder Performanz der späteren Realisierung. Dadurch wird eine vollständige Trennung von Darstellung und fertigem Code erreicht. So ist es möglich, im Editor eine leicht verständliche Notation zu verwenden, die dann beim Kompilieren zu einem leistungsfähigen Code optimiert wird, der jedoch nur schwer lesbar ist.

Der Vorgang des Kompilierens wird *Reduktion* genannt und entspricht einer Baumtransformation, bei der die Konstrukte des abstrakten Syntaxbaums auf konkretere Ebenen transformiert werden. Dieser Vorgang ist in Abbildung 2.10 schematisch dargestellt. Durch Aufruf der Methode *rcode* auf dem Wurzelknoten des abstrakten Syntaxbaums wird die Reduktion gestartet und die reduzierte Repräsentation wird zurückgegeben. Der Aufruf wird jeweils an die Kindknoten weitergegeben, so dass auf diese Art und Weise der komplette Syntaxbaum traversiert wird. Die Ergebnisse der Reduktion der Kindsknoten werden um eigene Informationen ergänzt und an den Vaterknoten hoch gereicht. Während der Reduktion findet auch eine Überprüfung und Optimierung des Syntaxbaumes statt. Die Optimierung kann dabei auch domänenspezifisch durchgeführt werden, da die Optimierungsregeln von den Sprachabstraktionen selbst bereitgestellt werden können und nicht auf einen externen Optimierer angewiesen sind.

Abbildung 2.10 Baumtransformation durch *Reduktion*



Durch die freie Wahl der Sprache ist das intentionale Programmieren sehr gut zur Spezifizierung von Aspekten im Rahmen der aspektorientierten Programmierung geeignet. Die Aspekte können in einer speziell zugeschnittenen Sprache definiert und in verschiedenen Darstellungen bearbeitet werden. Das Weben wird dann bei der Reduktion durchgeführt. Intentionales Programmieren verbindet die Ausdrucksmächtigkeit von domänenspezifischen Sprachen mit der Leistungsfähigkeit und dem Verbreitungsgrad herkömmlicher Programmiersprachen, da während der Entwicklung domänenspezifische Konstrukte verwendet werden, die dann bei der Reduktion auf Konstrukte herkömmlicher Programmiersprachen übertragen werden. Die von uns in Kapitel 5 beschriebenen domänenspezifischen Sprachen für Versionierungssysteme stellen auch Reduktionsmethoden bereit.

2.6 Modellgetriebene Architekturen

Bei modellgetriebenen Architekturen wird die Spezifikation der eigentlichen Funktionalität von den Details der späteren Realisierung getrennt. Dabei wird eine Anwendung unabhängig von einer Implementierungsplattform und Aspekten der technischen Realisierung in Form eines Modells entworfen. Dieses Modell kann dann schrittweise verfeinert werden. Zum Schluss werden dann auch implementierungsplattformabhängige Aspekte integriert, um ausführbaren Code für die entsprechende Plattform zu erhalten. In der Praxis der Softwareentwicklung hat sich gezeigt, dass sich das Modell und die eigentliche Implementierung eines Softwaresystems mit der Zeit auseinander entwickeln. Häufig werden Veränderungen, die durch Wartungsarbeiten hervorgerufen wurden, im Modell nicht mehr nachvollzogen. Durch modellgetriebene Architekturen soll dieses Problem vermieden werden, da die Änderungen nur am Modell durchgeführt werden sollen und die Implementierung vollständig und zum größten Teil automatisch aus dem Modell erzeugt werden soll. Auf diese Weise soll das Modell und der Code synchron gehalten werden.

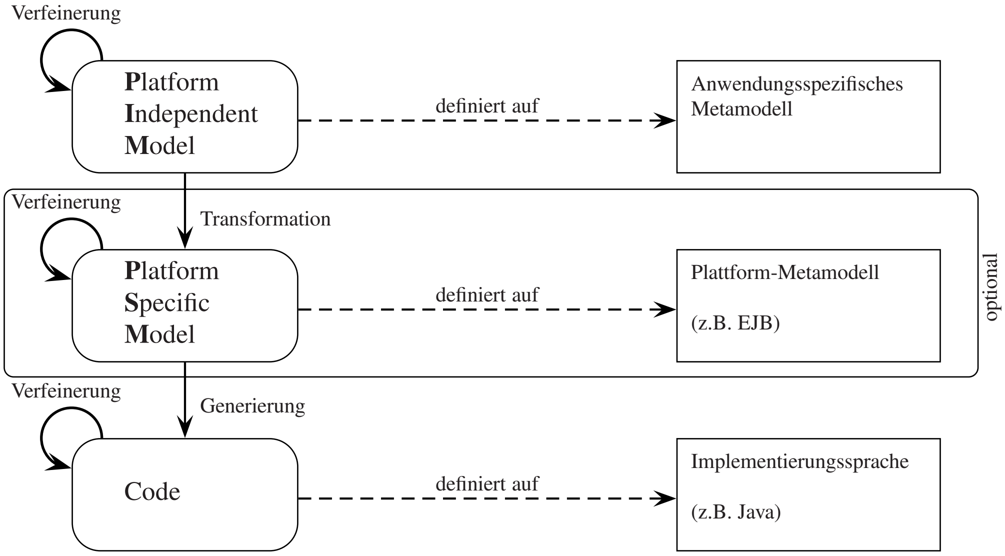
Die wohl bekannteste modellgetriebene Architektur ist die *Model Driven Architecture* (MDA) [OMG02b] , die von der Object Management Group (OMG) als Standard zur Beschreibung von Softwarearchitekturen definiert wurde. Dabei werden die bereits etablierten Standards wie die Unified Modeling Language (UML) [OMG03b], Meta-Object Facility (MOF) [OMG02a] und XML Metadata Interchange (XMI) [OMG03c] genutzt. Die MDA bietet sowohl Definitionen, die als gemeinsames Vokabular für Softwarearchitekturen dienen, als auch eine Strategie, um Softwaremodelle zu formulieren. MDA unterstützt die Existenz unterschiedlicher Plattformen (z.B. J2EE [Sun04a], CORBA [OMG03a] oder .NET [Mic03a]) durch Konzepte von plattformunabhängigen und plattformspezifischen Modellen.

Unter einer Plattform werden hier technische Details und Aspekte der Realisierung zusammengefasst, die für die eigentliche Funktionalität einer Komponente keinerlei Bedeutung haben. Zu jeder Plattform existiert eine Spezifikation in Form eines Modells oder eines UML-Profils. Mit einem plattformunabhängigen Modell (Platform Independent Model, PIM) wird ein System unter der Abstraktion von technischen Details und Aspekten der Realisierung beschrieben. Im Gegensatz dazu wird bei einem plattformspezifischen Modell (Platform Specific Model, PSM) auf diese Details eingegangen und die Realisierung der in einem PIM spezifizierten Funktionalität für eine bestimmte Plattform beschrieben. Durch eine, in der Regel mit Werkzeugen unterstützte, Modelltransformation werden aus den plattformunabhängigen fachlichen Spezifikationen plattformabhängige Modelle gewonnen. Mit einer weiteren werkzeuggestützten Transformation auf Basis eines PSM wird dann für eine konkrete Zielplattform eine Implementierung erstellt. Diese werkzeuggestützten Transformationen werden in Abbildung 2.11 dargestellt.

Um eine Transformation auf einer Plattform beschreiben zu können, wird eine präzise Definition der jeweiligen Plattform benötigt. Einen Ansatz dafür bieten UML-Profile. Mit Hilfe von UML-Profilen werden plattformspezifische Markierungen zur Steuerung von plattformspezifischen Generatoren bereitgestellt [Koc02].

Die Transformation von einem PIM zu einem PSM kann auf unterschiedliche Arten durchgeführt werden [OMG02b]:

Abbildung 2.11 MDA Grundprinzip



- Die Transformation wird vollständig manuell durchgeführt.
- Die Transformation wird vollständig manuell unter Anwendung von vorgegebenen Verfeinerungsmustern durchgeführt.
- Durch einen Algorithmus wird aus dem PIM ein PSM-Skelett generiert. Dieses PSM-Skelett muss dann manuell nach bearbeitet und verfeinert werden. Dabei können auch Verfeinerungsmuster eingesetzt werden.
- Durch einen Algorithmus wird das PSM komplett aus einem vollständigen und eindeutigen PIM generiert.

Die unterschiedlichen Arten der Transformation unterscheiden sich im Automatisierungsgrad.

Aus einem plattformspezifischen Modell einer Anwendung wird nach ausreichender Verfeinerung der Code für die entsprechende Zielplattform generiert. Dazu können neben dem eigentlichen Programmcode auch Schnittstellendefinitionen, Konfigurationsdateien oder Ähnliches gehören. Ob diese Generierung komplett automatisiert durchgeführt werden kann, hängt von der Genauigkeit und Vollständigkeit der Spezifikation ab.

Der Zwischenschritt über das plattformspezifische Modell, um den plattformspezifischen Code zu generieren, ist optional. Er kann weggelassen werden, wenn die Spezifikation der Anwendung ausreichend genau vorhanden und der Codegenerator mächtig genug ist, diese Transformation durchzuführen.

Die MDA ist ein viel versprechender Ansatz, um Arbeit bei der Entwicklung von Software einzusparen. Sie ist heute praktisch einsetzbar, wenn der OMG-Standard geeignet interpretiert wird. Es bestehen nur geringe Abhängigkeiten zu Werkzeugherstellern, da bis auf die Transformation, die noch nicht standardisiert ist, die Interoperabilität gegeben ist.

In dieser Arbeit wird beschrieben, wie domänenspezifische Modelle für datenbankorientierte Anwendungen definiert und anschließend Datenbankschemata, die auf die Modelle zugeschnitten sind, generiert werden. Die Modellierung erfolgt unabhängig vom verwendeten Datenbanksystem und dem dabei verwendeten SQL-Dialekt. Die domänenspezifischen Modelle sind plattformunabhänig und werden auf weitere Modelle abgebildet werden.

Kapitel 3

Produktlinie für Versionierungssysteme

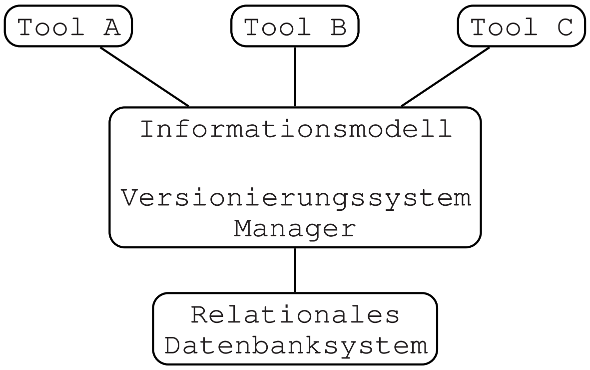
In diesem Kapitel betrachten wir objektorientierte Versionierungssysteme als ein Beispiel für eine Software-Produktlinie. In objektorientierten Versionierungssystemen können Daten sowie Beziehungen zwischen Daten gespeichert werden. Die Speicherung der Daten ist versionierbar, so dass mehrere Versionen eines Datenobjektes parallel existieren können. Durch die versionierte Speicherung der Daten ist es möglich, den zeitlichen Verlauf der Datenveränderung nachzuvollziehen. Die versionierte Speicherung der Daten ermöglicht außerdem, dass in einer Mehrbenutzerumgebung eine parallele Bearbeitung eines Objektes durch verschiedene Benutzer möglich ist. Zum Einsatz kommen derartige Versionierungssysteme beispielsweise bei der Entwicklung von Softwaresystemen, bei denen mehrere Entwickler parallel auf dem gespeicherten Datenbestand arbeiten, ohne sich dabei gegenseitig zu blockieren. Es ist jedoch auch möglich, den Datenbestand einer Unternehmensanwendung versioniert zu speichern.

3.1 Objektorientierte Versionierungssysteme

Versionierungssysteme stellen, wie in Abbildung 3.1 dargestellt, eine parallel nutzbare Schnittstelle für Anwendungsprogramme und Entwicklungswerkzeuge bereit. Durch die Speicherung der Daten in Versionierungssystemen können die Daten anwendungsübergreifend genutzt werden. Es ist nicht notwendig, für jede Anwendung eine anwendungsspezifische Datenbank zu entwickeln [BD94]. Das Informationsmodell stellt für Anwendungen, die die Daten des Versionierungssystems nutzen möchten, ein Modell für die Struktur der im Versionierungssystem gespeicherten Daten bereit. Der Zugriff auf die im Versionierungssystem gespeicherten Daten erfolgt über den Versionierungssystem-Manager. Durch diesen werden den Anwendungen die notwendigen Methoden für den Zugriff und die Manipulation der im Versionierungssystem gespeicherten Daten zur Verfügung gestellt [Ber97]. Diese Methoden werden später noch im Detail vorgestellt. Die Speicherung der Daten erfolgt in relationalen Datenbanken, da diese in der Lage sind, große Datenmengen effizient zu verwalten und mit Hilfe von SQL komplexe Anfragen und Transaktionen zu unterstützen [Ber96].

Versionierungssysteme eignen sich besonders gut, um sie im Rahmen einer Software-Produktlinie zu realisieren. Aus dem Informationsmodell, das für jedes Mitglied der Produktlinie definiert und im Verlauf dieses Kapitels vorgestellt wird, kann beispielsweise ein auf die Anwendung zugeschnittenes Datenbankschema generiert werden, das passende Tabellen für die Speicherung der definierten Datenobjekte enthält. Besondere Eigenschaften können dabei individuell festgelegt werden. Dazu gehören unter anderem die Propagierung von Operationen, die auf einem Objekt ausgeführt wurden, an Objekte, die mit diesem Objekt in Beziehung stehen. Die domänenspezifischen Eigenschaften werden im Verlauf dieses Kapitels vorgestellt.

Abbildung 3.1 Anwendungsbeispiel für Versionierungssysteme



3.2 Datenspeicherung

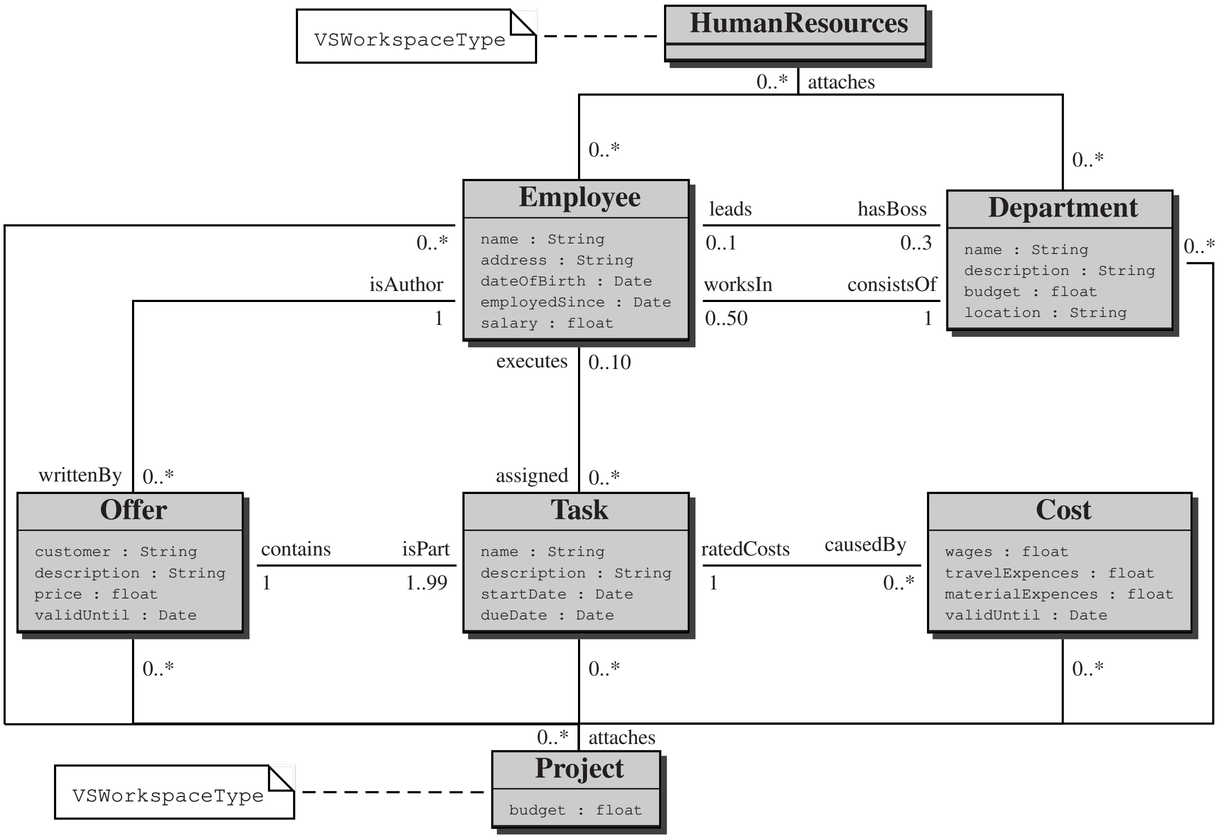
Für die strukturelle Darstellung von Daten und Beziehungen zwischen den Daten werden im Versionierungssystem entsprechende Objekt- und Beziehungstypen angelegt. Ein Datenobjekt ist eine Instanz des entsprechenden Objekttyps. Bei der Speicherung der Datenobjekte wird zwischen versionierbaren und nicht versionierbaren Datenobjekten unterschieden. Dementsprechend werden vom Versionierungssystem unterschiedliche Objekttypen für versionierbare und nicht versionierbare Objekte bereitgestellt. Für versionierbare Objekttypen lässt sich im Informationsmodell die maximale Versionsanzahl spezifizieren. Nicht versionierbare Objekttypen sind ein trivialer Sonderfall der versionierbaren Objekttypen, bei denen maximal eine Version existieren darf. Bei der Speicherung von nicht versionierten Objekten kann auf versionierungsspezifische Verwaltungsattribute verzichtet werden. Die Beziehungen, die zwischen Objekten existieren können, werden mit Hilfe von Instanzen eines Beziehungstyps gespeichert. Objekttypen und Beziehungstypen können Attribute enthalten, in denen Daten gespeichert werden können. Die Beziehungen zwischen Objekten sind bi-direktional, so dass eine Traversierung der Beziehungen in beide Richtungen möglich ist. Die Kardinalität der Beziehungen kann 1:1, 1:n oder n:m betragen.

Die in einem Versionierungssystem enthaltenen Objekttypen und Beziehungstypen werden in einem so genannten Informationsmodell beschrieben. Ein solches Modell für ein objektorientiertes Versionierungssystem ist in Abbildung 3.2 dargestellt. Das Beispiel zeigt ein einfaches Datenmodell eines Versionierungssystem für die Speicherung von Daten aus dem Bereich Projektmanagement. Die beiden Objekttypen für die versionierten Objekte *Employee, Department, Offer, Task* und *Cost* werden dabei als Klassen dargestellt. Die Beziehungstypen für die Beziehungen *leads_hasBoss, worksIn_consistsOf, isAuthor_writtenBy, executes_assigned, contains_isPart* und *ratedCosts_causedBy* werden als Assoziationen zwischen den Klassen dargestellt.

3.3 Arbeitskontexte

Objekte können mit Hilfe von Workspaceobjekten zu Arbeitskontexten zusammengefasst werden. Ein Arbeitskontext beinhaltet meistens semantisch zusammenhängende Objekte. Im In-

Abbildung 3.2 Beispiel eines Informationsmodells für ein Versionierungssystem



formationsmodell des Versionierungssystems wird mit Hilfe der *Attach*-Beziehung festgelegt, welche Objekte in Arbeitskontexten des entsprechenden Workspaceobjekttyps gespeichert werden können. Zu einem Arbeitskontext kann immer nur eine Version eines Objektes hinzugefügt werden. Ein Arbeitskontext bietet eine versionsfreie Sicht auf das Versionierungssystem. Dadurch kann bei einem Zugriff auf ein Objekt innerhalb eines Arbeitskontextes auf die Angabe der *Versions-ID* verzichtet werden. Dies vereinfacht die Handhabung der Objekte innerhalb von Arbeitskontexten.

3.4 Transaktionsunterstützung

Da für die Datenspeicherung in Versionierungssystemen Datenbanken verwendet werden, kann für die Ausführung der in Abschnitt 3.6 beschriebenen Operationen, die von Datenbanksystemen bereitgestellte Transaktionsunterstützung genutzt werden. Diese Transaktionen besitzen die ACID-Eigenschaften, durch die bei einer entsprechend hohen Isolationsstufe sichergestellt wird, dass Änderungen, die während der Transaktion vorgenommen werden, erst nach erfolgreicher Beendigung der Transaktion für andere Anwender sichtbar sind. Außerdem kann dadurch sichergestellt werden, dass nach Beendigung der Transaktion alle Integritätsbedingungen, die im Informationsmodell spezifiziert wurden, erfüllt sind.

Ein derartiger Transaktionsschutz ist nicht nur bei der Ausführung von elementaren Operationen wichtig, sondern wird auch für die Ausführung von Arbeitsvorgängen benötigt, die aus mehreren elementaren Operationen bestehen. Bei einem solchen Arbeitsvorgang soll es erst nach dessen Beendigung möglich sein, dass andere Benutzter die am Arbeitsvorgang beteiligten Objekte verändern können, um zu verhindern, das vorgenommene Änderungen versehentlich überschrieben werden. Um diesen Schutz gewährleisten zu können, bestehen zwei unterschiedliche Möglichkeiten. Zum einen kann der ACID-Transaktionsschutz der Datenbank genutzt werden, um eine Abfolge von elementaren Operationen ausführen zu können. Diese Möglichkeit beinhaltet jedoch den Nachteil, dass für längere Arbeitsvorgänge eventuell zu viele Datenbanksperren gehalten werden müssen, was den Mehrbenutzerbetrieb erheblich beeinträchtigen kann. Außerdem kann ein derartiger Schutz nicht über mehrere Verbindungsaufbauten zum Versionierungssystem hinweg gewährleistet werden. Zum anderen bieten Versionierungssysteme einen integrierten Schutz, der eine Alternative zu langen ACID-Transaktionen darstellt.

Um diesen Schutzmechanismus nutzen zu können, müssen sich die zu schützenden Objekte in einem Arbeitskontext befinden. Durch Ausführung der Operation `checkout` auf dem zu schützenden Objekt, wird der Schutz aktiviert. Danach kann das Objekt zwar noch von allen Benutzern gelesen werden, aber nur noch innerhalb des Arbeitskontextes manipuliert werden, von dem aus die Sperre aktiviert wurde. Durch die Operation `checkin` wird dieser Schutz wieder aufgehoben. Falls ein anderer Benutzer das geschützte Objekt manipulieren will, so muss er mit der Operation CreateSuccessor eine neue Version des Objektes erzeugen und nach Aufhebung des Schutzes die beiden Objekte mit Hilfe der Operation `merge` wieder zusammenführen.

3.5 Versionsunterstützung

Versionierungssysteme bieten die Möglichkeit, sowohl versionierte als auch nicht versionierte Objekte zu verwalten. Die Handhabung von versionierten Objekten ist komplexer als bei nicht versionierten Objekten, da bei einem Objektzugriff neben der *Objekt-ID* auch die entsprechende *Versions-ID* angegeben werden muss. Es existieren mehrere Möglichkeiten, um beim Zugriff auf versionierte Objekte auf die Angabe der *Versions-ID* zu verzichten, um so die Handhabung der versionierten Objekte zu vereinfachen:

- Explizite Festlegung
  Der Benutzer kann für ein Objekt eine bevorzugte Version angeben, die bei nachfolgenden Zugriffen auf ein Objekt ohne Angabe der Versionsnummer zurückgeliefert wird. Dieser Vorgang wird als *Pinning* bezeichnet. Wird das *Pinning* aufgehoben, so spricht man von *Unpinning*.
- Regelbasierte Auswahl
  Bei der regelbasierten Auswahl wird beim Zugriff auf ein versioniertes Objekt ohne Angabe einer Version vom Versionierungssystem anhand einer Regel die Version bestimmt, die zurückgeliefert wird. Diese Regel kann verschiedene Kriterien enthalten und muss zuvor entsprechend spezifiziert werden. Eine Regel kann beispielsweise immer die neueste Version eines Objektes auswählen.
- Arbeitskontexte
  Ein Arbeitskontext ist ein versionierter Objekttyp, mit dessen Hilfe semantisch zusammenhängende Objekte aggregiert werden können.

Versionierte Objekte enthalten eine Referenz auf ihre Vorgängerversion. Mit Hilfe dieser Referenzen kann der Versionsgraph der Objekte rekonstruiert werden. Dadurch ist ein Zugriff auf Vorgänger-, Nachfolger-, und auch alternative Objektversionen möglich.

3.5.1 Gleitende Beziehungsenden

Gleitende Beziehungsenden (*floating relationship ends*) können an Beziehungsenden zu versionierbaren Objekten im Informationsmodell definiert werden. Abbildung 3.3 zeigt ein Beispiel für ein gleitendes Beziehungsende. Die Objekttypen A und B sind Objekttypen, die eine Versionierung der Objekte unterstützen. Das Beziehungsende der Assoziation am Objekttyp B ist ein gleitendes Beziehungsende. So ist es möglich, dass eine Assoziation, die von einem Objekt vom Objekttyp A ausgeht, nicht auf eine bestimmte Version der Objekte vom Objekttyp B beschränkt ist, sondern mehrere, potentiell auch alle, Versionen der Objekte vom Objekttyp B umfasst. Die Menge der assoziierten Versionen eines Objektes wird als Kandidatenmenge der Versionen (*candidate version collection*) bezeichnet. Ein gleitendes Beziehungsende verändert nicht die Kardinalität einer Beziehung, da vom Benutzer zur Laufzeit eine Version aus der Kandidatenmenge auswählt wird. [KH03b]

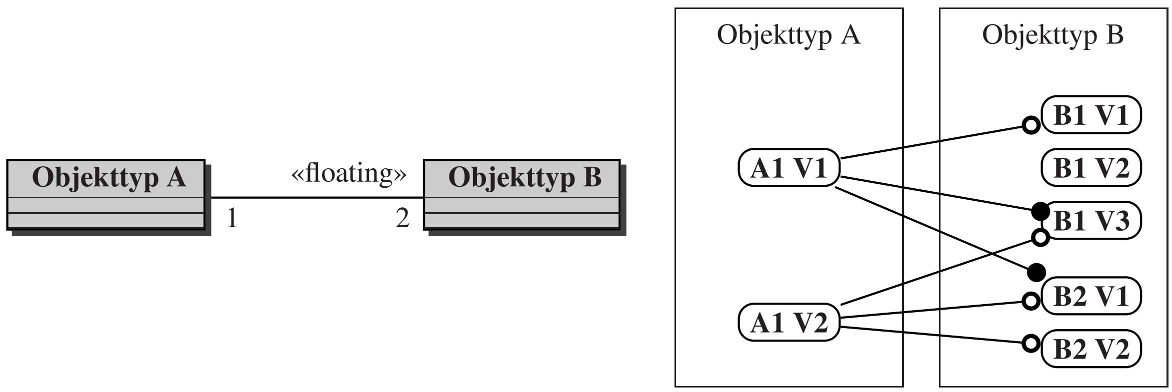
Für den Zugriff auf versionierte Objekte über eine Beziehung mit gleitendem Beziehungsende stehen folgende Varianten zur Auswahl:

- Pinning
  Wenn zuvor eine bevorzugte Version des Objekts angegeben wurde, wird diese Version beim Zugriff auf das versionierte Objekt zurückgeliefert. In Abbildung 3.3 sind die vorausgewählten Objektversionen durch die ausgefüllten Kreise zu erkennen.

- Regelbasierte Auswahl
  Falls keine bevorzugte Version des Objektes definiert wurde, wird mit Hilfe einer Regel eine Version aus der Kandidatenmenge ausgewählt. Dabei handelt es sich um die neuste Version des Objektes.

- gesamte Kandidatenmenge
  Der Benutzer kann sich auch die gesamte Kandidatenmenge zurück liefern lassen und selbst entscheiden, mit welcher Version des Objektes er arbeiten möchte.

Abbildung 3.3 Beispiel: Gleitendes Beziehungsende



3.6 Operationen

Bei der Verwendung von Versionierungssystemen werden nicht nur Operationen für das Lesen und Schreiben von Daten benötigt, sondern auch Operationen für die Objekt- und Versionsverwaltung sowie für die Navigation zwischen den Objekten. Die hier beschriebene Semantik der Operationen wurde bei den in Kapitel 6 beschriebenen Untersuchungen berücksichtigt.

3.6.1 Objekt- und Versionsverwaltung

Die im Versionierungssystem abgelegten Objekte enthalten neben den Attributen für die Nutzdaten auch Attribute, die für die Verwaltung der Objekte benötigt werden. Jedes Objekt besitzt mit der *Global-ID* eine versionierungssystemweit eindeutige ID. Versionierbare Objekte besitzen außerdem noch eine *Objekt-ID* und eine *Versions-ID*, die sich zur *Global-ID* zusammensetzten.

Neben den IDs enthalten die versionierbaren Objekte auch noch die Attribute `predecessor`,`frozen` und `checkedout`. Im Attribut `predecessor` wird eine Referenz auf die Vorgängerversion gehalten. Wenn das Attribut `frozen` den Wert *true* besitzt, kann das Objekt nicht mehr modifiziert werden. Falls auf ein Objekt in einem Arbeitskontext eine Langzeitsperre gehalten wird, wird im Attribut `checkedout` die *Global-ID* des Workspaceobjekts, das die Langzeitsperre auf dem Objekt hält, gespeichert.

Mit den nachfolgend beschriebenen Operationen können die Objekte verwaltet werden. Je nach Objekttyp, auf dem die Operation aufgerufen wird, kann sich die Semantik der Operationen unterscheiden. Außerdem kann nicht jede Operation auf jedem Objekttyp ausgeführt werden. Im Rahmen der Realisierung von Versionierungssystemen als Software-Produktlinie wäre es möglich, dass der Produktlinienhersteller auch unterschiedliche Semantiken für die Methoden anbietet.

- `new`
  Erzeugt eine neue Instanz eines Objekttyps. Falls für die Attribute keine Werte übergeben werden, werden die Attribute mit voreingestellten Standardwerten initialisiert. Nicht versionierten Objekten wird eine eindeutige *Global-ID* zugewiesen. Versionierbare Objekte erhalten eine neue noch nicht verwendete *Objekt-ID*. Die *Versions-ID* wird auf 1 gesetzt und die dazugehörige *Global-ID* berechnet.

- `copy`
  Erzeugt ebenfalls eine neue Instanz eines Objekttyps. Dabei werden die Attribute mit den Werten der bestehenden Instanz initialisiert. Der neuen Instanz eines nicht versionierten Objektes wird eine eindeutige *Global-ID* zugewiesen. Die neue Instanz eines versionierbaren Objektes erhält analog zur Operation `new` eine neue noch nicht verwendete *Objekt-ID*. Die *Versions-ID* wird auf 1 gesetzt und die dazugehörige *Global-ID* berechnet.

- `delete`
  Löscht eine Version eines Objektes, falls diese Objektversion keine Nachfolgerversionen besitzt. Wird `delete` auf einer Version eines Workspaceobjekts aufgerufen, so wird es gelöscht, falls keine Nachfolgerversionen existieren. Die Referenzen auf die im Arbeitskontext befindlichen Objekte werden ebenfalls gelöscht. Die Objekte selbst bleiben im Versionierungssystem erhalten.

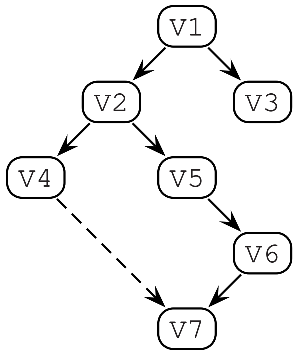
- `freeze`
  Sperrt eine Version eines Objektes gegen eine Änderung der Attributwerte. Sollen dennoch

Änderungen an den Attributwerten vorgenommen werden, so muss mit Hilfe der Operation `CreateSuccessor` eine Nachfolgerversion angelegt werden.

- `CreateSuccessor`
  Erzeugt eine Nachfolgerversion eines versionierten Objektes, falls die maximale Versionsanzahl des Objektes noch nicht erreicht ist. Es werden die Attributwerte und die *Objekt-ID* der aktuellen Version übernommen und die *Versions-ID* wird erhöht. Des weiteren wird eine Referenz auf die Vorgängerversion der Instanz gespeichert. Das Feld *frozen* wird auf *false* gesetzt. Durch die Erzeugung einer Nachfolgerversion ist es möglich, Änderungen an Objekten durchzuführen, deren aktuelle Version durch die Operation `freeze` gesperrt wurde. Wird `CreateSuccessor` auf einem Workspaceobjekt aufgerufen, wird eine neue Version des Workspaceobjekts angelegt und die Referenzen auf die im Arbeitskontext enthaltenen Objekte kopiert.

- `merge`
  Mit der Operation `merge` können zwei Entwicklungszweige wieder vereinigt werden. Abbildung 3.4 zeigt ein Beispiel für die Operation `merge(V4,V7)`. Dabei wird *V4* als primäre und *V7* als sekundäre Version bezeichnet. Beim Zusammenführen der beiden Versionen wird zunächst mit Hilfe des Versionsgraphen die gemeinsame Basisversion bestimmt. Die gemeinsame Basisversion ist die am spätesten erzeugte Version, die sowohl im Pfad der Version *V4* als auch im Pfad der Version *V7* zur Wurzel (*V1*) vorhanden ist. Im gezeigten Beispiel ist *V2* die gemeinsame Basisversion. Die Konflikte, die beim Zusammenführen der Versionen entstehen können, werden für jedes Attribut mit Hilfe von Regeln aufgelöst. Wenn ein Attribut in der primären Version im Vergleich zur Basisversion geändert wurde, enthält die zusammengeführte Version den Wert des Attributs der primären Version. Ist ein Attributwert nur in der sekundären Version geändert, so wird dieser in die zusammengeführte Version übernommen [Mic03b]. Die Tabelle in Abbildung 3.4 zeigt die Ergebnisse für die Operation `merge(V4,V7)`.

Abbildung 3.4 Beispiel: Operation `merge`



Version	Attribut A	Attribut B	Attribut C
V2	31.03.04	blau	351
V4	16.03.04	blau	371
V7	31.03.04	grün	375
V7 (merged)	16.03.04	grün	371

- `attach/detach`
  Mit der Operation `attach` kann ein Objekt zu einem Arbeitskontext hinzugefügt werden. Durch die Operation detach wird das Objekt wieder aus dem Arbeitskontext entfernt.

- `checkout/checkin`
  Durch die Operation `checkout` wird eine Langzeitsperre für ein Objekt, dass in einem Arbeitskontext enthalten ist, gesetzt. Im Attribut *checkout* wird eine Referenz auf den Arbeitskontext gehalten, durch den das Objekt gesperrt wurde. Wenn ein Objekt durch die Operation

Tabelle 3.1 Übersicht der Anwendbarkeit von Methoden auf Objekttypen

Operation	unversionierter Objekttyp	versionierter Objekttyp	Workspace Objekttyp
`new`	x	x	x
`copy`	x	x	x
`delete`	x	x	x
`freeze`	-	x	x
`CreateSuccessor`	-	x	x
`merge`	-	x	-
`attach/detach`	x	x	-
`checkout/checkin`	x	x	-
`getRoot`	-	x	x
`getAllVersions`	-	x	x
`getPredecessor`	-	x	x
`getSuccessors`	-	x	x
`getBaseVersion`	-	x	x

`checkout` gesperrt wurde, können Änderungen nur noch im Arbeitskontext vorgenommen werden, der die Sperre auf dem Objekt hält. Eine weitere Sperre in einem anderem Arbeitskontext ist nicht möglich. Um parallel zur Sperre Änderungen vornehmen zu können, muss mit `CreateSuccessor` eine Nachfolgerversion des Objektes angelegt werden. Mit Hilfe der Operation `checkin` wird die Langzeitsperre wieder aufgehoben. Wenn auf einem Objekt in einem Arbeitskontext eine Langzeitsperre gehalten wird, dürfen innerhalb dieses Arbeitskontextes die Methoden `CreateSuccessor` und `freeze` nicht auf dem gesperrten Objekt angewendet werden.

- `getRoot`
  Liefert die Objektversion mit der *Versions-ID* 1, d.h. die Wurzel des Versionsbaums, zurück.
- `getAllVersions`
  Liefert alle Versionen eines Objektes zurück.
- `getPredecessor`
  Liefert die direkte Vorgängerversion einer Objektversion zurück.
- `getSuccessors`
  Liefert alle direkten Nachfolgerversionen einer Objektversion zurück.
- `getAlternatives`
  Liefert alle Objektversionen, die die gleiche Objektversion als direkte Vorgängerversion besitzen, zurück.
- `getBaseVersion`
  Liefert die gemeinsame Basisversion zweier Objektversionen zurück. Die gemeinsame Basisversion ist die am spätesten erzeugte Version, die in beiden Pfaden der Objektversionen zur Wurzel vorhanden ist.

In Tabelle 3.1 ist die Anwendbarkeit der einzelnen Methoden auf die unterschiedlichen Objekttypen nochmals übersichtlich dargestellt.

Tabelle 3.2 Beziehungsenden für das Beispiel aus Abbildung 3.2

Rollenname	Objekttyp	Beziehung	minimale Kardinalität	maximale Kardinalität	gleitendes Beziehungsende	Akzeptiert Propagierung von						
						New	Copy	Delete	CreateSuccessor	AttachDetach	Freeze	CheckoutCheckin
leads	Employee	leads_hasBoss	0	1	-	-	-	-	-	x	x	x
hasBoss	Department	leads_hasBoss	0	3	-	-	-	-	-	-	-	-
worksIn	Employee	worksIn_consistsOf	0	50	-	-	-	-	-	x	x	x
consistsOf	Department	worksIn_consistsOf	1	1	-	-	-	-	-	-	-	-
isAuthor	Employee	isAuthor_writtenBy	1	1	-	-	-	-	-	-	-	-
writtenBy	Offer	isAuthor_writtenBy	0	*	x	-	-	-	-	-	-	-
executes	Employee	executes_assigned	0	10	-	-	-	-	-	-	-	-
assigned	Task	executes_assigned	0	*	x	-	-	-	-	-	-	-
contains	Offer	contains_isPartOf	1	1	-	-	-	-	-	-	x	-
isPartOf	Task	contains_isPartOf	1	99	-	x	x	x	-	x	x	x
ratedCosts	Task	ratedCosts_causedBy	1	1	x	-	-	-	-	-	-	-
causedBy	Costs	ratedCosts_causedBy	0	*	x	x	x	x	x	x	x	x

3.6.2 Propagierung von Operationen

Bei der Anwendung einer Operation auf ein Objekt im Versionierungssystem kann es nützlich sein, dass diese Operation an Objekte, die mit dem Objekt in einer Beziehung stehen, propagiert wird. Die Propagierung von Operationen wird im Informationsmodell bei der Definition der Beziehungsenden festgelegt. Die Propagierungseigenschaften für das in Abbildung 3.2 vorgestellte Informationsmodell werden in Tabelle 3.2 gezeigt. So wird zum Beispiel die Propagierung der Operation `freeze` von Objekten des Typs *Task* über die Beziehung *ratedCosts_causedBy* von Objekten vom Typ *Cost* akzeptiert.

In der domänenspezifischen Anfragesprache für Versionierungssysteme, die im Kapitel 5.2.2 vorgestellt wird, ist es bei einem Aufruf der Operationen `copy`, `delete`, `freeze` und `CreateSuccessor` auf einem Workspaceobjekt auch möglich, die entsprechende Operation, an die im Arbeitskontext befindlichen Objekte, zu propagieren. Diese Propagierungsmöglichkeit dient nur zur Automatisierung von Vorgängen und führt nie zu einer Verletzung der semantischen Integrität.

3.6.3 Manipulation von Beziehungen zwischen Objekten

Es existiert eine Reihe von Methoden, um Beziehungen zwischen Objekten manipulieren zu können. Dabei unterscheiden wir drei Arten von Beziehungen, die sich in der Anzahl der gleitenden Beziehungsenden unterscheiden. Es gibt Beziehungen mit keinem, einem oder zwei gleitenden Beziehungsenden. Bei Beziehungen, die mindestens ein gleitendes Beziehungsende besitzen, kann durch bestimmte Befehle die entsprechende Kandidatenmenge verändert werden. Die verschiedenen Befehle, die zur Manipulation der Kandidatenmenge oder der Beziehungen zwischen Objekten verwendet werden, werden nachfolgend erklärt.

- `CreateNewRelationship`
  Erzeugt eine neue Beziehung zwischen zwei Objekten. Dabei wird an beiden Beziehungsen-

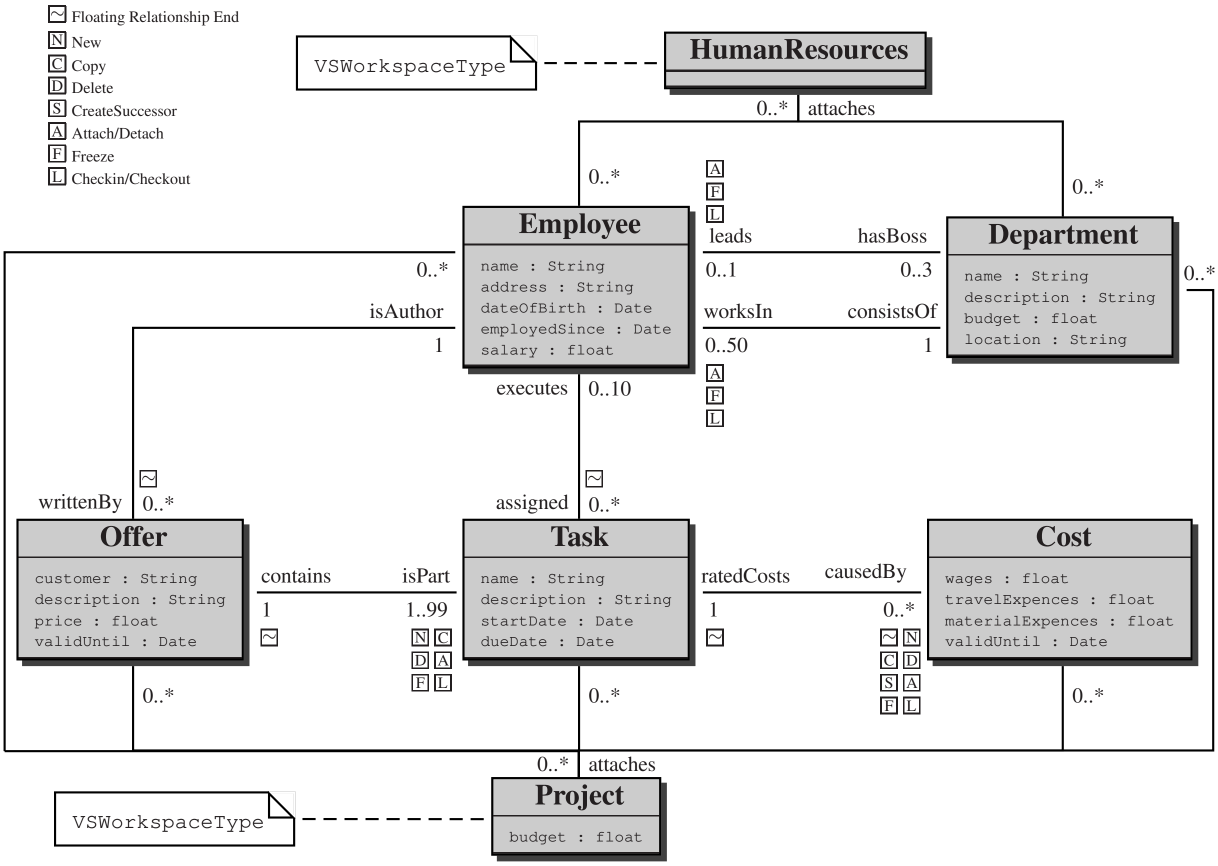
den, unabhängig davon, ob es sich um ein gleitendes Beziehungsende handelt oder nicht, die im Informationsmodell spezifizierte maximale Kardinalität überprüft.

- `DeleteRelationship`
  Löscht eine bestehende Beziehung zwischen zwei Objekten. Dabei wird an beiden Beziehungsenden, unabhängig davon, ob es sich um ein gleitendes Beziehungsende handelt, die im Informationsmodell spezifizierte minimale Kardinalität überprüft.
- `PinVersion`
  Markiert eine Version innerhalb der Kandidatenmenge als bevorzugte Version, die dann bei der Navigation (außerhalb von Arbeitskontexten) über diese Beziehung genutzt wird.
- `UnPinVersion`
  Hebt die Markierung einer bevorzugten Version innerhalb der entsprechenden Kandidatenmenge auf.
- `GetCVC`
  Liefert die gesamte Kandidatenmenge unabhängig von der Pinning-Einstellung zurück.
- `AddToCVC`
  Fügt eine Version eines versionierten Objektes zur Kandidatenmenge hinzu.
- `DeleteFromCVC`
  Entfernt eine Version eines versionierten Objektes aus der Kandidatenmenge.
- `GetRelatedObjects`
  Liefert alle Objekte, die mit einem bestimmten Objekt in Beziehung stehen, zurück. Falls die Objekttypen, die mit diesem Objekttyp in Verbindung stehen, ein gleitendes Beziehungsende auf ihrer Seite der Beziehung besitzen, werden die zuvor vorausgewählten Objektversionen zurückgegeben. Wenn innerhalb der entsprechenden Kandidatenmenge keine Objektversion vorausgewählt ist, wird eine regelbasierte Auswahl durchgeführt, das heißt, es wird die neueste Objektversion zurückgegeben.

3.6.4 Beispiel für ein Mitglied der Produktlinie

In Abbildung 3.2 wurde bereits ein Beispiel für ein Informationsmodell vorgestellt. Dieses Beispiel wurde in Abbildung 3.5 durch domänenspezifische Eigenschaften erweitert, und dient als Grundlage für die im Kapitel 6 durchgeführten Leistungsuntersuchungen. Zu diesen Eigenschaften gehören die gleitenden Beziehungsenden sowie die Akzeptanz der Propagierung einer bestimmten Operation, die in diesem Kapitel vorgestellt wurden. Durch die Symbole an den Beziehungsenden werden die im Versionierungssystem spezifizierten Eigenschaften, die von dem Mitglied der Produktlinie genutzt werden, gekennzeichnet. Die Bedeutung der Symbole wird in der Legende links oben erklärt. Bei dem Beziehungsende *causedBy* am Objekttyp *Cost* handelt es sich um ein gleitendes Beziehungsende. Außerdem wird die Propagierung der Operationen *New*, *Copy*, *Delete*, *CreateSuccessor*, *Attach/Detach*, *Freeze* und *Checkin/Checkout* akzeptiert.

Abbildung 3.5 Beispiel: Informationsmodell eines Versionierungssystems mit domänenspezifischen Erweiterungen



Kapitel 4

Domänenspezifische Sprachen für datenbankintensive Anwendungen

In diesem Kapitel werden zunächst die Eigenschaften von domänenspezifischen Sprachen vorgestellt. Anschließend werden die Konzepte einer Meta-Produktlinie für die Spezifikation von Mitgliedern verschiedenster datenbankintensiver Produktlinien vorgestellt.

4.1 Domänenspezifische Sprachen

Domänenspezifische Sprachen sind problemorientierte Sprachen, die speziell für eine bestimmte Anwendungsdomäne zugeschnitten sind. Dabei werden die Begriffe und Merkmale, die spezifisch für eine bestimmte Anwendungsdomäne sind, verwendet, um Zustände oder Prozesse auf einer höheren Abstraktionsebene als bei herkömmlichen Programmiersprachen beschreiben zu können. Derartige Sprachen spielen vor allem im Zusammenhang mit der generativen Programmierung eine wichtige Rolle. Wie bereits in Kapitel 2.2.1 beschrieben, werden bei der generativen Programmierung domänenspezifische Sprachen genutzt, um ein Mitglied einer Systemfamilie spezifizieren und generieren zu können.

Im Kontext der domänenspezifischen Sprachen ist der Begriff *Sprache* sehr allgemein zu verstehen. Bei einer domänenspezifischen Sprache kann es sich sowohl um eine textuelle Sprache (z.B. SQL) als auch um eine grafische Notation handeln. Des weiteren können sich domänenspezifische Sprachen auch im Grad ihrer Spezialisierung unterscheiden. Um eine komplette Anwendung zu beschreiben, werden im allgemeinen mehrere domänenspezifische Sprachen benötigt. Je nach Anwendungsfall kann es sinnvoll sein, für verschiedene Anwendergruppen unterschiedliche Varianten der domänenspezifischen Sprache bereitzustellen, die sich in ihrer Komplexität unterscheiden.

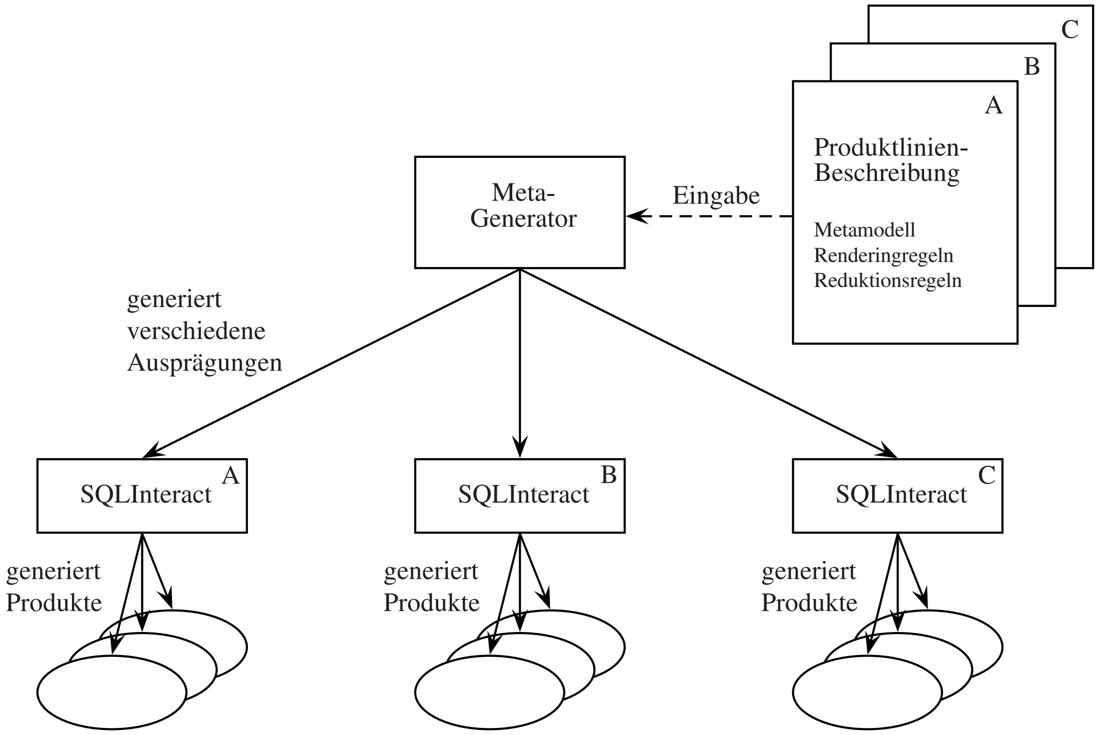
In Abhängigkeit von der genutzten Implementierungstechnologie kann zwischen drei Arten von domänenspezifischen Sprachen unterschieden werden. Statische separate domänenspezifische Sprachen, wie beispielsweise TeX oder SQL, werden typischerweise durch einen eigenen Compiler oder Interpreter implementiert. Diese Art der domänenspezifischen Sprachen ist wohl die bekannteste, wenn auch zu gleich die problematischste, da die Entwicklung des Compilers von Grund auf sehr teuer ist. Diese separaten domänenspezifischen Sprachen führen oft zu nicht kompatiblen technologischen Inseln. Eingebettete domänenspezifische Sprachen sind gewöhnlich in herkömmliche Programmiersprachen integriert. Die einfachste Möglichkeit, eine derartige domänenspezifische Sprache zu realisieren, ist die Nutzung von Klassen und Prozeduren als abstrakte Schnittstelle. Jedoch sind damit einige Einschränkungen verbunden. Beispielsweise können keine domänenspezifischen Optimierungen durchgeführt oder keine domänenspezifische Syntax verwendet werden. Einige herkömmliche Programmiersprachen stellen Möglichkeiten zur Metaprogrammierung bereit, durch die diese Merkmale in Prozeduren oder Klassen eingefügt werden können. Beispielsweise bietet C++ *template metaprogramming*, durch das domänenspezifische Codegenerierung und Optimierung durchgeführt werden können. Die dritte Art der domänenspezifischen Sprachen sind die modular zusammensetzbaren domänenspezifischen Sprachen, bei denen jede domänenspezifische Sprache als eine Komponente betrachtet wird. Die modular zusammensetzbaren domänenspezifischen Sprachen können in die gekapselten und die aspektuellen domänenspezifischen Sprachen aufgeteilt werden. Die gekapselten domänenspezifischen Sprachen sind einfacher zu handhaben, da bei einer Komposition andere domänenspezifische Sprachen nicht beeinflusst werden. Aspektorientierte domänenspezifische Sprachen beeinflussen die Semantik anderer Sprachen sowie ihre Implementierung. Für die Implementierung modular zusammensetzbarer domänenspezifischer Sprachen wird eine gemeinsame Implementierungsplattform, die die notwendige Infrastruktur bereitstellt, benötigt. Eine derartige Plattform bietet zum Beispiel die intentionale Programmierung. Modular zusammensetzbare domänenspezifische Sprachen haben im Vergleich zu monolithischen domänenspezifischen Sprachen Vorteile in Bezug auf Wiederverwendbarkeit und Erweiterbarkeit [CE00].

SQL ist, wie bereits beschrieben, eine Sprache für Datenbankanwendungen, die universal einsetzbar ist. Je nach Anwendungsdomäne, in der SQL eingesetzt wird, wären domänenspezifische Spracherweiterungen für SQL wünschenswert, um die domänenspezifischen Arbeitsabläufe besser unterstützen zu können. Die Unterstützung von domänenspezifischen Spracherweiterungen sind ein wesentlicher Bestandteil dieser Arbeit. Im weiteren Verlauf dieses Kapitels werden die Möglichkeiten zur Generierung von Software-Produkten mit Hilfe von domänenspezifischen Sprachen erläutert. In Kapitel 5 werden dann am Beispiel von Versionierungssystemen domänenspezifische Sprachen zur Spezifikation und Nutzung von datenbankintensiven Anwendungen vorgestellt.

4.2 SQLInteract als Meta-Produktlinie

Die in Kapitel 3 vorgestellten Versionierungssysteme sind lediglich ein Beispiel für eine Produktlinie im Bereich datenbankintensiver Anwendungen. Das *Enterprise Ressource Planning* (ERP) oder das *Customer Relationship Management* (CRM) sind nur zwei weitere Anwendungsgebiete für datenbankintensive Anwendungen, die sich im Rahmen einer Produktlinie realisieren lassen. SQLInteract, ist eine Meta-Produktlinie, mit der die Spezifikation von Mitgliedern verschiedenster datenbankintensiver Produktlinien unterstützt wird. Abbildung 4.1 zeigt einen Meta-Generator, der mit Hilfe einer Produktlinienbeschreibung, die aus einem Metamodell und entsprechenden Rendering- und Reduktionsregeln besteht, eine Ausprägung von SQLInteract generiert. Mit der generierten Ausprägung von SQL-Interact können Mitglieder dieser Produktlinie spezifiziert und generiert werden. Der Generator, mit dem die verschiedenen Ausprägungen von SQLInteract generiert werden können, ist ein Generator für Generatoren, also ein Meta-Generator.

Abbildung 4.1 SQLInteract als Metagenerator



4.2.1 Technologien

Um den genauen Ablauf der Generierung einer datenbankintensiven Anwendung beschreiben zu können, müssen zunächst noch zwei Technologien erläutert werden, die in diesem Zusammenhang eine wichtige Rolle spielen. Dabei handelt es sich um *Meta-Object Facility* (MOF) und *Common Warehouse Metamodel* (CWM), die von der Object Management Group spezifiziert wurden.

Meta-Object Facility

Die Meta-Object Facility (MOF) Spezifikation definiert eine abstrakte Sprache und ein Framework zur Spezifizierung, Konstruktion und Verwaltung von plattformunabhängigen Metamodellen. Ein Metamodell ist eine abstrakte Sprache zur Beschreibung von Metadaten. Metadaten sind Daten, mit denen andere Daten beschrieben werden. Beispiele für derartige Metamodelle sind das Metamodell für UML oder CWM [OMG02a].

Tabelle 4.1 zeigt die typische vierschichtige Architektur des MOF-Frameworks. Ein Modell ist eine Beschreibung eines Ausschnitts der Realität. Im Kontext von MOF wird unter einem Modell eine Menge von Metadaten verstanden. Metadaten sind Informationen, die ebenfalls durch Metadaten beschrieben werden können. Diese Metadaten werden auch als Meta-Metadaten bezeichnet. Ein Modell, das aus derartigen Meta-Metadaten besteht, wird Metamodell genannt. Um Metamodelle beschreiben zu können, werden Meta-Metamodelle verwendet. Das MOF-Modell ist ein Modell für die Beschreibung von Metamodellen. Daher ist es ein Meta-Metamodell.

Die MOF-Spezifikation enthält eine Spezifikation des MOF Meta-Metamodells. Dadurch

Tabelle 4.1 Vierschichtige Metadatenarchitektur von MOF

MOF-Ebene	MOF-Bezeichnung	Beispiel
M3	Meta-Metamodell	Abbildung 4.3
M2	Metamodell, Metadaten	Abbildung 5.1
M1	Modell, Metadaten	Abbildung 3.2
M0	Daten	

existiert eine abstrakte Sprache zur Spezifikation von MOF Metamodellen. Die Modellierungskonzepte von MOF sind ähnlich wie die Konzepte in UML, so dass MOF-Metamodelle in UML-Notation dargestellt werden können. Neben dem MOF-Modell beschreibt die MOF-Spezifikation das MOF-IDL-Mapping und die MOF-Interfaces. Mit dem MOF-IDL-Mapping können IDL (CORBA Interface Definition Language) Schnittstellen aus einem MOF-Metamodell generiert werden. Über diese Schnittstellen können Meta-Objekte verwaltet werden. Bei den MOF-Interfaces handelt es sich um IDL-Schnittstellen für CORBA-Objekte, die ein MOF-Metamodell repräsentieren. Über sie kann per *Reflection* auf die durch ein MOF basierendes Metamodell beschriebenen Metadaten zugegriffen werden.

Common Warehouse Metamodel

Das Common Warehouse Metamodel (CWM) ist eine Spezifikation zur Vereinfachung des Austausches von Metadaten bei Data-Warehouse-Anwendungen und Metadaten-Repositories in verteilten heterogenen Umgebungen. CWM baut auf den OMG Standards UML, MOF und XMI auf. Dabei wird UML als Notation, MOF als Framework für die Organisation und XMI als Format für den Datenaustausch verwendet.

Abbildung 4.2 zeigt den Aufbau des CWM-Metamodells. Die Grobstruktur wird von vier aufeinander aufbauenden Paketen gebildet, die jeweils mehrere Unterpakete enthalten:

- *Foundation*:
  Die Foundation-Ebene enthält allgemeine Konstrukte, die von den anderen Ebenen genutzt oder erweitert werden. Dabei baut sie direkt auf UML bzw. dem Objekt-Modell auf.

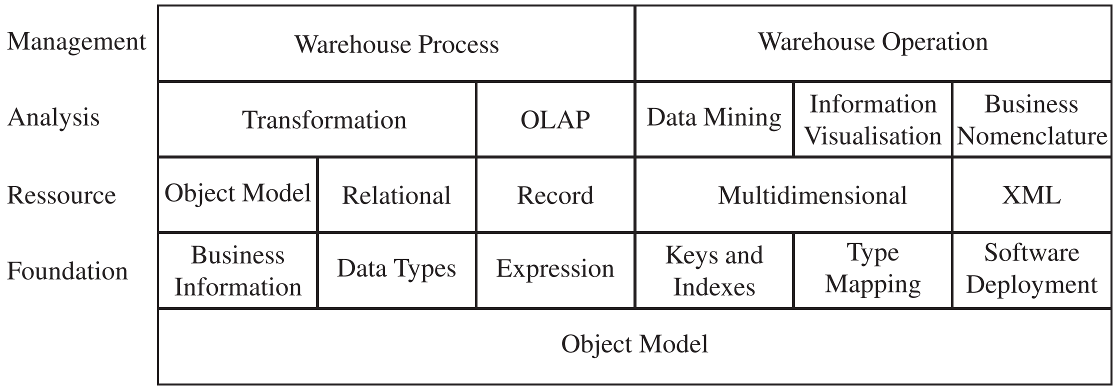
- *Data Ressource*:
  Hierunter fallen Metamodelle, die objektorientierte, relationale, mehrdimensionale und XML-Datenquellen repräsentieren.

- *Data Analysis*:
  Hier werden Metamodelle eingeordnet, die Datentransformation, OLAP (On-Line Analytical Processing), Data-Mining, Informationsvisualisierung und geschäftsbezogene Beziehungen repräsentieren.

- *Warehouse Management*:
  Metamodelle für Warehouse-Prozesse und Warehouse-Operationen werden in dieser Kategorie eingeordnet.

Für die Konzepte, die in den folgenden Abschnitten beschrieben werden, ist das Relational-Package von grundlegender Bedeutung. Das Relational-Package basiert auf dem SQL-Standard und beschreibt Metadaten. In diesem Package werden unter anderem Konstrukte für Tabellen, Primär- und Fremdschlüssel definiert.

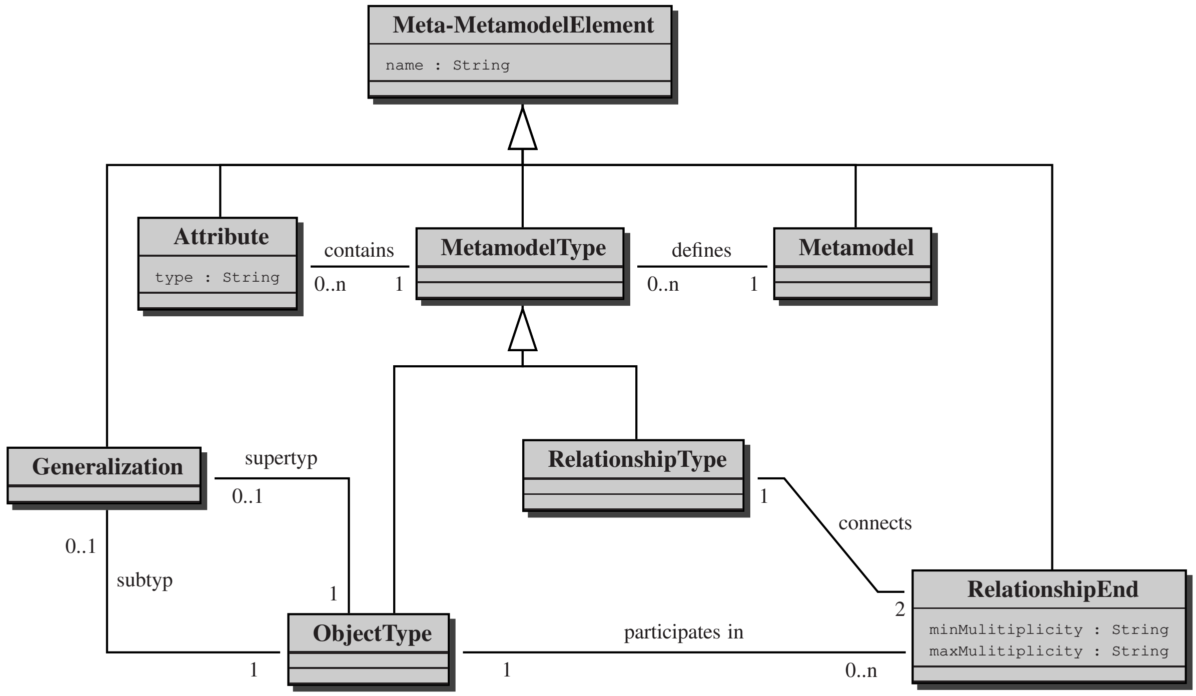
Abbildung 4.2 CWM-Metamodell



4.2.2 Meta-Metamodell

Ein wichtiger Bestandteil einer Produktlinienbeschreibung, die zum Generieren einer Ausprägung von SQLInteract benötigt wird, ist das domänenspezifische Metamodell. Dieses Metamodell ist ein Ergebnis der Domänenanalyse. Mit Hilfe von SQLInteract kann ein Kunde domänenspezifische Modelle definieren, die konform zum domänenspezifischen Metamodell sind. Abbildung 4.3 zeigt das auf MOF basierende Meta-Metamodel zur Beschreibung der domänenspezifischen Metamodelle.

Abbildung 4.3 Meta-Metamodell



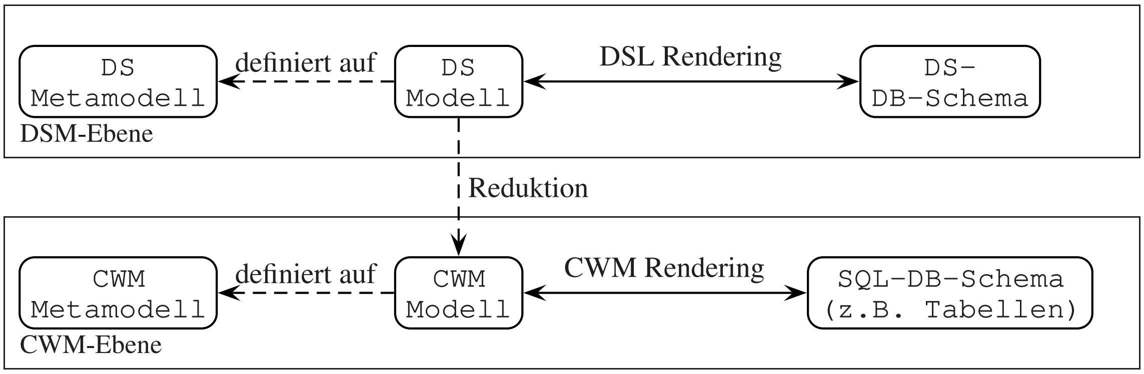
Ein domänenspezifisches Metamodell definiert eine Menge von Metamodelltypen, die Attribute enthalten können. Metamodelltypen sind entweder Objekttypen oder Beziehungstypen. Verschiedene Objekttypen können durch Spezialisierung und Generalisierung in einer *Supertyp-*

oder *Subtypbeziehung* zueinander stehen. Beziehungstypen sind jeweils mit zwei Beziehungsenden verbunden, die jeweils einen Objekttyp referenzieren. Mit den Beziehungsenden können die minimalen und maximalen Kardinalitäten festgelegt werden.

4.2.3 Domänenspezifischer Schemaentwurf

Anhand von Abbildung 4.4 lassen sich die Architektur und die Konzepte, die der Generierung von Produkten in SQLInteract zugrunde liegen, erklären. Die Architektur von SQLInteract gliedert sich in zwei Ebenen, die *CWM-Ebene* und die *DSM-Ebene*.

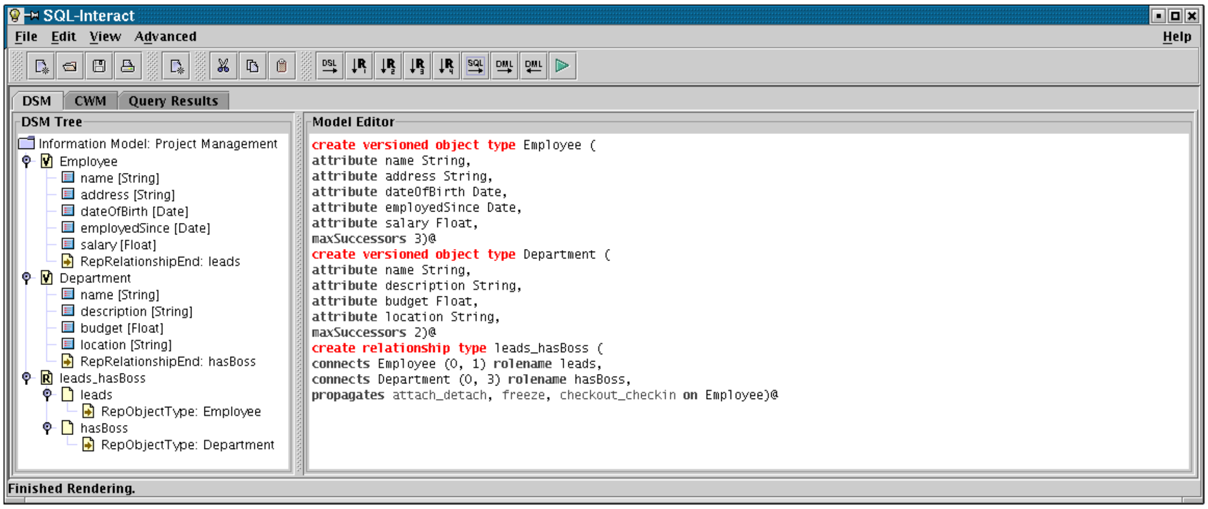
Abbildung 4.4 Zusammenhänge zwischen DSM- und CWM-Ebene



Auf der DSM-Ebene muss der Produktlinienhersteller ein MOF basiertes DS-Metamodel definieren, um die Konstrukte zu beschreiben, die in einem DS-Modell oder DS-DB-Schema auftreten können. Dieses Metamodel ist ein Ergebnis der Domänenanalyse und lässt sich durch das in Abschnitt 4.2.2 vorgestellte Meta-Metamodell beschreiben. Auf Grundlage dieses Metamodells kann die Konfiguration eines Mitgliedes der entsprechenden Produktlinie erfolgen. Zu Konfiguration des Produktlinienmitglieds existieren zwei äquivalente Möglichkeiten. Zum einen kann ein DS-Modell durch Instanziierung von DS-Metamodellkonstrukten erstellt werden. Dabei wird das DS-Modell in einer baumartigen Struktur repräsentiert und *DS-Baum* genannt. Alternativ kann ein DS-Datenbankschema in textueller Form mit Hilfe eines DS-Editors eingegeben werden. Zwischen dem DS-Baum und dem DS-Datenbankschema besteht eine eins-zu-eins Abbildung, so dass Änderungen am DS-Baum automatisch im DS-DB-Schema im Editor nachgeführt werden können und umgekehrt. Um diese Abbildung zu ermöglichen, muss der Produktlinienhersteller so genannte *DS-Rendering* Regeln bereitstellen, durch die definiert wird, wie ein Konstrukt im DS-Model in einem DS-Datenbankschema dargestellt werden muss. Dabei kann auch mehr als ein Regelsatz verwendet werden, um unterschiedliche textuelle Darstellungsformen zu ermöglichen. Die Darstellungen können sich sowohl in der Syntax des DS-Datenbankschemas, als auch in der Sicht auf das DS-Modell unterscheiden. Abbildung 4.5 zeigt die Konfigurationsoberfläche in der domänenspezifischen Ebene. Auf der linken Seite befindet sich ein DS-Datenbankschema im Model-Editor. Das dazu äquivalente DS-Modell in Form einer Baumstruktur ist im linken Teil der Abbildung zu sehen. Die Objektgraphen, die bei der Instanziierung und der Verknüpfung von Modellklassen entstehen, sind normalerweise nicht zyklenfrei. Um die nicht zyklenfreien Objektgraphen im DS-Baum darstellen zu können, müssen einige Assoziationen als LinkNodes dargestellt werden. Mit Hilfe dieser LinkNodes ist eine Navigation entlang der Assoziationen zwischen den Modellklassen möglich. In Abbildung 4.5

werden mit Hilfe dieser LinkNodes beispielsweise die Assoziationen zwischen der Beziehung *leads_hasBoss* und den dadurch verbundenen Objekttypen *Employee* und *Department* realisiert.

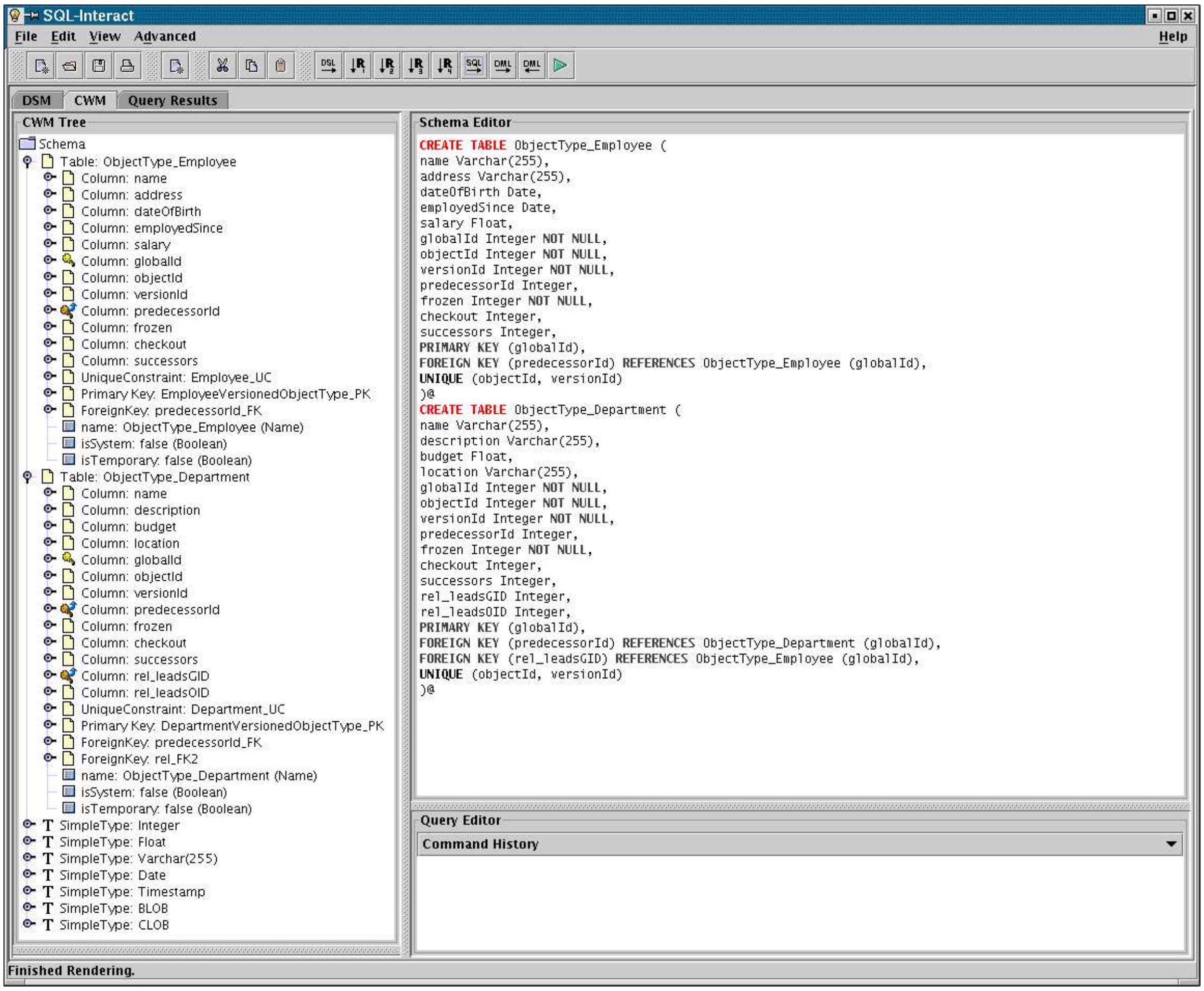
Abbildung 4.5 DSM-Ebene in SQLInteract



Neben den Tabellendefinitionen werden noch Komponenten zur Ausführung, der im DS-DB-Schema spezifizierten Anwendungslogik, der datenbankintensiven Anwendung benötigt. Dafür stehen zwei unterschiedliche Möglichkeiten zur Verfügung. Die erste Möglichkeit ist die Nutzung von Fähigkeiten einer neuen Generation von objektrelationalen Datenbanksystemen, die auf dem SQL:1999 und SQL:2003 Standard beruhen. Mit ihnen kann Anwendungslogik im Datenbanksystem ausgeführt und objektrelationale Erweiterungen auf einem relationalen Modell genutzt werden. Zur diesen Erweiterungen gehören *user defined routines*, *user-defined types*, *triggers*, *constraints* und *assertions*. Durch die Nutzung dieser Erweiterungen wird eine vollständige Realisierung einer datenbankintensiven Anwendung in einem Datenbanksystem ermöglicht. Die zweite Möglichkeit besteht darin, die benötigte Anwendungslogik in einem domänenspezifischen Datenbanktreiber zu kapseln. Diese Möglichkeit wird in Abschnitt 4.2.4 genauer beschrieben.

Die CWM-Ebene ist analog zur DSM-Ebene aufgebaut. Das Common Warehouse Metamodel ist ein Metamodell, dass zum einfachen Austausch von Metadaten zwischen Warehouse Tools, Warehouse Platforms und Warehouse Metadata Repositories geeignet ist. Durch das CWM-Package *Relational* werden Modellierungskonstrukte, die in Datenbankschemata von objektrelationalen Datenbanken verwendet werden, zur Verfügung gestellt. Aus diesem Grund wird zur Präsentation der im DS-Modell beschriebenen Eigenschaften der datenbankintensiven Anwendung ein CWM-Modell genutzt. Durch eine Reduktion, die in diesem Fall einer Modell-zu-Modell Transformation entspricht, wird ein DS-Modell in ein CWM-Modell transformiert. Der Begriff der Reduktion stammt aus der intentionalen Programmierung und beschreibt dort die Abbildung der modellierten Semantik auf eine niedrigere Abstraktionsstufe. Die hier beschriebene Reduktion ist ein Generatorlauf im Sinne der generativen Programmierung. Um diese Reduktion durchführen zu können, müssen vom Produktlinienhersteller domänenspezifische Reduktionsregeln bereitgestellt werden, die eine Abbildung zwischen den beiden Modellen definieren. Analog zu den DSL-Rendering-Regeln existieren CWM-Rendering-Regeln, die ein entsprechendes CWM-Modell, das durch einen CWM-Baum repräsentiert wird, in eine textuelle Repräsentation des Datenbankschemas in SQL überführen. Die CWM-Ebene von SQLInteract ist in Abbildung 4.6 gezeigt. Das CWM-Modell ist das Ergebnis einer Reduktion des in Abbildung 4.5 gezeigten DSM-Modells. Die Regeln für die Durchführung der Reduktion werden in Kapitel 5.3 genauer vorgestellt.

Abbildung 4.6 CWM-Ebene in SQLInteract



Falls es erforderlich ist, kann einem Kunde das entsprechende Schema mit zusätzlicher Funktionalität erweitern, die nicht in der domänenspezifischen Sprache spezifiziert werden konnte. Dies kann sowohl durch eine Manipulation des CWM-Baums oder des Datenbankschemas erreicht werden [KH03a]. Dieser Prozess entspricht auch der Beschreibung von partiellen *round-trip engineering* bei der modellgetriebenen Entwicklung, bei der es erlaubt ist, die generierten Fragmente durch Implementierungen zu erweitern, die auf der Spezifikationsebene nicht beschrieben werden konnten [Fra03].

Die Architektur von SQLInteract erlaubt einen problemlosen Austausch des domänenspezifischen Moduls, das für die Darstellung und Bearbeitung der modellierten Konstrukte auf der DSM-Ebene sowie für die Reduktion auf der CWM-Ebene zuständig ist [Hus03].

4.2.4 Domänenspezifische Datenbanktreiber

Im vorherigen Abschnitt wurde beschrieben, wie aus einem domänenspezifischen Modell ein entsprechendes Datenbankschema generiert werden kann. Die benötigte domänenspezifische Anwendungslogik kann entweder durch Nutzung der objektrelationalen Erweiterungen, die auf dem SQL:1999 und SQL:2003 Standard beruhen, realisiert werden. Alternativ kann die Anwendungslogik auch in einem domänenspezifischen Datenbanktreiber gekapselt werden. Dieser Datenbanktreiber kann dann von verschiedenen Anwendungsprogrammen genutzt werden, um auf die Daten zuzugreifen und sie mit Hilfe einer domänenspezifischen Sprache manipulieren zu können. Im Rahmen dieser Arbeit wurden verschiedene domänenspezifische Datenbanktreiber für Versionierungssysteme entwickelt. Diese Datenbanktreiber wurden in Java [Sun04b] implementiert und stellen die komplette Java Database Connectivity (JDBC [Sun04c]) Anwendungsprogrammierschnittstelle zur Verfügung. Durch Bereitstellung derartiger Schnittstellen können domänenspezifische Datenbanktreiber durch Anwendungsentwickler intuitiv genutzt werden.

Die Anfragen, die von einem domänenspezifischen Datenbanktreiber verarbeitet werden, müssen nicht dem SQL-Standard entsprechen. Sie können Spracherweiterungen enthalten, die speziell für eine Anwendungsdomäne entwickelt wurden. Alternativ zu möglichen Spracherweiterungen, kann auch eine vollständig domänenspezifische Datenmanipulationssprache (Data Manipulation Language, DML) entwickelt werden. Diese domänenspezifischen Datenmanipulationssprachen weisen jedoch häufig Ähnlichkeiten mit der herkömmlichen reinen SQL-Datenmanipulationssprache auf. Eine domänenspezifische Datenmanipulationssprache für Versionierungssysteme wird in Abschnitt 5.2.2 vorgestellt.

Mit einem domänenspezifischen Datenbanktreiber kann noch nicht direkt auf ein Datenbanksystem zugegriffen werden. Wie Abbildung 4.7 zeigt, ist ein domänenspezifischer Datenbanktreiber zwischen der Anwendung und einem herkömmlichen Datenbanktreiber, der vom Hersteller des Datenbanksystems zur Verfügung gestellt wird, angeordnet. Er übernimmt die Rolle eines Übersetzers, der eine domänenspezifischen Datenmanipulationsanweisung in eine oder auch mehrere Anweisungen im SQL-Dialekt des verwendeten Datenbanksystems überführt. Aus diesem Grund kann durch die Verwendung eines domänenspezifischen Datenbanktreibers keine Performanzsteigerung erreicht werden. In Kapitel 6.2 werden unter anderem die Kosten bestimmt, die durch die Nutzung eines domänenspezifischen Datenbanktreiber zusätzlich entstehen.

Abbildung 4.7 Kommunikation mittels domänenspezifischen Datenbanktreiber

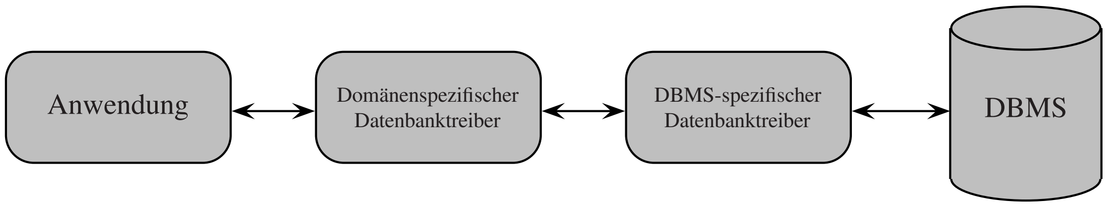
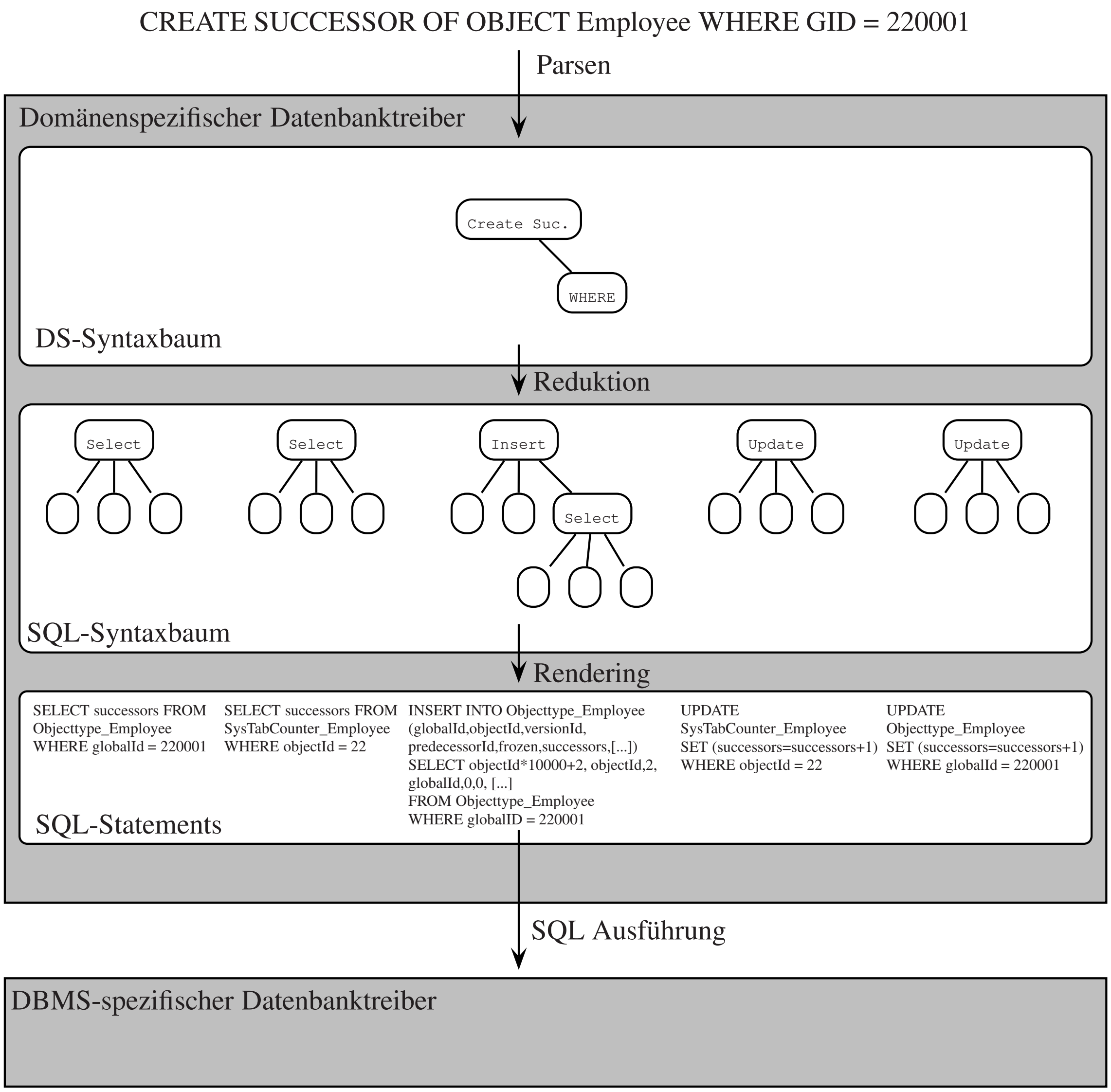


Abbildung 4.8 veranschaulicht den Aufbau und die Arbeitsabläufe eines domänenspezifischen Datenbanktreibers. Genau wie bei einem herkömmlichen Datenbanktreiber liegt die Anforderung in Form einer Zeichenkette vor. Diese Zeichenkette wird einem Parser übergeben, der diese in einen abstrakten Syntaxbaum, der aus Java-Objekten besteht, überführt. Auf dem Wurzelelement dieses Objektbaums wird dann eine Reduktionsmethode aufgerufen, um die Anweisung,

die der Baum repräsentiert, auszuführen. Die Erfahrungen, die am Beispiel einer domänenspezifischen Datenmanipulationssprache für Versionierungssysteme gesammelt wurden, haben gezeigt, dass sich nicht jede domänenspezifische Anweisung mit einem SQL-Befehl realisieren lässt. Häufig werden mehrere SQL-Befehle benötigt, die voneinander abhänig sind, so dass es auch nicht immer möglich ist, diese Anweisungen parallel ausführen zu lassen. An einigen Stellen, ist eine parallele Ausführung der SQL-Anweisungen möglich. Im Rahmen dieser Arbeit sind jedoch alle Anweisungen sequentiell ausgeführt worden. Bei der Ausführung der Reduktionsmethode werden SQL-Syntaxbäume erstellt, die einer SQL-Anweisung entsprechen. Durch den Aufruf einer Rendering-Methode auf dem Wurzelknoten des SQL-Syntaxbaums erhält man eine Zeichenkette, mit der dann der Aufruf bei einem herkömmlichen Datenbanktreiber erfolgt. Durch den Zwischenschritt über einen SQL-Syntaxbaum zur Erzeugung der SQL-Anweisungen kann von unterschiedlichen SQL-Dialekten abstrahiert werden. Durch die Tatsache, dass es häufig Abhängigkeiten zwischen den SQL-Anweisungen, die innerhalb einer Reduktionsmethode erstellt werden, gibt, müssen auch die Ergebnisse, die nach der Ausführung von Anweisungen vom herkömmlichen Datenbanktreiber zurückgeliefert werden, ausgewertet werden. Daraus folgt, dass eine Reduktionsmethode neben dem Code, der die SQL-Syntaxbäume erstellt, auch Code enthält, der die einzelnen Anweisungen verknüpft, auswertet und gegebenenfalls domänenspezifische Fehlermeldungen generiert. Während der Reduktion muss auf das DS-Modell zugegriffen werden, da die Speicherung der Informationen im Versionierungssystem von den im Imformationsmodell festgelegten Eigenschaften abhängt.

Der im vorherigen Absatz vorgestellte Ablauf der Reduktion wird in Listing 4.1 am Beispiel der Reduktionsmethode der Operation *Freeze Object* gezeigt. Zu Beginn der Reduktionsmethode wird das Informationsmodell benötigt, um Informationen über den Objekttyp zu erhalten. Falls der Objekttypname nicht bekannt ist, wird eine entsprechende domänenspezifische Fehlermeldung generiert. Anschließend kann leicht erkannt werden, wie eine SQL-Anweisung in Form eines Baumes aus Java-Objekten erzeugt wird. Durch den Aufruf der Rendering-Methode auf dem Objektbaum erhält man die auszuführende Anweisung in Form einer Zeichenkette. Diese Anweisung wird dann an den datenbankspezifischen JDBC-Treiber übergeben und ausgeführt. Damit ist die Operation *Freeze Object* auf dem Objekt, auf dem sie aufgerufen wurde, erfolgreich ausgeführt. Wie im Abschnitt 3.6.2 beschrieben, kann eine Operation über Beziehungen an andere Objekte propagiert werden. Eine derartige Propagierung wird im Informationsmodell festgelegt. Aus diesem Grund muss an dieser Stelle erneut auf das Informationsmodell zugegriffen werden, um festzustellen, ob und an welche Objekttypen die Operation *Freeze Object* propagiert werden muss. Falls die Operation an andere Objekte propagiert werden muss, wird analog wie beim Parsen einer DS-DML-Anweisung ein abstrakter Syntaxbaum erzeugt und anschließend die Reduktionsmethode auf dem Wurzelknoten dieses Baumes aufgerufen. Auf diese Weise kann die Operation *Freeze Object* über mehrere Beziehungen hinweg propagiert werden.

Abbildung 4.8 Aufbau eines domänenspezifischen Datenbanktreibers



Listing 4.1 Reduktionsmethode der Operation *Freeze Object*

```
public void reduce(Statement stmt, InformationModel iModel) throws SQLException {

  // Informationsbeschaffung aus dem Informationsmodell
  String objectTypeName = objectSelectClause.getObjectTypeName();
  VSVersionedObjectType vsObjectType = (VSVersionedObjectType)iModel.getObjectType(objectTypeName);
  String globalId = objectSelectClause.getGid();

  // Erzeugung des SQL-Objektbaums
  WhereClause whereClause1 = new WhereClause("globalId="+globalId);
  SelectStatement selStmt1 = new SelectStatement(new DisplayedColumnClause("checkout"),
                             new FromClause("ObjectType_"+objectTypeName),whereClause1);
  // Rendering
  String sql1 = selStmt1.render();
  // Ausfuehrung
  ResultSet rs = stmt.executeQuery(sql1);

  if (rs.next()){
    int checkout = rs.getInt(1);
    if (checkout == 0) {
      // Erzeugung des SQL-Objektbaums
      UpdateClause upClause = new UpdateClause("frozen","1");
      UpdateStatement upStmt1 = new UpdateStatement(
                                 new ReferencedTableClause("ObjectType_"+objectTypeName),
                                 upClause,new WhereClause("globalId="+globalId));
      // Rendering
      String sql3 = upStmt1.render();
      // Ausfuehrung
      stmt.executeUpdate(sql3);

      // Auf Propagierung von Operationen pruefen
      Vector ends = vsObjectType.getVsRelationshipEnds();

      for (Iterator iter = ends.iterator(); iter.hasNext();) {
        VSRelationshipEnd relEndA = (VSRelationshipEnd)iter.next();
        VSRelationshipEnd relEndB = relEndA.getRelationshipEnd();

        if (relEndB.getPropFreeze() == 1) {
          // Erzeugung eins DS-Objektbaums
          Statement stmt2 = stmt.getConnection().createStatement();
          GetRelatedObjects relObj = new GetRelatedObjects(vsObjectType, relEndA,
                                     objectSelectClause.getGid());
          // Reduktion
          ResultSet rs2 = relObj.reduce(stmt2);

          while (rs2.next()) {
            // Erzeugung eines DS-Objektbaums
            FreezeObjectCommand command1 = new FreezeObjectCommand(
                                          new DSObjectSelectClause(relEndB.getObjectType().getName(),
                                          Integer.toString(rs2.getInt(1))));
            // Reduktion
            command1.reduce(stmt, iModel);
          }
          rs2.close();
        }
      }
    } else {
      throw new CanNotFreezeCheckedOutObjectException();
    }
    rs.close();
  } else {
    rs.close();
    throw new SQLException();
  }
}
```

Kapitel 5 Anwendung domänenspezifischer Sprachen bei Versionierungssystemen

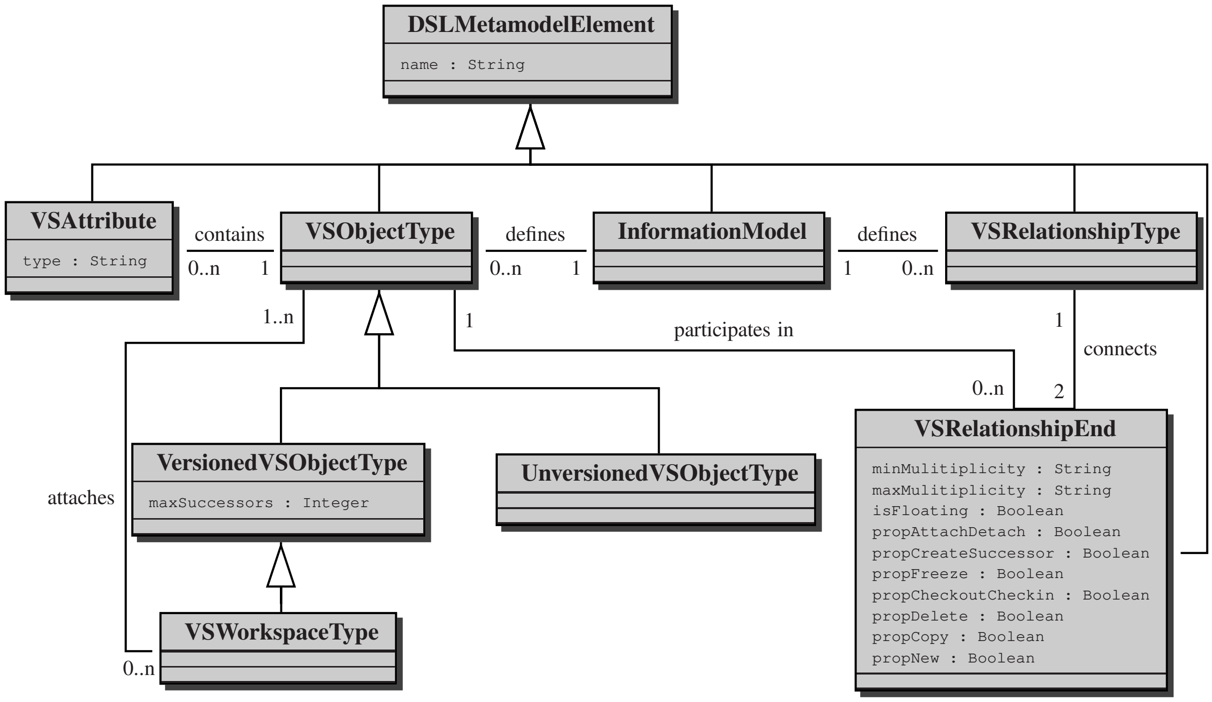
Die im vorherigen Kapitel vorgestellten Konzepte zur Generierung von Softwareprodukten einer Produktlinie mit Hilfe eines Generators werden in diesem Kapitel am Beispiel von Versionierungssystemen betrachtet. Dazu wird zunächst das Metamodell für die Konfiguration von Versionierungssystemen vorgestellt. Anschließend werden domänenspezifische Sprachen, sowohl für die Konfiguration als auch für die Nutzung von Versionierungssystemen eingeführt. Danach werden vier unterschiedliche Varianten für die Reduktion eines produktspezifischen Modells auf ein Datenbankschema erläutert und die Auswirkungen auf die Reduktion von domänenspezifischen Datenmanipulationsanweisungen betrachtet.

5.1 Metamodell für Versionierungssysteme

In Abschnitt 4.2.2 wurde das Meta-Metamodell für die Beschreibung von Metamodellen im System SQL-Interact vorgestellt. Die wichtigsten Elemente des Meta-Metamodells sind *ObjectType*, *Generalization*, *RelationshipType* sowie *RelationshipEnd*. Alle Objekte im Metamodell sind Instanzen von *ObjectType*. Mittels *Generalization* können die Objekte in einer Subtyp- oder Supertypbeziehung zueinander stehen. Objekte können durch Instanzen von *RelationshipType* und den beiden dazugehörigen *RelationshipEnds* verbunden werden. Diese Beziehungsenden können mit minimalen und maximalen Kardinalitäten beschrieben werden. Abbildung 5.1 zeigt ein Metamodell für Versionierungssysteme, das durch Instanziierung der Klassen des Meta-Metamodells entstanden ist. Durch dieses Metamodell können Informationsmodelle für konkrete Anwendungsszenarien modelliert werden. Das *InformationModel* ist Wurzelelement des domänenspezifischen Modells. Das Informationsmodell kann Instanzen von Objekttypen (*VSObjectType*) und Beziehungstypen (*VSRelationshipTyps*) enthalten. Instanzen von *VSObjectType* können beliebig viele Instanzen von *VSAttribute* zugeordnet werden. Mit Hilfe von Beziehungstypen können binäre Beziehungen zwischen Objekttypen realisiert werden. Aus diesem Grund enthält jede Instanz von *VSRelationshipType* zwei Instanzen von *VSRelationshipEnd*. Diese Beziehungsenden enthalten Attribute zur Festlegung von gleitenden Beziehungsenden oder Propagierung von Operationen. Im Metamodell für Versionierungssysteme werden aus dem *VSObjectType* die Objekttypen *VersionedVSObjectType* und *UnversionedObjectType* abgeleitet. Der Objekttyp für Arbeitskontexte (*VSWorkspaceType*) ist eine Spezialisierung des

VersionedVSObjectType. Instanzen vom *VSWorkspaceType* können mit Hilfe einer Attachment-Beziehung Referenzen auf andere Objekttypen enthalten.

Abbildung 5.1 Domänenspezifisches Metamodell für Versionierungssysteme



5.2 Domänenspezifische Sprachen für Versionierungssysteme

Mit Hilfe des vorgestellten Metamodells für Versionierungssysteme können produktspezifische Informationsmodelle erzeugt werden, aus denen durch einen Generator die Mitglieder der Produktlinie für Versionierungssysteme generiert werden. Dabei kommen auch domänenspezifische Sprachen zum Einsatz. Bei den domänenspezifischen Sprachen muss zwischen den Sprachen zur Konfiguration und denen zur domänenspezifischen Anfrage oder Datenmanipulation unterschieden werden.

5.2.1 Konfigurationssprache

Die Konfiguration eines Produktlinienmitglieds kann entweder durch einen DS-Baum oder durch ein DS-DB-Schema vorgenommen werden. Für die Spezifikation eines Versionierungssystems mittels DS-DB-Schema wird eine Sprache benötigt, die Konstrukte enthält, um die verschiedenen Objekttypen und Beziehungen zwischen den Objekten zu beschreiben.

Beispiel 5.1 zeigt einen Ausschnitt aus einem DS-DB-Schema, welches in Abbildung 3.2 dargestellte Informationsmodell definiert. Bei der Spezifikation von Objekttypen kann definiert werden, ob es sich um einen unversionierten Objekttyp, versionierten Objekttpy oder auch um einen Objekttyp für Arbeitskontexte handelt und welche benutzerdefinierten Attribute enthalten sind. Bei versionierten Objekttypen kann zudem die Anzahl der direkten Nachfolger einer Objektversion festgelegt werden, um die Breite des Versionsbaums begrenzen zu können. Bei der Definition von Beziehungen werden die beiden Beziehungsenden mit Namen der verbundenen Objekttypen, Rollennamen sowie die minimalen und maximalen Kardinalitäten angegeben. Des weiteren wird beschrieben welche Operationen an ein Beziehungsende propagiert werden können. Falls es sich um eine Beziehung mit mindestens einem gleitenden Beziehungsende handelt, wird dies ebenfalls durch domänenspezifische Sprachkonstrukte beschrieben. Das vollständige DS-Datenbankschema für das untersuchte Versionierungssystem befindet sich in Anhang B. Anhang A.1 enthält die vollständige Spezifikation der Konfigurationssprache für das Versionierungssystem.

Beispiel 5.1 Ausschnitt aus einem DS-DB-Schema

```
create versioned object type Employee (
attribute name String,
attribute address String,
attribute dateOfBirth Date,
attribute salary Float,
attribute employedSince Date,
maxSuccessors 3)

create versioned object type Department (
attribute name String,
attribute description String,
attribute budget Float,
attribute location String,
maxSuccessors 2)

create relationship type leads_hasBoss (
connects Employee (0, 1) rolename leads,
connects Department (0, 3) rolename hasBoss,
propagates attach_detach, freeze, checkout_checkin on Employee)
```

5.2.2 Anfrage- und Manipulationssprache

Mit Hilfe einer domänenspezifischen Anfrage- und Datenmanipulationssprache kann einem Anwender eine wesentlich komfortablere Schnittstelle für den Umgang mit den im Versionierungssystem gespeicherten Daten bereitgestellt werden. Der Anwender muss sich weder mit dem Datenbankschema auf der SQL-DDL Ebene auseinandersetzen, noch muss er wissen, welche Menge von Anweisungen ausgeführt werden muss, um beispielsweise ein Nachfolgerobjekt zu erzeugen. Er muss sich auch keinerlei Gedanken über die Propagierung von Operationen an andere Objekte machen. Alle diese Aufgaben werden von einem domänenspezifischen Datenbanktreiber erledigt. Dieser Datenbanktreiber ist speziell auf die domänenspezifische Anfrage- und Datenmanipulationssprache für Versionierungssysteme zugeschnitten. Beispiel 5.2 enthält einige Beispiele für domänenspezifische Anweisungen für Versionierungssysteme. Die vollständige Spezifikation der Anfrage- und Datenmanipulationssprache für Versionierungssysteme befindet sich im Anhang A.2 in der erweiterten Backus-Naur-Form (EBNF).

Anhand zweier Beispiele aus Beispiel 5.2 wird klar, welche Vorteile die Nutzung einer domänenspezifischen Manipulationssprache mit sich bringt.

- `CREATE SUCCESSOR OF OBJECT Offer WHERE GID = 770001`
  Diese Anweisung bewirkt, dass eine Nachfolgerversion eines Objektes angelegt wird. Sie wird vom domänenspezifischen Datenbanktreiber in eine Reihe von SQL-Anweisungen reduziert. Auf Grund der Möglichkeit, dass Operationen an andere Objekte propagiert werden können, ist Anzahl der SQL-Anweisungen abhängig vom aktuellen Datenbestand des Versionierungssystems. Das Ergebnis der Reduktion ohne Propagierung an andere

Beispiel 5.2 Domänenspezifische Datenmanipulationsbefehle in DS-DML

```
CREATE SUCCESSOR OF OBJECT Offer WHERE GID = 770001

CREATE NEW RELATIONSHIP leads_hasBoss WHERE leads.GID = 980002 AND hasBoss.GID = 1520001

PIN Costs WHERE GID = 1310001 IN CVC Task WHERE GID = 1300001 ROLENAME ratedCosts

SELECT Offer.* FROM Department -- consistsOf --> Employee -- isAuthor --> Offer
WHERE Department.globalId = 430001

SELECT Employee.* FROM Department -- consistsOf --> Employee
USE WORKSPACE HumanResources WHERE GID = 2330001 AND Department WHERE OID = 8
```

Objekte sieht folgendermaßen aus:

```
SELECT successors FROM Objecttype_Offer
WHERE globalId = 770001

SELECT successors FROM SysTabCounter_Offer
WHERE objectId = 77

INSERT INTO Objecttype_Offer
(globalId,objectId,versionId,predecessorId,frozen,successors,[...])
SELECT objectId*10000+2, objectId,2,globalId,0,0, [...]
FROM Objecttype_Offer
WHERE globalId = 770001

UPDATE SysTabCounter_Offer
SET (successors=successors+1)
WHERE objectId = 77

UPDATE Objecttype_Offer
SET (Successors=successors+1)
WHERE globalId = 770001
```

- `SELECT Offer.* FROM Department – consistsOf –> Employee – isAuthor –> Offer WHERE Department.globalId = 430001`
  Diese Anfrage beinhaltet eine Navigation über verschiedene Beziehungen. Für den Anwender spielen dabei die unterschiedlichen Beziehungsarten und deren unterschiedliche Verwaltung keine Rolle. Das Ergebnis der Reduktion bei Verwendung des in Abschnitt 5.3.3 vorgestellten Datenbankschema sieht folgendermaßen aussehen:

```
SELECT ObjectType_Offer.* FROM ObjectType_Offer
WHERE ObjectType_Offer.globalId in (
   SELECT writtenByGID AS globalID
   FROM Rel_CVC_isAuthor_to_writtenBy
   WHERE isAuthorGID  in (
      SELECT Rel_worksIn_consistsOf.worksInGID
      FROM ObjectType_Department , Rel_worksIn_consistsOf
      WHERE ObjectType_Department.globalId=430001
      AND Rel_worksIn_consistsOf.consistsOfGID = ObjectType_Department.globalId )
   AND pinned_writtenBy = 1 OR writtenByGID IN (
      SELECT max(writtenByGID) AS globalId
      FROM Rel_CVC_isAuthor_to_writtenBy
      WHERE isAuthorGID  in (
         SELECT Rel_worksIn_consistsOf.worksInGID
         FROM ObjectType_Department , Rel_worksIn_consistsOf
         WHERE ObjectType_Department.globalId=430001
         AND Rel_worksIn_consistsOf.consistsOfGID = ObjectType_Department.globalId )
   GROUP BY writtenByOID, isAuthorGID HAVING MAX(pinned_writtenBy) = 0 )
   )
```

5.3 Reduktion von DS-DDL

Im Abschnitt 4.2.3 wurden die allgemeinen Abläufe beim domänenspezifischen Schemaentwurf beschrieben. In diesem Abschnitt werden vier verschiedene Varianten für den Schemaentwurf am Beispiel von Versionierungssystemen vorgestellt. Bei der ersten Variante handelt es sich um eine sehr einfache Reduktion. Im Rahmen dieser Arbeit wurden für alle vier Varianten domänenspezifische JDBC-Treiber implementiert, um die Unterschiede in der Reduktion sowohl von DDL als auch bei DML-Befehlen zu untersuchen. Die Ergebnisse dieser Untersuchungen werden in Kapitel 6 vorgestellt.

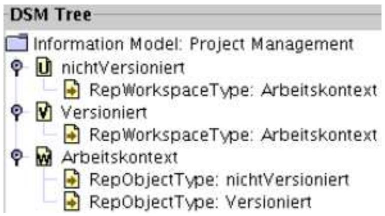
Bei der Reduktion muss zwischen Objekttypen und Beziehungstypen unterschieden werden. Die vier untersuchten Varianten unterscheiden sich nicht in der Reduktion von Objekttypen, sondern nur bei Beziehungstypen. Bei der Reduktion von Objekttypen wären auch unterschiedliche Reduktionen denkbar. So könnten beispielsweise bei versionierten Objekttypen die Versionsinformationen getrennt von den Nutzdaten verwaltet werden.

Im Metamodell für Versionierungssysteme sind die drei unterschiedlichen Objekttypen *UnversionedVSObjectType*, *VersionedVSObjectType* und *VSWorkspaceType* enthalten, die bei der Reduktion unterschiedlich behandelt werden. Für jede Instanz eines Objekttyps wird bei der Reduktion eine eigene Tabelle angelegt, die sowohl Spalten für Nutzdaten als auch für objekttypspezifische Verwaltungsdaten enthält. Bei der Reduktion von einem *UnversionedVSObjectType* wird der entsprechenden Tabelle eine Spalte zur Speicherung der *GlobalId* hinzugefügt. Tabellen, die zur Speicherung von *VersionedVSObjectTypes* genutzt werden, enthalten neben der *GlobalId* noch Spalten *ObjectId*, *VersionId*, *PredecessorId*, *successors*, *checkout* und *frozen*. Instanzen von *VSWorkspaceTypes* werden genau wie *VersionedVSObjectTypes* behandelt. Für die Objekttypinstanzen, die über eine Attachment-Beziehung mit einer *VSWorkspaceType*-Instanz assoziiert sind, wird jeweils eine Tabelle zur Speicherung von Referenzen auf Objekte, die im Arbeitskontext enthalten sind, benötigt. Abbildung 5.2 zeigt die Reduktion eines DS-DDL-Baums, der je eine Instanz eines *UnversionedVSObjectType*, *VersionedVSObjectType* und *VSWorkspaceType* enthält. Die Instanzen des *UnversionedVSObjectTypes* und des *VersionedVSObjectTypes* sind über eine Attachment-Beziehung mit der Instanz des *VSWorkspaceType* verbunden. Das generierte SQL-Schema veranschaulicht die, von den verschiedenen Objekttypen benötigten, Verwaltungsattribute.

Neben den in der Abbildung dargestellten Tabellen werden bei der Reduktion von *VersionedVSObjectTypes* und *VSWorkspaceTypes* noch Tabellen benötigt, um die Anzahl der Versionen je Objektinstanz zu materialisieren. Die vollständigen SQL-Schemata, die sich bei der Reduktion des in Abbildung 3.5 vorgestellten Informationsmodell für ein Versionierungssystem ergeben, befinden sich im Anhang C. Dieser Anhang enthält auch die Definitionen der Systemtabellen, in denen alle Informationen des Informationsmodell gespeichert werden. Diese Tabellen werden bei der Initialisierung des domänenspezifischen Datenbanktreibers benötigt und ausgelesen. Darauf wird im Abschnitt 5.4 genauer eingegangen.

Bei der Reduktion von Beziehungstypen muss die Art der Beziehung berücksichtigt werden. Beziehungsenden an versionierbaren Objekten können, wie in Abschnitt 3.5.1 beschrieben, gleitend sein. Dadurch ergeben sich drei Arten von Beziehungen, die sich in der Anzahl der gleitenden Beziehungsenden unterscheiden. Abbildung 5.3 zeigt Beispiele für Beziehungen mit keinem, einem oder zwei gleitenden Beziehungsenden. Die Kreise an den Beziehungsenden kennzeichnen ein gleitendes Beziehungsende. Ausgefüllte Kreise kennzeichnen die vorausgewählte Version innerhalb der Kandidatenmenge. Anhand von Abbildung 5.3 werden in den

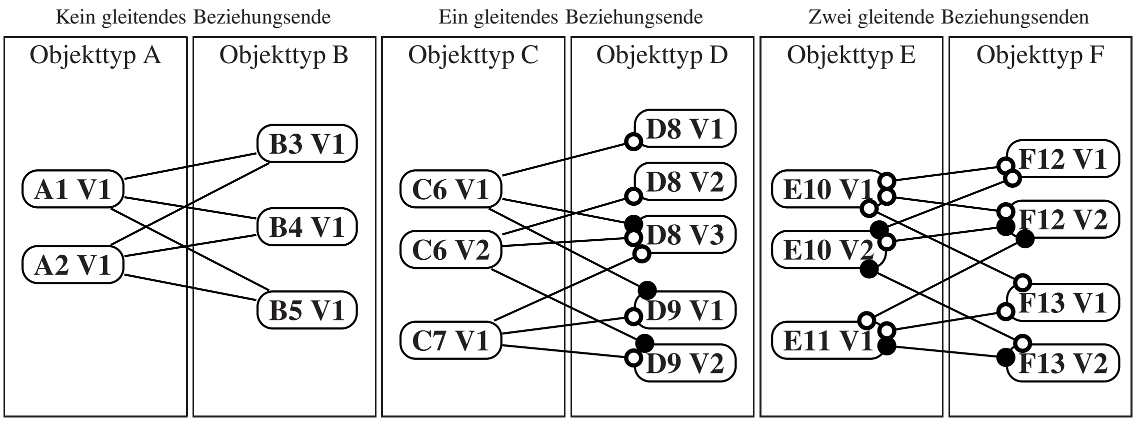
Abbildung 5.2 Reduktion der unterschiedlichen Objekttypen




```
CREATE TABLE ObjectType_nichtVersioniert (
globalId Integer NOT NULL,
PRIMARY KEY (globalId))

CREATE TABLE ObjectType_Versioniert (
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Versioniert (globalId),
UNIQUE (objectId, versionId))

CREATE TABLE ObjectType_Arbeitskontext (
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Arbeitskontext (globalId),
UNIQUE (objectId, versionId))

CREATE TABLE attached_nichtVersioniert_Arbeitskontext (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_nichtVersioniert (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_Arbeitskontext (globalId),
UNIQUE (wsGlobalId, otObjectId))

CREATE TABLE attached_Versioniert_Arbeitskontext (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
otObjectId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_Versioniert (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_Arbeitskontext (globalId),
UNIQUE (wsGlobalId, otObjectId))
```

nachfolgenden Abschnitten die Unterschiede in den vier Reduktionsvarianten verdeutlicht.

Abbildung 5.3 Drei Arten von Beziehungen



5.3.1 Variante 1

Bei der ersten Variante existiert nur eine generische Tabelle für die Verwaltung von Beziehungen zwischen den Objekten. Um eine bidirektionale Beziehung zu speichern, werden zwei

Einträge benötigt, da die Einträge in der generischen Tabelle jeweils nur eine Beziehung von der Quelle zum Ziel darstellen. Diese Art der Speicherung von Beziehungen kann im Vergleich zu den später vorgestellten Varianten vom Datenbanksystem nicht durch Fremdschlüssel unterstützt werden.

Abbildung 5.4 verdeutlicht die Beziehungsverwaltung anhand des Beispiels aus Abbildung 5.3. Zur größeren Übersichtlichkeit enthält die Tabelle *Relationships* zwei Leerzeilen, die die Einträge der unterschiedlichen Beziehungsarten voneinander trennen. Die Tabelle *Relationships* enthält neben der *GlobalId*, *ObjectId* und *VersionId* vom Quell- und Zielobjekt auch den Rollennamen am Quellobjekt sowie Informationen über Pinningeinstellungen.

Zum besseren Verständnis der Einträge in der Tabelle *Relationships* sei darauf hingewiesen, dass bei Beziehungstypen, die über genau ein gleitendes Beziehungsende verfügen, nicht alle Beziehungen bidirektional sind. Falls eine Objektversion, auf der Seite des gleitenden Beziehungsendes, sich in verschiedenen Kandidatenmengen unterschiedlicher Objektversionen mit gleicher *ObjectId* befindet, so gibt es genau eine bidirektionale Beziehung. Die anderen Beziehungen sind unidirektional in Richtung des gleitenden Beziehungsende.

Abbildung 5.4 Reduktionsvariante 1: Generische Tabelle zur Verwaltung von Beziehungsinformation

ObjektTyp A

GID	OID	VID	...
101	1	1	
201	2	1	

ObjektTyp C

GID	OID	VID	...
601	6	1	
602	6	2	
701	7	1	

ObjektTyp E

GID	OID	VID	...
1001	10	1	
1002	10	2	
1101	11	1	

Relationships

sourceGID	sourceOID	sourceVID	source_rolename	destGID	destOID	destVID	pinned
101	1	1	a	301	3	1	0
301	3	1	b	101	1	1	0
101	1	1	a	401	4	1	0
401	4	1	b	101	1	1	0
101	1	1	a	501	5	1	0
501	5	1	b	101	1	1	0
201	2	1	a	301	3	1	0
301	3	1	b	201	2	1	0
201	2	1	a	401	4	1	0
401	4	1	b	201	2	1	0
201	2	1	a	501	5	1	0
501	5	1	b	201	2	1	0
501	5	1	b	201	2	1	0
601	6	1	c	801	8	1	0
801	8	1	d	601	6	1	0
601	6	1	c	803	8	3	1
601	6	1	c	901	9	1	1
901	9	1	d	601	6	1	0
602	6	2	c	802	8	2	0
602	6	2	c	803	8	3	0
803	8	3	d	602	6	2	0
602	6	2	c	902	9	2	1
902	9	2	d	602	6	2	0
701	7	1	c	803	8	3	0
803	8	3	d	701	7	1	0
701	7	1	c	901	9	1	0
701	7	1	c	902	9	2	0
902	9	2	d	701	7	1	0
1001	10	1	e	1201	12	1	0
1201	12	1	f	1001	10	1	0
1001	10	1	e	1202	12	2	0
1202	12	2	f	1001	10	1	0
1001	10	1	e	1301	13	1	0
1301	13	1	f	1001	10	1	0
1002	10	2	e	1201	12	1	0
1201	12	1	f	1002	10	2	1
1002	10	2	e	1202	12	2	1
1202	12	2	f	1002	10	2	0
1002	10	2	e	1302	13	2	0
1302	13	2	f	1002	10	2	1
1101	11	1	e	1202	12	2	1
1202	12	2	f	1101	11	1	0
1101	11	1	e	1301	13	1	0
1301	13	1	f	1101	11	1	0
1101	11	1	e	1302	13	2	1
1302	13	2	f	1101	11	1	1

ObjektTyp B

GID	OID	VID	...
301	3	1	
401	4	1	
501	5	1	

ObjektTyp D

GID	OID	VID	...
801	8	1	
802	8	2	
803	8	3	
901	9	1	
902	9	2	

ObjektTyp F

GID	OID	VID	...
1201	12	1	
1202	12	2	
1301	13	1	
1302	13	2	

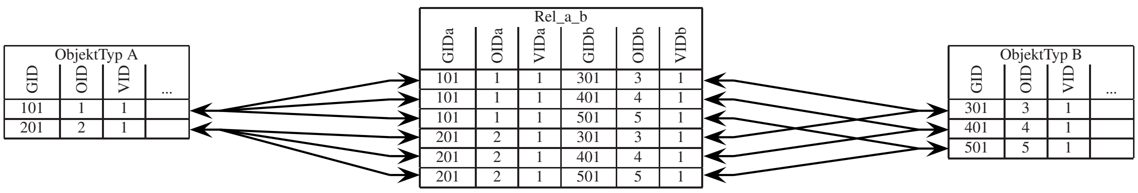
5.3.2 Variante 2

Bei dieser Variante werden die Beziehungen nicht mehr in einer generischen Tabelle verwaltet. Stattdessen werden für jeden Beziehungstyp zwischen zwei Objekttypen Tabellen angelegt, um die Beziehungen zwischen Instanzen der Objekttypen speichern zu können. Dabei wird zwischen drei unterschiedlichen Beziehungsarten unterschieden. Abbildung 5.5 zeigt eine Tabelle zur Verwaltung einer Beziehung ohne gleitendes Beziehungsende. Die Tabelle *Rel_a_b* besitzt Spalten zur Speicherung der *GlobalId*, *ObjectId* und *VersionId* der Objekttypinstanzen, die miteinander in Verbindung stehen. Die in der Abbildung enthaltenen Pfeile sollen die Fremdschlüsselbeziehungen verdeutlichen.

Bei der Verwaltung von Beziehungen mit einem gleitenden Beziehungsende werden, wie in Abbildung 5.6 dargestellt, zwei Tabellen benötigt. In der Tabelle *CVC_c_to_d* befinden sich Einträge, mit denen die Kandidatenmengen der einzelnen Objektversionen aus der Tabelle *ObjektTyp C* gespeichert werden. Im Vergleich zur Tabelle *Rel_a_b* bei Beziehungen ohne gleitende Beziehungsenden, enthält die Tabelle *CVC_c_to_d* zusätzlich die Spalte *pinned_d*, um die vorausgewählte Objektversionen markieren zu können. Wie bereits erwähnt, sind die Beziehungen zwischen Objektinstanzen nicht alle bidirektional, wenn es sich um einen Beziehungstyp mit genau einem gleitenden Beziehungsende handelt. Aus diesem Grund enthält die Tabelle *CVC_c_to_d* nur Beziehungsinformationen vom *Objekttyp C* in Richtung von *Objekttyp D*. Die Beziehungsinformationen für die Gegenrichtung werden in der Tabelle *Rel_d_to_c* verwaltet.

Bei Beziehungstypen mit zwei gleitenden Beziehungsenden ist die Verwaltung der Beziehungsinformationen einfacher, da hier alle Beziehungen bidirektional sind. Aus diesem Grund reicht eine Tabelle zur Speicherung der Informationen aus. Die Tabelle *CVC_c_d* in der Abbildung 5.7 enthält daher Spalten für Pinningeinstellungen in beide Richtungen.

Abbildung 5.5 Reduktionsvariante 2: Tabelle für Beziehungstyp ohne gleitende Beziehungsenden



5.3.3 Variante 3

Diese Variante der Reduktion ist eine Weiterentwicklung der vorherigen Variante, bei der nun die maximalen Kardinalitäten berücksichtigt werden, um Optimierungen vornehmen zu können. Diese Optimierungen können vorgenommen werden, falls die maximale Kardinalität 1 beträgt. Im bisherigen Beispiel wurden nur Beziehungen der Kardinalität *n:m* betrachtet. Abbildung 5.8 zeigt die Kardinalitäten der unterschiedlichen Beziehungsarten, die eine optimierte Speicherung zulassen.

Bei Beziehungen ohne gleitende Beziehungsenden müssen zwei neue Fälle betrachtet werden, die in Abbildung 5.8 dargestellt sind. Bei einer maximalen Kardinalität von *1:1* oder *1:n* kann auf eine zusätzliche Tabelle zur Speicherung der Beziehungsinformationen verzichtet werden. Diese Informationen werden durch zusätzliche Spalten in den Objekttyp-Tabellen verwaltet.

Abbildung 5.6 Reduktionsvariante 2: Tabellen für Beziehungstyp mit einem gleitenden Beziehungsende

Rel_d_to_c

GID_c	OID_c	GID_d	OID_d
601	6	801	8
602	6	803	8
601	6	901	9
602	6	902	9
701	7	803	8
701	7	902	9

CVC_c_to_d

GID_c	OID_c	VID_c	GID_d	OID_d	VID_d	pinned_d
601	6	1	801	8	1	0
601	6	1	803	8	3	1
601	6	1	901	9	1	1
602	6	2	802	8	2	0
602	6	2	803	8	3	0
602	6	2	902	9	2	1
701	7	1	803	8	3	0
701	7	1	901	9	1	0
701	7	1	902	9	2	0

ObjektTyp C

GID	OID	VID	...
601	6	1	
602	6	2	
701	7	1	

ObjektTyp D

GID	OID	VID	...
801	8	1	
802	8	2	
803	8	3	
901	9	1	
902	9	2	

Abbildung 5.7 Reduktionsvariante 2: Tabellen für Beziehungstyp mit zwei gleitenden Beziehungsenden

CVC_e_to_f

GID_e	OID_e	VID_e	pinned_e	GID_f	OID_f	VID_f	pinned_f
1001	10	1	0	1201	12	1	0
1001	10	1	0	1202	12	2	0
1001	10	1	0	1301	13	1	0
1002	10	2	0	1201	12	1	0
1002	10	2	0	1202	12	2	0
1002	10	2	0	1302	13	2	0
1101	11	1	0	1202	12	2	0
1101	11	1	0	1301	13	1	0
1101	11	1	0	1302	13	2	0

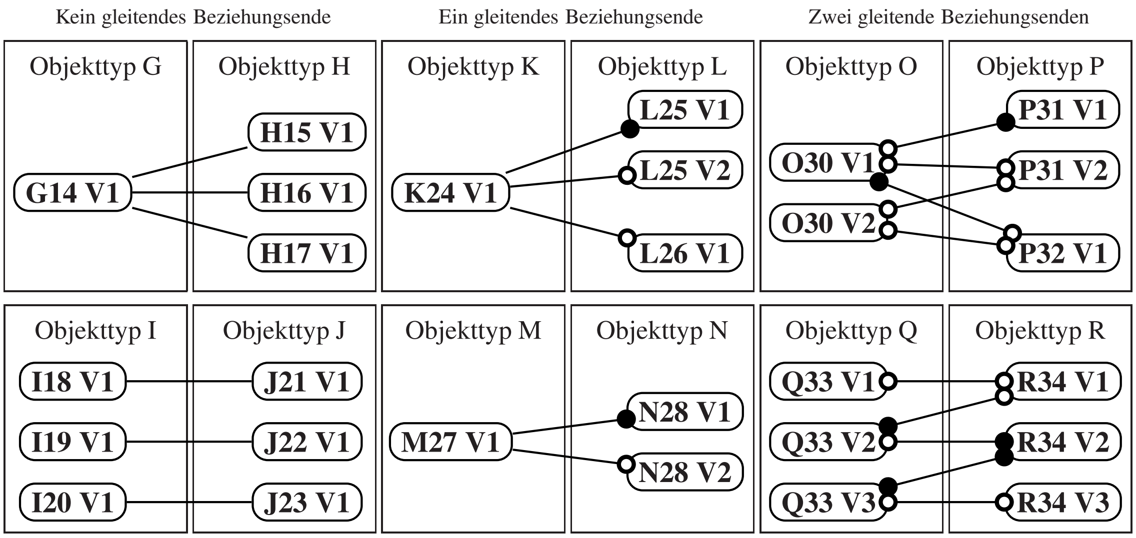
ObjektTyp E

GID	OID	VID	...
1001	10	1	
1002	10	2	
1101	11	1	

ObjektTyp F

GID	OID	VID	...
1201	12	1	
1202	12	2	
1301	13	1	
1302	13	2	

Abbildung 5.8 Beziehungsarten mit Optimierungsmöglichkeit auf Grund der maximalen Kardinalität



Bei einer *1:n* Beziehung wird der Objekttyp-Tabelle am Beziehungsende mit der Kardinalität *n* eine Spalte hinzugefügt, in der eine Referenz auf das eine Objekt am anderen Beziehungsende gespeichert wird. Falls die Beziehung eine *1:1* Beziehung ist, wird in jeder der beiden Objekttyptabellen eine Referenz auf das Objekt am jeweiligen anderen Beziehungsende gehalten.

Handelt es sich um eine Beziehung mit einem gleitenden Beziehungsende, so ist eine Optimierung möglich, falls die maximale Kardinalität am nicht gleitenden Beziehungsende 1 beträgt. In diesem Fall kann, wie in Abbildung 5.10, auf die Tabelle *Rel_l_to_k* verzichtet und durch die Spalte *rel_k* ersetzt werden. Dabei spielt die maximale Kardinalität am gleitenden Beziehungsende keine Rolle. Sie darf sowohl *1* als auch *n* betragen.

Bei Beziehungen mit zwei gleitenden Beziehungsenden sind keine Optimierungen in Abhängigkeit der maximalen Kardinalität möglich. Diese Optimierungen können erst bei der Reduktionsvariante 4 durchgeführt werden.

Abbildung 5.9 Reduktionsvariante 3: Tabelle für Beziehungstyp ohne gleitende Beziehungsenden unter Berücksichtigung der maximalen Kardinalität

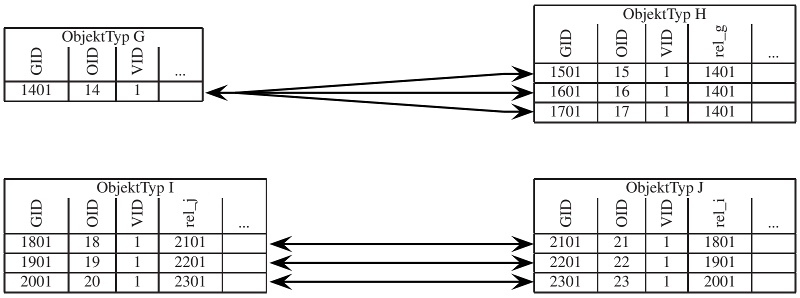
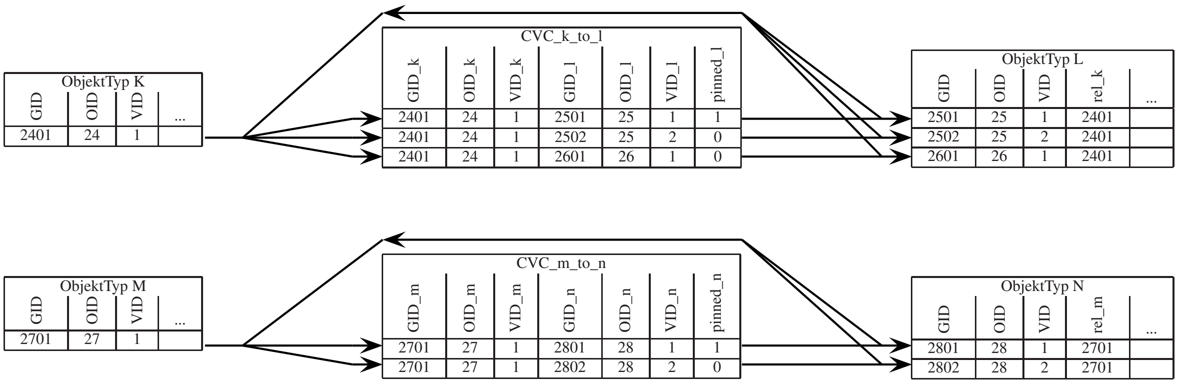


Abbildung 5.10 Reduktionsvariante 3: Tabelle für Beziehungstyp mit einem gleitenden Beziehungsende unter Berücksichtigung der maximalen Kardinalität

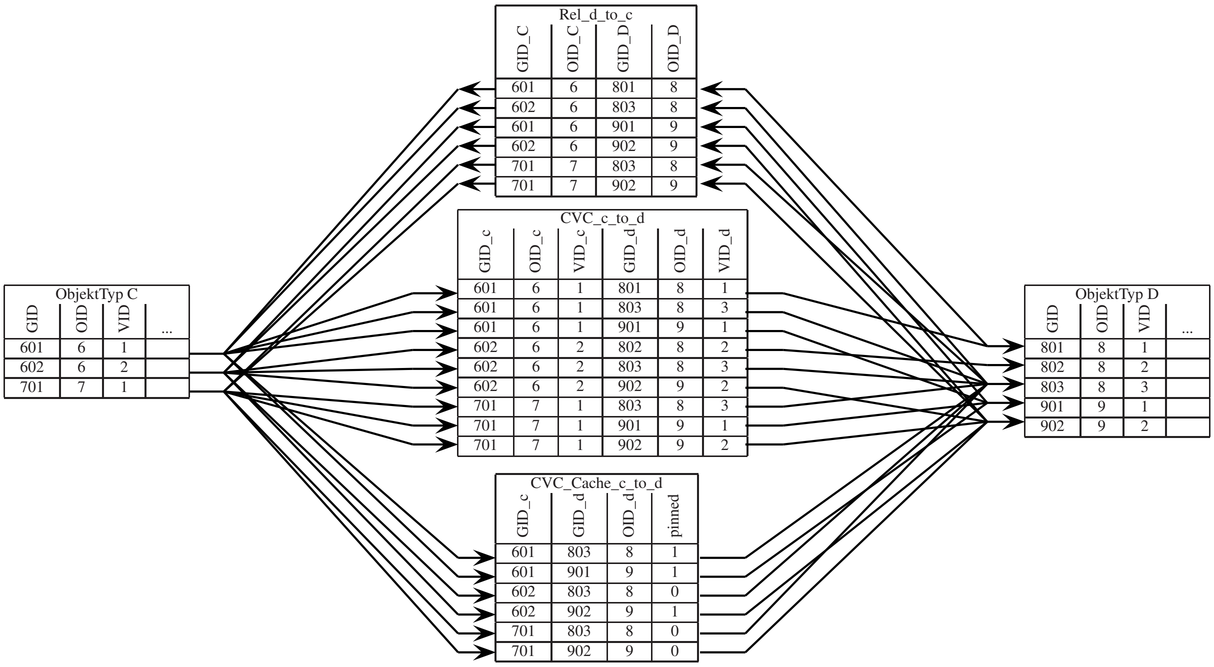


5.3.4 Variante 4

Bei der Navigation über eine Beziehung, deren Beziehungsende gleitend ist, wird bei den vorherigen Varianten der Reduktion zur Laufzeit überprüft, ob eine vorausgewählte Version existiert und falls keine vorausgewählte Version vorhanden ist, die neueste Objektversion bestimmt und verwendet. Dieser Aufwand zur Laufzeit für die Ausführung einer Navigation kann reduziert werden, indem diese Auswahl materialisiert und in einer zusätzlichen Tabelle bereitgestellt wird. Dadurch wird jedoch der Aufwand beim Erstellen und Löschen von Beziehungen erhöht, da diese Materialisierung nach jeder Änderung an der Tabelle, in der Beziehungen gespeichert werden, aktualisiert werden muss. Wir vermuten jedoch, dass die Anzahl der Navigationen über eine Beziehung deutlich größer ist als die Anzahl der Anlege- und Löschoperationen von Beziehungen.

Abbildung 5.11 und Abbildung 5.12 enthalten die Tabellen *CVC_Cache_c_to_d*, *CVC_Cache_e_to_f* und *CVC_Cache_f_to_e* in denen die Materialisierung der vorausgewählten oder durch eine Regel ausgewählten Objektversionen vorgehalten wird. Diese Tabellen enthalten die *GlobalId* der Quellobjektes und die *GlobalId* sowie *ObjectId* auf der Zielseite. Außerdem wird festgehalten, ob die materialisierte Objektversion vorausgewählt wurde, oder es sich um eine Auswahl durch eine Regel handelt.

Abbildung 5.11 Reduktionsvariante 4: Tabelle für Beziehungstyp mit einem gleitenden Beziehungsende mit Materialisierung der Pinning- und regelbasierten Auswahl



Für die Materialisierung muss nicht immer eine eigene Tabelle verwendet werden. Bei einer maximalen Kardinalität von 1 am gleitenden Beziehungsende kann diese Tabelle durch zwei Spalten in der Objekttyptabelle am gegenüberliegenden Beziehungsende ersetzt werden. Diese Optimierungsmöglichkeit wurde in den Abbildungen 5.13 und 5.14 ausgenutzt. Die Spalte *cache_n* der Tabelle *ObjektTyp M* in Abbildung 5.13 enthält eine Referenz auf die vorausgewählte oder durch eine Regel ausgewählte Objektversion. Zur Markierung, ob es sich bei der Referenz in der Spalte *cache_n* um eine vorausgewählte oder durch eine Regel ausgewählte Objektversion handelt, wird die Spalte *pinned_n* verwendet.

Abbildung 5.12 Reduktionsvariante 4: Tabelle für Beziehungstyp mit einem gleitenden Beziehungsende mit Materialisierung der Pinning- und regelbasierten Auswahl

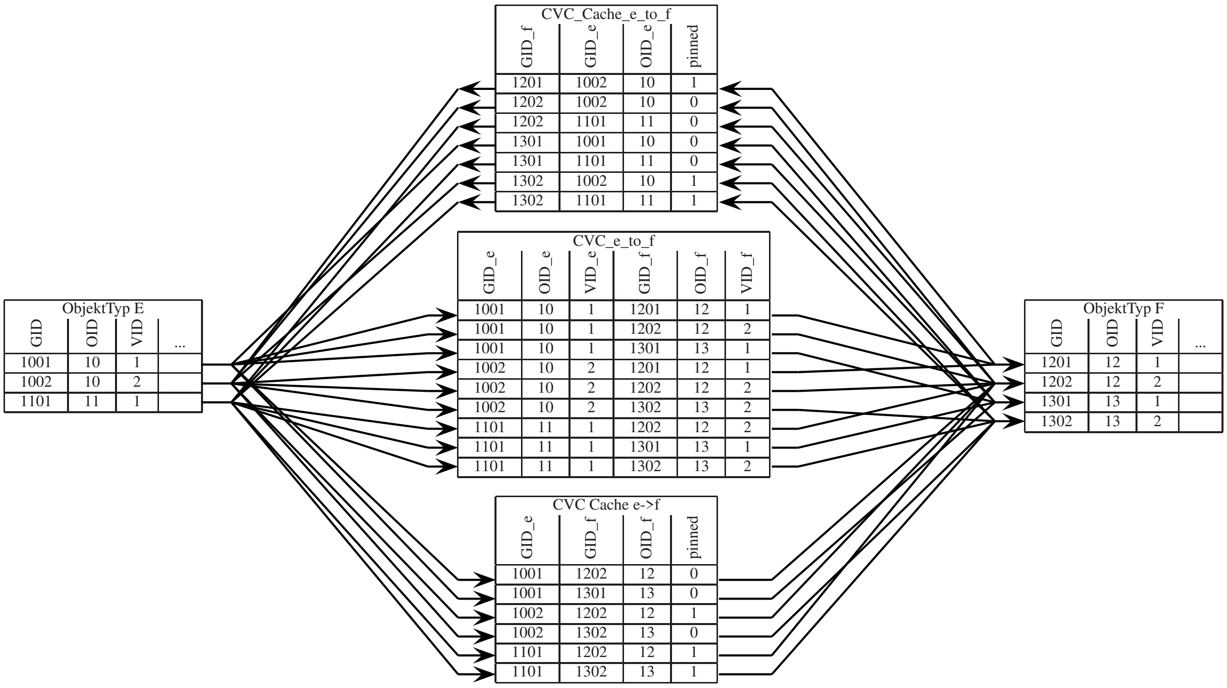


Abbildung 5.13 Reduktionsvariante 4: Tabelle für Beziehungstyp mit einem gleitenden Beziehungsende mit Materialisierung der Pinning- und regelbasierten Auswahl und Berücksichtigung der maximalen Kardinalitäten

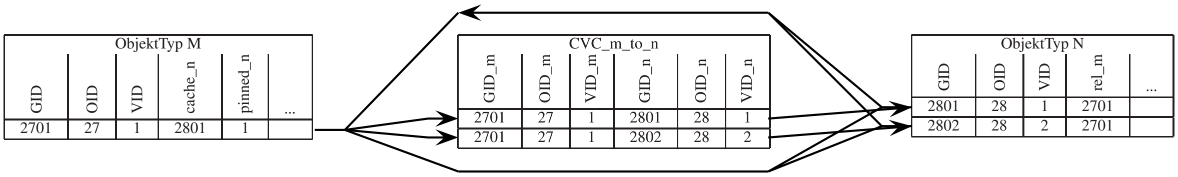
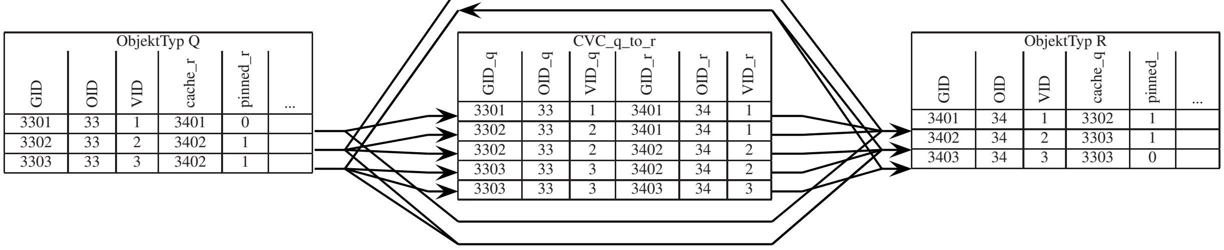


Abbildung 5.14 Reduktionsvariante 4: Tabelle für Beziehungstyp mit zwei gleitenden Beziehungsenden mit Materialisierung der Pinning- und regelbasierten Auswahl und Berücksichtigung der maximalen Kardinalitäten



5.4 Reduktion von DS-DML

Die Reduktion von DS-DML-Anweisungen ist eng mit der Reduktion von DS-DDL-Anweisungen verknüpft. Die Reduktionsmethoden müssen jeweils auf das generierte Datenbankschema zugeschnitten werden. Aus diesem Grund ist das Informationsmodell von enormer Bedeutung für die Abläufe der DS-DML-Reduktion. Das Informationsmodell enthält beispielsweise wichtige Informationen über die Propagierung von Operationen oder Art und Kardinalitäten der Beziehungen. Dies soll hier am Beispiel der Reduktionsmethode der Operation *GetRelatedObjects* für die im vorherigem Abschnitt vorgestellten Reduktionsvarianten vorgestellt werden. Die Operation *GetRelatedObjects* wird verwendet, um die *GlobalIds* der Objekte zu erhalten, die mit einem bestimmten Objekt über eine bestimmte Beziehung verbunden sind. Dies wird vor allem benötigt, um herauszufinden, an welche Objekte eine Operation propagiert werden muss, falls das entsprechende Beziehungsende eine Propagierung unterstützt. Beim Aufruf der Reduktionsmethode ist nur der Objekttyp und die *GlobalId* eines Objektes, sowie das damit verbundene Beziehungsende, über das die verbundenen Objekte gesucht werden sollen, bekannt. Falls die gesamte Kandidatenmenge eines gleitenden Beziehungsende zurückgegeben werden soll, so wird dies mit Hilfe der Variablen *getAll* ausgedrückt. Alle weiteren Informationen, die notwendig sind, um einen SQL-DML-Baum zu erzeugen, müssen aus dem Informationsmodell entnommen werden.

Listing 5.1 Reduktionsmethode der Operation *GetRelatedObjects* bei Variante 1

```
public ResultSet reduce(Statement stmt) throws SQLException{

   VSRelationshipEnd relEndB = relEndA.getRelationshipEnd();
   String rolenameA = relEndA.getName();

   FromClause fromClause = null;
   WhereClause whereClause = null;
   DisplayedColumnClause displayedColumnClause = null;

   if (relEndB.getFloating() == 0 || getAll) {
      displayedColumnClause = new DisplayedColumnClause("destGID AS globalID");
      fromClause = new FromClause("Relationships");
      whereClause = new WhereClause("sourceGID="+gid);
      whereClause.addCondition("source_rolename='"+rolenameA+"'");
   } else {
      displayedColumnClause = new DisplayedColumnClause("destGID AS globalID");
      fromClause = new FromClause("Relationships");
      WhereClause whereClauseSub = new WhereClause("sourceGID="+gid);
      whereClauseSub.addCondition("source_rolename='"+rolenameA+"'");
      SelectStatement subSel = new SelectStatement(new DisplayedColumnClause("max(destGID) AS globalId"),
                              fromClause,whereClauseSub,new GroupByClause("destOID","MAX(pinned) = 0"));
      whereClause = new WhereClause("sourceGID="+gid);
      whereClause.addCondition("source_rolename='"+rolenameA+"'");
      whereClause.addCondition("pinned = 1");
      whereClause.addCondition(WhereClause.OR,"destGID","IN",subSel);
   }

   SelectStatement selStmt1 = new SelectStatement(displayedColumnClause, fromClause ,whereClause);

   String sql1 = selStmt1.render();
   ResultSet rs = stmt.executeQuery(sql1);

   return rs;
}
```

Listing 5.1 zeigt die Reduktionsmethode der Operation *GetRelatedObjects* für das, in der Variante 1, generierte Datenbankschema mit einer generischen Tabelle für alle Beziehungsinformationen. Zunächst wird ein Objekt, das alle Informationen über die Eigenschaften des gegenüberliegenden Beziehungsende enthält, über das Informationsmodell angefragt. Dieses Objekt wird benötigt, um herauszufinden, ob das gegenüberliegende Beziehungsende gleitend ist. Falls es sich nicht um ein gleitendes Beziehungsende handelt oder die gesamte Kandidatenmenge zurückgeliefert werden soll, so wird eine Anfrage in der Form:

```
SELECT destGID AS globalID FROM Relationships
WHERE sourceGID = <gid> AND source_rolename = '<rollenname>'
```

als SQL-DML-Baum erzeugt. Handelt es sich jedoch um ein gleitendes Beziehungsende, so ist die zu erzeugende Anfrage komplexer, da überprüft werden muss, ob eine vorausgewählte Objektversion existiert und andernfalls die neueste Objektversion bestimmt werden muss. Diese Aufgabe wird von einer Anfrage der Form:

```
SELECT destGID AS globalID FROM Relationships
WHERE sourceGID = <gid> AND source_rolename = '<rollenname>'
      AND pinned = 1 OR destGID IN (
         SELECT max(destGID) AS globalId FROM Relationships
         WHERE  sourceGID = <gid> AND source_rolename = '<rollenname>'
        GROUP BY destOID HAVING MAX(pinned) = 0
      )
```

erledigt. Auf dem SQL-Objektbaum wird dann abschließend die Rendering-Methode aufgerufen und die erhaltene Anweisung dem Datenbanktreiber übergeben.

Listing 5.2 Reduktionsmethode der Operation *GetRelatedObjects* bei Variante 2

```
public ResultSet reduce(Statement stmt) throws SQLException{

   VSRelationshipEnd relEndB = relEndA.getRelationshipEnd();
   String rolenameA = relEndA.getName();
   String rolenameB = relEndB.getName();

   FromClause fromClause = null;
   WhereClause whereClause = null;
   DisplayedColumnClause displayedColumnClause = null;

   if (relEndA.getFloating() == 0 && relEndB.getFloating() == 0) { // keine Seite Floating

     displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
     VSRelationshipType rt = relEndA.getRelationshipType();
     fromClause = new FromClause("Rel_"+rt.getName());
     whereClause = new WhereClause(rolenameA+"GID="+gid);

   } else if (relEndB.getFloating() == 1) { // andere Seite Floating
      if (getAll) {
         displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
         if (relEndA.getFloating() == 1) {
            fromClause = new FromClause("Rel_CVC_"+relEndA.getRelationshipType().getName());
         } else {
            fromClause = new FromClause("Rel_CVC_"+rolenameA+"_to_"+rolenameB);
         }
         whereClause = new WhereClause(rolenameA+"GID="+gid);
      } else {
         displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
         if (relEndA.getFloating() == 1) {
            fromClause = new FromClause("Rel_CVC_"+relEndA.getRelationshipType().getName());
         } else {
            fromClause = new FromClause("Rel_CVC_"+rolenameA+"_to_"+rolenameB);
         }
         SelectStatement subSel = new SelectStatement(new DisplayedColumnClause("max("+rolenameB+"GID) AS globalId"),
                                 fromClause,new WhereClause(rolenameA+"GID="+gid),
                                 new GroupByClause(rolenameB+"OID","MAX(pinned_"+rolenameB+") = 0"));
         whereClause = new WhereClause(rolenameA+"GID="+gid);
         whereClause.addCondition("pinned_"+rolenameB+" = 1");
         whereClause.addCondition(WhereClause.OR,rolenameB+"GID","IN",subSel);
      }
   } else { // andere Seite nicht Floating
      displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
      fromClause = new FromClause("Rel_"+rolenameA+"_to_"+rolenameB);
      whereClause = new WhereClause(rolenameA+"GID="+gid);
   }

   SelectStatement selStmt1 = new SelectStatement(displayedColumnClause, fromClause ,whereClause);

   String sql1 = selStmt1.render();
   ResultSet rs = stmt.executeQuery(sql1);

   return rs;
}
```

Listing 5.2 enthält die Reduktionmethode für die Variante 2. Da in dieser Reduktionsvariante je nach Beziehungsart unterschiedliche Tabellen generiert werden, muss zunächst mit Hilfe des Informationsmodells festgestellt werden, ob es sich um eine Beziehung mit keinem, einem oder zwei gleitenden Beziehungsenden handelt. In Abhängigkeit der Beziehungsart wird ein passender SQL-DML-Baum erzeugt. Bei Beziehungen ohne gleitende Beziehungsenden wird folgende Anfrage verwendet.

```
SELECT <rollennameB>GID AS globalID FROM Rel_<beziehungsname>
WHERE <rollenname>GID = <gid>
```

Falls das gegenüberliegende Beziehungsende gleitend ist, wird folgende Anfrage benötigt:

```
SELECT <rollenname>GID AS globalID
FROM {Rel_CVC_<beziehungsname>,Rel_CVC_<rollenname>_to_<rollenname>}
WHERE <rollenname>GID = <gid> AND pinned_<rollenname> = 1 OR <rollenname>GID IN (
      SELECT max(<rollenname>GID) AS globalId
      FROM {Rel_CVC_<beziehungsname>,Rel_CVC_<rollenname>_to_<rollenname>}
      WHERE <rollenname>GID = <gid>
      GROUP BY <rollenname>OID HAVING MAX(pinned_<rollenname>) = 0
)
```

Dabei ist zu beachten, dass der Tabellenname davon abhängt, ob nur das gegenüberliegende Beziehungsende oder beide Beziehungsenden gleitend sind. Ist das gegenüberliegende Beziehungsende nicht gleitend, so wird folgende Anfrage erzeugt:

```
SELECT <rollennameB>GID AS globalID FROM Rel_<rollennameA>_to_<rollennameB>
WHERE <rollennameA>GID = <gid>
```

Da bei der Variante 3 die maximalen Kardinalitäten betrachtet werden, um Optimierungen bei der Datenverwaltung vorzunehmen, müssen in der Reduktionsmethode, die im Listing 5.3 dargestellt ist, mehr Fallunterscheidungen durchgeführt werden. Die Informationen, die für die Fallunterscheidungen benötigt werden, müssen aus dem Informationsmodell entnommen werden. Im Vergleich zu Listing 5.2 ist klar zu erkennen, dass sich der Codeabschnitt, der sich mit einem gegenüberliegenden gleitenden Beziehungsende beschäftigt, als einziger nicht verändert wird, da hier keine Optimierungen vorgenommen werden können. So ergeben sich bei Beziehungen ohne gleitende Beziehungsenden Anfragen in der Form:

```
-- n:m
SELECT <rollennameB>GID AS globalID FROM Rel_<beziehungsname>
WHERE <rollennameA>GID = <gidA>

-- 1:1 oder n:1
SELECT rel_<rollennameB>GID AS globalID
FROM ObjectType_<objekttypnameA>
WHERE globalGID = <gidA> AND rel_<rollennameB>GID IS NOT NULL

-- 1:n
SELECT globalID FROM ObjectType_<objekttypnameB>
WHERE rel_<rollennameA> = <gidA>
```

Beziehungen, bei denen nur das Beziehungsende A gleitend ist, werden folgende Anfragen zusammengebaut:

```
-- 1:1 oder n:1
SELECT rel_<rollennameB>GID AS globalID
FROM ObjectType_<objekttypnameA>
WHERE globalGID = <gidA> AND rel_<rollennameB>GID IS NOT NULL

-- n:1 oder m:1
SELECT <rollennameB>GID AS globalID
FROM Rel_<rollennameA>_to_<rollennameB>
WHERE <rollennameA>GID = <gidA>
```

Bei der letzten untersuchten Reduktionsvariante wird der Zugriff auf vorausgewählte oder durch eine Regel ausgewählte Objekte vereinfacht, da das Ergebnis des Auswahlprozesses materialisiert wird. Aus diesem Grund ändert sich im Listing 5.4 nur der Codeabschnitt, der relevant ist, falls das gegenüberliegende Beziehungsende gleitend ist. In Abhängigkeit der maximalen Kardinalität ergeben sich folgende zwei Anfragetypen:

```
-- 1:1 oder n:1
SELECT <rollennameB>GID AS globalID
FROM Rel_CVC_Cache_<rollennameA>_to_<rollennameB>
WHERE <rollennameA>GID = <gidA>

-- 1:m oder n:m
SELECT rel_<rolenameB>GID AS globalID
FROM ObjectType_<objekttypnameA>
WHERE globalID = <gidA> AND rel_<rollennameB>GID IS NOT NULL
```

Je nach Operation und DDL-Reduktionsvariante unterscheidet sich nicht nur die erzeugte SQL-Anweisung. Die Anzahl der Anweisungen, die bei der Reduktion erstellt werden kann variieren. Dies ist beispielsweise bei der Operation *NewRelationship* der Fall. Dabei werden Select-, Insert- und Updateanweisungen erzeugt.

Listing 5.3 Reduktionsmethode der Operation *GetRelatedObjects* bei Variante 3

```
public ResultSet reduce(Statement stmt) throws SQLException{

    VSRelationshipEnd relEndB = relEndA.getRelationshipEnd();
    String rolenameA = relEndA.getName();
    String rolenameB = relEndB.getName();

    FromClause fromClause = null;
    WhereClause whereClause = null;
    DisplayedColumnClause displayedColumnClause = null;

    if (relEndA.getFloating() == 0 && relEndB.getFloating() == 0) { // keine Seite Floating

       // n:m
       if (relEndA.getMaxMultiplicity() != 1 && relEndB.getMaxMultiplicity() != 1) {
          displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
          VSRelationshipType rt = relEndA.getRelationshipType();
          fromClause = new FromClause("Rel_"+rt.getName()+"_"+rt.getRelationshipEnd1().getName()+"_"+rt.getRelationshipEnd2().getName());
          whereClause = new WhereClause(rolenameA+"GID="+gid);

       // 1:1 oder n:1
       } else if (relEndB.getMaxMultiplicity() == 1) {
          displayedColumnClause = new DisplayedColumnClause("rel_"+rolenameB+"GID AS globalID");
          fromClause = new FromClause("ObjectType_"+relEndA.getObjectType().getName());
          whereClause = new WhereClause("globalID="+gid);
          whereClause.addCondition("rel_"+rolenameB+"GID IS NOT NULL");

       // 1:n
            } else {
         displayedColumnClause = new DisplayedColumnClause("globalID");
         fromClause = new FromClause("ObjectType_"+relEndB.getObjectType().getName());
         whereClause = new WhereClause("rel_"+rolenameA+"GID = "+gid);
       }
    } else if (relEndB.getFloating() == 1) { // andere Seite Floating
      if (getAll) {
         displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
         if (relEndA.getFloating() == 1) {
            fromClause = new FromClause("Rel_CVC_"+relEndA.getRelationshipType().getName());
         } else {
            fromClause = new FromClause("Rel_CVC_"+rolenameA+"_to_"+rolenameB);
         }
         whereClause = new WhereClause(rolenameA+"GID="+gid);
      } else {
         displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
         if (relEndA.getFloating() == 1) {
            fromClause = new FromClause("Rel_CVC_"+relEndA.getRelationshipType().getName());
         } else {
            fromClause = new FromClause("Rel_CVC_"+rolenameA+"_to_"+rolenameB);
         }

         SelectStatement subSel = new SelectStatement(new DisplayedColumnClause("max("+rolenameB+"GID) AS globalId"),
                                  fromClause,new WhereClause(rolenameA+"GID="+gid),
                                  new GroupByClause(rolenameB+"OID","MAX(pinned_"+rolenameB+") = 0"));

         whereClause = new WhereClause(rolenameA+"GID="+gid);
         whereClause.addCondition("pinned_"+rolenameB+" = 1");
         whereClause.addCondition(WhereClause.OR,rolenameB+"GID","IN",subSel);
      }
   } else { // andere Seite nicht Floating
      if (relEndB.getMaxMultiplicity() == 1) {
         displayedColumnClause = new DisplayedColumnClause("rel_"+rolenameB+"GID AS globalID");
         fromClause = new FromClause("ObjectType_"+relEndA.getObjectType().getName());
         whereClause = new WhereClause("globalID="+gid);
         whereClause.addCondition("rel_"+rolenameB+"GID IS NOT NULL");
      } else {
         displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
         fromClause = new FromClause("Rel_"+rolenameA+"_to_"+rolenameB);
         whereClause = new WhereClause(rolenameA+"GID="+gid);
      }

   }


   SelectStatement selStmt1 = new SelectStatement(displayedColumnClause, fromClause ,whereClause);

   String sql1 = selStmt1.render();
   ResultSet rs = stmt.executeQuery(sql1);

   return rs;
}
```

Listing 5.4 Reduktionsmethode der Operation *GetRelatedObjects* bei Variante 4

```
public ResultSet reduce(Statement stmt) throws SQLException{

   VSRelationshipEnd relEndB = relEndA.getRelationshipEnd();
   String rolenameA = relEndA.getName();
   String rolenameB = relEndB.getName();

   FromClause fromClause = null;
   WhereClause whereClause = null;
   DisplayedColumnClause displayedColumnClause = null;

   if (relEndA.getFloating() == 0 && relEndB.getFloating() == 0) { // keine Seite Floating

      // n:m
      if (relEndA.getMaxMultiplicity() != 1 && relEndB.getMaxMultiplicity() != 1) {

         displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
         VSRelationshipType rt = relEndA.getRelationshipType();
         fromClause = new FromClause("Rel_"+rt.getName()+"_"+rt.getRelationshipEnd1().getName()+"_"+rt.getRelationshipEnd2().getName());
         whereClause = new WhereClause(rolenameA+"GID="+gid);

         // 1:1 oder n:1
      } else if (relEndB.getMaxMultiplicity() == 1) {
         displayedColumnClause = new DisplayedColumnClause("rel_"+rolenameB+"GID AS globalID");
         fromClause = new FromClause("ObjectType_"+relEndA.getObjectType().getName());
         whereClause = new WhereClause("globalID="+gid);
         whereClause.addCondition("rel_"+rolenameB+"GID IS NOT NULL");

      // 1:n
      } else {
          displayedColumnClause = new DisplayedColumnClause("globalID");
          fromClause = new FromClause("ObjectType_"+relEndB.getObjectType().getName());
          whereClause = new WhereClause("rel_"+rolenameA+"GID = "+gid);
      }


   } else if (relEndB.getFloating() == 1) { // andere Seite Floating
           if (getAll) {
               displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
               if (relEndA.getFloating() == 1) {
                   fromClause = new FromClause("Rel_CVC_"+relEndA.getRelationshipType().getName());
               } else {
                   fromClause = new FromClause("Rel_CVC_"+rolenameA+"_to_"+rolenameB);
               }
               whereClause = new WhereClause(rolenameA+"GID="+gid);
           } else {
               if (relEndB.getMaxMultiplicity() == 1) {
                   displayedColumnClause = new DisplayedColumnClause("cache_"+rolenameB+" AS globalID");
                   fromClause = new FromClause("ObjectType_"+relEndA.getObjectType().getName());
                   whereClause = new WhereClause("globalID="+gid);
                   whereClause.addCondition("cache_"+rolenameB+" IS NOT NULL");
               } else {
                   displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
                   fromClause = new FromClause("Rel_CVC_Cache_"+rolenameA+"_to_"+rolenameB);
                   whereClause = new WhereClause(rolenameA+"GID="+gid);
               }
           }
        } else { // andere Seite nicht Floating
          if (relEndB.getMaxMultiplicity() == 1) {
                   displayedColumnClause = new DisplayedColumnClause("rel_"+rolenameB+"GID AS globalID");
                   fromClause = new FromClause("ObjectType_"+relEndA.getObjectType().getName());
                   whereClause = new WhereClause("globalID="+gid);
                   whereClause.addCondition("rel_"+rolenameB+"GID IS NOT NULL");
               } else {
                   displayedColumnClause = new DisplayedColumnClause(rolenameB+"GID AS globalID");
                   fromClause = new FromClause("Rel_"+rolenameA+"_to_"+rolenameB);
                   whereClause = new WhereClause(rolenameA+"GID="+gid);
               }
          }


   SelectStatement selStmt1 = new SelectStatement(displayedColumnClause, fromClause ,whereClause);

   String sql1 = selStmt1.render();
   ResultSet rs = stmt.executeQuery(sql1);

   return rs;
}
```

Kapitel 6 Bewertung

Die vier entwickelten domänenspezifischen Datenbanktreiber werden in diesem Kapitel nach verschiedenen Kriterien bewertet. Diese Bewertung untergliedert sich in zwei Teile. Zunächst wird eine Bewertung der Reduktionsmethoden durch Softwaremetriken vorgenommen. Anschließend wird eine Leistungsbewertung der vier verschiedenen Datenbanktreiber und den dazugehörigen Datenbankschemata durchgeführt. Mit Hilfe der Softwaremetriken wird die Komplexität der Reduktionsmethoden der vier unterschiedlichen Implementierungen verglichen. Die Leistungsuntersuchungen dienen zum Vergleich der unterschiedlichen Datenbankschemata und den dazugehörigen domänenspezifischen Datenbanktreibern.

6.1 Bewertung durch Softwaremetriken

In diesem Abschnitt werden zunächst einige Softwaremetriken vorgestellt, mit denen anschließend die Reduktionsmethoden der vier Datenbanktreiber bewertet werden.

6.1.1 Vorstellung von Softwaremetriken

Die wohl bekannteste Metrik, um die Größe von Quellcode zu messen, ist die Zählung der Quellcodezeilen (Lines of Code, LOC). Das Maß erlaubt jedoch nur eine grobe Abschätzung des Umfangs eines Programms. Es spiegelt nicht notwendigerweise den Aufwand, der zur Programmerstellung nötig war, wider, da beispielsweise Kommentarzeilen und Formatierung des Quellcodes störend wirken und komplexe Kontrollflüsse nicht erfasst werden.

In diesem Abschnitt werden drei Metriken vorgestellt, die bei der Bewertung eingesetzt wurden. Zu ihnen gehören die Zählung von Anweisungen, die zyklomatische Komplexität und der Halstead-Aufwand.

Zählung der Anweisungen

Wie bereits beschrieben, ist die LOC-Metrik nicht besonders aussagekräftig, da der Quellcode neben dem eigentlichen Programm auch Leerzeilen und Kommentare enthalten kann. Die spezifische Formatierung des Quellcodes kann aber auch hier Verfälschungen in der Aufwandsabschätzung zur Folge haben, wenn etwa die Zeilen stark unterschiedlich lang sind. Um den Einfluss der Formatierung des Quellcodes auf die Messung aufzuheben, kann die Anzahl der Anweisungen gezählt werden. Dazu ist jedoch eine aufwändige Syntaxanalyse notwendig. Die Zählung der Anweisungen gestattet nicht nur die bessere Abschätzung des Erstellungsaufwands, sondern liefert auch eine bessere Abschätzung des Testaufwands, da beim Testen typischerweise nur die funktionalen Teile einer Anwendung betrachtet werden. Eine Zählung der Quellcodeanweisungen anstelle von Quellcodezeilen ist vor diesem Hintergrund sinnvoller, weil

Tabelle 6.1 Faustregeln für die zyklomatische Komplexität C(G)

C(G)	Programm	Risiko
1-10	einfach	gering
11-20	komplexer	erträglich
21-50	komplex	hoch
> 50	untestbar	extrem hoch

Deklarationen von Datenstrukturen ausgeblendet werden.

Zyklomatische Komplexität

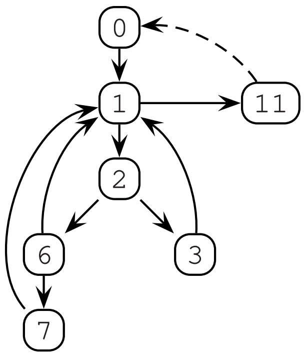
Die zyklomatische Komplexität nach McCabe [McC76] ist ein Maß zur Bestimmung der strukturellen Komplexität. Grundlage dieses Maßes ist der Kontrollflussgraph einer Funktion. Für einen Graphen G ist die zyklomatische Komplexität $C(G)$ definiert als die Anzahl von Schleifen im Kontrollflussgraphen, bei dem der Endknoten künstlich mit dem Startknoten verbunden wird. Die zyklomatische Komplexität kann nach Analyse des Quellcodes durch folgende Formel bestimmt werden:

$C(G) = e - n + 2$, wobei
e für die Anzahl der Kanten und
n für Anzahl der Knoten steht.

Abbildung 6.1 zeigt ein einfaches Beispiel zur Berechnung der zyklomatischen Komplexität. Die Abbildung enthält ein Programmcodeausschnitt und den dazugehörigen Kontrollflussgraphen. In diesem Beispiel beträgt die Anzahl der Knoten 7 und die Anzahl der Kanten zwischen den Knoten ist 9. Die Komplexität beträgt $C(G) = 9 - 7 + 2 = 4$ [Fra04].

Abbildung 6.1 Programmcodeausschnitt und zugehöriger Kontrollflussgraph

```
n=a; m=b;
while(n!=m){
  if(n<m) {
    n=n+a;
  } else {
    if (m<n) {
      m=m+b;
    }
  }
}
println(n);
```



Mit Hilfe der zyklomatischen Komplexität können Aussagen über die Testbarkeit oder Wartbarkeit von Software-Modulen getroffen werden. Ein Modul kann leicht getestet werden, wenn die Anzahl der Testfälle überschaubar ist. Module sind nur dann wartbar, wenn die Funktionsweise der Module durch Analyse des Programmcodes leicht verstanden werden kann. Die Testbarkeit und Wartbarkeit von Modulen wird durch einen einfachen Kontrollfluss begünstigt. Tabelle 6.1 zeigt einige Faustregeln für die zyklomatische Komplexität und das damit verbundene Risiko.

Metriken nach Halstead

Die Komplexität eines Programms ist nicht nur von der Komplexität des Kontrollflusses abhängig. Die Anzahl der zu verarbeitenden Daten und die dazu notwendigen Operationen spielen dabei auch eine wesentliche Rolle. Die dadurch entstehende Komplexität berechnet Halstead [Hal77] an hand von Metriken. Halstead baut seine Metriken auf dem Quellcode von Software-Modulen auf und unterstellt zwei grundlegende Bausteinarten: Operanden und Operatoren. Als Grundlage für die Metriken werden folgende Variablen definiert:

n_1	Anzahl unterschiedlicher Operatoren
n_2	Anzahl unterschiedlicher Operanden
N_1	Gesamtanzahl Operatoren
N_2	Gesamtanzahl Operanden

Halstead leitet daraus folgende Metriken ab:

Länge (N)	$N = N_1 + N_2$
Vokabular (n)	$n = n_1 + n_2$
Volumen(V)	$V = N * log2(n)$
Schwierigkeit (D)	$D = \frac{n_1 * N_2}{2 * n_2}$
Aufwand (E)	$E = D * V$

6.1.2 Analyse

Die Reduktionsmethoden der vier unterschiedlichen Implementierungen des domänenspezifischen JDBC-Treibers wurden mit Hilfe des Analysewerkzeuges [Vir03] analysiert. Dabei wurden verschiedene Metriken bestimmt. Tabelle 6.2 zeigt die Ergebnisse für die Implementierung der Variante 1. Gemessen wurden die Anzahl der Ausdrücke[1], Anweisungen, Fallunterscheidungen, Schleifen, so wie die maximalen Verschachtelungstiefen. Aus den gemessenen Werten wurde die zyklomatische Komplexität sowie der Halstead-Aufwand berechnet. Die Ergebnisse der Analyse für die anderen Implementierungen befinden sich im Anhang E.

Durch eine Kapselung des Zugriffs auf Tabellen und Spalten, die Beziehungsinformationen enthalten, waren für die Implementierung der unterschiedlichen JDBC-Treibervarianten nicht in allen Reduktionsmethoden Anpassungen nötig. Die Reduktionsmethoden für das Anlegen und Löschen von Beziehungen, den Zugriff auf mit einem Objekt in Beziehung stehenden Objekten, der Festlegung der vorausgewählten Objektversion, sowie die Reduktionsmethoden für die Navigation über verschiedene Beziehungen mussten angepasst werden. Die anderen Reduktionsmethoden konnten unverändert übernommen werden und weisen daher bei der Analyse die gleichen Messwerte auf.

Die Abbildungen 6.2 und 6.3 zeigen, das die Optimierungen, die in den verschiedenen Varianten vorgenommen wurden, zu einen erhöhten Aufwand bei der Implementierung der Reduktionsmethode geführt haben. Die Ergebnisse der Analyse der Reduktionsmethoden für die Operation `New Relationship` sind in Abbildung 6.2 veranschaulicht. Dabei ist festzustellen, dass die zyklomatische Komplexität bei Variante 4 um den Faktor 2,5 größer ist als bei Variante 1. Ein Vergleich des Halstead-Aufwands ergibt einen Faktor von 2,4. Bei der Operation `Delete Relationship`, deren Vergleich in Abbildung 6.3 dargestellt ist, ist der größte Unterschied zwischen den einzelnen Varianten feststellbar. So ist die zyklomatische Komplexität bei Varian-

[1] Ein Ausdruck ist eine Befehl, der zu einen Ergebnis ausgewertet werden kann. Dabei kann das Ergebnis entweder numerisch oder eine Referenz auf ein Objekt sein.

Tabelle 6.2 Softwaremetriken für Variante 1

Reduktionsmethode	Ausdrücke	Anweisungen	Fallunter-scheidungen	Schleifen	Verschachtel-ungstiefe	Zyklomatische Komplexität	Halstead Aufwand
Attach	49	16	5	2	3	4	10011
Copy Object	94	28	5	3	3	4	28740
Create Successor	224	81	15	5	8	14	186046
Delete Object	88	33	7	2	6	6	25026
Delete Relationship	44	18	4	0	1	3	11787
Detach	59	24	7	2	3	6	14468
Freeze Object	111	48	12	4	6	11	53477
Get Alternatives	38	14	3	1	1	2	8665
Get Baseversion	111	40	6	2	2	5	49382
Get CVC	50	25	6	1	2	5	18890
Get Predecessor	38	14	3	1	1	2	8665
Get RelatedObjects (a.v.W)	45	16	2	0	1	1	10368
Get RelatedObjects (i.v.W)	22	7	1	0	0	0	3522
Get Root	30	12	3	1	1	2	6137
Get Successors	38	14	3	1	1	2	8665
Merge Objects	116	55	14	2	3	13	34379
New Object	193	27	5	2	2	4	33080
New Relationship	210	79	6	0	2	5	116729
Pinning	45	15	3	0	1	2	12206
Select	18	7	2	0	1	1	1947
Update Object	30	14	3	0	2	2	5312
ColumnList	5	5	2	1	1	1	462
DisplayedColumn	9	9	3	1	2	2	2091
FromClause	16	11	4	2	2	3	2631
FromItem (a.v.W)	76	45	10	1	4	9	52140
FromItem (i.v.W)	29	29	8	1	3	7	17291
ObjectSelect	8	8	4	0	0	3	1275
Order	6	5	2	1	1	1	552
OrderItem	3	2	1	0	0	0	31
Update	7	5	2	1	1	1	513
UpdateItem	26	22	7	0	2	6	6061
Where	9	9	3	1	2	2	2091
Summe	1847	737	161	38	68	129	732640

te 4 5,3 mal so hoch wie bei Variante 1. Der Faktor, der sich bei einem Vergleich des Halstead-Aufwands ergibt ist 17,9. Dieser Trend ist auch bei den anderen Befehlen zu erkennen, deren Abbildungen sich im Anhang F befinden. Beim Vergleich der Reduktionsmethoden von Variante 1 und Variante 4 konnte bei keiner Operation ein Rückgang der Komplexität festgestellt werden. Bei allen Reduktionsmethoden hat die Komplexität zugenommen. Die geringste Zunahme der Komplexität mit einem Faktor 2 konnte bei der Reduktionsmethode des Knotens `From Item` (außerhalb von Arbeitskontexten) festgestellt werden.

Abbildung 6.2 Vergleich der Operation `New Relationship`

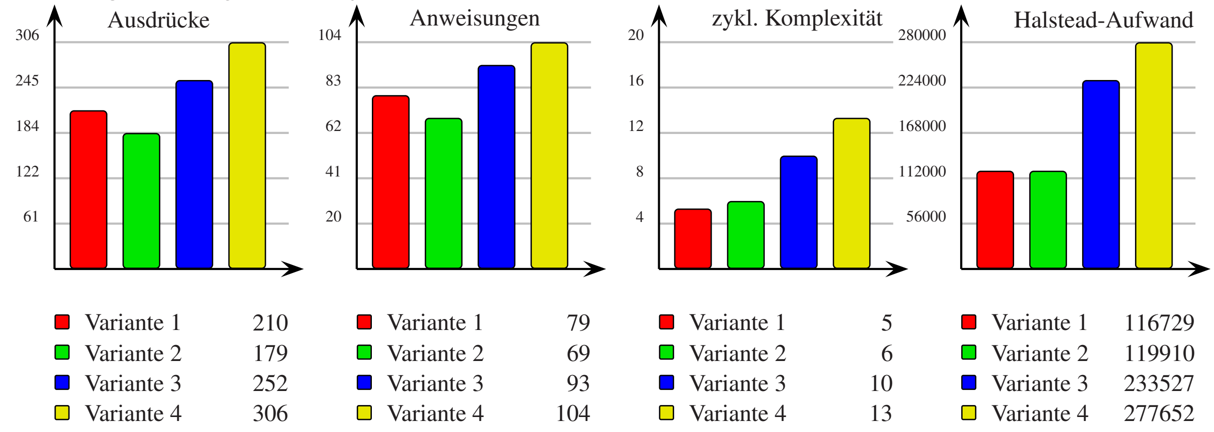


Abbildung 6.3 Vergleich der Operation `Delete Relationship`

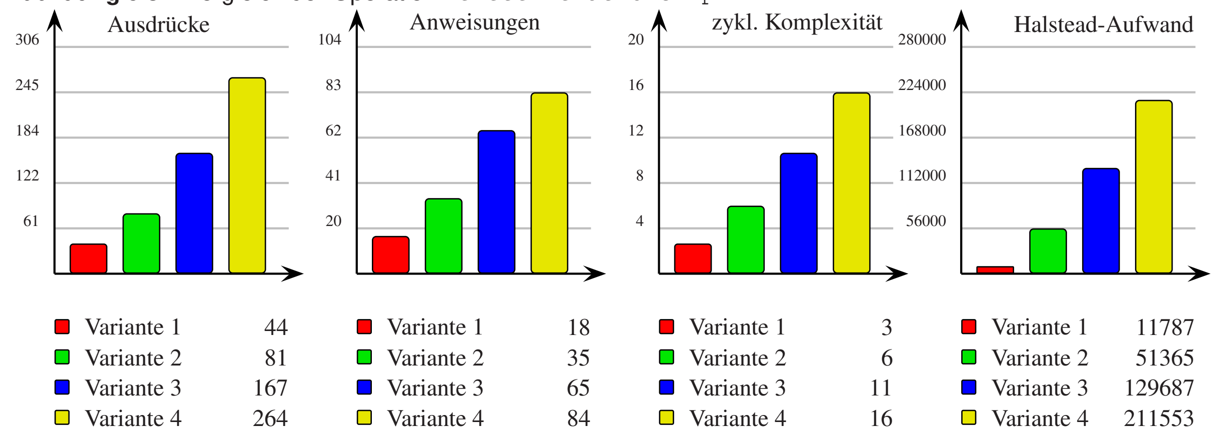


Abbildung 6.4 zeigt die zyklomatische Komplexität für alle implementierten Reduktionsmethoden. Die Reduktionsmethoden, die in den vier Varianten nicht verändert wurden, sind nur einmal eingezeichnet. Für die Reduktionsmethoden, die Aufgrund der Schemaänderung angepasst werden mussten, existiert für jede Variante ein eigener Eintrag. Die Notiz in der Klammer hinter dem Namen des Knotens, der die Reduktionsmethode enthält, gibt die Zugehörigkeit der Reduktionsmethode zu der entsprechenden Variante an. Dabei wird der eben beschriebene Trend, dass die Komplexität der Reduktionsmethoden von Variante 1 zur Variante 4 zunimmt, bestätigt. Die vier senkrechten Striche kennzeichnen die durchschnittliche Komplexität der einzelnen Varianten. Die Reduktionsmethoden, bei denen häufig auf das Informationsmodell zugegriffen wird besitzen eine überdurchschnittliche Komplexität. Zu diesen Reduktionsmethoden

gehört beispielsweise die Reduktionsmethode des Knotens `New Relationship`. Innerhalb dieser Reduktionsmethoden existieren viele Fallunterscheidungen, die je nach Ergebnis eines Zugriffs auf das Informationsmodell dazu führen, dass unterschiedlicher Reduktionscode ausgeführt wird. Die Speicherung von Beziehungen in Abhängigkeit der maximalen Kardinalität ist ein Beispiel für den unterschiedlichen Reduktionscode. Eine unterdurchschnittliche Komplexität, ist bei Knoten feststellbar, bei denen die Reduktion unabhängig vom Informationsmodell durchgeführt wird und der domänenspezifischen Syntaxbaum auf wenige SQL-Anweisungen reduziert werden kann. Die Reduktionsmethoden, in denen einzelne Teile des domänenspezifischen Syntaxbaums nach einem festen Schema ohne Fallunterscheidungen auf Teile des SQL-Syntaxbaums reduziert werden, weisen ebenfalls eine unterdurchschnittliche Komplexität auf. Ein Beispiel dafür ist die Reduktionsmethode des Knotens `ColumnList`. Auf Grundlage der in Tabelle 6.1 vorgestellten Faustregeln für die zyklomatische Komplexität, ist festzustellen, dass alle Reduktionsmethoden trotz der Zunahme der Komplexität wartbar sind.

Analog zur Abbildung 6.4 zeigt Abbildung 6.5 den Vergleich aller Reduktionsmethoden bezüglich des Halstead-Aufwands. Die Reduktionsmethoden bei denen häufig auf das Informationsmodell zugegriffen wird besitzen eine überdurchschnittlichen Halstead-Aufwand. Dazu gehören unter anderem die Methoden zur Reduktion der Anweisungen zum Anlegen und Löschen von Beziehungen. Im Vergleich zur Abbildung 6.4 kann festgestellt werden, dass es keinen direkten Zusammenhang zwischen der zyklomatischen Komplexität und dem Halstead-Aufwand gibt, da die Reduktionsmethoden mit der größten zyklomatischen Komplexität nicht den größten Halstead-Aufwand besitzen. Der durchschnittliche Halstead-Aufwand nimmt jedoch genau so wie die zyklomatische Komplexität von Variante 1 nach Version 4 zu.

Abbildung 6.4 Vergleich der zyklomatischen Komplexität

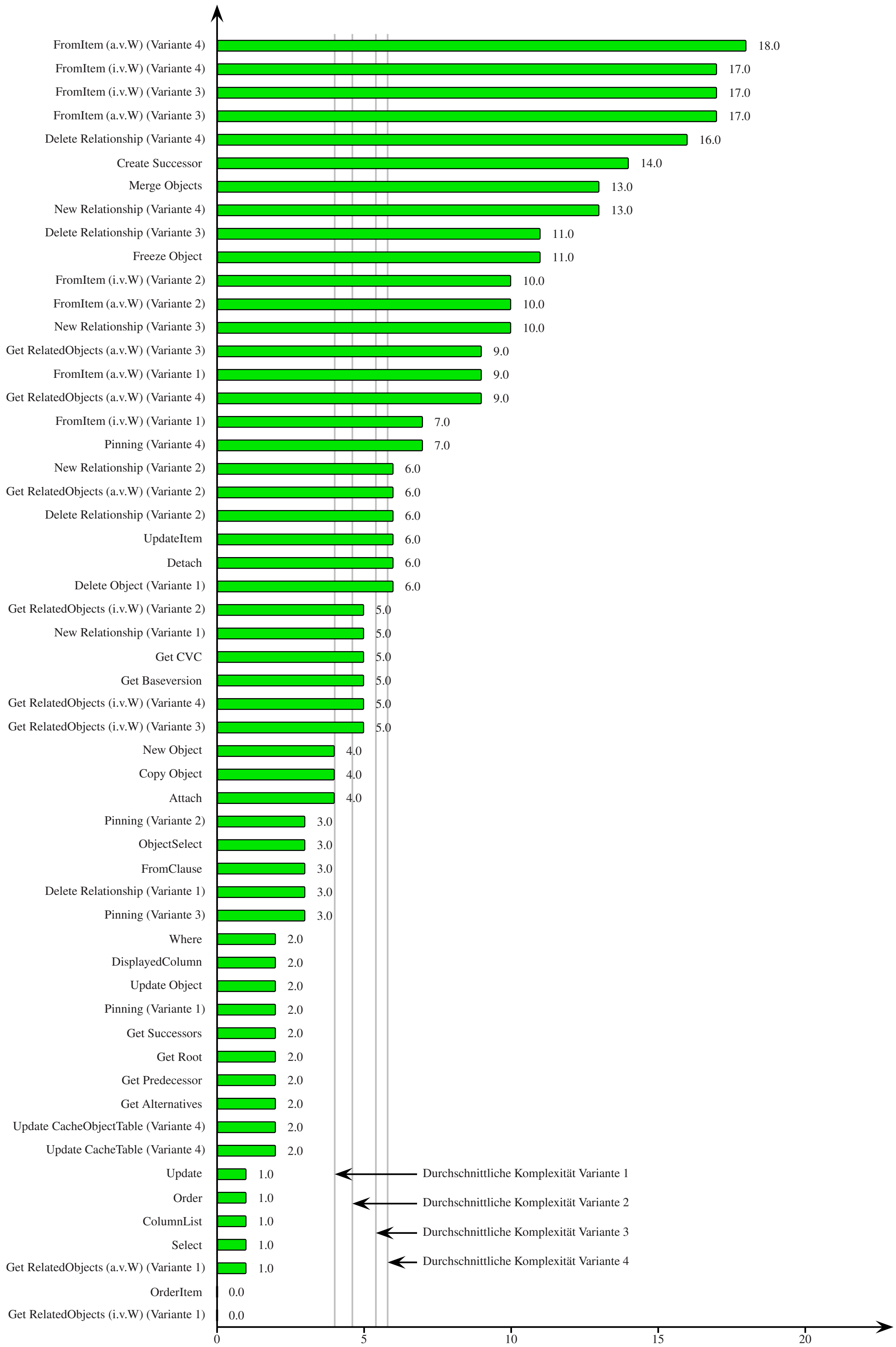
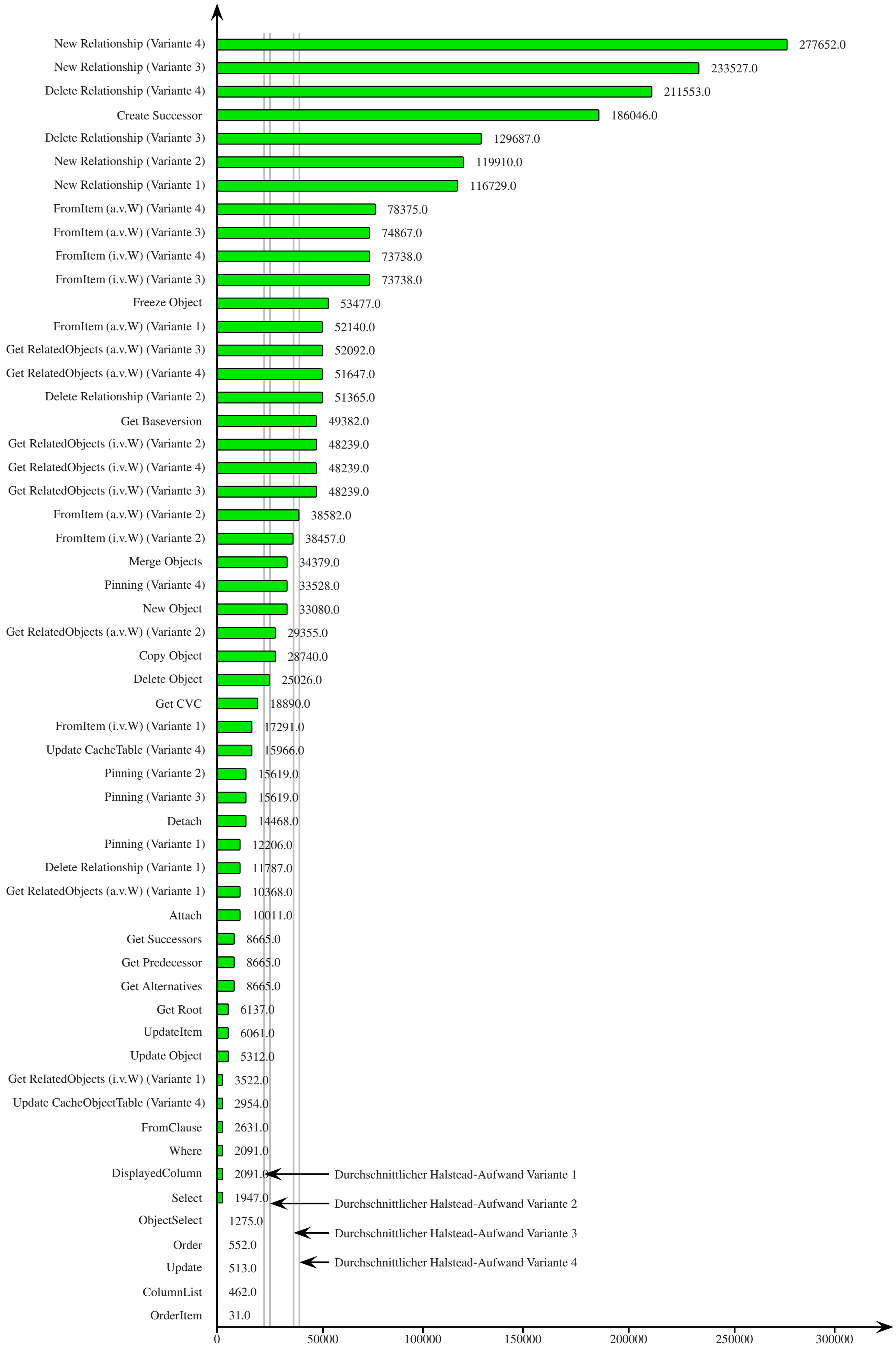


Abbildung 6.5 Vergleich des Halstead-Aufwands



6.2 Bewertung durch Leistungsuntersuchungen

Um die verschiedenen Varianten der JDBC-Treiberimplementierungen durch einen Leistungstest bewerten zu können, ist eine große Menge von DS-DML-Anweisungen notwendig, die auf einem entsprechend großen Datenbestand in der Datenbank ausgeführt werden. Die Größe des Datenbestandes ist beispielsweise entscheidend für die Kosten eines Joins zweier Tabellen. Da unseren Implementierungsvarianten unterschiedliche SQL-DDL Datenbankschemata zugrunde liegen, muss der Datenbestand groß genug sein, um die Unterschiede in der Leistung der Datenbankschemata feststellen zu können. Außerdem ist es wichtig, dass jeder Befehl ausreichend oft ausgeführt wird, um statistische Ausreißer vernachlässigen zu können. Um die Größe des Datenbestandes und eine ausreichend häufige Ausführung der einzelnen Befehle sicher zu stellen, wurde ein Generator für domänenspezifische Anweisungen entwickelt.

6.2.1 Benchmarkgenerator für DS-DML-Anweisungen

Für die Generierung von DS-DML-Anweisungen musste ein Verfahren entwickelt werden, durch das sichergestellt wird, dass die Anweisungen syntaktisch korrekt sind und gültige Aktionen darstellen. Ein Beispiel für eine Anweisung, die nicht fehlerfrei ausgeführt werden kann und somit keine gültige Aktion darstellt, ist die Ausführung der Operation `Update` auf einem Objekt, das den Status *frozen* besitzt. Aus diesem Grund wurde ein auf Vorlagen basierendes Verfahren eingesetzt. Diese Vorlagen enthalten Platzhalter, die dynamisch zur Laufzeit durch gültige Werte aus der Datenbank ersetzt werden. Dadurch kann garantiert werden, dass das referenzierte Objekt in der Datenbank enthalten ist. Der Zugriff auf den aktuellen Datenbestand zur Generierung von DS-DML-Anweisungen ist mit Ausnahme der Regeln mit denen neue Objekte erstellt werden bei allen Regeln notwendig. Bei der Generierung von DS-DML-Anweisungen zum Anlegen neuer Objekte wird lediglich auf Tabellen zugegriffen, in den zufällige Werte für die Nutzdaten gespeichert sind. Der Generator erhält als Eingabe eine Konfigurationsdatei im XML-Format. Diese Datei enthält eine Menge von Regeln, nach denen eine DS-DML-Anweisung generiert wird. Die Beispiele 6.1 und 6.3 zeigen Ausschnitte aus der verwendeten Konfigurationsdatei. Für jede Regel wird über das Attribut *count* festgelegt, wie viele Anweisungen nach dieser Regel generiert werden sollen. Die Auswahl der Regeln erfolgt mit Hilfe eines Zufallszahlengenerators, um die unterschiedlichen Anweisungen zu vermischen. Wenn eine Regel ausgewählt wurde, werden die in der Regel definierten Anweisungen genau einmal ausgeführt, um eine Anweisung zu generieren. Die Ausführung einer Anweisung erfolgt sofort, damit der, bei der Generierung von Anweisungen, benötigte Datenbestand aktuell gehalten wird. Falls die Anweisung fehlerfrei ausgeführt werden konnte wird es für die spätere Leistungsuntersuchung gespeichert und der Zähler, für die Anzahl der nach dieser Regel noch zu generierenden Anweisungen dekrementiert. Falls der Zähler den Wert Null erreicht, wird die gesamte Regel aus dem Speicher entfernt, so dass sie nicht mehr vom Zufallszahlengenerator ausgewählt werden kann. Die DS-DML-Anweisungen lassen sich in Anweisungen zur Datenmanipulation und die für Anfragen unterteilen. Damit Anfrage-Anweisungen für die spätere Leistungsuntersuchung gespeichert werden, müssen sie fehlerfrei ausgeführt werden können und dabei eine nicht leere Ergebnismenge zurückliefern. Diese Bedingung stellt sicher, dass beispielsweise bei einer Navigation über mehrere Beziehungen hinweg, alle notwendigen Joins ausgeführt werden und der Vorgang nicht vorzeitig durch die Datenbank abgebrochen wird. Über das Attribut *autocommit* wird festgelegt, ob jede domänenspezifische Anweisung in einer eigenen Transaktion ausgeführt werden soll. Alternativ werden die Anweisungen, die bei einer Abarbeitung einer Regel generiert wurden, in einer Transaktion ausgeführt. Mit diesem Benchmarkgenerator wurden 141775 DS-DML-Anweisungen für die später durchgeführten Leistungsuntersuchungen generiert. Dafür wurden 26 Stunden benötigt, da die Generierung von Anfragen, bei denen über bis zu vier Beziehungen navigiert wird, sehr aufwändig ist. Viele Versuche derartige Anweisungen zu generieren, führten zu einer leeren Ergebnismenge und wurden somit nicht für die Leistungsuntersuchung gespeichert.

Beispiel 6.1 Ausschnitt aus der Konfigurationsdatei

```
<rule id="4" count="10000" autocommit="false">
    <statement type="update">
        CREATE NEW OBJECT Task ( name , description , startDate , dueDate )
        VALUES ( $value1$ , $value2$ , $value3$ , $value4$ )
    </statement>
    <statement type="update">
        CREATE NEW RELATIONSHIP contains_isPart WHERE contains.GID = $gid$
        AND isPart.GID = $returnvalue1$
    </statement>
    <select>
        <query sqltype="sql"> SELECT value as value1 FROM string</query>
        <variable type="String">value1</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT value as value2 FROM string</query>
        <variable type="String">value2</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT value as value3 FROM date</query>
        <variable type="String">value3</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT value as value4 FROM date</query>
        <variable type="String">value4</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT globalId as gid FROM Objecttype_Offer</query>
        <variable type="int">gid</variable>
    </select>
</rule>
```

Durch Regel 4 im Beispiel 6.1 werden zwei domänenspezifische Anweisungen erzeugt, die in einer gemeinsamen Transaktion ausgeführt werden. Mit dieser Regel wird ein neues Objekt vom Typ *Task* angelegt und anschließend eine neue Beziehung zwischen diesem Objekt und einem existierenden Objekt vom Typ *Offer* angelegt. Auf Grund der im Informationsmodell definierten minimalen Kardinalitäten muss jedes Objekt vom Typ *Task* mit einem Objekt vom Typ *Offer* in Beziehung stehen. Das Anlegen eines Objekts vom Typ *Task* ohne eine Beziehung zu einem Objekt vom Typ *Offer* würde vom domänenspezifischen Datenbanktreiber verhindert, da dort die im Informationsmodell definierten minimalen und maximalen Kardinalitäten überprüft werden. Die Regelelemente (`<rule>`) in der Konfigurationsdatei enthalten die Elemente `<statement>` und `<select>`. Im Element `statement` wird das Grundgerüst für eine zu generierende domänenspezifische Anweisung gespeichert. Dieses Grundgerüst enthält jedoch noch Platzhalter, die durch Dollarzeichen eingeschlossen sind. Durch Anfragen auf dem aktuellen Datenbestand müssen die Werte für die Platzhalter noch bestimmt werden. Mit Hilfe des Attributs *type* im Element `statement` wird festgehalten, ob es sich um einen Anfragebefehl oder einen Befehl zur Datenmanipulation handelt. Beispiel 6.2 zeigt DS-DML-Anweisungen, die mit der im Beispiel 6.1 vorgestellten Regel erzeugt wurden.

Beispiel 6.2 DS-DML-Anweisungen, die mit der im Beispiel 6.1 vorgestellten Regel erzeugt wurden

```
CREATE NEW OBJECT Task (name,description,startDate,dueDate ) VALUES ('Diplomarbeit','DSL untersuchen',2003-09-17,2004-03-16)

CREATE NEW RELATIONSHIP contains_isPart WHERE contains.GID = 3350001 AND isPart.GID = 24080001
```

Die Werte, durch die die Platzhalter in den domänenspezifischen Anweisungen ersetzt werden, werden mit Hilfe der Anweisungen bestimmt, die im Element `select` beschrieben werden.

Beispiel 6.3 Ausschnitt aus der Konfigurationsdatei

```
<rule id="30" count="500"  autocommit="true">
    <statement type="update">
        PIN Costs WHERE GID = $globalid$ IN CVC Task WHERE GID = $gid$ ROLENAME ratedCosts
    </statement>
    <select>
        <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_Task</query>
        <variable type="int">gid</variable>
    </select>
    <select>
        <query sqltype="dssql">GET CVC FROM Task WHERE GID = $gid$ ROLENAME ratedCosts</query>
        <variable type="int">globalid</variable>
    </select>
</rule>
```

Dabei enthält das Element `query` die notwendige SQL-Anweisung. Über das Attribut *sqltype* wird angegeben, ob es sich um eine herkömmliche SQL Anweisung oder um eine domänenspezifische Anweisung handelt, mit der die notwendigen Informationen bestimmt werden. Ein Beispiel für die Verwendung einer domänenspezifischen Anweisung zur Informationsbeschaffung beinhaltet Regel 30 im Beispiel 6.3. Bei dieser Regel wird mit Hilfe des Befehls `Get CVC` zunächst die Kandidatenmenge bestimmt, um anschließend ein Objekt dieser Menge mit der Operation `PIN` als bevorzugte Version zu definieren. Das Ergebnis der Anwendung der Regel aus Beispiel 6.3 ist in Beispiel 6.4 zu sehen. Neben dem Element `query` enthält das Element `select` auch noch Elemente mit dem Namen `variable`. Die dort definierten Variablen müssen aus der Ergebnismenge der zuvor ausgeführten Anfrage entnommen werden. Das Attribut *type* gibt den benötigten Datentyp an. Da auch in den Elementen vom Typ `select`, analog zu den domänenspezifischen Anweisungen, Platzhalter enthalten sein können, werden diese Anfragen in der Reihenfolge ihrer Definition ausgeführt.

Beispiel 6.4 DS-DML-Anweisung, die mit der im Beispiel 6.3 vorgestellten Regel erzeugt wurde

```
PIN Costs WHERE GID = 2320001 IN CVC Task WHERE GID = 24770203  ROLENAME ratedCosts
```

Für die Leistungsbewertung wurden auf diese Art mit 47 unterschiedlichen Regeln insgesamt 141775 DS-DML-Anweisungen generiert. Die vollständige Konfigurationsdatei, die für die Generierung verwendet wurde, befindet sich im Anhang D. Die generierten DS-DML-Anweisungen lassen sich in unterschiedliche Kategorien einteilen. Tabelle 6.3 zeigt den prozentualen Anteil der einzelnen Kategorien an der Leistungsuntersuchung.

6.2.2 Durchführung

Ausgangspunkt für die Leistungsuntersuchung der vier Varianten war jeweils eine Datenbank, in der mit Hilfe von SQLInteract das zur Variante passende Datenbankschema angelegt wurde. Die Daten aus dem Informationsmodell wurden in die dafür vorgesehenen Systemtabellen eingetragen. Zum Start der Leistungsuntersuchung haben die Datenbanken keinerlei Daten eines Versionierungssystems enthalten. Der Datenbestand, der für die Ausführung bestimmter Operationen notwendig war, wurde als Bestandteil der Leistungsmessung angelegt. Die Semantik der 141775 ausgeführten domänenspezifischen Anweisungen entspricht denen der im Kapitel 3.6 vorgestellten Operationen. Während der Ausführung der einzelnen DS-DML-Anweisungen wurden unterschiedliche Messwerte erfasst. Es wurde die Zeit gemessen, die der Parser benötigt hat, um die Zeichenkette in einen domänenspezifischen Objektbaum zu überführen. Diese Zeit wurde *Parser-Zeit* genannt. Die drei folgenden Zeiten wurden innerhalb der Reduktionsme-

Tabelle 6.3 Anteile der einzelnen Kategorien an der Leistungsuntersuchung

Kategorie	Anzahl	Anteil
Create New Object	26250	18,5 %
Create New Relationship	46100	32,5 %
Attach / Detach	5325	3,8 %
CreateSuccessor	20500	14,5 %
Copy Object	20000	14,1 %
Freeze Object	1100	0,8 %
Update Object	5000	3,5 %
Delete Object	5000	3,5 %
Get	2500	1,8 %
Merge	500	0,4 %
Get CVC	1500	1,1 %
Pinning	1500	1,1 %
Select	6500	4,6 %
Summe	141775	100 %

thoden gemessen. Zur *Reduktionszeit* gehören alle Aktionen, die ausgeführt werden, um einen SQL-Syntaxbaum zusammenzubauen. Dazu gehören auch Zugriffe auf das Informationsmodell sowie die Auswertung bereits ausgeführter Anfragen an die Datenbank. Des weiteren wurde die Zeit festgehalten, die benötigt wird, um den SQL-Syntaxbaum in eine textuelle Repräsentation zu überführen, die dann zur Ausführung an die Datenbank geschickt wird. Diese Zeit wird als *Rendering-Zeit* bezeichnet. Die Zeit, die die Datenbank zu Ausführung der erzeugten Anweisungen benötigt, wurde ebenfalls gemessen und trägt den Namen *SQL-Ausführungszeit*.

Die *Reduktionszeit*, *Rendering-Zeit* und die *SQL-Ausführungszeit* wurden ebenfalls bei der Überprüfung der im Informationsmodell definierten minimalen und maximalen Kardinalitäten gemessen, um die dadurch zusätzlich entstandenen Kosten mit denen der eigentlichen Befehlsausführung vergleichen zu können.

Durch diese Leistungsuntersuchungen der unterschiedlichen Implementierungen sollen verschiedene Aspekte untersucht werden. Eine zentrale Frage, die sich bei der Entwicklung einer domänenspezifischen Sprache stellt, ist, wie hoch die zusätzlichen Kosten sind, die bei der Nutzung der domänenspezifischen Spracherweiterungen anfallen. Des weiteren wird untersucht, welchen Einfluss die Veränderungen am Datenbankschema auf die unterschiedlichen Zeiten haben. Wie bereits erwähnt, werden von den Datenbanktreibern die im Informationsmodell definierten minimalen und maximalen Kardinalitäten der Beziehungen überprüft. Durch die Leistungsuntersuchung soll geklärt werden, wie hoch die Kosten einer derartigen Überprüfung sind.

Die Leistungsuntersuchung wurden auf einem Linux-Server mit folgender Konfiguration durchgeführt:

Hardwarekonfiguration

Prozessor:	Intel Pentium 4 CPU 3.00GHz HT
Hauptspeicher:	512 MByte DDR-RAM, 400 MHz
Massenspeicher:	80 GByte, 7.200 U/Min, Ultra-ATA 100

Softwarekonfiguration

Betriebssystem:	Mandrake Linux 9.2
Kernel:	Linux 2.4.22
Java-Laufzeitumgebung:	Sun-SDK 1.4.2_03
Datenbanksystem:	IBM DB2 8.1.0

Um negative Einflüsse auf die Leistungsuntersuchung zu minimieren, wurden alle nicht benötigten Systemdienste deaktiviert.

6.2.3 Auswertung

Die bei der durchgeführten Leistungsuntersuchung gesammelten Messwerte wurden statistisch ausgewertet und sind im Anhang G in tabellarischer Form zu finden. Zum Vergleich der unterschiedlichen JDBC-Treiberimplementierungen wurden verschiedene Abbildungen erzeugt, die sich im Anhang H befinden. In diesem Abschnitt werden einige dieser Abbildungen verwendet, um die im vorherigen Abschnitt vorgestellten Fragestellungen beantworten zu können.

Mit der Nutzung einer domänenspezifischen Sprache für Versionierungssysteme ist kein Performanzgewinn verbunden, da die domänenspezifischen Anweisungen auf herkömmliche SQL-Anweisungen reduziert werden. Der Vorteil der domänenspezifischen Sprache besteht in einer komfortableren Schnittstelle für Anwendungsentwickler. Um die Kosten, die bei der Nutzung einer domänenspezifischen Schnittstelle zusätzlich zur Ausführung der eigentlichen SQL-Anweisungen entstehen, abschätzen zu können, wurden Tortendiagramme erstellt, die die unterschiedlichen Anteile an der Gesamtausführungszeit eines domänenspezifischen Befehls enthalten. Die Gesamtausführungszeit setzt sich damit aus der *Parser-Zeit*, der *Reduktionszeit*, der *Rendering-Zeit* so wie der *SQL-Ausführungszeit* zusammen.

Die Tortendiagramme wurden für die verschiedenen Kategorien der domänenspezifischen Anweisungen erstellt. Abbildung 6.6 zeigt die Zusammensetzung der Gesamtausführungszeit für das Erstellen neuer Beziehungen zwischen zwei Objekten. Bei dieser Kategorie handelt es sich um eine übergeordnete Kategorie, die die Messwerte für das Erstellen der unterschiedlichen Beziehungsarten zusammenfasst. Hierbei ist zu erkennen, dass der Anteil der zusätzlich zur SQL-Ausführungszeit entstehenden Kosten maximal 1,5 Prozent beträgt. Dieser Anteil ist beispielsweise bei der Operation `Create Successor`, wie in Abbildung 6.7 gezeigt, deutlich höher. Dieser erhöhte Aufwand für die Reduktion lässt sich durch die notwendigen Zugriffe auf das Informationsmodell erklären. So muss bei Operation `Create Successor` beispielsweise kontrolliert werden, wie viele Nachfolgerversionen ein Objekt besitzen darf oder ob die Operation an andere Objekte propagiert werden muss. Derartige Informationen sind im Informationsmodell definiert und müssen zur Laufzeit der Reduktionsmethode nachgefragt werden. Aus diesen Gründen werden bei den vier Varianten zwischen 4,4 und 5,6 Prozent der Ausführungszeit für die Reduktion benötigt. Ein derartiger Wert wurde auch bei der Operation `Create New Object` beobachtet. Bei zwei Drittel der domänenspezifischen Befehlen liegt der Anteil der Reduktion zwischen 0,5 und 3 Prozent. Der größte Reduktionsaufwand wurde bei der Operation `Merge Objects` gemessen. Da diese Reduktionsmethode die komplette Logik zum Zusammenführen der unterschiedlichen Objektversionen enthält, erklärt sich der, in Abbildung 6.8 gezeigte, Anteil von 8,8 und 9,3 Prozent der Gesamtausführungszeit. Im Vergleich dazu betragen die Reduktionszeiten bei `Select`-Operationen, wie in Abbildung 6.9 gezeigt, zwischen 0,2 und 0,5 Prozent. Die Zeit, die für das Parsen der Zeichenkette und die Erzeugung des domänenspezifischen Syntaxbaumes, auf dem die Reduktionsmethode aufgerufen wird, benötigt wird, ist sehr gering und liegt im Durchschnitt aller ausgeführten Befehle bei 0,1 Prozent. Bei Variante 4 beträgt der durchschnittliche Anteil der Reduktionszeit 1,8 Prozent und der für die Rendering-Zeit 0,4 Prozent.

Neben dem relativen Vergleich der gemessenen Werte wurde auch ein Vergleich der absoluten Werte durchgeführt, um die Auswirkungen der unterschiedlichen Datenbankschemata und Datenbanktreiberimplementierungen bewerten zu können. Dieser Vergleich wurde ebenfalls für

Abbildung 6.6 Zusammensetzung der Gesamtausführungszeit: CREATE NEW RELATIONSHIP

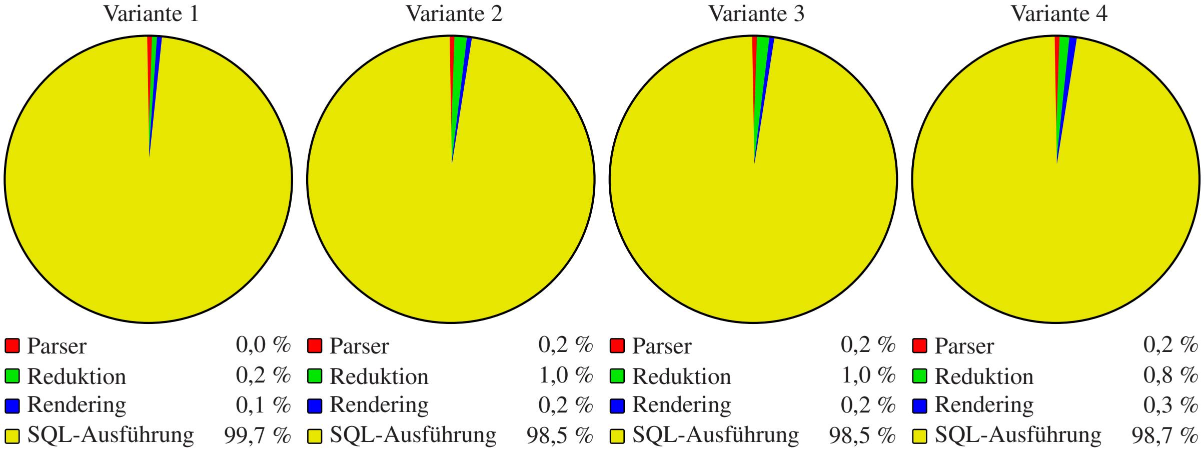


Abbildung 6.7 Durchschnittszeitenvergleich: CREATE SUCCESSOR

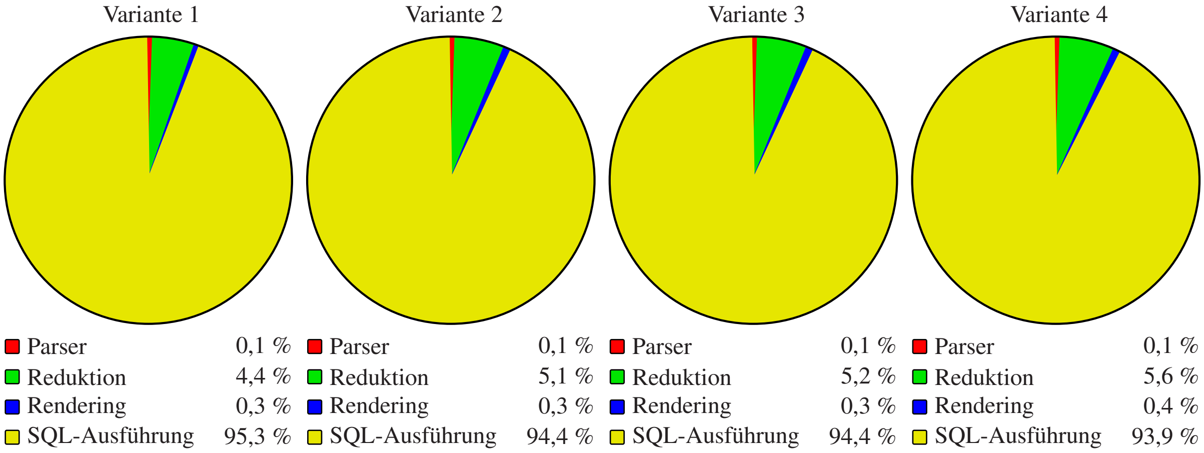
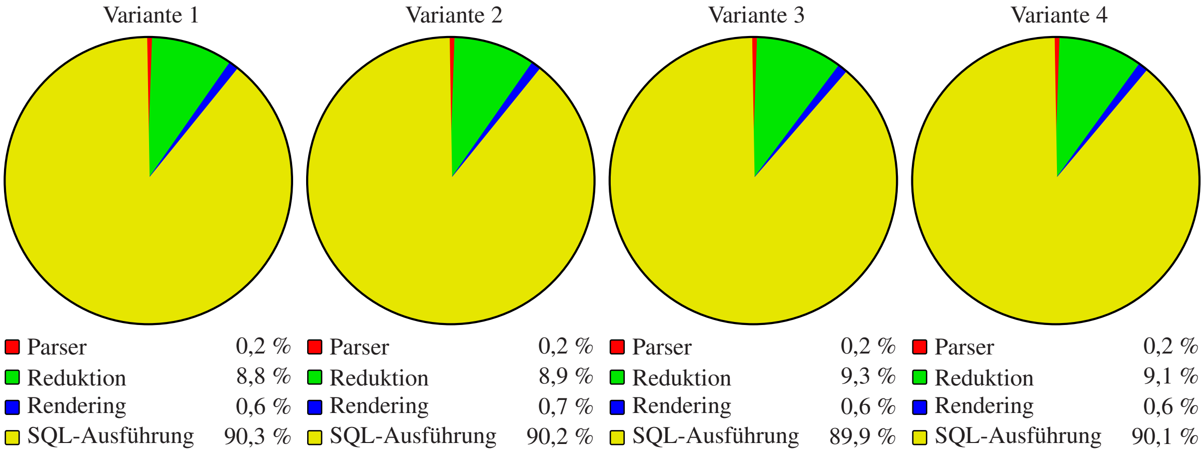
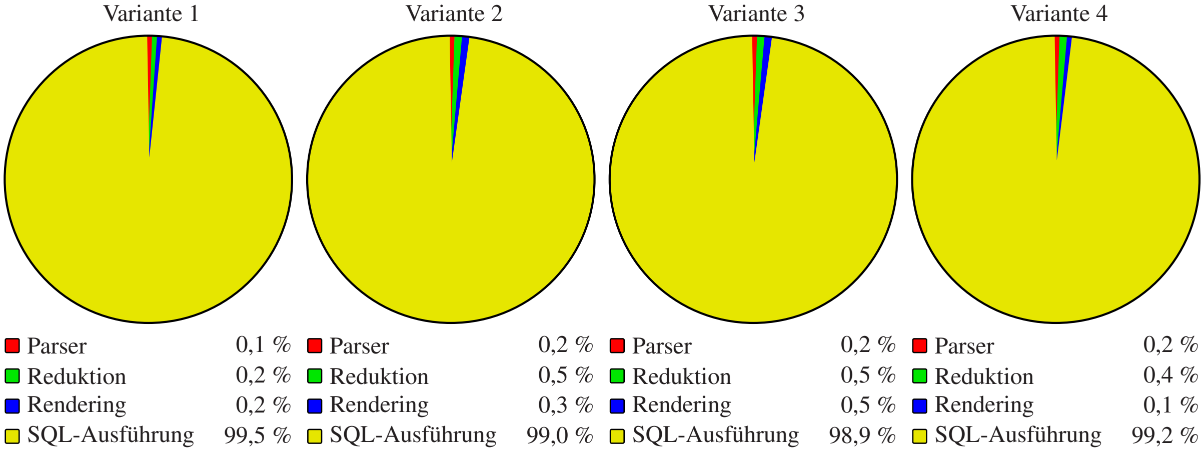


Abbildung 6.8 Durchschnittszeitenvergleich: MERGE OBJECTS



die unterschiedlichen Kategorien von DS-DML-Anweisungen durchgeführt. Abbildung 6.10 zeigt eine Gegenüberstellung der Zeiten für die Reduktion, das Rendering sowie die SQL-

Abbildung 6.9 Durchschnittszeitenvergleich: SELECT



Ausführungszeit der vier untersuchten Varianten bei allen Befehlen. Bei den Diagrammen, die sich in der Abbildung befinden, wurde jeweils der maximale Wert im Vergleich der vier Varianten auf einhundert Prozent gesetzt und die Werte der anderen Varianten entsprechend angepasst. Im Vergleich der Gesamtzeiten hat die Variante 1 eindeutig am schlechtesten abgeschnitten, da die Gesamtzeit für die Ausführung der Leistungsuntersuchung fast doppelt so groß ist wie bei Variante 4. Bei der durchgeführten Leistungsuntersuchung hat sich die Variante 2 als schnellste Variante herausgestellt. Dieses Ergebnis ist jedoch darauf zurückzuführen, dass 7,5 Prozent der domänenspezifischen Befehle, die bei der Leistungsmessung verwendet wurden, Befehle für Anfragen im Versionierungssystem gewesen sind. Die vorgenommenen Optimierungen zwischen den verschiedenen Varianten dienten zur Beschleunigung der Anfragebearbeitung. Dabei wurden auch höhere Kosten für das Anlegen von Objekten und Beziehungen in Kauf genommen. Abbildung 6.11 zeigt wie sich die vorgenommenen Optimierungen bei Select-Anweisungen ausgewirkt haben. Dabei ist die Variante 4 eindeutig die Schnellste. Da davon auszugehen ist, dass die Quote der Anfragen in einem Versionierungssystem deutlich mehr als sieben Prozent beträgt, kann die Variante 4 als die Schnellste angesehen werden. Der Mehraufwand der bei der Beschleunigung der Anfragen in Kauf genommen wird, ist in Abbildung 6.12 zu erkennen. Da bei Veränderungen an den Beziehungen mit mindestens einem gleitenden Beziehungsende zwischen zwei Objekten die Materialisierung der vorausgewählten oder anhand einer Regel ausgewählten Objektversion aktualisiert wird, werden zusätzliche SQL-Anweisungen benötigt, die sich in einer größeren SQL-Ausführungszeit widerspiegeln. Anhand des Vergleichs der *Rendering-Zeiten* der SQL-Syntaxbäume in Abbildung 6.11, ist erkennbar, dass die Select-Anweisungen bei der Variante 4 deutlich einfacher sind. Darauf wurde bereits in Abschnitt 5.4 hingewiesen. Die zunehmende zyklomatische Komplexität und der wachsende Halstead-Aufwand der Reduktionsmethoden steht nicht mit der für die Reduktion benötigten Zeit in Beziehung.

Abbildung 6.13 zeigt die Beschleunigung der einzelnen Befehle die mit der Variante 4 im Vergleich zur Variante 2 erzielt wird. Dabei wird erkennbar, dass Anfragen beschleunigt werden. Für das Anlegen von Objekten und Beziehungen wird mehr Zeit benötigt. In Abbildung 6.14 sind die Befehle der Variante 4 nach ihren Kosten sortiert angeordnet.

Bisher wurden nur die Kosten für die Ausführung der domänenspezifischen Befehle betrachtet. Die entwickelten Datenbanktreiber unterstützen zusätzlich die Überprüfung der im Informationsmodell definierten minimalen und maximalen Kardinalitäten der Beziehungen. Anhand von

Abbildung 6.10 Zeitvergleich: Summe aller Befehle

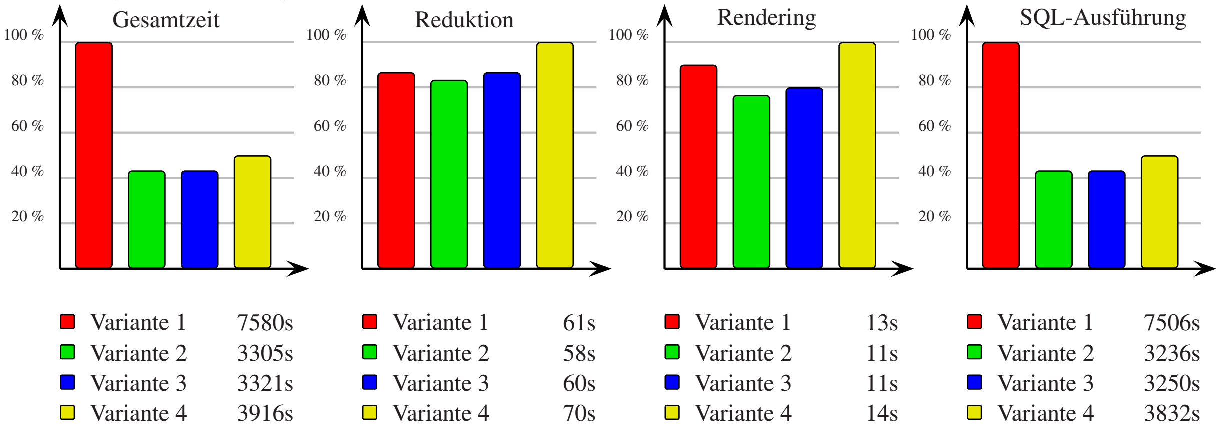


Abbildung 6.11 Zeitvergleich: SELECT

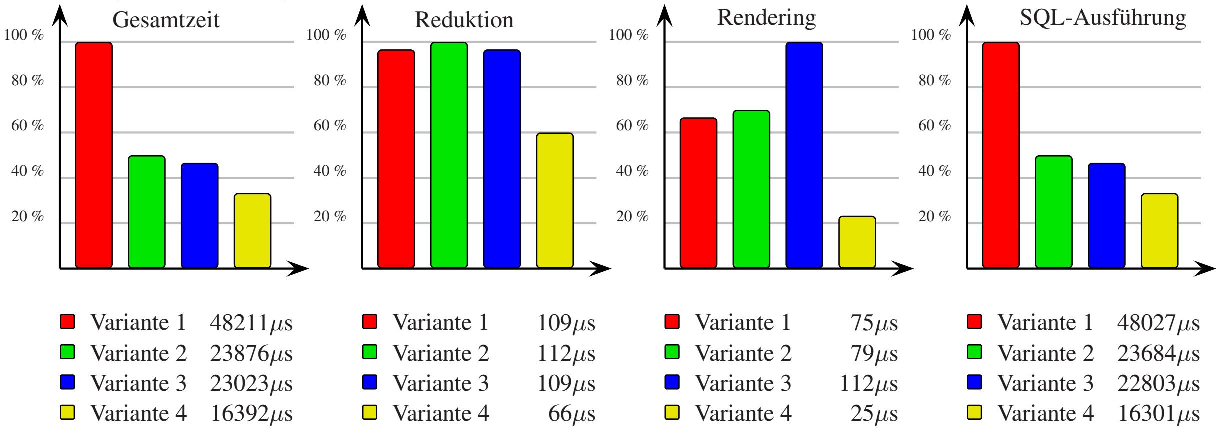
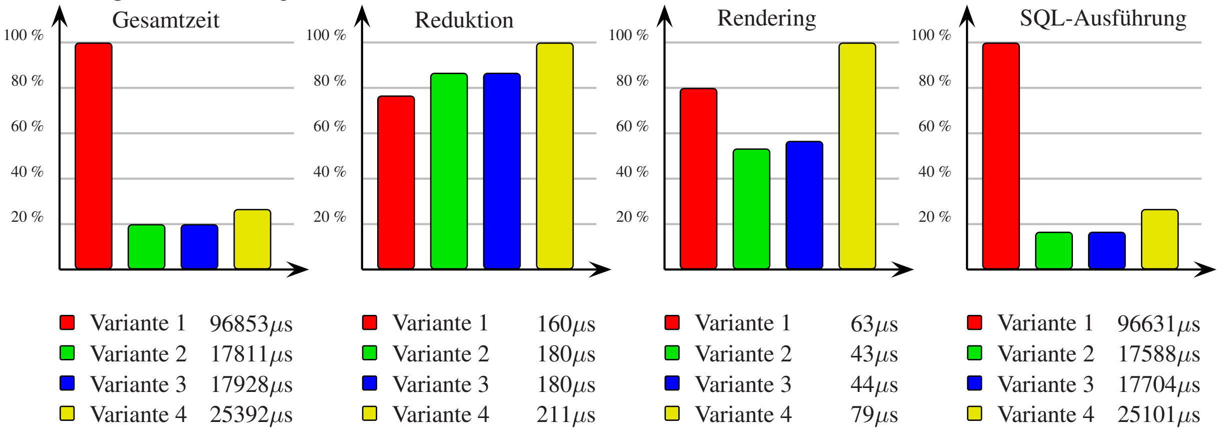
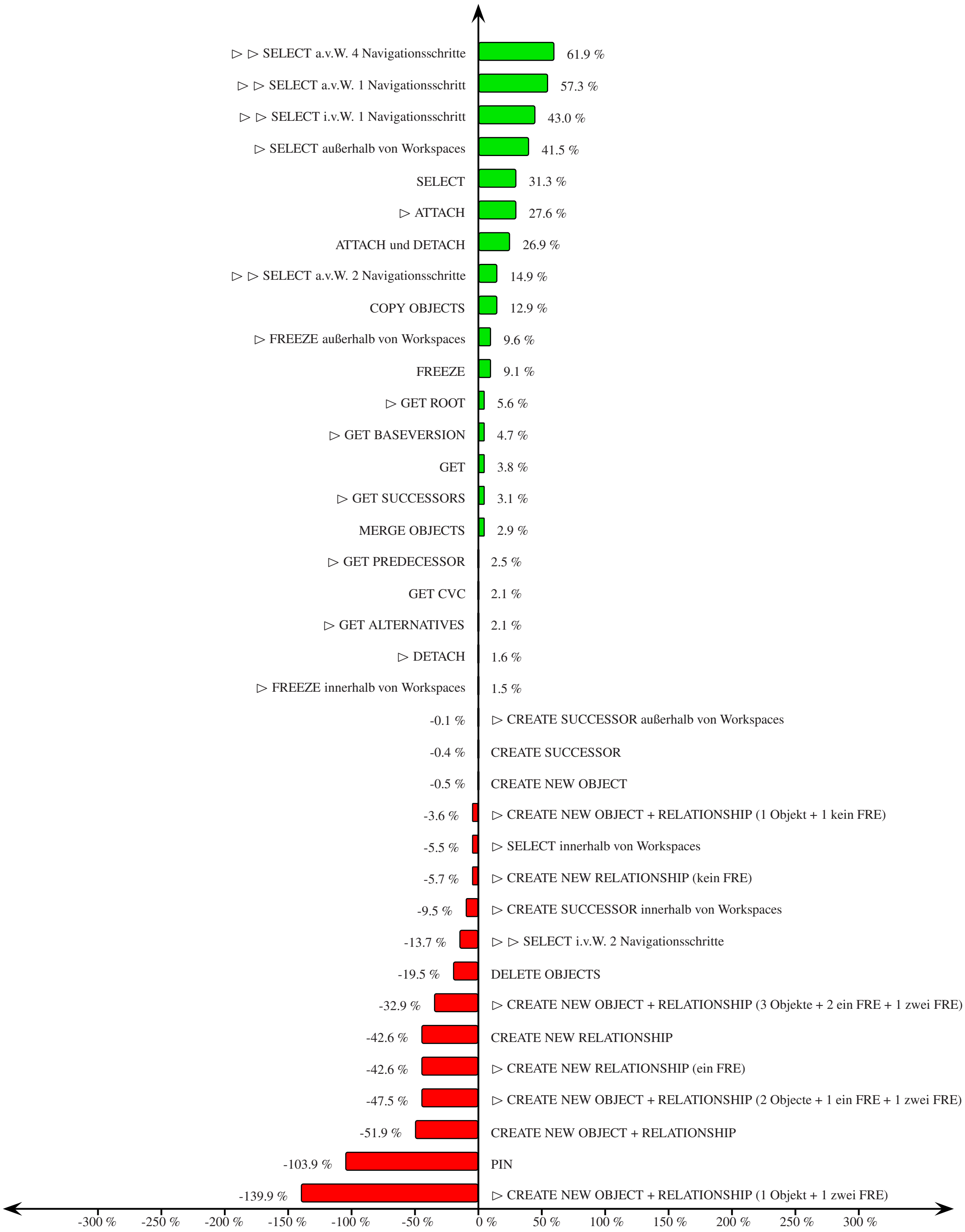


Abbildung 6.12 Zeitvergleich: CREATE NEW RELATIONSHIP



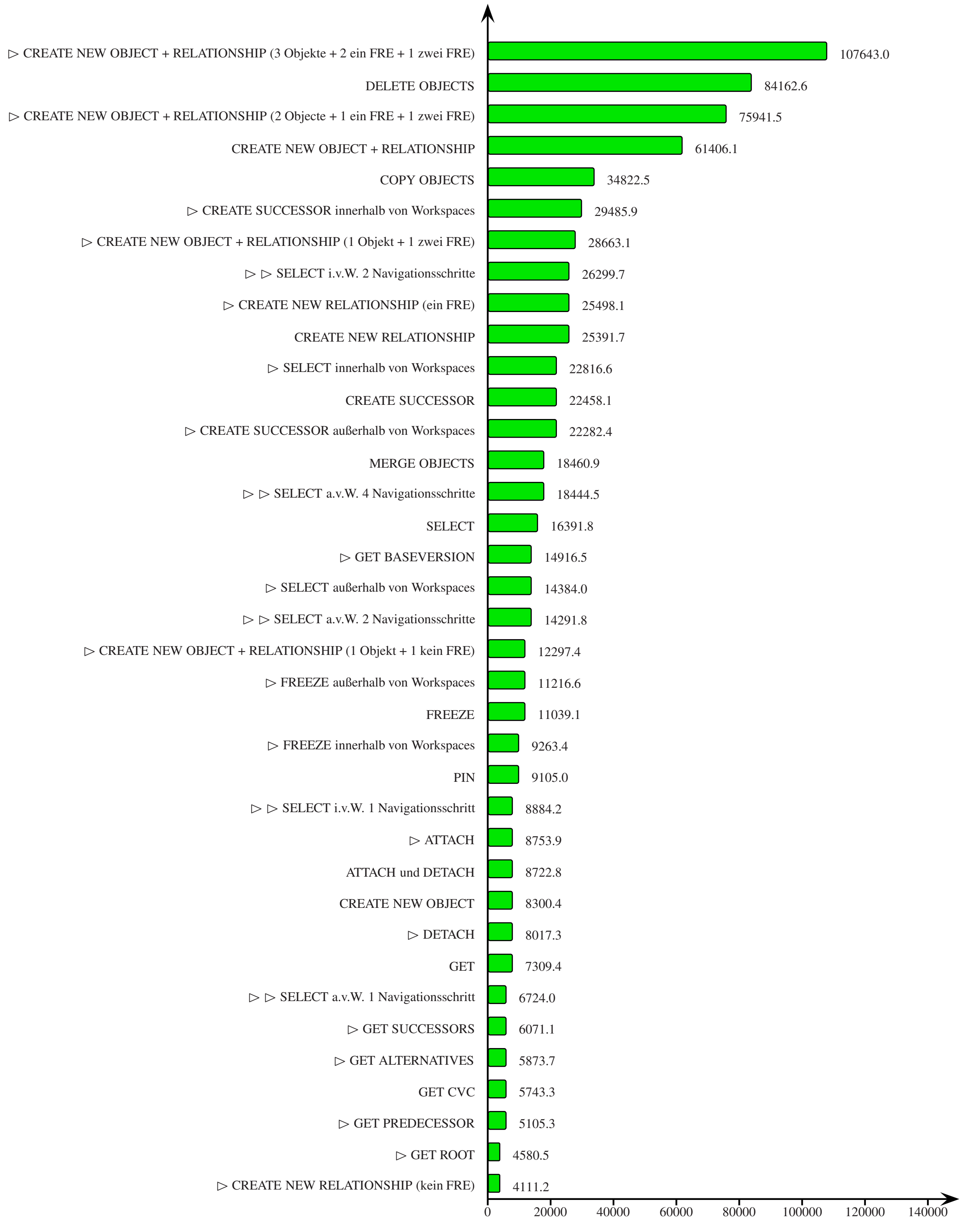
zwei Abbildungen soll die Frage nach der Höhe der Kosten für eine derartige Überprüfung beantwortet werden. Abbildung 6.15 zeigt, dass der Aufwand für die Überprüfung bei Varian-

Abbildung 6.13 Beschleunigung der Variante 4 im Vergleich zu Variante 2



te 1 am größten ist. Durch die Verteilung der Beziehungsinformationen auf unterschiedliche Tabellen wird die Überprüfung wesentlich beschleunigt. Auch bei der Überprüfung der Kardinalitäten schneidet Variante 4 am besten ab. Die Diagramme in Abbildung 6.16 geben das Verhältnis zwischen der eigentlichen Ausführungszeit der Anweisung und der anschließenden Überprüfung der minimalen und maximalen Kardinalitäten an. Der Anteil für die Überprüfung

Abbildung 6.14 Vergleich der Kosten der einzelnen Befehle bei Variante 4



beträgt bei Variante 1 mehr als 60 Prozent. Meiner Meinung nach kann dieser Wert bei einer Anwendung die große Datenmengen effizient verarbeiten muss, nicht toleriert werden. Bei Variante 4 liegt dieser Anteil nur noch bei etwa 20 Prozent.

Abbildung 6.15 Zeitvergleich bei der Kardinalitätenkontrolle: Summe aller Befehle

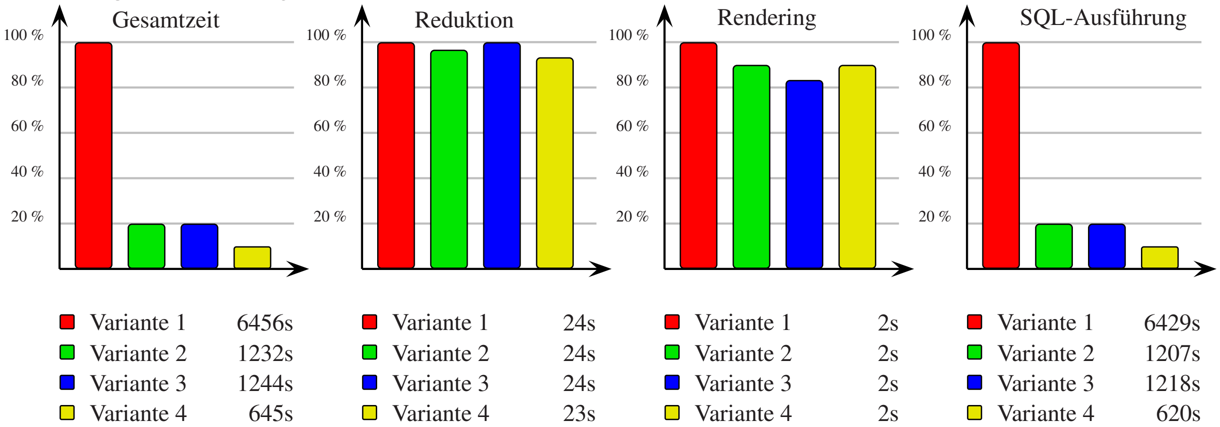
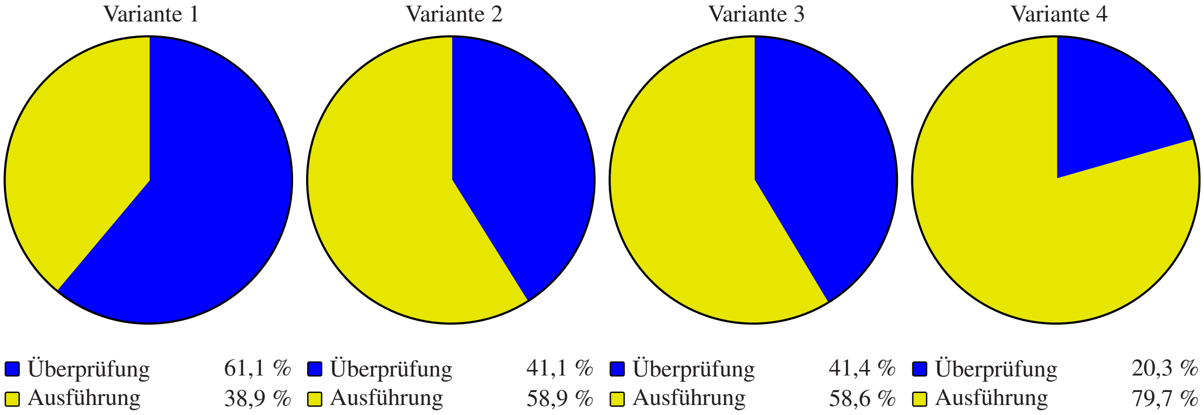


Abbildung 6.16 Kostenanteile: Summe aller Befehle



6.2.4 Untersuchungen mit gespeicherten Prozeduren

Wie bereits in Kapitel 4.2.3 beschrieben wurde, kann die benötigte Anwendungslogik entweder in einem domänenspezifschen Datenbanktreiber gekapselt oder mit Hilfe von objektrelationalen Erweiterungen komplett in der Datenbank realisiert werden. Bei den durchgeführten Leistungsuntersuchungen, fiel auf, dass eine DS-DML-Anweisung bei Variante 4 auf bis zu 338 SQL-Anweisungen reduziert wurde. Dies hat zur Folge, dass ein großer Kommunikationsaufwand zwischen dem domänenspezifischen Datenbanktreiber und der Datenbank besteht.

Mit dem Einsatz von gespeicherten Prozeduren, wurde in einer zweiten Leistungsuntersuchung versucht, diese Kommuniationskosten auf die Kosten für den Aufruf einer einzigen gespeicherten Prozedur zu senken. Dazu wurde eine Auswahl von Operationen, die in einem Versionierungssystem ausgeführt werden können, durch gespeicherten Prozeduren realisiert. Das zu grunde liegende Datenbankschema entsprach dem Schema der Variante 4. Im Gegensatz zu den generischen Reduktionsmethoden in den implementierten Datenbanktreibern muss bei den gespeicherten Prozeduren eine Prozedur für jeden Anwendungsfall erstellt bzw. generiert werden. Dies hat unteranderem den Vorteil, dass die Anfragen an das Informationsmodell entfallen können und die Informationen fest in der gespeicherten Prozedur enthalten sind. Dadurch muss beispielsweise die Operation `CreateSucessor` für jeden versionierbaren Objekttyp, der im

Informationsmodell definiert wurde, bereitgestellt werden. Listing 6.1 zeigt eine gespeicherte Prozedur zum Anlegen von Beziehungen zwischen Objekten vom Typ *Task* und *Cost*. Dabei ist erkennbar, dass keinerlei Anfragen durchgeführt werden, um festzustellen, dass es sich hier um einen Beziehungstyp mit zwei gleitenden Beziehungsenden handelt. Die Informationen über den Typ der Beziehung als auch über die maximalen Kardinalitäten der Beziehungen, müssen bei der Erstellung der Prozedur berücksichtigt werden.

Listing 6.1 Gespeicherte Prozedur zum Anlegen von Beziehungen zwischen Objekten vom Typ *Task* und *Cost*

```
CREATE PROCEDURE NewRelationship_ratedCosts_causedBy
                        (IN ratedCostsGIDv Integer,
                        IN causedByGIDv Integer)
                        LANGUAGE SQL
BEGIN ATOMIC
        DECLARE cachePinned Integer;
        DECLARE pinnedValue Integer;
        DECLARE sqlcode Integer DEFAULT 0;
        DECLARE newCVCGID Integer;

        DECLARE ratedCostsOIDv Integer;
        DECLARE ratedCostsVIDv Integer;
        DECLARE causedByOIDv Integer;
        DECLARE causedByVIDv Integer;
        SET ratedCostsOIDv = FLOOR(ratedCostsGIDv/10000);
        SET ratedCostsVIDv = ratedCostsGIDv - (ratedCostsOIDv*10000);
        SET causedByOIDv = FLOOR(causedByGIDv/10000);
        SET causedByVIDv = causedByGIDv - (causedByOIDv*10000);

        -- CVC Tabelle
        INSERT INTO Rel_CVC_ratedCosts_causedBy (ratedCostsGID,ratedCostsOID,ratedCostsVID,causedByGID,causedByOID,causedByVID)
        VALUES (ratedCostsGIDv,ratedCostsOIDv,ratedCostsVIDv,causedByGIDv,causedByOIDv,causedByVIDv);

        -- CVC Cache 1
        SELECT cache_pinned_ratedCosts INTO cachePinned
        FROM ObjectType_Costs
        WHERE globalID=causedByGIDv;

        IF cachePinned IS NULL OR cachePinned = 0 THEN
                UPDATE ObjectType_Costs
                SET cache_ratedCosts=ratedCostsGIDv,cache_pinned_ratedCosts=0
                WHERE globalID=causedByGIDv;
        END IF;

        -- CVC Cache 2
        SELECT pinned INTO pinnedValue
        FROM Rel_CVC_Cache_ratedCosts_to_causedBy
        WHERE ratedCostsGID=ratedCostsGIDv AND causedByOID=causedByOIDv;

        IF sqlcode = 0 AND pinnedValue = 0 THEN
                SELECT causedByGID INTO newCVCGID
                FROM Rel_CVC_ratedCosts_causedBy
                WHERE ratedCostsGID=ratedCostsGIDv AND causedByOID=causedByOIDv
                ORDER BY causedByGID DESC
                FETCH FIRST 1 ROWS ONLY;

                UPDATE Rel_CVC_Cache_ratedCosts_to_causedBy
                SET causedByGID=newCVCGID
                WHERE ratedCostsGID=ratedCostsGIDv AND causedByOID=causedByOIDv;
        ELSE
                INSERT INTO Rel_CVC_Cache_ratedCosts_to_causedBy (ratedCostsGID,causedByGID,causedByOID,pinned)
                        SELECT ratedCostsGID, causedByGID, causedByOID, 0
                        FROM Rel_CVC_ratedCosts_causedBy
                        WHERE ratedCostsGID=ratedCostsGIDv AND causedByOID=causedByOIDv
                        ORDER BY causedByGID DESC
                        FETCH FIRST 1 ROWS ONLY;
        END IF;

END
```

Die Informationen über eine eventuelle Propagierung einer Operationen an andere Objekte wird nicht zur Laufzeit der Prozedur überprüft. Beim Erstellen der gespeicherten Prozedur muss dies bereits berücksichtig werden. Zur Laufzeit müssen nur noch die Objekte bestimmt werden, an die eine Operation propagiert werden soll. Listing 6.2 zeigt die gespeicherte Prozedur für die Operation `CreateSuccessor` bei Objekten vom Typ *Task*. Dabei werden mit Hilfe des *Cursors c1* die Objekte vom Typ *Cost* bestimmt, an die die Operation `CreateSuccessor` propagiert werden muss. Um die Propagierung auszuführen, wird die Prozedur `CreateSuccessor_Costs` mit entsprechenden Parametern aufgerufen. Anschließend wird durch die Prozedur `NewRelationship_ratedCosts_causedBy` eine neue Beziehung zwischen den Nachfolgerobjekten angelegt.

Listing 6.2 Gespeicherte Prozedur für die Operation `CreateSuccessor` bei Objekten vom Typ *Task*

```
CREATE PROCEDURE CreateSuccessor_Task
                        (OUT newGID Integer,
                        IN GID Integer)
                        LANGUAGE SQL
BEGIN ATOMIC
        DECLARE OID Integer;
        DECLARE sucCount Integer;
        DECLARE maxVersionId Integer;
        DECLARE oldCostGID Integer;
        DECLARE newCostGID Integer;
        DECLARE SQLCODE INT DEFAULT 0;

        DECLARE c1 CURSOR FOR
                SELECT causedByGID
                FROM Rel_CVC_Cache_ratedCosts_to_causedBy
                WHERE ratedCostsGID=GID;


        SELECT objectId, successors INTO OID, sucCount
        FROM ObjectType_Task
        WHERE globalId=GID;

        IF sucCount < 10 THEN
                SELECT successors INTO maxVersionId
                FROM SysTabCounter_Task
                WHERE objectId=OID;

                SET newGID = OID * 10000 +maxVersionId+2;

                INSERT INTO ObjectType_Task (globalId,objectId,versionId,predecessorId,frozen,successors,
                name,description,startDate,dueDate)
                        SELECT newGID,objectId,maxVersionId+2,GID,0,0,name,description,startDate,dueDate
                        FROM ObjectType_Task
                        WHERE globalId=GID;

                UPDATE SysTabCounter_Task
                SET successors=successors+1
                WHERE objectId=OID;

                UPDATE ObjectType_Task
                SET successors=successors+1
                WHERE globalId=GID;


                -- Propagiere an Cost
                OPEN c1;
                FETCH c1 INTO oldCostGID;
                WHILE SQLCODE = 0 DO
                        CALL CreateSuccessor_Costs(newCostGID,oldCostGID);
                        CALL NewRelationship_ratedCosts_causedBy(newGID,newCostGID);
                        FETCH c1 INTO oldCostGID;
                END WHILE;
                CLOSE c1;
        END IF;

END
```

Um untersuchen zu können, ob durch die Verwendung von gespeicherten Prozeduren Zeit eingespart werden kann, wurde eine fünfte Variante des domänenspezifischen Datenbanktreibers entwickelt, die genau wie die vier zuvor betrachteten Varianten, DS-DML-Anweisungen als Eingabe erhält. In den Reduktionsmethoden werden jedoch keine SQL-Anweisungen mehr erstellt, sondern der Name sowie die Parameter der benötigenten gespeicherten Prozedur bestimmt. Daher wurde bei Variante 5 auch keine Rendering-Zeit gemessen.

Da nicht für alle Operationen gespeicherten Prozeduren erstellt wurden, wurde für die Leistungsuntersuchung mit Hilfe des Benchmarkgenerators eine Menge von 30760 DS-DML-Anweisungen generiert. Tabelle 6.4 zeigt die Verteilung der Anweisungen auf die verschiedenen Kategorien von Befehlen. Diese DS-DML-Anweisungen wurden mit den Varianten 4 und 5 der domänenspezifischen Datenbanktreiber ausgeführt. Die Ergebnisse dieser Leistungsuntersuchung befinden sich in Anhang I.

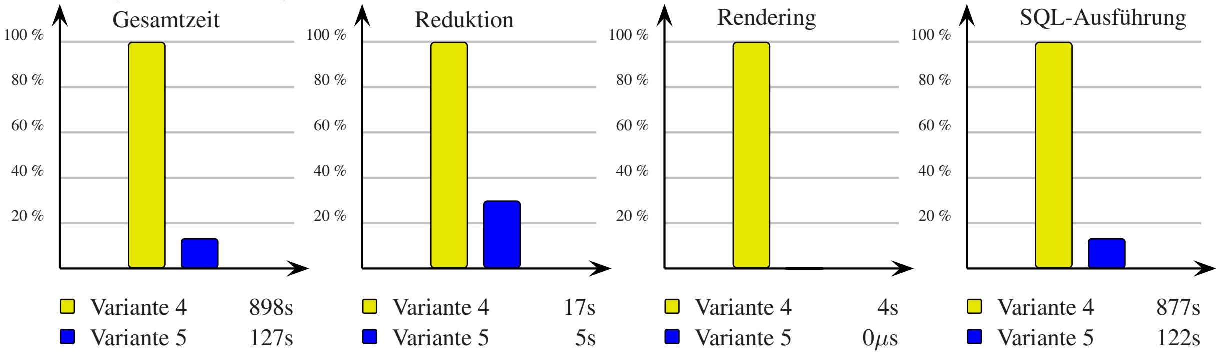
Abbildung 6.17 zeigt den Vergleich der Summen der gemessenen Zeiten aller ausgeführten

Tabelle 6.4 Anteile der einzelnen Kategorien an der Leistungsuntersuchung

Kategorie	Anzahl	Anteil
Create New Object	8330	27,1 %
Create New Relationship	8330	27,1 %
CreateSuccessor	6000	19,5 %
Copy Object	6000	19,5 %
Update Object	1000	3,3 %
Get	500	1,6 %
Get CVC	300	0,9 %
Select	300	0,9 %
Summe	30760	100 %

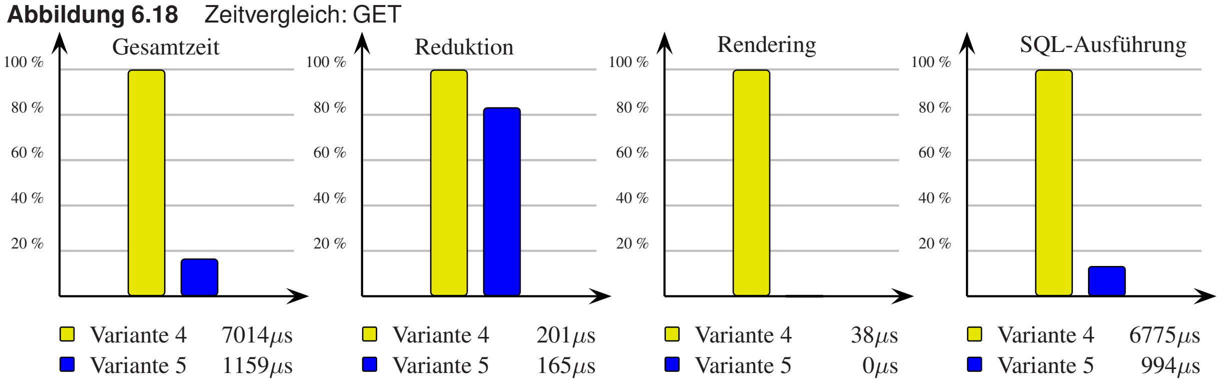
Befehle bei Variante 4 und Variante 5. Dabei ist deutlich zu sehen, dass die Variante, die auf gespeicherten Prozeduren basiert im Vergleich zur Varante 4 nur 16 Prozent der Zeit benötigt hat. Diese große Beschleunigung ist auf verschiedene Gründe zurückzuführen. Zum einen kann, wie zuvor vermutet, der Kommunikationsaufwand zwischen dem domänenspezifischen Datenbanktreiber und der Datenbank reduziert werden. Zum andernen müssen keine Informationen aus dem Informationsmodell geholt werden.

Abbildung 6.17 Zeitvergleich: Summe aller Befehle



Die Leistungsuntersuchung hat ergeben, dass durch die gespeicherten Prozeduren nicht nur die Ausführung von DS-DML-Anweisungen, bei denen mehrere SQL-Anweisungen erzeugt werden, beschleunigt wird, sondern auch große Zeiteinsparungen bei einzelnen Select-Anweisungen möglich sind. Abbildung 6.18 zeigt den Vergleich der gemessenen Zeiten für die Kategorie `Get`. Die Gesamtausführungszeit beträgt dabei ebenfalls nur 17 Prozent im Vergleich zur Variante 4. Dies lässt vermuten, dass beim Anlegen der gespeicherten Prozeduren bereits Vorbereitungen für die spätere Ausführung getroffen werden, so dass die Ausführung beschleunigt wird. Dieser Vorteil kann jedoch nur genutzt werden, wenn die genaue Anfrage bereits im Voraus bekannt ist, da beim Aufruf der gespeicherten Prozedur zur Laufzeit lediglich die Parameter angepasst werden können und keine Tabellen- und Spaltennamen.

Abbildung 6.18 Zeitvergleich: GET



Kapitel 7 Zusammenfassung und Ausblick

In diesem Kapitel werden die wichtigsten Erkenntnisse dieser Arbeit abschließend zusammengefasst und ein Ausblick auf weiterführende Forschungsbereiche gegeben.

7.1 Zusammenfassung

In dieser Diplomarbeit wurden die Konzepte zur Unterstützung von datenbankorientierten Software-Produktlinien durch domänenspezifische Sprachen am Beispiel von Versionierungssystemen untersucht. Dabei wurde zunächst eine Sprache zur Konfiguration von Versionierungssystemen entwickelt, mit der ein Versionierungssystem spezifiziert wurde, das als Grundlage der später durchgeführten Leistungsuntersuchungen gedient hat. Um diese Leistungsuntersuchungen durchführen zu können, wurde das System SQLInteract um die in Kapitel 5 beschriebenen verschiedenen Reduktionsvarianten für produktspezifische Modelle auf CWM-Modelle erweitert. Dadurch konnte mit SQLInteract das in den Leistungsuntersuchungen verwendete Beispiel eines Versionierungssystems spezifiziert werden und die vier unterschiedlichen Datenbankschemata generiert werden.

Bei den verwendeten Datenbankschemata wurden verschiedene Arten zur Speicherung von Beziehungen zwischen Objekten untersucht. Dabei wurde mit einen einfachen Schema begonnen, bei dem alle Beziehungsinformationen in einer einzigen Tabelle gespeichert werden. Danach wurden schrittweise Optimierungen vorgenommen, um die Navigation zwischen den Objekten zu beschleunigen.

Anschließend wurde eine Anfrage- und Datenmanipulationssprache für Versionierungssysteme entwickelt, um eine komfortable Schnittstelle für den Umgang mit Versionierungssystemen bereitzustellen. Da die Reduktion von domänenspezifischen Anweisungen auf herkömmliche SQL-Anweisungen abhängig vom zugrunde liegenden Datenbankschema ist, wurde für jedes der vier Datenbankschemata ein eigener domänenspezifischer Datenbanktreiber entwickelt. Bei der Analyse der Reduktionsmethoden mit Hilfe von Softwaremetriken, wurde festgestellt, dass die Komplexität der Reduktionsmethoden zunimmt, je mehr Optimierungen am Datenbankschema durchgeführt wurden. Trotz einer Zunahme der Komplexität der Reduktionsmethoden liegt die Komplexität in einen wartbaren Bereich.

Um die unterschiedlichen Datenbankschemata und Datenbanktreiber im Rahmen einer Leistungsuntersuchung vergleichen zu können, wurde ein Generator für domänenspezifische Anweisungen entwickelt. Dieser Generator generiert die Anweisungen in Abhängigkeit der im Versionierungssystem gespeicherten Daten. Auf diese Weise sollte ein möglichst realistisches Anwendungsszenario simuliert werden. Die generierten domänenspezifischen Anweisungen wurden mit allen vier Datenbanktreibern ausgeführt. Dabei wurden unterschiedliche Messwerte erfasst und gespeichert.

Die gesammelten Messwerte wurden statistisch analysiert und verschiedene Aspekte anhand von Abbildungen, die sich im Anhang H befinden, verdeutlicht. Als Fazit kann festgehalten werden, dass durch den Einsatz von domänenspezifischen Sprachen eine komfortable Schnittstelle für Entwickler bereitgestellt werden kann. Diese Schnittstelle bietet zwei Vorteile. Zum einen kann eine Menge von SQL-Anweisungen, die zur Ausführung einer bestimmten Operation in der betrachteten Anwendungsdomäne durch einen domänenspezifischen Befehl ersetzt werden. Der zweite Vorteil bei der Verwendung dieser Schnittstelle ist, dass die domänenspezifischen Anweisungen eine einfache Struktur besitzen und unabhängig vom verwendeten Datenbankschema sind. Der zusätzliche Zeitbedarf, der durch die Nutzung einer derartigen Schnittstelle entsteht, liegt mit 2,3 Prozent der eigentlichen SQL-Ausführungszeit für Anwendungen, bei denen die Leistung nicht zu den wichtigsten Anforderungen gehört, in einem Bereich der akzeptabel ist. Es lohnt sich komplexe Reduktionsmethoden einzusetzen.

7.2 Ausblick

Während der Erstellung dieser Diplomarbeit sind Fragestellungen entstanden, die sich als weiterführende Forschungsthemen anbieten. Eine Auswahl dieser Themen wird nachfolgend vorgestellt.

7.2.1 Metasprachen

Wie in dieser Arbeit gezeigt wurde, kann die Entwicklung von datenbankorientierten Software-Produktlinien, durch domänenspezifische Spracherweiterungen unterstützt werden. Dazu mussten unterschiedliche Reduktionsmethoden entwickelt werden. Zum einen die Reduktionsmethoden mit denen die Elemente des domänenspezifischen Metamodells auf passende CWM-Konstrukte reduziert und zum anderen die Reduktionsmethoden, in denen die domänenspezifischen Anfrage- und Datenmanipulationsbefehle auf herkömmliche SQL-Anweisungen reduziert werden.

Die Entwicklung dieser Reduktionsmethoden kann durch die Verwendung von Metasprachen unterstützt werden. Mit Hilfe von Metasprachen kann die Entwicklung auf einer höheren Abstraktionsebene und unabhängig von der später verwendeten Programmiersprache durchgeführt werden. Dabei können beispielsweise mehrere Anweisungen, die in den Reduktionsmethoden benötigt werden, durch ein bestimmtes Sprachkonstrukt beschrieben werden. Durch eine Reihe von Abbildungsregeln, können die Reduktionsmethoden aus den Spezifikationen in der Metasprache generiert werden.

Bei Betrachtung der Reduktionsmethoden für domänenspezifische Anfrage- und Datenmanipulationsanweisungen, die im Kapitel 5.4 genauer vorgestellt wurden, fällt auf, dass häufig auf das Informationsmodell zugegriffen werden muss, um beispielsweise festzustellen, ob es sich um ein gleitendes Beziehungsende handelt oder ob die maximale Kardinalität eines Beziehungsendes 1 beträgt. In Abhängigkeit der Ergebnisse beim Zugriff auf das Informationsmodell wird unterschiedlicher Reduktionscode ausgeführt. Durch Verwendung einer besonderen Art von Polymorphie, könnte auf den Zugriff auf das Informationsmodell verzichtet werden und es wird jeweils die passende Reduktionsmethode ausgeführt [CE00].

7.2.2 Domänenspezifische Schemaevolution

In dieser Arbeit wurde ein Ansatz zum domänenspezifischen Datenbankschemaentwurf betrachtet. Dieser Ansatz könnte um die Möglichkeiten einer domänenspezifischen Schemaevolution erweitert werden. Dabei wird ein bereits vorhandenes produktspezifsches Datenbankschema durch eine Veränderung der DS-DDL-Anweisungen an geänderte Anforderungen angepasst. Inwieweit können die vorhandenen Reduktionsmethoden genutzt werden, um eine Veränderung des Datenbankschemas durchzuführen. Dabei muss nach Lösungen gesucht werden, um bereits vorhandene Daten an das geänderte Datenbankschema anzupassen. Dabei stellt sich unteranderem die Frage, ob es Situationen gibt, in denen die Daten nicht verlustfrei übernommen werden können.

7.2.3 Product-Line Mining

Das Konzept des *Reverse Engineering* das bereits im Bereich der modellgetriebenen Architekturen eingesetzt wird, sieht die Entwicklung eines Softwaresystems nicht als linearen Vorgang sondern es werden Endprodukte analysiert, um Informationen für die Erstellung neuer Produkte daraus abzuleiten. Dieses Konzept könnte auch auf die Entwicklung von Produktlinien übertragen werden.

Häufig werden in einer Anwendungsdomäne mehrere datenbankorientierte Anwendungen eingesetzt, die getrennt von einander entwickelt wurden. Aus den Datenbankschemata, die den Anwendungen zugrunde liegen, kann versucht werden ein gemeinsames Metamodell zu finden. Dieser Prozess nennt sich *Product Line Mining* und beschäftigt sich mit der Analyse der Gemeinsamkeiten, der vorhandenen Anwendungen einer Anwendungsdomäne. Dabei sollen die Gemeinsamkeiten extrahiert werden, um auf diese Weise ein Metamodell für diese Produkte automatisch erstellen zu können.

7.2.4 Genetische Algorithmen

Auf Grundlage der durch Benchmarkgenerator erzeugten domänenspezifischen Anweisungen und den vorgenommen Leistungsuntersuchungen könnte versucht werden, die Suche nach dem optimalen Datenbankschema, mit Hilfe von genetischen Algorithmen, zu automatisieren.

In der Informatik existieren viele Probleme, von denen man vermutet, dass kein Algorithmus existiert, der das Problem in akzeptabler Zeit effizient löst. Derartige Probleme treten in der Praxis häufig auf und es wird versucht Algorithmen zu entwickeln, die möglichst schnell Näherungslösungen für ein Problem liefern. Teilweise ist es nur sehr schwer möglich, einen Algorithmus zu finden, der schnell zufriedenstellende Näherungslösungen berechnet. Bei der Suche nach einem Algorithmus, der möglichst schnell Näherungslösungen berechnet, können genetische Algorithmen eingesetzt werden. Ein genetischer Algorithmus ist ein allgemeines, einfach einzusetzendes Lösungsverfahren, das von der Natur inspiriert ist. Er ahmt die natürliche Evolution nach, um für ein Problem eine gute Lösung zu finden. Dabei arbeitet er auf einer Menge von möglichen Lösungen, die sukzessive durch die Evolutionsprinzipien Selektion, Kreuzung und Mutation verändert wird. Das Ziel ist dabei, mit der Zeit von Generation zu Generation immer bessere Lösungen zu entwickeln.

Um auf diese Weise bessere Verfahren entwickeln zu können, muss das Prinzip der Evolution nur noch auf dem Computer nachgebildet werden. Die natürliche Selektion wird simuliert, indem die schlechteren Verfahren bzw. Lösungen für das Problem aus der Population entfernt werden. Die Paarung bzw. Kreuzung zweier Lösungen wird so nachgebildet, dass zwei in der Population vorhandenen Verfahren zu einem neuen Verfahren zusammengesetzt werden. Die Mutationen werden durch punktuelle Änderungen an den vorhandenen Verfahren nachempfunden. Die Menge der vorhandenen Verfahren wird durch Selektion, Kreuzung und Mutation kontinuierlich verändert. So wird versucht kontinuierlich bessere Verfahren und nach Möglichkeit ein optimales Verfahren zu finden.

Überträgt man das Verfahren der genetischen Algorithmen, auf das Problem ein optimales Datenbankschema zu finden, so bedeutet dies, das die Evolution auf die Menge der vorhandenen Datenbankschemata und die dazugehörigen Reduktionsmethoden angewendet wird. Das Ergebnis der Evolution wird dann durch eine Leistungsuntersuchung mit vorgegebenen Anweisungen mit den bisherigen Verfahren verglichen.

Anhang A Syntax der DS-DDL und DS-DML für Versionierungssysteme

Zur Spezifiaktion der domänenspezifischen DDL und DML für Versionierungssysteme wurde die erweitere Backus-Naur-Form verwendet.

A.1 DS-DDL

```
create_objecttype_command ::=
create_unversioned_objecttype_clause | create_versioned_objecttype_clause |
create_workspacetype_clause;

create_unversioned_objecttype_clause ::=
"CREATE UNVERSIONED OBJECTTYPE" objecttypename "("
{attribute_clause ","} [attribute_clause]
")";

create_versioned_objecttype_clause ::=
"CREATE VERSIONED OBJECTTYPE" objecttypename "("
{attribute_clause ","}
"MAXSUCCESSORS" n
")";

create_workspacetype_clause ::= ;
"CREATE VERSIONED OBJECTTYPE" objecttypename "("
{attribute_clause ","}
{attachment_clause ","}
"MAXSUCCESSORS" maxValue_clause
")";

create_relationship_command ::=
"CREATE RELATIONSHIP TYPE" relationshiptypename "("
"CONNECTS" objecttypename "(" n "," maxValue_clause ")" [ "FLOATING" ] ","
"CONNECTS" objecttypename "(" n "," maxValue_clause ")" [ "FLOATING" ] [ "," ]
[ "PROPAGATES" ]  proptype { "," proptype } "ON" objecttypename [ "," ]
[ "PROPAGATES" ]  proptype { "," proptype } "ON" objecttypename
")";

attribute_clause ::=
"ATTRIBUTE" attributename datatype_clause;

attachment_clause ::=
"ATTACHMENT" objecttypename "(" n "," maxValue_clause ")";

datatype_clause ::=
"String" | "Integer" |"Float" | "Boolean" |
"Date" | "DateTime" | "BLOB" | "CLOB";
```

```
proptype ::=
"attach_detach" | "create_successor" | "freeze" |
"checkout_checkin" | "copy" | "new" | "delete";

objecttypename ::= identifier;

attributename ::= identifier;

ralationshiptypename ::= identifier;

maxValue_clause ::= n | "*";

identifier ::=
"letter" { "letter" | "digit" | "_" | "$" | "#" };

n ::= "digit" { "digit" };
```

A.2 DS-DML

Eigenständige DML-Anweisungen

```
new_object_command ::=
"CREATE NEW OBJECT" ObjectTypeName [ "(" column_list ")"
"VALUES (" value_list ")" ] [ use_Workspace_clause ];

new_relationship_command ::=
"CREATE NEW RELATIONSHIP" RelationshipTypeName "WHERE"
Rolename".GID=" id_value AND Rolename".GID=" id_value;

copy_object_command ::=
"COPY OBJECT"  ( object_select_clause_with_workspace |
workspace_object_select_clause
[ "PROPAGATE TO ATTACHED OBJECTS" ] );

createSuccessor_command ::=
"CREATE SUCCESSOR OF OBJECT" (
versioned_object_select_clause_with_workspace |
workspace_object_select_clause
[ "PROPAGATE TO ATTACHED OBJECTS" ] );

freeze_object_command ::=
"FREEZE OBJECT" (
versioned_object_select_clause_with_workspace |
workspace_object_select_clause
[ "PROPAGATE TO ATTACHED OBJECTS" ] );

delete_object_command ::=
"DELETE OBJECT" ( ObjectTypeName "WHERE OID =" id_value
[ use_Workspace_clause ] | WorkspaceTypeName
"WHERE OID =" id_value [ "PROPAGATE TO ATTACHED OBJECTS" ] );


delete_relationship_command ::=
"DELETE RELATIONSHIP RelationshipTypeName WHERE
Rolename".OID=" id_value AND Rolename".OID=" id_value;

merge_objects_command ::=
"MERGE OBJECTS" ObjectTypeName "WHERE" ( "OID =" id_value
"AND PRIM_VID ="id_value "AND SEC_VID =" id_value |
PRIM_GID =" id_value "AND SEC_GID =" id_value );

update_object_command ::=
"UPDATE" ObjectTypeName update_clause ( object_select_clause2 |
"WHERE OID=" id_value [ "AND" condition ] );
```

```
select_command ::=
"SELECT" [ "DISTINCT" ] ( "*" |
( displayed_column { "," displayed_column } ) )
"FROM"  from_clause { "," from_clause }  [ "WHERE" condition ]
{ group_clause } { set_clause } { order_clause }
[use_Workspace_clause];

getRoot_command ::=
"GET ROOT OF" versioned_object_select_clause_with_workspace;

getPredecessor_command ::=
"GET PREDECESSOR OF"
versioned_object_select_clause_with_workspace;

getSuccessors_command ::=
"GET SUCCESSORS OF"
versioned_object_select_clause_with_workspace;

getAlternatives_command ::=
"GET ALTERNATIVES OF"
versioned_object_select_clause_with_workspace;

getBaseVersion_command ::=
"GET BASEVERSION OF" VersionedObjectTypeName "WHERE" (
"OID =" id_value "AND VID1 =" id_value "AND VID2 =" id_value |
"GID1 =" id_value "AND GID2 =" id_value );

get_cvc_command ::=
"GET CVC FROM" object_select
"ROLENAME" Rolename;

pinning_command ::=
"PIN" versioned_object_select "IN CVC"
object_select "ROLENAME" Rolename;

unpinning_command ::=
"UNPIN" versioned_object_select IN CVC
versioned_object_select" "ROLENAME" Rolename;

attach_command ::=
"ATTACH" ObjectTypeName "GID =" id_value
"TO WORKSPACE" WorkspaceTypeName "GID =" id_value;

detach_command ::=
"DETACH" ObjectTypeName "GID =" id_value
"FROM WORKSPACE" WorkspaceTypeName "GID =" id_value;

checkout_command ::=
"CHECKOUT" ObjectTypeName "GID =" id_value
"FROM WORKSPACE" WorkspaceTypeName "GID =" id_value;

checkin_command ::=
"CHECKIN" ObjectTypeName "GID =" id_value
"TO WORKSPACE" WorkspaceTypeName "GID =" id_value;
```

Bestandteile der DML-Anweisungen

```
object_select_clause  ::=
unversioned_object_select | versioned_object_select;

object_select_clause_with_workspace ::=
object_select [ use_Workspace_clause ];

unversioned_object_select ::=
UnversionedObjectTypeName "WHERE"
```

```
( "OID" | "GID" ) =" id_value;

versioned_object_select ::=
VersionedObjectTypeName "WHERE" ( "GID =" id_value |
"OID =" id_value "AND VID=" id_value );

versioned_object_select_with_workspace ::=
versioned_object_select [ use_Workspace_clause ];

workspace_object_select ::=
WorkspaceTypeName "WHERE" ( "GID =" id_value |
"OID =" id_value "AND VID=" id_value );

use_Workspace_clause ::=
"USE WORKSPACE" workspace_object_select;


from_clause ::=
selected_table { object_navigation_clause };


object_navigation_clause ::=
"--" Rolename "--> " selected_table;

select_end_clauses ::= { group_clause } { set_clause } { order_clause };

update_clause ::=
"SET"  column_name "=" expression
{ "," column_name "=" expression };


order_clause ::=
"ORDER BY" column_name [( "ASC" | "DESC" )]
{ "," column_name [( "ASC" | "DESC" )]};

set_clause ::=
( ( "UNION ALL" ) | "INTERSECT" | "MINUS" ) select_command;

group_clause ::=
"GROUP BY" expression { "," expression } [ "HAVING" condition ];

condition ::= [ "NOT" ] logical_term { "OR" logical_term };

logical_term ::= logical_factor { "AND" logical_factor };

logical_factor ::=
( expression comparaison_op exprssion )
| ( [ "NOT" ] "LIKE" match_string )
| ( "IS" [ "NOT" ] "null" )
| ( "(" condition ")" );

displayed_column ::=
( ObjectTypeName "." ( "*" | column_name)) |
( expression [ "AS" alias ] );

selected_table ::= ObjectTypeName [ alias ];

expression  ::= [ "+" | "-" ] term { ( "+" | "-" ) term };

comparaison_op ::=
"=" | "<" | ">" | "<>" | "!=" | "^=" | "<=" | ">=";

match_string ::= "'" { "any_character" | "_" | "%" } "'";

column_list ::=  column_name { "," column_name };

value_list ::= value { "," value };

ObjectTypeName ::= identifier;
```

```
VersionedObjectTypeName ::= identifier;

WorkspaceTypeName ::= identifier;

RelationshipTypeName ::= identifier;

Rolename ::= identifier;

alias ::= identifier;

id_value ::= n { n };

term ::= factor { ( "*" | "/" ) factor };

factor ::=
n | column_name
| ( group_function "(" [ "DISTINCT" ] expression ")" );

group_function ::= "AVG" | "COUNT" | "MAX" | "MIN" | "SUM";

value ::= ( [ "+" | "-" ] ( identifier | n ) ) | quoted_string;

n ::= "digit" { "digit" };

identifier ::=
( "letter" { "letter" | "digit" | "_" | "$" | "#" } )
| quoted_string;

quoted_string ::= "'" { "any_character" } "'";
```

Anhang B DS-DB-Schema für ein Versionierungssystem

```
create versioned object type Task (
attribute name String,
attribute description String,
attribute startDate Date,
attribute dueDate Date,
maxSuccessors 10)

create versioned object type Costs (
attribute wages Float,
attribute travelExpences Float,
attribute materialExpences Float,
attribute validUntil Date,
maxSuccessors 15)

create versioned object type Offer (
attribute customer String,
attribute description String,
attribute price Float,
attribute validUntil Date,
maxSuccessors 10)

create versioned object type Employee (
attribute name String,
attribute address String,
attribute dateOfBirth Date,
attribute salary Float,
attribute employedSince Date,
maxSuccessors 3)

create versioned object type Department (
attribute name String,
attribute description String,
attribute budget Float,
attribute location String,
maxSuccessors 2)

create workspace type Project (
attribute budget Float,
attachment Task (0, *),
attachment Costs (0, *),
attachment Offer (0, *),
attachment Employee (0, *),
attachment Department (0, *),
maxSuccessors *)

create workspace type HumanResources (
attachment Employee (0, *),
attachment Department (0, *),
maxSuccessors *)

create relationship type ratedCosts_causedBy (
```

```
connects Task (1, 1) rolename ratedCosts floating,
connects Costs (0, *) rolename causedBy floating,
propagates attach_detach, create_successor, freeze, checkout_checkin, copy, new, delete on Costs)

create relationship type contains_isPart (
connects Offer (1, 1) rolename contains floating,
connects Task (1, 99) rolename isPart,
propagates attach_detach, freeze, checkout_checkin, copy, new, delete on Task)

create relationship type executes_assigned (
connects Employee (0, 10) rolename executes,
connects Task (0, *) rolename assigned floating)

create relationship type writtenBy_isAuthor (
connects Offer (0, *) rolename writtenBy floating,
connects Employee (1, 1) rolename isAuthor)

create relationship type leads_hasBoss (
connects Employee (0, 1) rolename leads,
connects Department (0, 3) rolename hasBoss,
propagates attach_detach, freeze, checkout_checkin on Employee)

create relationship type worksIn_consistsOf (
connects Employee (0, 50) rolename worksIn,
connects Department (1, 1) rolename consistsOf,
propagates attach_detach, freeze, checkout_checkin on Employee)
```

Anhang C Ergebnisse des Schemaentwurfs für Versionierungssysteme

C.1 Gemeinsamkeiten der vier Datenbankschemata

```
CREATE TABLE SysTabCounter_Task (
objectId Integer NOT NULL,
successors Integer NOT NULL,
PRIMARY KEY (objectId)
)

CREATE TABLE SysTabCounter_Costs (
objectId Integer NOT NULL,
successors Integer NOT NULL,
PRIMARY KEY (objectId)
)

CREATE TABLE SysTabCounter_Offer (
objectId Integer NOT NULL,
successors Integer NOT NULL,
PRIMARY KEY (objectId)
)

CREATE TABLE SysTabCounter_Employee (
objectId Integer NOT NULL,
successors Integer NOT NULL,
PRIMARY KEY (objectId)
)

CREATE TABLE SysTabCounter_Department (
objectId Integer NOT NULL,
successors Integer NOT NULL,
PRIMARY KEY (objectId)
)

CREATE TABLE ObjectType_Project (
budget Float,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Project (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE SysTabCounter_Project (
objectId Integer NOT NULL,
successors Integer NOT NULL,
PRIMARY KEY (objectId)
)

CREATE TABLE attached_Task_Project (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
otObjectId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_Project (globalId),
UNIQUE (wsGlobalId, otObjectId)
```

```
)

CREATE TABLE attached_Costs_Project (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
otObjectId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_Costs (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_Project (globalId),
UNIQUE (wsGlobalId, otObjectId)
)

CREATE TABLE attached_Offer_Project (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
otObjectId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_Offer (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_Project (globalId),
UNIQUE (wsGlobalId, otObjectId)
)

CREATE TABLE attached_Employee_Project (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
otObjectId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_Project (globalId),
UNIQUE (wsGlobalId, otObjectId)
)

CREATE TABLE attached_Department_Project (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
otObjectId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_Department (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_Project (globalId),
UNIQUE (wsGlobalId, otObjectId)
)

CREATE TABLE ObjectType_HumanResources (
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_HumanResources (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE SysTabCounter_HumanResources (
objectId Integer NOT NULL,
successors Integer NOT NULL,
PRIMARY KEY (objectId)
)

CREATE TABLE attached_Employee_HumanResources (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
otObjectId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_HumanResources (globalId),
UNIQUE (wsGlobalId, otObjectId)
)

CREATE TABLE attached_Department_HumanResources (
wsGlobalId Integer NOT NULL,
otGlobalId Integer NOT NULL,
otObjectId Integer NOT NULL,
PRIMARY KEY (wsGlobalId, otGlobalId),
FOREIGN KEY (otGlobalId) REFERENCES ObjectType_Department (globalId),
FOREIGN KEY (wsGlobalId) REFERENCES ObjectType_HumanResources (globalId),
UNIQUE (wsGlobalId, otObjectId)
)
```

Tabellen zur Speicherung des Informationsmodells

```
CREATE TABLE SysTabObjectTypes (
name Varchar(255) NOT NULL,
objecttype Varchar(255) NOT NULL,
maxSuccessors Integer NOT NULL,
PRIMARY KEY (name)
)

CREATE TABLE SysTabObjectTypeAttributes (
ObjectTypeName Varchar(255) NOT NULL,
```

```
AttributeName Varchar(255) NOT NULL,
AttributeType Varchar(255) NOT NULL,
PRIMARY KEY (ObjectTypeName, AttributeName)
)

CREATE TABLE SysTabWorkspaceAttachments (
ObjectTypeName Varchar(255) NOT NULL,
WorkspaceTypeName Varchar(255) NOT NULL,
PRIMARY KEY (ObjectTypeName, WorkspaceTypeName)
)

CREATE TABLE SysTabRelShipTypes (
name Varchar(255) NOT NULL,
rolenameA Varchar(255) NOT NULL,
rolenameB Varchar(255) NOT NULL,
objectTypeA Varchar(255) NOT NULL,
objectTypeB Varchar(255) NOT NULL,
PRIMARY KEY (name)
)

CREATE TABLE SysTabRelShipEnds (
name Varchar(255) NOT NULL,
objectType Varchar(255) NOT NULL,
relShipType Varchar(255) NOT NULL,
isFloating Integer NOT NULL,
minMultiplicity Integer NOT NULL,
maxMultiplicity Integer NOT NULL,
propAttachDetach Integer NOT NULL,
propCreateSuccessor Integer NOT NULL,
propFreeze Integer NOT NULL,
propCheckoutCheckin Integer NOT NULL,
propCopy Integer NOT NULL,
propNew Integer NOT NULL,
propDelete Integer NOT NULL,
PRIMARY KEY (name),
FOREIGN KEY (objectType) REFERENCES SysTabObjectTypes (name),
FOREIGN KEY (relShipType) REFERENCES SysTabRelShipTypes (name),
UNIQUE (objectType, relShipType)
)

CREATE TABLE SysTabCounters (
name Varchar(255) NOT NULL,
value Integer NOT NULL,
PRIMARY KEY (name)
)
```

C.2 Datenbankschema bei Variante 1

```
CREATE TABLE ObjectType_Task (
name Varchar(255),
description Varchar(255),
startDate Date,
dueDate Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Task (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Costs (
wages Float,
travelExpences Float,
materialExpences Float,
validUntil Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Costs (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Offer (
customer Varchar(255),
description Varchar(255),
price Float,
validUntil Date,
globalId Integer NOT NULL,
```

```
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Offer (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Employee (
name Varchar(255),
address Varchar(255),
dateOfBirth Date,
salary Float,
employedSince Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Employee (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Department (
name Varchar(255),
description Varchar(255),
budget Float,
location Varchar(255),
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Department (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE Relationships (
sourceGID Integer NOT NULL,
sourceOID Integer NOT NULL,
sourceVID Integer NOT NULL,
source_rolename Varchar(255) NOT NULL,
destGID Integer NOT NULL,
destOID Integer NOT NULL,
destVID Integer NOT NULL,
pinned Integer NOT NULL,
PRIMARY KEY (sourceGID, destGID, source_rolename)
)
```

C.3 Datenbankschema bei Variante 2

```
CREATE TABLE ObjectType_Task (
name Varchar(255),
description Varchar(255),
startDate Date,
dueDate Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Task (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Costs (
wages Float,
travelExpences Float,
materialExpences Float,
validUntil Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
```

```
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Costs (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Offer (
customer Varchar(255),
description Varchar(255),
price Float,
validUntil Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Offer (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Employee (
name Varchar(255),
address Varchar(255),
dateOfBirth Date,
salary Float,
employedSince Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Employee (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Department (
name Varchar(255),
description Varchar(255),
budget Float,
location Varchar(255),
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Department (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE Rel_CVC_ratedCosts_causedBy (
ratedCostsGID Integer NOT NULL,
ratedCostsOID Integer NOT NULL,
ratedCostsVID Integer NOT NULL,
pinned_ratedCosts Integer NOT NULL,
causedByGID Integer NOT NULL,
causedByOID Integer NOT NULL,
causedByVID Integer NOT NULL,
pinned_causedBy Integer NOT NULL,
PRIMARY KEY (ratedCostsGID, causedByGID),
FOREIGN KEY (ratedCostsGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (causedByGID) REFERENCES ObjectType_Costs (globalId)
)

CREATE TABLE Rel_CVC_isPart_to_contains (
isPartGID Integer NOT NULL,
isPartOID Integer NOT NULL,
isPartVID Integer NOT NULL,
containsGID Integer NOT NULL,
containsOID Integer NOT NULL,
containsVID Integer NOT NULL,
pinned_contains Integer NOT NULL,
PRIMARY KEY (isPartGID, containsGID),
FOREIGN KEY (isPartGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (containsGID) REFERENCES ObjectType_Offer (globalId)
)

CREATE TABLE Rel_contains_to_isPart (
containsGID Integer NOT NULL,
containsOID Integer NOT NULL,
isPartGID Integer NOT NULL,
isPartOID Integer NOT NULL,
PRIMARY KEY (containsGID, isPartGID),
FOREIGN KEY (containsGID) REFERENCES ObjectType_Offer (globalId),
FOREIGN KEY (isPartGID) REFERENCES ObjectType_Task (globalId)
)
```

```
CREATE TABLE Rel_CVC_executes_to_assigned (
executesGID Integer NOT NULL,
executesOID Integer NOT NULL,
executesVID Integer NOT NULL,
assignedGID Integer NOT NULL,
assignedOID Integer NOT NULL,
assignedVID Integer NOT NULL,
pinned_assigned Integer NOT NULL,
PRIMARY KEY (executesGID, assignedGID),
FOREIGN KEY (executesGID) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (assignedGID) REFERENCES ObjectType_Task (globalId)
)

CREATE TABLE Rel_assigned_to_executes (
assignedGID Integer NOT NULL,
assignedOID Integer NOT NULL,
executesGID Integer NOT NULL,
executesOID Integer NOT NULL,
PRIMARY KEY (assignedGID, executesGID),
FOREIGN KEY (assignedGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (executesGID) REFERENCES ObjectType_Employee (globalId)
)

CREATE TABLE Rel_CVC_isAuthor_to_writtenBy (
isAuthorGID Integer NOT NULL,
isAuthorOID Integer NOT NULL,
isAuthorVID Integer NOT NULL,
writtenByGID Integer NOT NULL,
writtenByOID Integer NOT NULL,
writtenByVID Integer NOT NULL,
pinned_writtenBy Integer NOT NULL,
PRIMARY KEY (isAuthorGID, writtenByGID),
FOREIGN KEY (isAuthorGID) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (writtenByGID) REFERENCES ObjectType_Offer (globalId)
)

CREATE TABLE Rel_writtenBy_to_isAuthor (
writtenByGID Integer NOT NULL,
writtenByOID Integer NOT NULL,
isAuthorGID Integer NOT NULL,
isAuthorOID Integer NOT NULL,
PRIMARY KEY (writtenByGID, isAuthorGID),
FOREIGN KEY (writtenByGID) REFERENCES ObjectType_Offer (globalId),
FOREIGN KEY (isAuthorGID) REFERENCES ObjectType_Employee (globalId)
)

CREATE TABLE Rel_leads_hasBoss (
leadsGId Integer NOT NULL,
leadsOId Integer NOT NULL,
hasBossGId Integer NOT NULL,
hasBossOId Integer NOT NULL,
PRIMARY KEY (leadsGId, hasBossGId),
FOREIGN KEY (leadsGId) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (hasBossGId) REFERENCES ObjectType_Department (globalId),
UNIQUE (leadsGId, hasBossOId),
UNIQUE (hasBossGId, leadsOId)
)

CREATE TABLE Rel_worksIn_consistsOf (
worksInGId Integer NOT NULL,
worksInOId Integer NOT NULL,
consistsOfGId Integer NOT NULL,
consistsOfOId Integer NOT NULL,
PRIMARY KEY (worksInGId, consistsOfGId),
FOREIGN KEY (worksInGId) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (consistsOfGId) REFERENCES ObjectType_Department (globalId),
UNIQUE (worksInGId, consistsOfOId),
UNIQUE (consistsOfGId, worksInOId)
)
```

C.4 Datenbankschema bei Variante 3

```
CREATE TABLE ObjectType_Task (
name Varchar(255),
description Varchar(255),
startDate Date,
dueDate Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Task (globalId),
UNIQUE (objectId, versionId)
```

```
)

CREATE TABLE ObjectType_Costs (
wages Float,
travelExpences Float,
materialExpences Float,
validUntil Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Costs (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Offer (
customer Varchar(255),
description Varchar(255),
price Float,
validUntil Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
rel_isAuthorGID Integer,
rel_isAuthorOID Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Offer (globalId),
FOREIGN KEY (rel_isAuthorGID) REFERENCES ObjectType_Employee (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Employee (
name Varchar(255),
address Varchar(255),
dateOfBirth Date,
salary Float,
employedSince Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
rel_consistsOfGID Integer,
rel_consistsOfOID Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (rel_consistsOfGID) REFERENCES ObjectType_Department (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Department (
name Varchar(255),
description Varchar(255),
budget Float,
location Varchar(255),
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
rel_leadsGID Integer,
rel_leadsOID Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Department (globalId),
FOREIGN KEY (rel_leadsGID) REFERENCES ObjectType_Employee (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE Rel_CVC_ratedCosts_causedBy (
ratedCostsGID Integer NOT NULL,
ratedCostsOID Integer NOT NULL,
ratedCostsVID Integer NOT NULL,
pinned_ratedCosts Integer NOT NULL,
causedByGID Integer NOT NULL,
causedByOID Integer NOT NULL,
causedByVID Integer NOT NULL,
pinned_causedBy Integer NOT NULL,
PRIMARY KEY (ratedCostsGID, causedByGID),
FOREIGN KEY (ratedCostsGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (causedByGID) REFERENCES ObjectType_Costs (globalId)
)
```

```
CREATE TABLE Rel_CVC_isPart_to_contains (
isPartGID Integer NOT NULL,
isPartOID Integer NOT NULL,
isPartVID Integer NOT NULL,
containsGID Integer NOT NULL,
containsOID Integer NOT NULL,
containsVID Integer NOT NULL,
pinned_contains Integer NOT NULL,
PRIMARY KEY (isPartGID, containsGID),
FOREIGN KEY (isPartGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (containsGID) REFERENCES ObjectType_Offer (globalId)
)

CREATE TABLE Rel_contains_to_isPart (
containsGID Integer NOT NULL,
containsOID Integer NOT NULL,
isPartGID Integer NOT NULL,
isPartOID Integer NOT NULL,
PRIMARY KEY (containsGID, isPartGID),
FOREIGN KEY (containsGID) REFERENCES ObjectType_Offer (globalId),
FOREIGN KEY (isPartGID) REFERENCES ObjectType_Task (globalId)
)

CREATE TABLE Rel_CVC_executes_to_assigned (
executesGID Integer NOT NULL,
executesOID Integer NOT NULL,
executesVID Integer NOT NULL,
assignedGID Integer NOT NULL,
assignedOID Integer NOT NULL,
assignedVID Integer NOT NULL,
pinned_assigned Integer NOT NULL,
PRIMARY KEY (executesGID, assignedGID),
FOREIGN KEY (executesGID) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (assignedGID) REFERENCES ObjectType_Task (globalId)
)

CREATE TABLE Rel_assigned_to_executes (
assignedGID Integer NOT NULL,
assignedOID Integer NOT NULL,
executesGID Integer NOT NULL,
executesOID Integer NOT NULL,
PRIMARY KEY (assignedGID, executesGID),
FOREIGN KEY (assignedGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (executesGID) REFERENCES ObjectType_Employee (globalId)
)

CREATE TABLE Rel_CVC_isAuthor_to_writtenBy (
isAuthorGID Integer NOT NULL,
isAuthorOID Integer NOT NULL,
isAuthorVID Integer NOT NULL,
writtenByGID Integer NOT NULL,
writtenByOID Integer NOT NULL,
writtenByVID Integer NOT NULL,
pinned_writtenBy Integer NOT NULL,
PRIMARY KEY (isAuthorGID, writtenByGID),
FOREIGN KEY (isAuthorGID) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (writtenByGID) REFERENCES ObjectType_Offer (globalId)
)
```

C.5 Datenbankschema bei Variante 4

```
CREATE TABLE ObjectType_Task (
name Varchar(255),
description Varchar(255),
startDate Date,
dueDate Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
cache_contains Integer,
cache_pinned_contains Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (cache_contains) REFERENCES ObjectType_Offer (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Costs (
wages Float,
travelExpences Float,
materialExpences Float,
validUntil Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
```

```
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
cache_ratedCosts Integer,
cache_pinned_ratedCosts Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Costs (globalId),
FOREIGN KEY (cache_ratedCosts) REFERENCES ObjectType_Task (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Offer (
customer Varchar(255),
description Varchar(255),
price Float,
validUntil Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
rel_isAuthorGID Integer,
rel_isAuthorOID Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Offer (globalId),
FOREIGN KEY (rel_isAuthorGID) REFERENCES ObjectType_Employee (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Employee (
name Varchar(255),
address Varchar(255),
dateOfBirth Date,
salary Float,
employedSince Date,
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
rel_consistsOfGID Integer,
rel_consistsOfOID Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (rel_consistsOfGID) REFERENCES ObjectType_Department (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE ObjectType_Department (
name Varchar(255),
description Varchar(255),
budget Float,
location Varchar(255),
globalId Integer NOT NULL,
objectId Integer NOT NULL,
versionId Integer NOT NULL,
predecessorId Integer,
frozen Integer NOT NULL,
checkout Integer,
successors Integer,
rel_leadsGID Integer,
rel_leadsOID Integer,
PRIMARY KEY (globalId),
FOREIGN KEY (predecessorId) REFERENCES ObjectType_Department (globalId),
FOREIGN KEY (rel_leadsGID) REFERENCES ObjectType_Employee (globalId),
UNIQUE (objectId, versionId)
)

CREATE TABLE Rel_CVC_ratedCosts_causedBy (
ratedCostsGID Integer NOT NULL,
ratedCostsOID Integer NOT NULL,
ratedCostsVID Integer NOT NULL,
causedByGID Integer NOT NULL,
causedByOID Integer NOT NULL,
causedByVID Integer NOT NULL,
PRIMARY KEY (ratedCostsGID, causedByGID),
FOREIGN KEY (ratedCostsGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (causedByGID) REFERENCES ObjectType_Costs (globalId)
)

CREATE TABLE Rel_CVC_Cache_ratedCosts_to_causedBy (
ratedCostsGID Integer NOT NULL,
causedByGID Integer NOT NULL,
causedByOID Integer NOT NULL,
pinned Integer NOT NULL,
PRIMARY KEY (ratedCostsGID, causedByGID),
FOREIGN KEY (ratedCostsGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (causedByGID) REFERENCES ObjectType_Costs (globalId)
```

```
)

CREATE TABLE Rel_CVC_isPart_to_contains (
isPartGID Integer NOT NULL,
isPartOID Integer NOT NULL,
isPartVID Integer NOT NULL,
containsGID Integer NOT NULL,
containsOID Integer NOT NULL,
containsVID Integer NOT NULL,
PRIMARY KEY (isPartGID, containsGID),
FOREIGN KEY (isPartGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (containsGID) REFERENCES ObjectType_Offer (globalId)
)

CREATE TABLE Rel_contains_to_isPart (
containsGID Integer NOT NULL,
containsOID Integer NOT NULL,
isPartGID Integer NOT NULL,
isPartOID Integer NOT NULL,
PRIMARY KEY (containsGID, isPartGID),
FOREIGN KEY (containsGID) REFERENCES ObjectType_Offer (globalId),
FOREIGN KEY (isPartGID) REFERENCES ObjectType_Task (globalId)
)

CREATE TABLE Rel_CVC_executes_to_assigned (
executesGID Integer NOT NULL,
executesOID Integer NOT NULL,
executesVID Integer NOT NULL,
assignedGID Integer NOT NULL,
assignedOID Integer NOT NULL,
assignedVID Integer NOT NULL,
PRIMARY KEY (executesGID, assignedGID),
FOREIGN KEY (executesGID) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (assignedGID) REFERENCES ObjectType_Task (globalId)
)

CREATE TABLE Rel_assigned_to_executes (
assignedGID Integer NOT NULL,
assignedOID Integer NOT NULL,
executesGID Integer NOT NULL,
executesOID Integer NOT NULL,
PRIMARY KEY (assignedGID, executesGID),
FOREIGN KEY (assignedGID) REFERENCES ObjectType_Task (globalId),
FOREIGN KEY (executesGID) REFERENCES ObjectType_Employee (globalId)
)

CREATE TABLE Rel_CVC_Cache_executes_to_assigned (
executesGID Integer NOT NULL,
assignedGID Integer NOT NULL,
assignedOID Integer NOT NULL,
pinned Integer NOT NULL,
PRIMARY KEY (executesGID, assignedGID),
FOREIGN KEY (executesGID) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (assignedGID) REFERENCES ObjectType_Task (globalId)
)

CREATE TABLE Rel_CVC_isAuthor_to_writtenBy (
isAuthorGID Integer NOT NULL,
isAuthorOID Integer NOT NULL,
isAuthorVID Integer NOT NULL,
writtenByGID Integer NOT NULL,
writtenByOID Integer NOT NULL,
writtenByVID Integer NOT NULL,
PRIMARY KEY (isAuthorGID, writtenByGID),
FOREIGN KEY (isAuthorGID) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (writtenByGID) REFERENCES ObjectType_Offer (globalId)
)

CREATE TABLE Rel_CVC_Cache_isAuthor_to_writtenBy (
isAuthorGID Integer NOT NULL,
writtenByGID Integer NOT NULL,
writtenByOID Integer NOT NULL,
pinned Integer NOT NULL,
PRIMARY KEY (isAuthorGID, writtenByGID),
FOREIGN KEY (isAuthorGID) REFERENCES ObjectType_Employee (globalId),
FOREIGN KEY (writtenByGID) REFERENCES ObjectType_Offer (globalId)
)
```

Anhang D Regeln für den Benchmarkgenerator

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<generatorrules>

    <!-- Erzeugen von Objekten und Beziehungen -->

    <rule id="1" count="100" autocommit="true">
        <statement type="update">
            OBJECT Department ( name , description , budget , location )
            VALUES ( $value1$ , $value2$ , $value3$ , $value4$ )
        </statement>
        <select>
            <query sqltype="sql"> SELECT value as value1 FROM string</query>
            <variable type="String">value1</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value2 FROM string</query>
            <variable type="String">value2</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value3 FROM int</query>
            <variable type="String">value3</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value4 FROM string</query>
            <variable type="String">value4</variable>
        </select>
    </rule>

    <rule id="2" count="1000" autocommit="false">
        <statement type="update">
            CREATE NEW OBJECT Employee ( name , address , dateOfBirth , employedSince , salary )
            VALUES ( $value1$ , $value2$ , $value3$ , $value4$ , $value5$ )
        </statement>
        <statement type="update">
            CREATE NEW RELATIONSHIP worksIn_consistsOf WHERE consistsOf.GID = $gid$ AND worksIn.GID = $returnvalue1$
        </statement>
        <select>
            <query sqltype="sql"> SELECT value as value1 FROM string</query>
            <variable type="String">value1</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value2 FROM string</query>
            <variable type="String">value2</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value3 FROM date</query>
            <variable type="String">value3</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value4 FROM date</query>
            <variable type="String">value4</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value5 FROM int</query>
            <variable type="String">value5</variable>
        </select>
        <select>
            <query sqltype="sql">SELECT globalId as gid FROM Objecttype_Department</query>
            <variable type="int">gid</variable>
        </select>
    </rule>

    <rule id="3" count="5000" autocommit="false">
        <statement type="update">
            CREATE NEW OBJECT Offer ( customer , description , price , validUntil )
            VALUES ( $value1$ , $value2$ , $value3$ , $value4$ )
        </statement>
        <statement type="update">
            CREATE NEW RELATIONSHIP writtenBy_isAuthor WHERE isAuthor.GID = $gid$ AND writtenBy.GID = $returnvalue1$
        </statement>
```

```
        <select>
            <query sqltype="sql"> SELECT value as value1 FROM string</query>
            <variable type="String">value1</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value2 FROM string</query>
            <variable type="String">value2</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value3 FROM int</query>
            <variable type="String">value3</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value4 FROM date</query>
            <variable type="String">value4</variable>
        </select>
        <select>
            <query sqltype="sql">SELECT globalId as gid FROM Objecttype_Employee</query>
            <variable type="int">gid</variable>
        </select>
    </rule>

    <rule id="4" count="10000" autocommit="false">
        <statement type="update">
            CREATE NEW OBJECT Task ( name , description , startDate , dueDate )
            VALUES ( $value1$ , $value2$ , $value3$ , $value4$ )
        </statement>
        <statement type="update">
            CREATE NEW RELATIONSHIP contains_isPart WHERE contains.GID = $gid$ AND isPart.GID = $returnvalue1$
        </statement>
        <select>
            <query sqltype="sql"> SELECT value as value1 FROM string</query>
            <variable type="String">value1</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value2 FROM string</query>
            <variable type="String">value2</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value3 FROM date</query>
            <variable type="String">value3</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value4 FROM date</query>
            <variable type="String">value4</variable>
        </select>
        <select>
            <query sqltype="sql">SELECT globalId as gid FROM Objecttype_Offer</query>
            <variable type="int">gid</variable>
        </select>
    </rule>

    <rule id="5" count="10000" autocommit="false">
        <statement type="update">
            CREATE NEW OBJECT Costs ( wages , travelExpences , materialExpences , validUntil )
            VALUES ( $value1$ , $value2$ , $value3$ , $value4$ )
        </statement>
        <statement type="update">
            CREATE NEW RELATIONSHIP ratedCosts_causedBy WHERE ratedCosts.GID = $gid$ AND causedBy.GID = $returnvalue1$
        </statement>
        <select>
            <query sqltype="sql"> SELECT value as value1 FROM int</query>
            <variable type="String">value1</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value2 FROM int</query>
            <variable type="String">value2</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value3 FROM int</query>
            <variable type="String">value3</variable>
        </select>
        <select>
            <query sqltype="sql"> SELECT value as value4 FROM date</query>
            <variable type="String">value4</variable>
        </select>
        <select>
            <query sqltype="sql">SELECT globalId as gid FROM Objecttype_Task</query>
            <variable type="int">gid</variable>
        </select>
    </rule>

    <rule id="6" count="100" autocommit="true">
        <statement type="update">
            CREATE NEW RELATIONSHIP leads_hasBoss WHERE leads.GID = $gidA$ AND hasBoss.GID = $gidB$
        </statement>
        <select>
            <query sqltype="sql">SELECT globalId as gidA FROM Objecttype_Employee</query>
            <variable type="int">gidA</variable>
        </select>
        <select>
            <query sqltype="sql">SELECT globalId as gidB FROM Objecttype_Department</query>
            <variable type="int">gidB</variable>
        </select>
```

```
</rule>

<rule id="7" count="20000" autocommit="true">
    <statement type="update">
        CREATE NEW RELATIONSHIP executes_assigned WHERE executes.GID = $gidA$ AND assigned.GID = $gidB$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalId as gidA FROM Objecttype_Employee</query>
        <variable type="int">gidA</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT globalId as gidB FROM Objecttype_Task</query>
        <variable type="int">gidB</variable>
    </select>
</rule>

<!-- Workspace Objeckte & AttachmentBeziehungen-->

<rule id="8" count="100" autocommit="true">
    <statement type="update">
        CREATE NEW OBJECT Project ( budget ) VALUES ( $value1$ )
    </statement>
    <select>
        <query sqltype="sql"> SELECT value as value1 FROM int</query>
        <variable type="String">value1</variable>
    </select>
</rule>

<rule id="9" count="5000" autocommit="true">
    <statement type="update">
        ATTACH $ObjectTypeName$ GID = $gidA$ TO WORKSPACE Project GID = $gidB$
    </statement>
    <select>
        <query sqltype="sql">SELECT ObjectTypeName FROM SysTabWorkspaceAttachments</query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT globalId as gidA FROM Objecttype_$ObjectTypeName$</query>
        <variable type="int">gidA</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT globalId as gidB FROM Objecttype_Project</query>
        <variable type="int">gidB</variable>
    </select>
</rule>

<rule id="10" count="100" autocommit="true">
    <statement type="update">
        DETACH $ObjectTypeName$ GID = $gidA$ FROM WORKSPACE Project GID = $gidB$
    </statement>
    <select>
        <query sqltype="sql">SELECT ObjectTypeName FROM SysTabWorkspaceAttachments</query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT wsglobalId as gidB, otglobalId as gidA FROM attached_$ObjectTypeName$_Project</query>
        <variable type="int">gidA</variable>
        <variable type="int">gidB</variable>
    </select>
</rule>

<rule id="11" count="50" autocommit="true">
    <statement type="update">CREATE NEW OBJECT HumanResources </statement>
</rule>


<rule id="12" count="200" autocommit="true">
    <statement type="update">
        ATTACH $ObjectTypeName$ GID = $gidA$ TO WORKSPACE HumanResources GID = $gidB$
    </statement>
    <select>
        <query sqltype="sql">
            SELECT ObjectTypeName FROM SysTabWorkspaceAttachments WHERE WorkspaceTypeName='HumanResources'
        </query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT globalId as gidA FROM Objecttype_$ObjectTypeName$</query>
        <variable type="int">gidA</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT globalId as gidB FROM Objecttype_HumanResources</query>
        <variable type="int">gidB</variable>
    </select>
</rule>


<rule id="13" count="25" autocommit="true">
    <statement type="update">
        DETACH $ObjectTypeName$ GID = $gidA$  FROM WORKSPACE HumanResources GID = $gidB$
    </statement>
    <select>
        <query sqltype="sql">
            SELECT ObjectTypeName FROM SysTabWorkspaceAttachments WHERE WorkspaceTypeName='HumanResources'
```

```
        </query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">
            SELECT wsglobalId as gidB, otglobalId as gidA FROM attached_$ObjectTypeName$_HumanResources
        </query>
        <variable type="int">gidA</variable>
        <variable type="int">gidB</variable>
    </select>
</rule>


<!-- Erzeugen von Nachfolgern -->

<rule id="14" count="10000" autocommit="true">
    <statement type="update">CREATE SUCCESSOR OF OBJECT $ObjectTypeName$ WHERE GID = $gid$</statement>
    <select>
        <query sqltype="sql"> SELECT name as ObjectTypeName FROM SysTabObjectTypes</query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_$ObjectTypeName$</query>
        <variable type="int">gid</variable>
    </select>
</rule>


<rule id="15" count="10000" autocommit="true">
    <statement type="update">CREATE SUCCESSOR OF OBJECT Employee WHERE GID = $gid$</statement>
    <select>
        <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_Employee</query>
        <variable type="int">gid</variable>
    </select>
</rule>


<rule id="16" count="500"  autocommit="true">
    <statement type="update">
        CREATE SUCCESSOR OF OBJECT $ObjectTypeName$ USE WORKSPACE $WorkspaceTypeName$
        WHERE GID = $gid$ AND $ObjectTypeName$ WHERE OID = $oid$
    </statement>
    <select>
        <query sqltype="sql">
            SELECT name as WorkspaceTypeName FROM SysTabObjectTypes WHERE ObjectType='workspace'
        </query>
        <variable type="String">WorkspaceTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">
            SELECT ObjectTypeName FROM SysTabWorkspaceAttachments WHERE WorkspaceTypeName='$WorkspaceTypeName$'
        </query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">
            SELECT wsglobalId as gid, otobjectId as oid FROM attached_$ObjectTypeName$_$WorkspaceTypeName$
        </query>
        <variable type="int">gid</variable>
        <variable type="int">oid</variable>
    </select>
</rule>


<!-- Erzeugen von Kopien -->

<rule id="17" count="20000" autocommit="true">
    <statement type="update">
        COPY OBJECT $ObjectTypeName$ WHERE OID = $oid$ AND VID = $vid$
    </statement>
    <select>
        <query sqltype="sql">
            SELECT name as ObjectTypeName FROM SysTabObjectTypes WHERE ObjectType='versioned'
        </query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT objectId as oid, versionId as vid FROM ObjectType_$ObjectTypeName$</query>
        <variable type="int">oid</variable>
        <variable type="int">vid</variable>
    </select>
</rule>

<!-- Einfrieren von Objekten -->

<rule id="18" count="1000" autocommit="true">
    <statement type="update">
        FREEZE OBJECT $ObjectTypeName$ WHERE OID = $oid$ AND VID = $vid$
    </statement>
    <select>
        <query sqltype="sql"> SELECT name as ObjectTypeName FROM SysTabObjectTypes</query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
```

```
        <query sqltype="sql"> SELECT objectId as oid, versionId as vid FROM ObjectType_$ObjectTypeName$</query>
        <variable type="int">oid</variable>
        <variable type="int">vid</variable>
    </select>
</rule>

<rule id="19" count="100" autocommit="true">
    <statement type="update">
        FREEZE OBJECT $ObjectTypeName$ USE WORKSPACE $WorkspaceTypeName$
        WHERE GID = $gid$ AND $ObjectTypeName$ WHERE OID = $oid$
    </statement>
    <select>
        <query sqltype="sql">
            SELECT name as WorkspaceTypeName FROM SysTabObjectTypes WHERE ObjectType='workspace'
        </query>
        <variable type="String">WorkspaceTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">
            SELECT ObjectTypeName FROM SysTabWorkspaceAttachments WHERE WorkspaceTypeName='$WorkspaceTypeName$'
        </query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">
            SELECT wsglobalId as gid, otobjectId as oid FROM attached_$ObjectTypeName$_$WorkspaceTypeName$
        </query>
        <variable type="int">gid</variable>
        <variable type="int">oid</variable>
    </select>
</rule>

<!-- Löschen von Objekten und Beziehungen -->

<rule id="20" count="5000" autocommit="true">
    <statement type="update">DELETE OBJECT $ObjectTypeName$ WHERE GID = $gid$</statement>
    <select>
        <query sqltype="sql">
            SELECT name as ObjectTypeName FROM SysTabObjectTypes WHERE ObjectType='versioned'
        </query>
        <variable type="String">ObjectTypeName</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT objectId as oid FROM ObjectType_$ObjectTypeName$</query>
        <variable type="int">oid</variable>
    </select>
    <select>
        <query sqltype="sql">
            SELECT globalId as gid FROM ObjectType_$ObjectTypeName$
            WHERE objectId = $oid$ ORDER BY globalId DESC FETCH FIRST 1 ROWS ONLY
        </query>
        <variable type="int">gid</variable>
    </select>
</rule>

<!-- Anfragen im Versionsbaum -->

<rule id="21" count="500" autocommit="true">
    <statement type="query">GET ROOT OF Employee WHERE GID = $gid$</statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Employee</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="22" count="500" autocommit="true">
    <statement type="query">GET ALTERNATIVES OF Employee WHERE GID = $gid$</statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Employee</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="23" count="500" autocommit="true">
    <statement type="query">GET SUCCESSORS OF Employee WHERE GID = $gid$</statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Employee</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="24" count="500"  autocommit="true">
    <statement type="query">GET PREDECESSOR OF Employee WHERE GID = $gid$</statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Employee</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="25" count="500"  autocommit="true">
    <statement type="query">
        GET BASEVERSION OF Employee WHERE OID = $oid$ AND VID1 = $vid1$ AND VID2 = $vid2$
    </statement>
    <select>
```

```
            <query sqltype="sql"> SELECT objectid as oid FROM ObjectType_Employee</query>
            <variable type="int">oid</variable>
        </select>
        <select>
            <query sqltype="sql">
                SELECT versionid as vid1 FROM ObjectType_Employee WHERE objectId=$oid$ AND versionId &gt; 1
            </query>
            <variable type="int">vid1</variable>
        </select>
        <select>
            <query sqltype="sql">
                SELECT versionid as vid2 FROM ObjectType_Employee
                WHERE objectId=$oid$ AND versionId &gt; 1 AND versionId &lt;&gt; $vid1$
            </query>
            <variable type="int">vid2</variable>
        </select>
    </rule>

    <!-- Merge -->

    <rule id="26" count="500"  autocommit="true">
        <statement type="update">
            MERGE OBJECTS Employee WHERE OID = $oid$ AND PRIM_VID = $vid1$ AND SEC_VID = $vid2$
        </statement>
        <select>
            <query sqltype="sql">SELECT objectid as oid FROM ObjectType_Employee</query>
            <variable type="int">oid</variable>
        </select>
        <select>
            <query sqltype="sql">
                SELECT versionid as vid1 FROM ObjectType_Employee WHERE objectId=$oid$ AND versionId &gt; 1
            </query>
            <variable type="int">vid1</variable>
        </select>
        <select>
            <query sqltype="sql">
                SELECT versionid as vid2 FROM ObjectType_Employee
                WHERE objectId=$oid$ AND versionId &gt; 1 AND versionId &lt;&gt; $vid1$
            </query>
            <variable type="int">vid2</variable>
        </select>
    </rule>

    <!-- GetCVC -->

    <rule id="27" count="500"  autocommit="true">
        <statement type="query">GET CVC FROM Task WHERE GID = $gid$ ROLENAME ratedCosts</statement>
        <select>
            <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_Task</query>
            <variable type="int">gid</variable>
        </select>
    </rule>

    <rule id="28" count="500"  autocommit="true">
        <statement type="query">GET CVC FROM Task WHERE GID = $gid$ ROLENAME isPart</statement>
        <select>
            <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_Task</query>
            <variable type="int">gid</variable>
        </select>
    </rule>

    <rule id="29" count="500"  autocommit="true">
        <statement type="query">GET CVC FROM Employee WHERE GID = $gid$ ROLENAME executes</statement>
        <select>
            <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_Employee</query>
            <variable type="int">gid</variable>
        </select>
    </rule>

    <!-- Pinning -->

    <rule id="30" count="500"  autocommit="true">
        <statement type="update">
            PIN Costs WHERE GID = $globalid$ IN CVC Task WHERE GID = $gid$ ROLENAME ratedCosts
        </statement>
        <select>
            <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_Task</query>
            <variable type="int">gid</variable>
        </select>
        <select>
            <query sqltype="dssql">GET CVC FROM Task WHERE GID = $gid$ ROLENAME ratedCosts</query>
            <variable type="int">globalid</variable>
        </select>
    </rule>

    <rule id="31" count="500"  autocommit="true">
        <statement type="update">
            PIN Task WHERE GID = $globalid$ IN CVC Costs WHERE GID = $gid$ ROLENAME causedBy
        </statement>
        <select>
            <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_Costs</query>
            <variable type="int">gid</variable>
        </select>
        <select>
```

```
        <query sqltype="dssql">GET CVC FROM Costs WHERE GID = $gid$ ROLENAME causedBy</query>
        <variable type="int">globalid</variable>
    </select>
</rule>

<rule id="32" count="500"  autocommit="true">
    <statement type="update">
        PIN Task WHERE GID = $globalid$ IN CVC Employee WHERE GID = $gid$ ROLENAME executes
    </statement>
    <select>
        <query sqltype="sql"> SELECT globalid as gid FROM ObjectType_Employee</query>
        <variable type="int">gid</variable>
    </select>
    <select>
        <query sqltype="dssql">GET CVC FROM Employee WHERE GID = $gid$ ROLENAME executes</query>
        <variable type="int">globalid</variable>
    </select>
</rule>

<!-- Update -->

<rule id="33" count="5000" autocommit="true">
    <statement type="update">
        UPDATE Employee SET name = $value1$ , address = $value2$ , dateOfBirth = $value3$ ,
        employedSince = $value4$ , salary = $value5$ WHERE GlobalID = $gid$
    </statement>
    <select>
        <query sqltype="sql"> SELECT value as value1 FROM string</query>
        <variable type="String">value1</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT value as value2 FROM string</query>
        <variable type="String">value2</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT value as value3 FROM date</query>
        <variable type="String">value3</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT value as value4 FROM date</query>
        <variable type="String">value4</variable>
    </select>
    <select>
        <query sqltype="sql"> SELECT value as value5 FROM int</query>
        <variable type="String">value5</variable>
    </select>
    <select>
        <query sqltype="sql">SELECT globalId as gid FROM Objecttype_Employee</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<!-- Select -->

<rule id="34" count="500" autocommit="true">
    <statement type="query">
        SELECT Employee.* FROM Department -- consistsOf --> Employee WHERE Department.globalId = $gid$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Department</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="35" count="500" autocommit="true">
    <statement type="query">
        SELECT Task.* FROM Department -- consistsOf --> Employee -- executes --> Task
        WHERE Department.globalId = $gid$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Department</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="36" count="500" autocommit="true">
    <statement type="query">
        SELECT Offer.* FROM Department -- consistsOf --> Employee -- isAuthor --> Offer
        WHERE Department.globalId = $gid$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Department</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="37" count="500" autocommit="true">
    <statement type="query">
        SELECT Employee.* FROM Offer -- contains --> Task -- assigned --> Employee
        WHERE Offer.globalId = $gid$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Offer</query>
        <variable type="int">gid</variable>
```

```
    </select>
</rule>

<rule id="38" count="500" autocommit="true">
    <statement type="query">
        SELECT Costs.* FROM Offer -- contains --> Task -- ratedCosts --> Costs
        WHERE Offer.globalId = $gid$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Offer</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="39" count="500" autocommit="true">
    <statement type="query">
        SELECT Employee.* FROM Offer -- contains --> Task -- assigned --> Employee
        WHERE Offer.globalId = $gid$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Offer</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="40" count="500" autocommit="true">
    <statement type="query">
        SELECT Costs.* FROM Department -- consistsOf --> Employee -- isAuthor --> Offer
        -- contains --> Task -- ratedCosts --> Costs WHERE Department.globalId = $gid$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Department</query>
        <variable type="int">gid</variable>
    </select>
</rule>

<rule id="41" count="500" autocommit="true">
    <statement type="query">
        SELECT Department.* FROM Costs -- causedBy --> Task -- isPart --> Offer
        -- writtenBy --> Employee -- worksIn --> Department WHERE Costs.globalId = $gid$
    </statement>
    <select>
        <query sqltype="sql">SELECT globalid as gid FROM ObjectType_Costs</query>
        <variable type="int">gid</variable>
    </select>
</rule>


<!-- Select mit Workspace -->

<rule id="42" count="500" autocommit="true">
    <statement type="query">
        SELECT Employee.* FROM Department -- consistsOf --> Employee
        USE WORKSPACE HumanResources WHERE GID = $gid$ AND Department WHERE OID = $oid$
    </statement>
    <select>
        <query sqltype="sql">SELECT wsGlobalId as gid, otObjectId as oid FROM attached_Department_HumanResources</query>
        <variable type="int">gid</variable>
        <variable type="int">oid</variable>
    </select>
</rule>

<rule id="43" count="500" autocommit="true">
    <statement type="query">
        SELECT Task.* FROM Department -- consistsOf --> Employee -- executes --> Task
        USE WORKSPACE Project WHERE GID = $gid$ AND Department WHERE OID = $oid$
    </statement>
    <select>
        <query sqltype="sql">SELECT wsGlobalId as gid, otObjectId as oid FROM attached_Department_Project</query>
        <variable type="int">gid</variable>
        <variable type="int">oid</variable>
    </select>
</rule>

<rule id="44" count="500" autocommit="true">
    <statement type="query">
        SELECT Offer.* FROM Department -- consistsOf --> Employee -- isAuthor --> Offer
        USE WORKSPACE Project WHERE GID = $gid$ AND Department WHERE OID = $oid$
    </statement>
    <select>
        <query sqltype="sql">SELECT wsGlobalId as gid, otObjectId as oid FROM attached_Department_Project</query>
        <variable type="int">gid</variable>
        <variable type="int">oid</variable>
    </select>
</rule>

<rule id="45" count="500" autocommit="true">
    <statement type="query">
        SELECT Employee.* FROM Offer -- contains --> Task -- assigned --> Employee
        USE WORKSPACE Project WHERE GID = $gid$ AND Offer WHERE OID = $oid$
    </statement>
    <select>
        <query sqltype="sql">SELECT wsGlobalId as gid, otObjectId as oid FROM attached_Offer_Project</query>
        <variable type="int">gid</variable>
```

```
            <variable type="int">oid</variable>
        </select>
    </rule>

    <rule id="46" count="500" autocommit="true">
        <statement type="query">
            SELECT Costs.* FROM Offer -- contains --> Task -- ratedCosts --> Costs
            USE WORKSPACE Project WHERE GID = $gid$ AND Offer WHERE OID = $oid$
        </statement>
        <select>
            <query sqltype="sql">SELECT wsGlobalId as gid, otObjectId as oid FROM attached_Offer_Project</query>
            <variable type="int">gid</variable>
            <variable type="int">oid</variable>
        </select>
    </rule>

</generatorrules>
```

Anhang E Ergebnisse der Analyse mit Softwaremetriken

E.1 Variante 1

Reduktionsmethode	Ausdrücke	Anweisungen	Fallunter-scheidungen	Schleifen	Verschachtel-ungstiefe	Zyklonmatische Komplexität	Halstead Aufwand
Attach	49	16	5	2	3	4	10011
Copy Object	94	28	5	3	3	4	28740
Create Successor	224	81	15	5	8	14	186046
Delete Object	88	33	7	2	6	6	25026
Delete Relationship	44	18	4	0	1	3	11787
Detach	59	24	7	2	3	6	14468
Freeze Object	111	48	12	4	6	11	53477
Get Alternatives	38	14	3	1	1	2	8665
Get Baseversion	111	40	6	2	2	5	49382
Get CVC	50	25	6	1	2	5	18890
Get Predecessor	38	14	3	1	1	2	8665
Get RelatedObjects (a.v.W)	45	16	2	0	1	1	10368
Get RelatedObjects (i.v.W)	22	7	1	0	0	0	3522
Get Root	30	12	3	1	1	2	6137
Get Successors	38	14	3	1	1	2	8665
Merge Objects	116	55	14	2	3	13	34379
New Object	193	27	5	2	2	4	33080
New Relationship	210	79	6	0	2	5	116729
Pinning	45	15	3	0	1	2	12206
Select	18	7	2	0	1	1	1947
Update Object	30	14	3	0	2	2	5312
ColumnList	5	5	2	1	1	1	462
DisplayedColumn	9	9	3	1	2	2	2091
FromClause	16	11	4	2	2	3	2631
FromItem (a.v.W)	76	45	10	1	4	9	52140
FromItem (i.v.W)	29	29	8	1	3	7	17291
ObjectSelect	8	8	4	0	0	3	1275
Order	6	5	2	1	1	1	552
OrderItem	3	2	1	0	0	0	31
Update	7	5	2	1	1	1	513
UpdateItem	26	22	7	0	2	6	6061
Where	9	9	3	1	2	2	2091
Summe	1847	737	161	38	68	129	732640

E.2 Variante 2

Reduktionsmethode	Ausdrücke	Anweisungen	Fallunter-scheidungen	Schleifen	Verschachtel-ungstiefe	Zyklonmatische Komplexität	Halstead Aufwand
Attach	49	16	5	2	3	4	10011
Copy Object	94	28	5	3	3	4	28740
Create Successor	224	81	15	5	8	14	186046
Delete Object	88	33	7	2	6	6	25026
Delete Relationship	81	35	7	0	2	6	51365
Detach	59	24	7	2	3	6	14468
Freeze Object	111	48	12	4	6	11	53477
Get Alternatives	38	14	3	1	1	2	8665
Get Baseversion	111	40	6	2	2	5	49382
Get CVC	50	25	6	1	2	5	18890
Get Predecessor	38	14	3	1	1	2	8665
Get RelatedObjects (a.v.W)	69	38	7	0	3	6	29355
Get RelatedObjects (i.v.W)	67	42	6	0	2	5	48239
Get Root	30	12	3	1	1	2	6137
Get Successors	38	14	3	1	1	2	8665
Merge Objects	116	55	14	2	3	13	34379
New Object	193	27	5	2	2	4	33080
New Relationship	179	69	7	0	2	6	119910
Pinning	48	18	4	0	2	3	15619
Select	18	7	2	0	1	1	1947
Update Object	30	14	3	0	2	2	5312
ColumnList	5	5	2	1	1	1	462
DisplayedColumn	9	9	3	1	2	2	2091
FromClause	16	11	4	2	2	3	2631
FromItem (a.v.W)	75	43	11	1	4	10	38582
FromItem (i.v.W)	46	43	11	1	4	10	38457
ObjectSelect	8	8	4	0	0	3	1275
Order	6	5	2	1	1	1	552
OrderItem	3	2	1	0	0	0	31
Update	7	5	2	1	1	1	513
UpdateItem	26	22	7	0	2	6	6061
Where	9	9	3	1	2	2	2091
Summe	1941	816	180	38	75	148	850124

E.3 Variante 3

Reduktionsmethode	Ausdrücke	Anweisungen	Fallunter-scheidungen	Schleifen	Verschachtel-ungstiefe	Zyklonmatische Komplexität	Halstead Aufwand
Attach	49	16	5	2	3	4	10011
Copy Object	94	28	5	3	3	4	28740
Create Successor	224	81	15	5	8	14	186046
Delete Object	88	33	7	2	6	6	25026
Delete Relationship	167	65	12	1	3	11	129687
Detach	59	24	7	2	3	6	14468
Freeze Object	111	48	12	4	6	11	53477
Get Alternatives	38	14	3	1	1	2	8665
Get Baseversion	111	40	6	2	2	5	49382
Get CVC	50	25	6	1	2	5	18890
Get Predecessor	38	14	3	1	1	2	8665
Get RelatedObjects (a.v.W)	92	57	10	0	3	9	52092
Get RelatedObjects (i.v.W)	67	42	6	0	2	5	48239
Get Root	30	12	3	1	1	2	6137
Get Successors	38	14	3	1	1	2	8665
Merge Objects	116	55	14	2	3	13	34379
New Object	193	27	5	2	2	4	33080
New Relationship	252	93	11	0	3	10	233527
Pinning	48	18	4	0	2	3	15619
Select	18	7	2	0	1	1	1947
Update Object	30	14	3	0	2	2	5312
ColumnList	5	5	2	1	1	1	462
DisplayedColumn	9	9	3	1	2	2	2091
FromClause	16	11	4	2	2	3	2631
FromItem (a.v.W)	95	68	18	1	5	17	74867
FromItem (i.v.W)	66	68	18	1	5	17	73738
ObjectSelect	8	8	4	0	0	3	1275
Order	6	5	2	1	1	1	552
OrderItem	3	2	1	0	0	0	31
Update	7	5	2	1	1	1	513
UpdateItem	26	22	7	0	2	6	6061
Where	9	9	3	1	2	2	2091
Summe	2163	939	206	39	79	174	1136366

E.4 Variante 4

Reduktionsmethode	Ausdrücke	Anweisungen	Fallunter-scheidungen	Schleifen	Verschachtel-ungstiefe	Zyklonmatische Komplexität	Halstead Aufwand
Attach	49	16	5	2	3	4	10011
Copy Object	94	28	5	3	3	4	28740
Create Successor	224	81	15	5	8	14	186046
Delete Object	88	33	7	2	6	6	25026
Delete Relationship	264	84	17	0	4	16	211553
Detach	59	24	7	2	3	6	14468
Freeze Object	111	48	12	4	6	11	53477
Get Alternatives	38	14	3	1	1	2	8665
Get Baseversion	111	40	6	2	2	5	49382
Get CVC	50	25	6	1	2	5	18890
Get Predecessor	38	14	3	1	1	2	8665
Get RelatedObjects (a.v.W)	83	58	10	0	3	9	51647
Get RelatedObjects (i.v.W)	67	42	6	0	2	5	48239
Get Root	30	12	3	1	1	2	6137
Get Successors	38	14	3	1	1	2	8665
Merge Objects	116	55	14	2	3	13	34379
New Object	193	27	5	2	2	4	33080
New Relationship	306	104	14	0	3	13	277652
Update Cache Table	69	24	3	0	2	2	15966
Update Object Table	25	11	3	0	2	2	2954
Pinning	86	36	8	0	3	7	33528
Select	18	7	2	0	1	1	1947
Update Object	30	14	3	0	2	2	5312
ColumnList	5	5	2	1	1	1	462
DisplayedColumn	9	9	3	1	2	2	2091
FromClause	16	11	4	2	2	3	2631
FromItem (a.v.W)	67	67	19	1	5	18	78375
FromItem (i.v.W)	66	68	18	1	5	17	73738
ObjectSelect	8	8	4	0	0	3	1275
Order	6	5	2	1	1	1	552
OrderItem	3	2	1	0	0	0	31
Update	7	5	2	1	1	1	513
UpdateItem	26	22	7	0	2	6	6061
Where	9	9	3	1	2	2	2091
Summe	2409	1024	225	38	85	191	1302249

Anhang F

Diagramme zur Bewertung durch Softwaremetriken

Abbildung F.1 Vergleich der Operation `New Relationship`

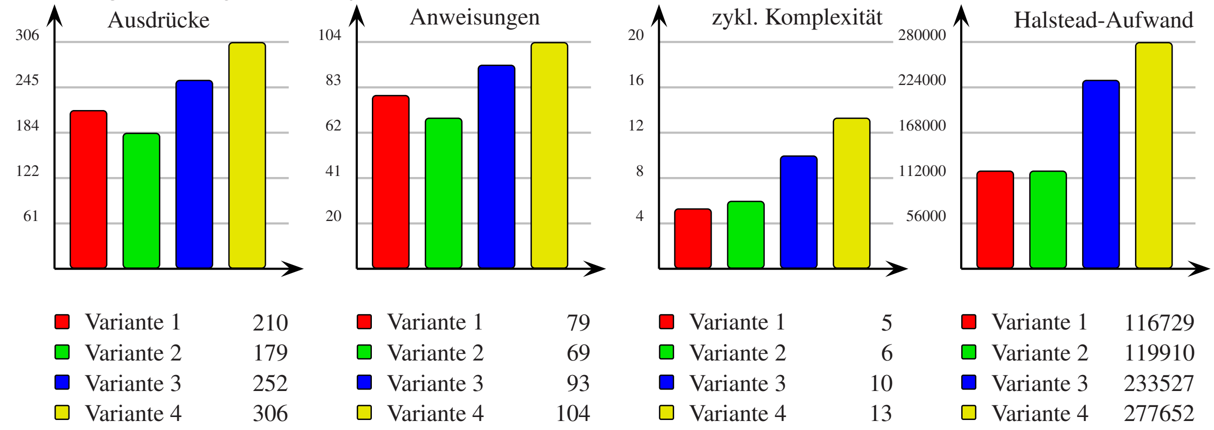


Abbildung F.2 Vergleich der Operation `Delete Relationship`

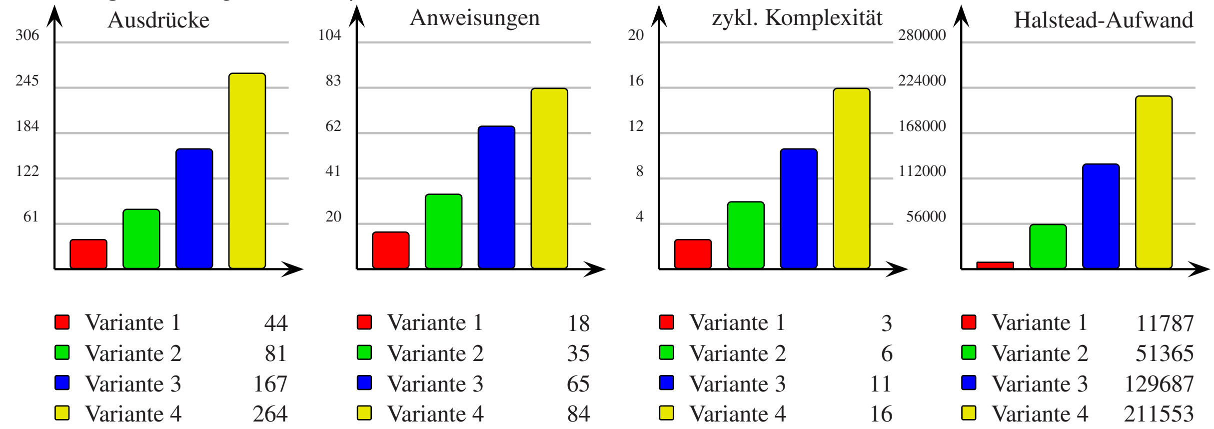


Abbildung F.3 Vergleich der Operation `Pinning`

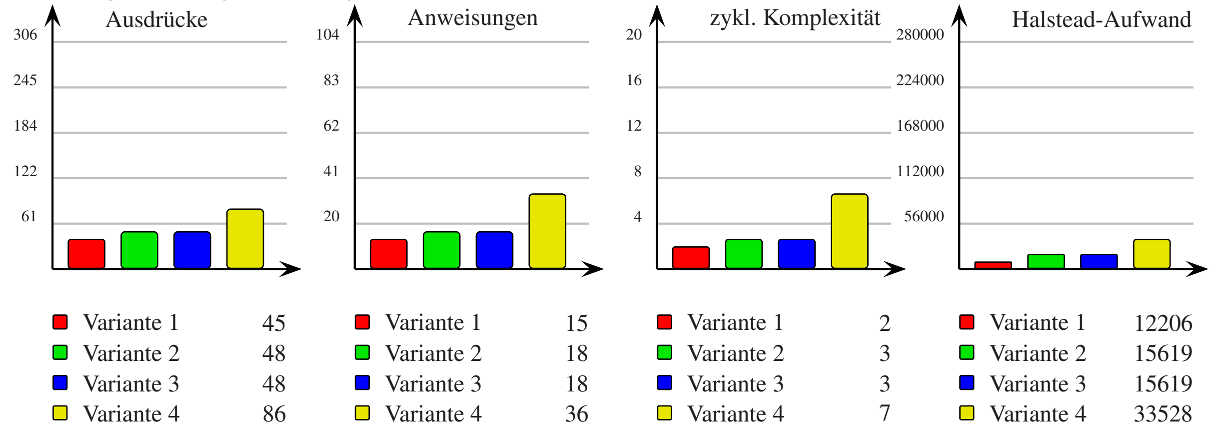


Abbildung F.4 Vergleich der Operation `Get RelatedObjects (i.v.W)`

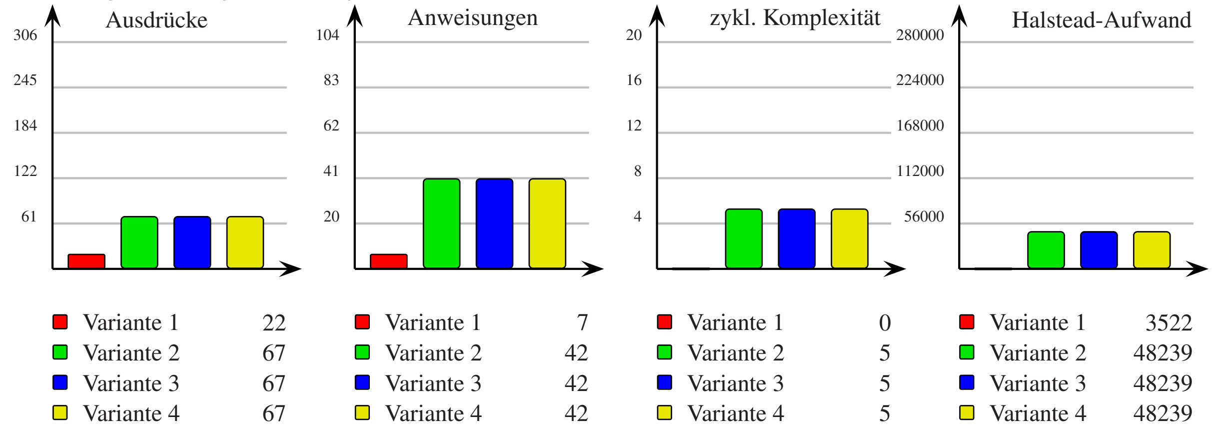


Abbildung F.5 Vergleich der Operation `Get RelatedObjects (a.v.W)`

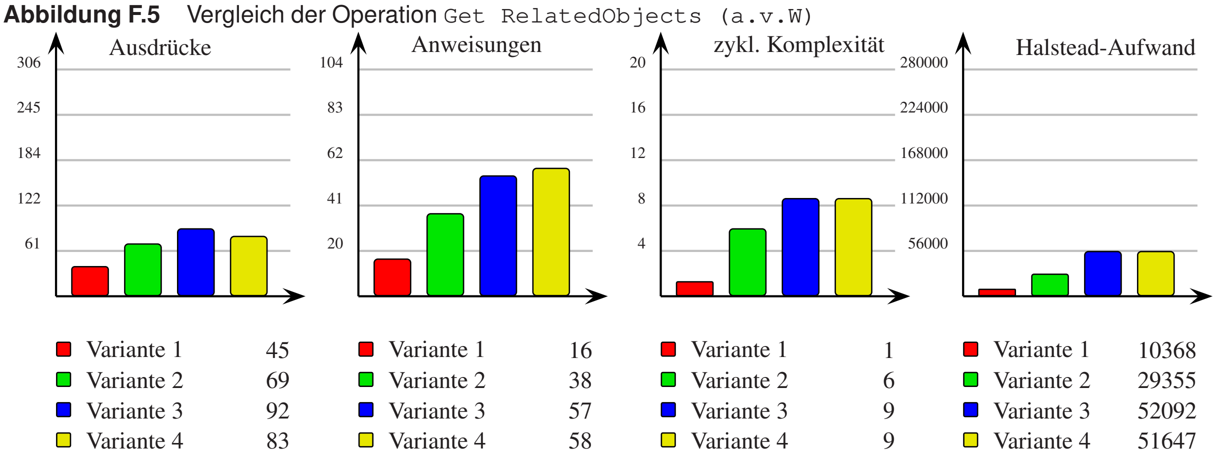


Abbildung F.6 Vergleich der Operation `FromItem (i.v.W)`

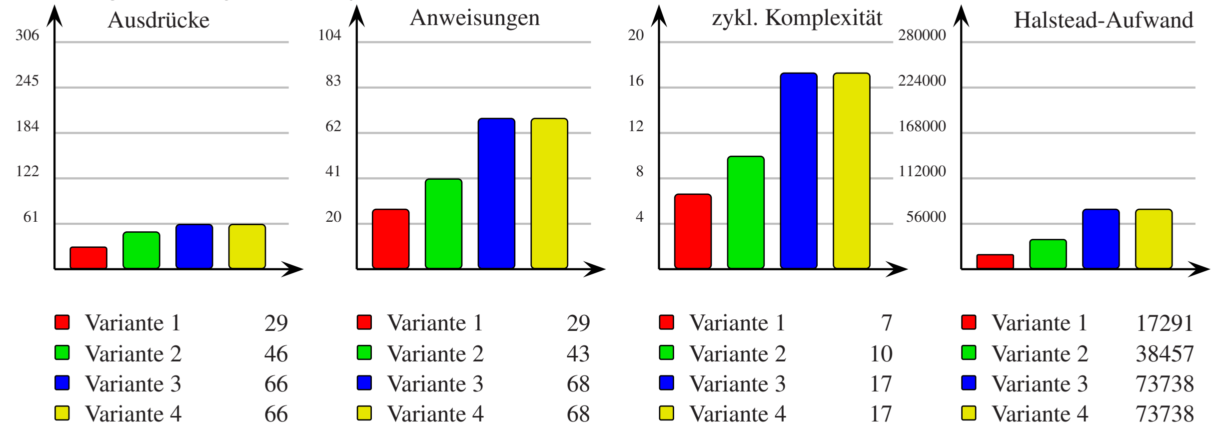
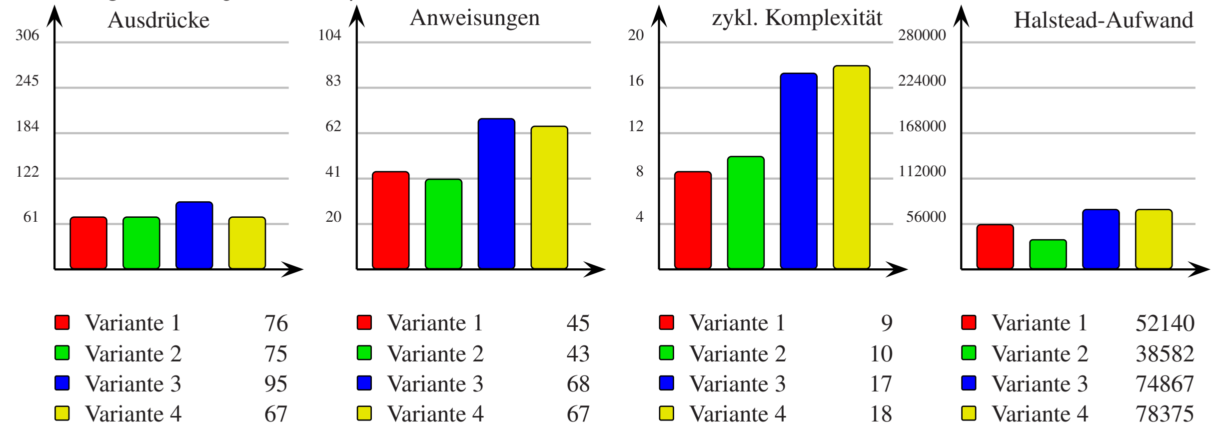


Abbildung F.7 Vergleich der Operation `FromItem (a.v.W)`



Anhang G Ergebnisse der Leistungsuntersuchung

G.1 Variante 1

G.1.1 Absolute Zeiten in μs

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
alle Befehle	SUM	7585666675	5229167	61253288	12703662	7506480558	733473
CREATE NEW OBJECT	SUM	1496710	22312	76765	13207	1384426	700
	MIN	7489	21	149	54	7265	4
	MAX	54228	10576	22725	1248	19679	4
	MEDIAN	8011	44	251	62	7654	4
	AVG	8553	127	439	75	7911	4
	STDDEV	3934	810	1723	97	1305	0
CREATE NEW RELATIONSHIP	SUM	1947598107	844668	3214728	1256996	1942281715	60200
	MIN	4011	36	99	35	3841	2
	MAX	318243	9982	31799	11415	265047	3
	MEDIAN	85792	37	122	57	85576	3
	AVG	96895	42	160	63	96631	2
	STDDEV	55105	116	233	217	54538	0
▷ kein FRE	SUM	445993	12799	11384	3835	417975	200
	MIN	4060	85	99	35	3841	2
	MAX	10652	1743	515	139	8255	2
	MEDIAN	4233	101	103	36	3993	2
	AVG	4460	128	114	38	4180	2
	STDDEV	905	188	43	11	664	0
▷ ein FRE	SUM	1947152114	831869	3203344	1253161	1941863740	60000
	MIN	8433	36	107	52	8238	3
	MAX	318243	9982	31799	11415	265047	3
	MEDIAN	85916	37	122	57	85700	3
	AVG	97358	42	160	63	97093	3
	STDDEV	54847	116	234	218	54280	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	1910660647	2549330	15042061	5342602	1887726654	296000
	MIN	11377	67	246	89	10975	6
	MAX	702163	188846	90576	159683	263058	20
	MEDIAN	72898	72	677	202	71947	13
	AVG	73487	98	579	205	72605	11
	STDDEV	63215	1409	648	1050	60108	5
▷ 1 Objekt + 1 kein FRE	SUM	12330019	154185	424459	110535	11640840	6000
	MIN	11584	84	375	95	11030	6
	MAX	49584	14988	2204	1924	30468	6
	MEDIAN	11929	104	408	102	11315	6
	AVG	12330	154	424	111	11641	6
	STDDEV	1964	520	117	83	1244	0
▷ 1 Objekt + 1 zwei FRE	SUM	122901977	1001230	2738325	1226980	117935442	60000
	MIN	11377	67	246	89	10975	6
	MAX	428496	188846	8818	159683	71149	6
	MEDIAN	11828	70	264	96	11398	6
	AVG	12290	100	274	123	11794	6
	STDDEV	5789	1896	186	1610	2098	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	1220610095	959109	7087762	2257666	1210305558	130000
	MIN	37656	70	577	196	36813	13
	MAX	403020	122569	8413	8980	263058	13
	MEDIAN	117009	73	700	206	116030	13
	AVG	122061	96	709	226	121031	13

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	STDDEV	46897	1239	194	290	45174	0
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	554818556	434806	4791515	1747421	547844814	100000
	MIN	65983	70	863	303	64747	20
	MAX	328962	3495	90576	37246	197645	20
	MEDIAN	110284	73	898	315	108998	20
	AVG	110964	87	958	349	109569	20
	STDDEV	24374	103	1291	587	22392	0
CREATE SUCCESSOR	SUM	649043061	434883	28695982	1676356	618235840	119788
	MIN	7357	15	643	46	6653	5
	MAX	792067	7772	305370	9100	469825	90
	MEDIAN	18250	16	1145	60	17029	5
	AVG	31661	21	1400	82	30158	5
	STDDEV	54849	115	3510	224	51000	3
▷ außerhalb von Workspaces	SUM	635427683	413066	28180585	1621330	605212702	115964
	MIN	7357	15	643	46	6653	5
	MAX	792067	7772	305370	9100	469825	90
	MEDIAN	18241	16	1150	60	17015	5
	AVG	31771	21	1409	81	30261	5
	STDDEV	55435	115	3551	226	51543	3
▷ innerhalb von Workspaces	SUM	13615378	21817	515397	55026	13023138	3824
	MIN	14140	34	696	70	13340	6
	MAX	232172	2688	8577	1335	219572	72
	MEDIAN	21408	38	767	77	20526	6
	AVG	27231	44	1031	110	26046	7
	STDDEV	19909	118	845	123	18822	6
COPY OBJECTS	SUM	2313813808	422247	7459647	2833759	2303098155	164570
	MIN	7862	17	107	53	7685	4
	MAX	1304181	10450	12425	11563	1269743	227
	MEDIAN	8728	18	130	64	8516	4
	AVG	115691	21	373	142	115155	8
	STDDEV	169559	105	507	287	168660	7
DELETE OBJECTS	SUM	250272530	77430	3049391	564327	246581382	47682
	MIN	11479	13	194	41	11231	4
	MAX	498482	3558	41674	5705	447545	65
	MEDIAN	38299	14	424	73	37788	7
	AVG	50055	15	610	113	49316	9
	STDDEV	35344	50	755	184	34354	6
MERGE OBJECTS	SUM	9489286	19300	837891	59752	8572343	2205
	MIN	5195	27	1282	98	3788	4
	MAX	46233	2797	14490	1695	27251	5
	MEDIAN	18881	31	1539	106	17205	4
	AVG	18979	39	1676	120	17145	4
	STDDEV	2550	127	755	104	1564	0
FREEZE	SUM	16747277	39691	300250	47642	16359694	4211
	MIN	1609	25	107	15	1462	2
	MAX	85916	3198	3239	1350	78129	20
	MEDIAN	6768	29	150	20	6569	2
	AVG	15225	36	273	43	14872	3
	STDDEV	14675	114	269	74	14218	2
▷ außerhalb von Workspaces	SUM	15806442	36234	277034	44408	15448766	3947
	MIN	1609	25	107	15	1462	2
	MAX	85916	3198	3239	1350	78129	20
	MEDIAN	6521	29	137	18	6337	2
	AVG	15806	36	277	44	15449	3
	STDDEV	15089	120	272	77	14620	3
▷ innerhalb von Workspaces	SUM	940835	3457	23216	3234	910928	264
	MIN	1779	31	149	21	1578	2
	MAX	66449	41	1731	228	64449	19
	MEDIAN	7262	34	156	23	7049	2
	AVG	9408	35	232	32	9109	2
	STDDEV	7316	2	230	27	7057	2
ATTACH und DETACH	SUM	150075293	148969	800496	185140	148940688	14867
	MIN	3084	17	13	9	3045	1
	MAX	213643	2838	6119	4407	200279	14
	MEDIAN	7257	26	164	29	7038	3
	AVG	28183	28	150	35	27970	2
	STDDEV	43617	68	181	73	43294	1
▷ ATTACH	SUM	148230758	136423	749656	177351	147167328	14234
	MIN	3113	17	13	9	3074	1
	MAX	213310	2505	6119	4407	200279	14
	MEDIAN	7259	26	164	29	7040	3
	AVG	29065	27	147	35	28856	2
	STDDEV	44307	44	172	74	44016	1
▷ DETACH	SUM	1844535	12546	50840	7789	1773360	633
	MIN	3091	22	15	9	3045	1
	MAX	32847	2838	2819	847	26343	9
	MEDIAN	7182	26	223	30	6903	3
	AVG	8198	56	226	35	7882	2
	STDDEV	5439	256	310	58	4814	1
GET ...	SUM	18927746	79872	286914	86275	18474685	3500
	MIN	799	14	11	14	760	1
	MAX	28370	3384	2811	1142	21033	3
	MEDIAN	5701	27	24	18	5632	1
	AVG	7571	32	115	35	7390	1

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	STDDEV	4273	105	207	59	3903	0
▷ ROOT	SUM	2451353	15934	10325	8530	2416564	500
	MIN	808	23	11	14	760	1
	MAX	9337	1649	757	941	5990	1
	MEDIAN	5047	28	23	15	4981	1
	AVG	4903	32	21	17	4833	1
	STDDEV	1074	73	36	41	924	0
▷ ALTERNATIVES	SUM	2977938	16100	20593	10548	2930697	500
	MIN	905	14	17	17	857	1
	MAX	11978	1817	1845	1026	7290	1
	MEDIAN	5955	26	31	18	5880	1
	AVG	5956	32	41	21	5861	1
	STDDEV	919	118	142	49	611	0
▷ SUCCESSORS	SUM	3097407	11298	10025	9873	3066211	500
	MIN	2373	14	16	16	2327	1
	MAX	12554	649	56	916	10933	1
	MEDIAN	6238	17	17	18	6186	1
	AVG	6195	23	20	20	6132	1
	STDDEV	699	29	6	40	624	0
▷ PREDECESSOR	SUM	2611923	14003	15523	9942	2572455	500
	MIN	869	14	16	17	822	1
	MAX	12604	1992	1720	911	7981	1
	MEDIAN	5170	25	31	18	5096	1
	AVG	5224	28	31	20	5145	1
	STDDEV	497	95	77	40	284	0
▷ BASEVERSION	SUM	7789125	22537	230448	47382	7488758	1500
	MIN	6287	31	305	83	5868	3
	MAX	28370	3384	2811	1142	21033	3
	MEDIAN	15577	34	424	86	15033	3
	AVG	15578	45	461	95	14978	3
	STDDEV	1607	159	191	74	1183	0
GET CVC	SUM	8603051	54518	141292	53643	8353598	1500
	MIN	931	24	24	17	866	1
	MAX	88733	3023	46416	23751	15543	1
	MEDIAN	5584	28	26	18	5512	1
	AVG	5735	36	94	36	5569	1
	STDDEV	2926	118	1614	614	580	0
PIN	SUM	17222010	56517	56369	30923	17078201	3000
	MIN	1582	27	30	17	1508	2
	MAX	39394	2455	1579	1235	34125	2
	MEDIAN	9886	32	34	19	9801	2
	AVG	11481	38	38	21	11385	2
	STDDEV	6176	90	63	40	5983	0
SELECT	SUM	253329683	222383	570816	393097	252143387	5250
	MIN	852	23	34	14	781	1
	MAX	469228	7181	61551	88253	312243	1
	MEDIAN	28406	36	73	30	28267	1
	AVG	48253	42	109	75	48027	1
	STDDEV	59929	130	911	1227	57662	0
▷ außerhalb von Workspaces	SUM	230521779	163333	476537	352939	229528970	4000
	MIN	880	23	34	14	809	1
	MAX	469228	7181	61551	88253	312243	1
	MEDIAN	33894	33	74	49	33738	1
	AVG	57630	41	119	88	57382	1
	STDDEV	64979	145	1035	1405	62394	0
▷▷ 1 Navigationsschritt	SUM	3768332	19929	22991	8971	3716441	500
	MIN	880	23	34	14	809	1
	MAX	12796	1392	1367	1228	8809	1
	MEDIAN	7522	33	40	15	7434	1
	AVG	7537	40	46	18	7433	1
	STDDEV	891	86	60	54	690	0
▷▷ 2 Navigationsschritte	SUM	80941873	98433	221309	120118	80502013	2500
	MIN	1013	26	42	19	926	1
	MAX	123332	7181	19386	2726	94039	1
	MEDIAN	24557	28	73	49	24407	1
	AVG	32377	39	89	48	32201	1
	STDDEV	24245	170	415	121	23539	0
▷▷ 4 Navigationsschritte	SUM	145811574	44971	232237	223850	145310516	1000
	MIN	4792	33	109	80	4570	1
	MAX	464084	2037	61551	88253	312243	1
	MEDIAN	147774	35	123	97	147519	1
	AVG	145812	45	232	224	145311	1
	STDDEV	65034	91	1959	2798	60186	0
▷ innerhalb von Workspaces	SUM	22807904	59050	94279	40158	22614417	1250
	MIN	894	37	56	20	781	1
	MAX	73590	1095	7958	1601	62936	1
	MEDIAN	12124	42	61	30	11991	1
	AVG	18246	47	75	32	18092	1
	STDDEV	18540	57	228	65	18189	0
▷▷ 1 Navigationsschritt	SUM	3016249	11846	17897	5308	2981198	250
	MIN	986	37	58	20	871	1
	MAX	23896	804	1067	69	21956	1
	MEDIAN	12146	43	62	21	12020	1
	AVG	12065	47	72	21	11925	1
	STDDEV	1880	49	87	3	1741	0
▷▷ 2 Navigationsschritte	SUM	19791655	47204	76382	34850	19633219	1000

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	MIN	904	39	56	28	781	1
	MAX	73590	1095	7958	1601	62936	1
	MEDIAN	1220	42	61	30	1087	1
	AVG	19792	47	76	35	19633	1
	STDDEV	20406	59	251	73	20023	0

G.1.2 Normierte Zeiten

		Parser	Reduction	Rendering	SQL-Ausführung
CREATE NEW OBJECT	MIN	0,3%	2,0%	0,7%	97,0%
	MAX	19,5%	41,9%	2,3%	36,3%
	AVG	1,5%	5,1%	0,9%	92,5%
CREATE NEW RELATIONSHIP	MIN	0,9%	2,5%	0,9%	95,8%
	MAX	3,1%	10,0%	3,6%	83,3%
	AVG	0,0%	0,2%	0,1%	99,7%
▷ kein FRE	MIN	2,1%	2,4%	0,9%	94,6%
	MAX	16,4%	4,8%	1,3%	77,5%
	AVG	2,9%	2,6%	0,9%	93,7%
▷ ein FRE	MIN	0,4%	1,3%	0,6%	97,7%
	MAX	3,1%	10,0%	3,6%	83,3%
	AVG	0,0%	0,2%	0,1%	99,7%
CREATE NEW OBJECT + RELATIONSHIP	MIN	0,6%	2,2%	0,8%	96,5%
	MAX	26,9%	12,9%	22,7%	37,5%
	AVG	0,1%	0,8%	0,3%	98,8%
▷ 1 Objekt + 1 kein FRE	MIN	0,7%	3,2%	0,8%	95,2%
	MAX	30,2%	4,4%	3,9%	61,4%
	AVG	1,3%	3,4%	0,9%	94,4%
▷ 1 Objekt + 1 zwei FRE	MIN	0,6%	2,2%	0,8%	96,5%
	MAX	44,1%	2,1%	37,3%	16,6%
	AVG	0,8%	2,2%	1,0%	96,0%
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	MIN	0,2%	1,5%	0,5%	97,8%
	MAX	30,4%	2,1%	2,2%	65,3%
	AVG	0,1%	0,6%	0,2%	99,2%
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	MIN	0,1%	1,3%	0,5%	98,1%
	MAX	1,1%	27,5%	11,3%	60,1%
	AVG	0,1%	0,9%	0,3%	98,7%
CREATE SUCCESSOR	MIN	0,2%	8,7%	0,6%	90,4%
	MAX	1,0%	38,6%	1,1%	59,3%
	AVG	0,1%	4,4%	0,3%	95,3%
▷ außerhalb von Workspaces	MIN	0,2%	8,7%	0,6%	90,4%
	MAX	1,0%	38,6%	1,1%	59,3%
	AVG	0,1%	4,4%	0,3%	95,2%
▷ innerhalb von Workspaces	MIN	0,2%	4,9%	0,5%	94,3%
	MAX	1,2%	3,7%	0,6%	94,6%
	AVG	0,2%	3,8%	0,4%	95,7%
COPY OBJECTS	MIN	0,2%	1,4%	0,7%	97,7%
	MAX	0,8%	1,0%	0,9%	97,4%
	AVG	0,0%	0,3%	0,1%	99,5%
DELETE OBJECTS	MIN	0,1%	1,7%	0,4%	97,8%
	MAX	0,7%	8,4%	1,1%	89,8%
	AVG	0,0%	1,2%	0,2%	98,5%
MERGE OBJECTS	MIN	0,5%	24,7%	1,9%	72,9%
	MAX	6,0%	31,3%	3,7%	58,9%
	AVG	0,2%	8,8%	0,6%	90,3%
FREEZE	MIN	1,6%	6,7%	0,9%	90,9%
	MAX	3,7%	3,8%	1,6%	90,9%
	AVG	0,2%	1,8%	0,3%	97,7%
▷ außerhalb von Workspaces	MIN	1,6%	6,7%	0,9%	90,9%
	MAX	3,7%	3,8%	1,6%	90,9%
	AVG	0,2%	1,8%	0,3%	97,7%
▷ innerhalb von Workspaces	MIN	1,7%	8,4%	1,2%	88,7%
	MAX	0,1%	2,6%	0,3%	97,0%
	AVG	0,4%	2,5%	0,3%	96,8%
ATTACH und DETACH	MIN	0,6%	0,4%	0,3%	98,7%
	MAX	1,3%	2,9%	2,1%	93,7%
	AVG	0,1%	0,5%	0,1%	99,2%
▷ ATTACH	MIN	0,5%	0,4%	0,3%	98,7%
	MAX	1,2%	2,9%	2,1%	93,9%
	AVG	0,1%	0,5%	0,1%	99,3%
▷ DETACH	MIN	0,7%	0,5%	0,3%	98,5%
	MAX	8,6%	8,6%	2,6%	80,2%
	AVG	0,7%	2,8%	0,4%	96,1%

		Parser	Reduction	Rendering	SQL-Ausführung
GET ...	MIN	1,8%	1,4%	1,8%	95,1%
	MAX	11,9%	9,9%	4,0%	74,1%
	AVG	0,4%	1,5%	0,5%	97,6%
▷ ROOT	MIN	2,8%	1,4%	1,7%	94,1%
	MAX	17,7%	8,1%	10,1%	64,2%
	AVG	0,7%	0,4%	0,3%	98,6%
▷ ALTERNATIVES	MIN	1,5%	1,9%	1,9%	94,7%
	MAX	15,2%	15,4%	8,6%	60,9%
	AVG	0,5%	0,7%	0,4%	98,4%
▷ SUCCESSORS	MIN	0,6%	0,7%	0,7%	98,1%
	MAX	5,2%	0,4%	7,3%	87,1%
	AVG	0,4%	0,3%	0,3%	99,0%
▷ PREDECESSOR	MIN	1,6%	1,8%	2,0%	94,6%
	MAX	15,8%	13,6%	7,2%	63,3%
	AVG	0,5%	0,6%	0,4%	98,5%
▷ BASEVERSION	MIN	0,5%	4,9%	1,3%	93,3%
	MAX	11,9%	9,9%	4,0%	74,1%
	AVG	0,3%	3,0%	0,6%	96,1%
GET CVC	MIN	1,7%	1,9%	1,1%	95,3%
	MAX	6,2%	4,0%	3,1%	86,6%
	AVG	0,3%	0,3%	0,2%	99,2%
PIN	MIN	1,7%	1,9%	1,1%	95,3%
	MAX	6,2%	4,0%	3,1%	86,6%
	AVG	0,3%	0,3%	0,2%	99,2%
SELECT	MIN	2,7%	4,0%	1,6%	91,7%
	MAX	1,5%	13,1%	18,8%	66,5%
	AVG	0,1%	0,2%	0,2%	99,5%
▷ außerhalb von Workspaces	MIN	2,6%	3,9%	1,6%	91,9%
	MAX	1,5%	13,1%	18,8%	66,5%
	AVG	0,1%	0,2%	0,2%	99,6%
▷▷ 1 Navigationsschritt	MIN	2,6%	3,9%	1,6%	91,9%
	MAX	10,9%	10,7%	9,6%	68,8%
	AVG	0,5%	0,6%	0,2%	98,6%
▷▷ 2 Navigationsschritte	MIN	2,6%	4,1%	1,9%	91,4%
	MAX	5,8%	15,7%	2,2%	76,2%
	AVG	0,1%	0,3%	0,1%	99,5%
▷▷ 4 Navigationsschritte	MIN	0,7%	2,3%	1,7%	95,4%
	MAX	0,4%	13,3%	19,0%	67,3%
	AVG	0,0%	0,2%	0,2%	99,7%
▷ innerhalb von Workspaces	MIN	4,1%	6,3%	2,2%	87,4%
	MAX	1,5%	10,8%	2,2%	85,5%
	AVG	0,3%	0,4%	0,2%	99,2%
▷▷ 1 Navigationsschritt	MIN	3,8%	5,9%	2,0%	88,3%
	MAX	3,4%	4,5%	0,3%	91,9%
	AVG	0,4%	0,6%	0,2%	98,8%
▷▷ 2 Navigationsschritte	MIN	4,3%	6,2%	3,1%	86,4%
	MAX	1,5%	10,8%	2,2%	85,5%
	AVG	0,2%	0,4%	0,2%	99,2%

G.1.3 Absolute Zeiten zum Überprüfen der Kardinalitäten in μs

		Summe	Reduction	Rendering	SQL-Ausführung	SQL
CREATE NEW OBJECT	SUM	878244	6052	3298	868894	400
	MIN	0	0	0	0	0
	MAX	10864	722	73	10069	4
	AVG	5019	35	19	4965	2
	STDDEV	4398	59	18	4321	1
CREATE NEW RELATIONSHIP	SUM	1512132336	4752099	435819	1506944418	40400
	MIN	2185	218	17	1950	2
	MAX	204273	10586	10194	183493	4
	AVG	75230	236	22	74972	2
	STDDEV	28751	133	154	28464	0
▷ kein FRE	SUM	960699	40141	3880	916678	400
	MIN	2346	369	27	1950	4
	MAX	13269	537	839	11893	4
	AVG	9607	401	39	9167	4
	STDDEV	1715	30	81	1604	0
▷ ein FRE	SUM	1511171637	4711958	431939	1506027740	40000
	MIN	5106	218	17	4871	2
	MAX	204273	10586	10194	183493	2
	AVG	75559	236	22	75301	2
	STDDEV	28438	133	154	28151	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	4849625613	19116351	1958505	4828550757	160000
	MIN	8325	338	24	7963	3

		Summe	Reduction	Rendering	SQL-Ausführung	SQL
	MAX	1065973	360718	161678	543577	10
	AVG	186524	735	75	185713	6
	STDDEV	109978	2647	1318	106013	2
▷ 1 Objekt + 1 kein FRE	SUM	52988695	847980	81559	52059156	10000
	MIN	22290	777	72	21441	10
	MAX	87903	4538	1860	81505	10
	AVG	52989	848	82	52059	10
	STDDEV	15736	183	83	15470	0
▷ 1 Objekt + 1 zwei FRE	SUM	1443847970	4066670	299646	1439481654	30000
	MIN	8325	338	24	7963	3
	MAX	723877	360718	27345	335814	3
	AVG	144385	407	30	143948	3
	STDDEV	74644	3612	284	70749	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	2489016539	8224296	970107	2479822136	70000
	MIN	24711	766	58	23887	7
	MAX	731464	26209	161678	543577	7
	AVG	248902	822	97	247982	7
	STDDEV	120587	347	1949	118291	0
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	863772409	5977405	607193	857187811	50000
	MIN	34272	1084	84	33104	10
	MAX	637252	219872	72641	344739	10
	AVG	172754	1195	121	171438	10
	STDDEV	77563	3107	1124	73331	0
DELETE OBJECTS	SUM	93185577	343119	36641	92805817	2761
	MIN	0	0	0	0	0
	MAX	170505	531	9055	160919	5
	AVG	18637	69	7	18561	0
	STDDEV	26965	76	129	26760	0

G.1.4 Normierte Zeiten zum Überprüfen der Kardinalitäten

		Reduction	Rendering	SQL-Ausführung
CREATE NEW OBJECT	MIN	0,0%	0,0%	0,0%
	MAX	6,6%	0,7%	92,7%
	AVG	0,7%	0,4%	99,0%
CREATE NEW RELATIONSHIP	MIN	10,0%	0,8%	89,2%
	MAX	5,2%	5,0%	89,8%
	AVG	0,3%	0,0%	99,7%
▷ kein FRE	MIN	15,7%	1,2%	83,1%
	MAX	4,0%	6,3%	89,6%
	AVG	4,2%	0,4%	95,4%
▷ ein FRE	MIN	4,3%	0,3%	95,4%
	MAX	5,2%	5,0%	89,8%
	AVG	0,3%	0,0%	99,7%
CREATE NEW OBJECT + RELATIONSHIP	MIN	4,1%	0,3%	95,7%
	MAX	33,8%	15,2%	51,0%
	AVG	0,4%	0,0%	99,6%
▷ 1 Objekt + 1 kein FRE	MIN	3,5%	0,3%	96,2%
	MAX	5,2%	2,1%	92,7%
	AVG	1,6%	0,2%	98,2%
▷ 1 Objekt + 1 zwei FRE	MIN	4,1%	0,3%	95,7%
	MAX	49,8%	3,8%	46,4%
	AVG	0,3%	0,0%	99,7%
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	MIN	3,1%	0,2%	96,7%
	MAX	3,6%	22,1%	74,3%
	AVG	0,3%	0,0%	99,6%
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	MIN	3,2%	0,2%	96,6%
	MAX	34,5%	11,4%	54,1%
	AVG	0,7%	0,1%	99,2%
DELETE OBJECTS	MIN	0,0%	0,0%	0,0%
	MAX	0,3%	5,3%	94,4%
	AVG	0,4%	0,0%	99,6%

G.2 Variante 2

G.2.1 Absolute Zeiten in μs

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
alle Befehle	SUM	3310712665	5361068	57796988	11193698	3236360911	700562
CREATE NEW OBJECT	SUM	1465402	20671	74474	13932	1356325	700
	MIN	7498	22	162	52	7262	4
	MAX	48385	10591	21718	1060	15016	4
	MEDIAN	7991	44	247	62	7638	4
	AVG	8374	118	426	80	7750	4
	STDDEV	3256	800	1654	108	694	0
CREATE NEW RELATIONSHIP	SUM	358893895	893453	3621208	868323	353510911	60100
	MIN	3760	36	100	11	3613	1
	MAX	117077	9656	10288	10022	87111	3
	MEDIAN	16968	38	136	39	16755	3
	AVG	17855	44	180	43	17588	2
	STDDEV	5199	170	198	162	4669	0
▷ kein FRE	SUM	400788	11824	10916	1230	376818	100
	MIN	3806	82	100	11	3613	1
	MAX	5671	1289	153	17	4212	1
	MEDIAN	3977	100	104	12	3761	1
	AVG	4008	118	109	12	3768	1
	STDDEV	279	134	12	1	132	0
▷ ein FRE	SUM	358493107	881629	3610292	867093	353134093	60000
	MIN	9028	36	120	36	8836	3
	MAX	117077	9656	10288	10022	87111	3
	MEDIAN	16993	38	136	39	16780	3
	AVG	17925	44	181	43	17657	3
	STDDEV	5108	170	198	163	4577	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	1053778025	2626493	15942413	4692012	1030517107	270000
	MIN	11236	68	253	66	10849	5
	MAX	894181	317473	205207	169005	202496	19
	MEDIAN	47877	73	715	164	46925	12
	AVG	40530	101	613	180	39635	10
	STDDEV	31769	2009	1449	1789	26521	5
▷ 1 Objekt + 1 kein FRE	SUM	12015725	143507	430531	83516	11358171	5000
	MIN	11372	84	373	66	10849	5
	MAX	57189	6861	2243	1393	46692	5
	MEDIAN	11658	103	409	76	11070	5
	AVG	12016	144	431	84	11358	5
	STDDEV	2063	274	132	73	1584	0
▷ 1 Objekt + 1 zwei FRE	SUM	120636671	1149171	2987113	819094	115681293	50000
	MIN	11241	68	253	68	10852	5
	MAX	584989	317473	205207	8692	53617	5
	MEDIAN	11587	71	268	75	11173	5
	AVG	12064	115	299	82	11568	5
	STDDEV	7398	3183	2056	184	1975	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	515808053	891455	7449735	2022423	505444440	120000
	MIN	37492	71	607	158	36656	12
	MAX	412009	57292	30833	169005	154879	12
	MEDIAN	51100	74	725	166	50135	12
	AVG	51581	89	745	202	50544	12
	STDDEV	7731	592	400	1748	4990	0
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	405317576	442360	5075034	1766979	398033203	95000
	MIN	69049	71	906	245	67827	19
	MAX	466818	4724	92335	167263	202496	19
	MEDIAN	80485	74	953	257	79201	19
	AVG	81064	88	1015	353	79607	19
	STDDEV	9989	119	1321	3228	5322	0
CREATE SUCCESSOR	SUM	459024704	398382	23611592	1547140	433467590	117630
	MIN	7388	14	665	45	6664	5
	MAX	349124	6347	40918	8636	293223	78
	MEDIAN	17847	16	923	58	16850	5
	AVG	22391	19	1152	75	21145	5
	STDDEV	17733	57	1095	200	16380	3
▷ außerhalb von Workspaces	SUM	445541124	375831	23086308	1496828	420582157	113895
	MIN	7388	14	665	45	6664	5
	MAX	349124	6347	40918	8636	293223	78
	MEDIAN	17838	16	925	58	16839	5
	AVG	22277	19	1154	75	21029	5
	STDDEV	17671	54	1099	202	16316	3
▷ innerhalb von Workspaces	SUM	13483580	22551	525284	50312	12885433	3735
	MIN	14007	34	712	65	13196	6
	MAX	229283	2658	10673	1438	214514	64
	MEDIAN	21515	38	768	74	20635	6
	AVG	26967	45	1051	101	25771	7
	STDDEV	19343	122	926	105	18190	5
COPY OBJECTS	SUM	800188679	405973	7355159	2471230	789956317	155595
	MIN	7867	16	106	51	7694	4

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	MAX	1161074	2392	11649	10160	1136873	190
	MEDIAN	8515	18	118	59	8320	4
	AVG	40009	20	368	124	39498	7
	STDDEV	52041	21	489	254	51277	6
DELETE OBJECTS	SUM	352150810	125656	3499974	588581	347936599	52004
	MIN	12163	12	195	39	11917	4
	MAX	886922	42806	7385	6753	829978	104
	MEDIAN	44998	14	427	63	44494	7
	AVG	70430	25	700	118	69587	10
	STDDEV	75386	612	733	257	73784	10
MERGE OBJECTS	SUM	9522460	19515	843820	65351	8593774	2205
	MIN	5201	27	1298	97	3779	4
	MAX	48306	3205	14143	3043	27915	5
	MEDIAN	18904	32	1542	105	17225	4
	AVG	19045	39	1688	131	17188	4
	STDDEV	2352	142	755	184	1270	0
FREEZE	SUM	13397116	43366	303018	45191	13005541	4211
	MIN	1589	26	108	14	1441	2
	MAX	70986	3204	3482	1055	63245	20
	MEDIAN	6734	30	149	19	6536	2
	AVG	12179	39	275	41	11823	3
	STDDEV	9449	128	268	66	8987	2
▷ außerhalb von Workspaces	SUM	12451677	38210	278170	41414	12093883	3947
	MIN	1589	26	108	14	1441	2
	MAX	65023	3204	3482	1055	57282	20
	MEDIAN	6543	30	134	17	6362	2
	AVG	12452	38	278	41	12094	3
	STDDEV	9598	129	268	66	9135	3
▷ innerhalb von Workspaces	SUM	945439	5156	24848	3777	911658	264
	MIN	2158	33	148	21	1956	2
	MAX	66366	852	1704	565	63245	19
	MEDIAN	7447	35	155	23	7234	2
	AVG	9454	52	248	38	9117	2
	STDDEV	7219	113	260	59	6786	2
ATTACH und DETACH	SUM	63684126	155770	812699	175362	62540295	14867
	MIN	3169	17	12	9	3131	1
	MAX	79661	2778	5667	3351	67865	14
	MEDIAN	7723	27	166	26	7504	3
	AVG	11959	29	153	33	11745	2
	STDDEV	10691	67	196	73	10355	1
▷ ATTACH	SUM	61838053	142790	764457	168876	60761930	14234
	MIN	3582	17	12	9	3544	1
	MAX	79475	2592	5667	3351	67865	14
	MEDIAN	7729	27	166	26	7510	3
	AVG	12125	28	150	33	11914	2
	STDDEV	10802	43	186	75	10498	1
▷ DETACH	SUM	1846073	12980	48242	6486	1778365	633
	MIN	3177	23	14	9	3131	1
	MAX	34665	2778	4485	117	27285	9
	MEDIAN	7181	27	221	27	6906	3
	AVG	8205	58	214	29	7904	2
	STDDEV	5545	254	340	20	4930	1
GET ...	SUM	19075498	78649	311172	84019	18601658	3500
	MIN	799	14	10	14	761	1
	MAX	59887	2863	31059	1681	24284	3
	MEDIAN	5734	27	24	17	5666	1
	AVG	7630	31	124	34	7441	1
	STDDEV	4713	97	651	61	3905	0
▷ ROOT	SUM	2441474	15946	9532	7620	2408376	500
	MIN	808	23	10	14	761	1
	MAX	7694	1680	245	70	5699	1
	MEDIAN	5088	28	23	15	5022	1
	AVG	4883	32	19	15	4817	1
	STDDEV	1079	74	13	3	989	0
▷ ALTERNATIVES	SUM	3011841	13516	13541	10575	2974209	500
	MIN	874	14	16	16	828	1
	MAX	11329	1796	319	1681	7533	1
	MEDIAN	6082	26	30	18	6008	1
	AVG	6024	27	27	21	5948	1
	STDDEV	826	80	17	74	656	0
▷ SUCCESSORS	SUM	3143751	11341	9744	8682	3113984	500
	MIN	2465	14	16	16	2419	1
	MAX	10092	655	47	24	9366	1
	MEDIAN	6354	17	17	17	6303	1
	AVG	6288	23	19	17	6228	1
	STDDEV	665	29	5	1	629	0
▷ PREDECESSOR	SUM	2633099	16091	14797	8873	2593338	500
	MIN	900	14	16	16	854	1
	MAX	8912	1930	1206	46	5730	1
	MEDIAN	5241	25	31	17	5168	1
	AVG	5266	32	30	18	5187	1
	STDDEV	402	123	55	2	222	0
▷ BASEVERSION	SUM	7845333	21755	263558	48269	7511751	1500
	MIN	3749	31	307	83	3328	3
	MAX	59378	2863	31059	1172	24284	3

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	MEDIAN	15609	35	423	86	15065	3
	AVG	15691	44	527	97	15024	3
	STDDEV	2716	136	1382	89	1109	0
GET CVC	SUM	8847861	49238	57843	29244	8711536	1500
	MIN	931	26	26	15	864	1
	MAX	16049	3159	1168	1201	10521	1
	MEDIAN	5560	29	28	16	5487	1
	AVG	5899	33	39	19	5808	1
	STDDEV	895	83	77	55	681	0
PIN	SUM	6754550	56381	58376	55911	6583882	3000
	MIN	1176	28	31	15	1102	2
	MAX	49805	2585	1476	22653	23091	2
	MEDIAN	4217	33	35	16	4133	2
	AVG	4503	38	39	37	4389	2
	STDDEV	1858	92	64	589	1113	0
SELECT	SUM	125579375	232740	589226	413872	124343537	5250
	MIN	1117	23	33	13	1048	1
	MAX	263362	9706	72471	105299	75886	1
	MEDIAN	17305	36	67	27	17175	1
	AVG	23920	44	112	79	23684	1
	STDDEV	17786	206	1246	1508	14826	0
▷ außerhalb von Workspaces	SUM	98483070	166309	488110	380646	97448005	4000
	MIN	1186	23	33	13	1117	1
	MAX	263362	9706	72471	105299	75886	1
	MEDIAN	17974	34	67	45	17828	1
	AVG	24621	42	122	95	24362	1
	STDDEV	18489	192	1415	1727	15156	0
▷▷ 1 Navigationsschritt	SUM	7900116	21429	74713	9314	7794660	500
	MIN	1630	23	33	13	1561	1
	MAX	76570	2753	48368	1145	24304	1
	MEDIAN	15825	34	38	14	15739	1
	AVG	15800	43	149	19	15589	1
	STDDEV	6587	151	2163	63	4211	0
▷▷ 2 Navigationsschritte	SUM	42107182	103449	271227	114659	41617847	2500
	MIN	1199	26	41	15	1117	1
	MAX	115314	9706	72471	2170	30967	1
	MEDIAN	16401	28	66	44	16263	1
	AVG	16843	41	108	46	16647	1
	STDDEV	5088	231	1498	125	3233	0
▷▷ 4 Navigationsschritte	SUM	48475772	41431	142170	256673	48035498	1000
	MIN	2474	33	94	69	2278	1
	MAX	185097	1238	2674	105299	75886	1
	MEDIAN	47999	35	108	88	47768	1
	AVG	48476	41	142	257	48035	1
	STDDEV	15385	46	229	3443	11668	0
▷ innerhalb von Workspaces	SUM	27096305	66431	101116	33226	26895532	1250
	MIN	1159	36	57	18	1048	1
	MAX	79635	8509	12084	1579	57463	1
	MEDIAN	16105	43	63	26	15973	1
	AVG	21677	53	81	27	21516	1
	STDDEV	14132	245	346	50	13491	0
▷▷ 1 Navigationsschritt	SUM	3906883	12526	18082	4761	3871514	250
	MIN	1264	36	60	18	1150	1
	MAX	22065	1466	1851	47	18701	1
	MEDIAN	15930	43	64	19	15804	1
	AVG	15628	50	72	19	15486	1
	STDDEV	2256	90	113	2	2051	0
▷▷ 2 Navigationsschritte	SUM	23189422	53905	83034	28465	23024018	1000
	MIN	1168	39	57	24	1048	1
	MAX	79635	8509	12084	1579	57463	1
	MEDIAN	16324	43	62	26	16193	1
	AVG	23189	54	83	28	23024	1
	STDDEV	15374	270	383	55	14666	0

G.2.2 Normierte Zeiten

		Parser	Reduction	Rendering	SQL-Ausführung
CREATE NEW OBJECT	MIN	0,3%	2,2%	0,7%	96,9%
	MAX	21,9%	44,9%	2,2%	31,0%
	AVG	1,4%	5,1%	1,0%	92,6%
CREATE NEW RELATIONSHIP	MIN	1,0%	2,7%	0,3%	96,1%
	MAX	8,2%	8,8%	8,6%	74,4%
	AVG	0,2%	1,0%	0,2%	98,5%
▷ kein FRE	MIN	2,2%	2,6%	0,3%	94,9%
	MAX	22,7%	2,7%	0,3%	74,3%

		Parser	Reduction	Rendering	SQL-Ausführung
	AVG	3,0%	2,7%	0,3%	94,0%
▷ ein FRE	MIN	0,4%	1,3%	0,4%	97,9%
	MAX	8,2%	8,8%	8,6%	74,4%
	AVG	0,2%	1,0%	0,2%	98,5%
CREATE NEW OBJECT + RELATIONSHIP	MIN	0,6%	2,3%	0,6%	96,6%
	MAX	35,5%	22,9%	18,9%	22,6%
	AVG	0,2%	1,5%	0,4%	97,8%
▷ 1 Objekt + 1 kein FRE	MIN	0,7%	3,3%	0,6%	95,4%
	MAX	12,0%	3,9%	2,4%	81,6%
	AVG	1,2%	3,6%	0,7%	94,5%
▷ 1 Objekt + 1 zwei FRE	MIN	0,6%	2,3%	0,6%	96,5%
	MAX	54,3%	35,1%	1,5%	9,2%
	AVG	1,0%	2,5%	0,7%	95,9%
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	MIN	0,2%	1,6%	0,4%	97,8%
	MAX	13,9%	7,5%	41,0%	37,6%
	AVG	0,2%	1,4%	0,4%	98,0%
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	MIN	0,1%	1,3%	0,4%	98,2%
	MAX	1,0%	19,8%	35,8%	43,4%
	AVG	0,1%	1,3%	0,4%	98,2%
CREATE SUCCESSOR	MIN	0,2%	9,0%	0,6%	90,2%
	MAX	1,8%	11,7%	2,5%	84,0%
	AVG	0,1%	5,1%	0,3%	94,4%
▷ außerhalb von Workspaces	MIN	0,2%	9,0%	0,6%	90,2%
	MAX	1,8%	11,7%	2,5%	84,0%
	AVG	0,1%	5,2%	0,3%	94,4%
▷ innerhalb von Workspaces	MIN	0,2%	5,1%	0,5%	94,2%
	MAX	1,2%	4,7%	0,6%	93,6%
	AVG	0,2%	3,9%	0,4%	95,6%
COPY OBJECTS	MIN	0,2%	1,3%	0,6%	97,8%
	MAX	0,2%	1,0%	0,9%	97,9%
	AVG	0,1%	0,9%	0,3%	98,7%
DELETE OBJECTS	MIN	0,1%	1,6%	0,3%	98,0%
	MAX	4,8%	0,8%	0,8%	93,6%
	AVG	0,0%	1,0%	0,2%	98,8%
MERGE OBJECTS	MIN	0,5%	25,0%	1,9%	72,7%
	MAX	6,6%	29,3%	6,3%	57,8%
	AVG	0,2%	8,9%	0,7%	90,2%
FREEZE	MIN	1,6%	6,8%	0,9%	90,7%
	MAX	4,5%	4,9%	1,5%	89,1%
	AVG	0,3%	2,3%	0,3%	97,1%
▷ außerhalb von Workspaces	MIN	1,6%	6,8%	0,9%	90,7%
	MAX	4,9%	5,4%	1,6%	88,1%
	AVG	0,3%	2,2%	0,3%	97,1%
▷ innerhalb von Workspaces	MIN	1,5%	6,9%	1,0%	90,6%
	MAX	1,3%	2,6%	0,9%	95,3%
	AVG	0,5%	2,6%	0,4%	96,4%
ATTACH und DETACH	MIN	0,5%	0,4%	0,3%	98,8%
	MAX	3,5%	7,1%	4,2%	85,2%
	AVG	0,2%	1,3%	0,3%	98,2%
▷ ATTACH	MIN	0,5%	0,3%	0,3%	98,9%
	MAX	3,3%	7,1%	4,2%	85,4%
	AVG	0,2%	1,2%	0,3%	98,3%
▷ DETACH	MIN	0,7%	0,4%	0,3%	98,6%
	MAX	8,0%	12,9%	0,3%	78,7%
	AVG	0,7%	2,6%	0,4%	96,3%
GET ...	MIN	1,8%	1,3%	1,8%	95,2%
	MAX	4,8%	51,9%	2,8%	40,5%
	AVG	0,4%	1,6%	0,4%	97,5%
▷ ROOT	MIN	2,8%	1,2%	1,7%	94,2%
	MAX	21,8%	3,2%	0,9%	74,1%
	AVG	0,7%	0,4%	0,3%	98,6%
▷ ALTERNATIVES	MIN	1,6%	1,8%	1,8%	94,7%
	MAX	15,9%	2,8%	14,8%	66,5%
	AVG	0,4%	0,4%	0,4%	98,8%
▷ SUCCESSORS	MIN	0,6%	0,6%	0,6%	98,1%
	MAX	6,5%	0,5%	0,2%	92,8%
	AVG	0,4%	0,3%	0,3%	99,1%
▷ PREDECESSOR	MIN	1,6%	1,8%	1,8%	94,9%
	MAX	21,7%	13,5%	0,5%	64,3%
	AVG	0,6%	0,6%	0,3%	98,5%
▷ BASEVERSION	MIN	0,8%	8,2%	2,2%	88,8%
	MAX	4,8%	52,3%	2,0%	40,9%
	AVG	0,3%	3,4%	0,6%	95,7%
GET CVC	MIN	2,4%	2,6%	1,3%	93,7%
	MAX	5,2%	3,0%	45,5%	46,4%
	AVG	0,8%	0,9%	0,8%	97,5%
PIN	MIN	2,4%	2,6%	1,3%	93,7%
	MAX	5,2%	3,0%	45,5%	46,4%
	AVG	0,8%	0,9%	0,8%	97,5%
SELECT	MIN	2,1%	3,0%	1,2%	93,8%
	MAX	3,7%	27,5%	40,0%	28,8%
	AVG	0,2%	0,5%	0,3%	99,0%
▷ außerhalb von Workspaces	MIN	1,9%	2,8%	1,1%	94,2%

		Parser	Reduction	Rendering	SQL-Ausführung
	MAX	3,7%	27,5%	40,0%	28,8%
	AVG	0,2%	0,5%	0,4%	98,9%
▷▷ 1 Navigationsschritt	MIN	1,4%	2,0%	0,8%	95,8%
	MAX	3,6%	63,2%	1,5%	31,7%
	AVG	0,3%	0,9%	0,1%	98,7%
▷▷ 2 Navigationsschritte	MIN	2,2%	3,4%	1,3%	93,2%
	MAX	8,4%	62,8%	1,9%	26,9%
	AVG	0,2%	0,6%	0,3%	98,8%
▷▷ 4 Navigationsschritte	MIN	1,3%	3,8%	2,8%	92,1%
	MAX	0,7%	1,4%	56,9%	41,0%
	AVG	0,1%	0,3%	0,5%	99,1%
▷ innerhalb von Workspaces	MIN	3,1%	4,9%	1,6%	90,4%
	MAX	10,7%	15,2%	2,0%	72,2%
	AVG	0,2%	0,4%	0,1%	99,3%
▷▷ 1 Navigationsschritt	MIN	2,8%	4,7%	1,4%	91,0%
	MAX	6,6%	8,4%	0,2%	84,8%
	AVG	0,3%	0,5%	0,1%	99,1%
▷▷ 2 Navigationsschritte	MIN	3,3%	4,9%	2,1%	89,7%
	MAX	10,7%	15,2%	2,0%	72,2%
	AVG	0,2%	0,4%	0,1%	99,3%

G.2.3 Absolute Zeiten zum Überprüfen der Kardinalitäten in μs

		Summe	Reduction	Rendering	SQL-Ausführung	SQL
CREATE NEW OBJECT	SUM	793046	6587	3628	782831	400
	MIN	0	0	0	0	0
	MAX	11621	1159	760	9702	4
	AVG	4532	38	21	4473	2
	STDDEV	4026	90	58	3878	1
CREATE NEW RELATIONSHIP	SUM	195568715	4722652	368307	190477756	40400
	MIN	2160	217	14	1929	2
	MAX	47298	10057	9816	27425	4
	AVG	9730	235	18	9477	2
	STDDEV	707	130	120	457	0
▷ kein FRE	SUM	807979	39425	2610	765944	400
	MIN	2326	375	22	1929	4
	MAX	9348	442	52	8854	4
	AVG	8080	394	26	7659	4
	STDDEV	1040	19	5	1016	0
▷ ein FRE	SUM	194760736	4683227	365697	189711812	40000
	MIN	4858	217	14	4627	2
	MAX	47298	10057	9816	27425	2
	AVG	9738	234	18	9486	2
	STDDEV	684	130	120	434	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	1020114493	18629161	1782797	999702535	160000
	MIN	7589	342	21	7226	3
	MAX	479758	59557	342995	77206	10
	AVG	39235	717	69	38450	6
	STDDEV	14512	548	2135	11829	2
▷ 1 Objekt + 1 kein FRE	SUM	24832545	835022	71205	23926318	10000
	MIN	21477	746	60	20671	10
	MAX	41139	10542	1277	29320	10
	AVG	24833	835	71	23926	10
	STDDEV	936	361	81	494	0
▷ 1 Objekt + 1 zwei FRE	SUM	304771097	3657337	275966	300837794	30000
	MIN	7589	342	21	7226	3
	MAX	79834	11062	8398	60374	3
	AVG	30477	366	28	30084	3
	STDDEV	10052	184	150	9717	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	454357846	8322431	977350	445058065	70000
	MIN	23664	777	50	22837	7
	MAX	422347	9586	342995	69766	7
	AVG	45436	832	98	44506	7
	STDDEV	13607	352	3435	9820	0
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	236153005	5814371	458276	229880358	50000
	MIN	32713	1092	73	31548	10
	MAX	142103	59557	5340	77206	10
	AVG	47231	1163	92	45976	10
	STDDEV	6868	864	199	5806	0
DELETE OBJECTS	SUM	15971996	372953	35778	15563265	3002
	MIN	0	0	0	0	0
	MAX	49432	5017	5622	38793	7
	AVG	3194	75	7	3113	0
	STDDEV	4079	128	95	3856	0

G.2.4 Normierte Zeiten zum Überprüfen der Kardinalitäten

		Reduction	Rendering	SQL-Ausführung
CREATE NEW OBJECT	MIN	0,0%	0,0%	0,0%
	MAX	10,0%	6,5%	83,5%
	AVG	0,8%	0,4%	98,7%
CREATE NEW RELATIONSHIP	MIN	10,0%	0,6%	89,3%
	MAX	21,3%	20,8%	58,0%
	AVG	2,4%	0,2%	97,4%
▷ kein FRE	MIN	16,1%	0,9%	82,9%
	MAX	4,7%	0,6%	94,7%
	AVG	4,9%	0,3%	94,8%
▷ ein FRE	MIN	4,5%	0,3%	95,2%
	MAX	21,3%	20,8%	58,0%
	AVG	2,4%	0,2%	97,4%
CREATE NEW OBJECT + RELATIONSHIP	MIN	4,5%	0,3%	95,2%
	MAX	12,4%	71,5%	16,1%
	AVG	1,8%	0,2%	98,0%
▷ 1 Objekt + 1 kein FRE	MIN	3,5%	0,3%	96,2%
	MAX	25,6%	3,1%	71,3%
	AVG	3,4%	0,3%	96,4%
▷ 1 Objekt + 1 zwei FRE	MIN	4,5%	0,3%	95,2%
	MAX	13,9%	10,5%	75,6%
	AVG	1,2%	0,1%	98,7%
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	MIN	3,3%	0,2%	96,5%
	MAX	2,3%	81,2%	16,5%
	AVG	1,8%	0,2%	98,0%
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	MIN	3,3%	0,2%	96,4%
	MAX	41,9%	3,8%	54,3%
	AVG	2,5%	0,2%	97,3%
DELETE OBJECTS	MIN	0,0%	0,0%	0,0%
	MAX	10,1%	11,4%	78,5%
	AVG	2,3%	0,2%	97,5%

G.3 Variante 3

G.3.1 Absolute Zeiten in μs

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
alle Befehle	SUM	3326277364	5264942	59874253	11333668	3249804501	696254
CREATE NEW OBJECT	SUM	1540524	25784	73113	12532	1429095	700
	MIN	7492	22	164	54	7252	4
	MAX	78618	16335	18786	623	42874	4
	MEDIAN	8347	44	246	62	7995	4
	AVG	8803	147	418	72	8166	4
	STDDEV	5494	1229	1447	59	2759	0
CREATE NEW RELATIONSHIP	SUM	361219058	862663	3614869	890313	355851213	60100
	MIN	3998	36	107	9	3846	1
	MAX	139391	9674	10171	42203	77343	3
	MEDIAN	17109	38	137	39	16895	3
	AVG	17971	43	180	44	17704	2
	STDDEV	5376	119	189	335	4733	0
▷ kein FRE	SUM	426301	10740	12790	1084	401687	100
	MIN	4046	84	107	9	3846	1
	MAX	7939	665	171	47	7056	1
	MEDIAN	4215	101	123	10	3981	1
	AVG	4263	107	128	11	4017	1
	STDDEV	411	68	11	4	328	0
▷ ein FRE	SUM	360792757	851923	3602079	889229	355449526	60000
	MIN	8913	36	121	37	8719	3
	MAX	139391	9674	10171	42203	77343	3
	MEDIAN	17139	38	137	39	16925	3
	AVG	18040	43	180	44	17772	3
	STDDEV	5288	119	190	336	4644	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	1031804106	2260332	15969643	4260085	1009314046	265000
	MIN	11301	68	253	63	10917	5
	MAX	776485	15033	344126	166663	250663	18

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	MEDIAN	48013	73	712	163	47065	12
	AVG	39685	87	614	164	38820	10
	STDDEV	28681	200	2192	1192	25096	4
▷ 1 Objekt + 1 kein FRE	SUM	12678271	137228	456579	80124	12004340	5000
	MIN	12101	85	387	63	11566	5
	MAX	42901	7476	2315	1739	31371	5
	MEDIAN	12417	103	431	74	11809	5
	AVG	12678	137	457	80	12004	5
	STDDEV	1489	261	152	76	1000	0
▷ 1 Objekt + 1 zwei FRE	SUM	120799952	820828	2747755	963234	116268135	50000
	MIN	11307	68	253	69	10917	5
	MAX	249036	11521	6699	166663	64153	5
	MEDIAN	11646	71	268	74	11233	5
	AVG	12080	82	275	96	11627	5
	STDDEV	3954	189	88	1671	2006	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	518841547	856620	7728405	1765630	508490892	120000
	MIN	37907	70	599	156	37082	12
	MAX	471323	15033	344126	9258	102906	12
	MEDIAN	51465	74	723	165	50503	12
	AVG	51884	86	773	177	50849	12
	STDDEV	9008	234	3450	197	5127	0
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	379484336	445656	5036904	1451097	372550679	90000
	MIN	60472	71	908	232	59261	18
	MAX	403612	3872	58505	90572	250663	18
	MEDIAN	75415	74	956	244	74141	18
	AVG	75897	89	1007	290	74510	18
	STDDEV	7592	114	851	1302	5325	0
CREATE SUCCESSOR	SUM	469047393	440800	24333245	1568394	442704954	117630
	MIN	7915	15	663	44	7193	5
	MAX	531132	8766	202974	29745	289647	78
	MEDIAN	18361	16	946	58	17341	5
	AVG	22880	22	1187	77	21595	5
	STDDEV	18652	119	1822	277	16435	3
▷ außerhalb von Workspaces	SUM	455440019	418219	23809988	1512342	429699470	113895
	MIN	7915	15	663	44	7193	5
	MAX	531132	8766	202974	29745	289647	78
	MEDIAN	18351	16	949	58	17328	5
	AVG	22772	21	1190	76	21485	5
	STDDEV	18605	119	1839	280	16368	3
▷ innerhalb von Workspaces	SUM	13607374	22581	523257	56052	13005484	3735
	MIN	14410	34	723	67	13586	6
	MAX	238670	2653	8776	1790	225451	64
	MEDIAN	21907	38	772	74	21023	6
	AVG	27215	45	1047	112	26011	7
	STDDEV	19480	121	838	152	18369	5
COPY OBJECTS	SUM	809032188	445795	7358059	2837539	798390795	155595
	MIN	7882	17	106	51	7708	4
	MAX	1466146	10011	10861	312228	1133046	190
	MEDIAN	8908	19	120	59	8710	4
	AVG	40452	22	368	142	39920	7
	STDDEV	54276	121	474	2232	51450	6
DELETE OBJECTS	SUM	368249791	86328	4831483	606507	362725473	52696
	MIN	11800	13	194	39	11554	4
	MAX	888052	5544	69815	7228	805465	104
	MEDIAN	50040	14	427	65	49534	7
	AVG	73650	17	966	121	72545	10
	STDDEV	74634	94	1599	264	72676	10
MERGE OBJECTS	SUM	9537289	20451	887704	56767	8572367	2205
	MIN	5241	28	1338	97	3778	4
	MAX	64809	2808	30261	1232	30508	5
	MEDIAN	18934	32	1566	104	17232	4
	AVG	19075	41	1775	114	17145	4
	STDDEV	3265	133	1756	56	1320	0
FREEZE	SUM	13519900	38041	303872	50519	13127468	4211
	MIN	1608	26	109	14	1459	2
	MAX	72098	3197	3480	1773	63648	20
	MEDIAN	6712	30	148	18	6516	2
	AVG	12291	35	276	46	11934	3
	STDDEV	9614	101	265	118	9130	2
▷ außerhalb von Workspaces	SUM	12571120	33932	280161	47253	12209774	3947
	MIN	1608	26	109	14	1459	2
	MAX	66010	3197	3480	1773	57560	20
	MEDIAN	6587	29	136	17	6405	2
	AVG	12571	34	280	47	12210	3
	STDDEV	9764	105	265	123	9272	3
▷ innerhalb von Workspaces	SUM	948780	4109	23711	3266	917694	264
	MIN	1785	33	148	22	1582	2
	MAX	66479	604	1996	231	63648	19
	MEDIAN	7398	35	157	24	7182	2
	AVG	9488	41	237	33	9177	2
	STDDEV	7347	57	269	27	6995	2
ATTACH und DETACH	SUM	68099449	157008	805626	183076	66953739	14867
	MIN	3342	17	12	9	3304	1
	MAX	89266	2793	4989	4348	77136	14
	MEDIAN	9531	27	166	27	9311	3

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	AVG	12789	29	151	34	12573	2
	STDDEV	10921	66	178	101	10577	1
▷ ATTACH	SUM	66175817	144433	753445	176269	65101670	14234
	MIN	3592	17	12	9	3554	1
	MAX	88466	2272	4710	4348	77136	14
	MEDIAN	9619	27	166	27	9399	3
	AVG	12976	28	148	35	12765	2
	STDDEV	11013	40	159	103	10710	1
▷ DETACH	SUM	1923632	12575	52181	6807	1852069	633
	MIN	3350	23	14	9	3304	1
	MAX	39618	2793	4989	120	31716	9
	MEDIAN	7335	27	226	29	7053	3
	AVG	8549	56	232	30	8231	2
	STDDEV	5971	254	410	21	5286	1
GET ...	SUM	19134831	106963	284796	87145	18655927	3500
	MIN	806	13	10	14	769	1
	MAX	56585	30991	2851	2072	20671	3
	MEDIAN	5779	27	24	17	5711	1
	AVG	7654	43	114	35	7462	1
	STDDEV	4787	625	203	76	3883	0
▷ ROOT	SUM	2465005	16632	9673	7588	2431112	500
	MIN	815	22	10	14	769	1
	MAX	7466	1652	241	34	5539	1
	MEDIAN	5096	28	23	15	5030	1
	AVG	4930	33	19	15	4862	1
	STDDEV	994	82	13	2	897	0
▷ ALTERNATIVES	SUM	3039223	13727	15061	8930	3001505	500
	MIN	897	14	17	17	849	1
	MAX	10609	1782	1158	68	7601	1
	MEDIAN	6107	26	30	18	6033	1
	AVG	6078	27	30	18	6003	1
	STDDEV	761	80	55	3	623	0
▷ SUCCESSORS	SUM	3185026	42159	9949	9560	3123358	500
	MIN	5162	14	16	16	5116	1
	MAX	39761	30991	42	884	7844	1
	MEDIAN	6353	17	18	17	6301	1
	AVG	6370	84	20	19	6247	1
	STDDEV	2025	1384	5	39	597	0
▷ PREDECESSOR	SUM	2627198	14209	15201	10077	2587711	500
	MIN	869	13	16	16	824	1
	MAX	10193	1930	1167	1234	5862	1
	MEDIAN	5235	25	31	17	5162	1
	AVG	5254	28	30	20	5175	1
	STDDEV	431	99	55	54	223	0
▷ BASEVERSION	SUM	7818379	20236	234912	50990	7512241	1500
	MIN	6296	31	311	82	5872	3
	MAX	28424	2830	2851	2072	20671	3
	MEDIAN	15618	34	430	86	15068	3
	AVG	15637	40	470	102	15024	3
	STDDEV	1461	125	202	137	997	0
GET CVC	SUM	8869348	47534	54248	28764	8738802	1500
	MIN	952	24	27	15	886	1
	MAX	16173	3149	1198	2027	9799	1
	MEDIAN	5586	29	29	16	5512	1
	AVG	5913	32	36	19	5826	1
	STDDEV	868	81	56	61	670	0
PIN	SUM	6687859	54143	55965	25358	6552393	3000
	MIN	1213	28	31	15	1139	2
	MAX	27628	2574	859	350	23845	2
	MEDIAN	4221	33	35	16	4137	2
	AVG	4459	36	37	17	4368	2
	STDDEV	1182	75	39	9	1060	0
SELECT	SUM	121094381	223461	570629	585609	119714682	5250
	MIN	1012	23	29	10	950	1
	MAX	265949	11768	57593	124243	72345	1
	MEDIAN	17418	36	67	27	17288	1
	AVG	23066	43	109	112	22803	1
	STDDEV	18769	183	926	2262	15399	0
▷ außerhalb von Workspaces	SUM	95403809	166913	402212	543860	94290824	4000
	MIN	1033	23	29	10	971	1
	MAX	233737	11768	25381	124243	72345	1
	MEDIAN	17989	34	68	45	17842	1
	AVG	23851	42	101	136	23573	1
	STDDEV	18841	208	451	2587	15595	0
▷▷ 1 Navigationsschritt	SUM	3496337	17362	22361	7199	3449415	500
	MIN	1033	23	29	10	971	1
	MAX	11357	854	1764	791	7948	1
	MEDIAN	7058	34	35	11	6978	1
	AVG	6993	35	45	14	6899	1
	STDDEV	902	38	107	49	708	0
▷▷ 2 Navigationsschritte	SUM	43887098	98944	233420	165200	43389534	2500
	MIN	1217	26	41	15	1135	1
	MAX	126312	11768	25381	47471	41692	1
	MEDIAN	17119	28	67	44	16980	1
	AVG	17555	40	93	66	17356	1

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	STDDEV	5145	241	551	958	3394	0
▷▷ 4 Navigationsschritte	SUM	48020374	50607	146431	371461	47451875	1000
	MIN	2482	33	100	68	2281	1
	MAX	202232	3462	2182	124243	72345	1
	MEDIAN	47961	35	110	88	47728	1
	AVG	48020	51	146	371	47452	1
	STDDEV	17060	167	211	4939	11743	0
▷ innerhalb von Workspaces	SUM	25690572	56548	168417	41749	25423858	1250
	MIN	1060	37	57	16	950	1
	MAX	135653	915	57593	8963	68182	1
	MEDIAN	12970	43	62	26	12839	1
	AVG	20552	45	135	33	20339	1
	STDDEV	16489	34	1717	257	14482	0
▷▷ 1 Navigationsschritt	SUM	2389107	11994	75344	5057	2296712	250
	MIN	1060	37	57	16	950	1
	MAX	69449	840	57593	795	10221	1
	MEDIAN	9551	44	61	17	9429	1
	AVG	9556	48	301	20	9187	1
	STDDEV	5123	51	3632	49	1391	0
▷▷ 2 Navigationsschritte	SUM	23301465	44554	93073	36692	23127146	1000
	MIN	1186	39	57	25	1065	1
	MAX	97102	915	19042	8963	68182	1
	MEDIAN	16143	43	62	26	16012	1
	AVG	23301	45	93	37	23127	1
	STDDEV	15854	28	614	286	14927	0

G.3.2 Normierte Zeiten

		Parser	Reduction	Rendering	SQL-Ausführung
CREATE NEW OBJECT	MIN	0,3%	2,2%	0,7%	96,8%
	MAX	20,8%	23,9%	0,8%	54,5%
	AVG	1,7%	4,7%	0,8%	92,8%
CREATE NEW RELATIONSHIP	MIN	0,9%	2,7%	0,2%	96,2%
	MAX	6,9%	7,3%	30,3%	55,5%
	AVG	0,2%	1,0%	0,2%	98,5%
▷ kein FRE	MIN	2,1%	2,6%	0,2%	95,1%
	MAX	8,4%	2,2%	0,6%	88,9%
	AVG	2,5%	3,0%	0,3%	94,2%
▷ ein FRE	MIN	0,4%	1,4%	0,4%	97,8%
	MAX	6,9%	7,3%	30,3%	55,5%
	AVG	0,2%	1,0%	0,2%	98,5%
CREATE NEW OBJECT + RELATIONSHIP	MIN	0,6%	2,2%	0,6%	96,6%
	MAX	1,9%	44,3%	21,5%	32,3%
	AVG	0,2%	1,5%	0,4%	97,8%
▷ 1 Objekt + 1 kein FRE	MIN	0,7%	3,2%	0,5%	95,6%
	MAX	17,4%	5,4%	4,1%	73,1%
	AVG	1,1%	3,6%	0,6%	94,7%
▷ 1 Objekt + 1 zwei FRE	MIN	0,6%	2,2%	0,6%	96,6%
	MAX	4,6%	2,7%	66,9%	25,8%
	AVG	0,7%	2,3%	0,8%	96,2%
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	MIN	0,2%	1,6%	0,4%	97,8%
	MAX	3,2%	73,0%	2,0%	21,8%
	AVG	0,2%	1,5%	0,3%	98,0%
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	MIN	0,1%	1,5%	0,4%	98,0%
	MAX	1,0%	14,5%	22,4%	62,1%
	AVG	0,1%	1,3%	0,4%	98,2%
CREATE SUCCESSOR	MIN	0,2%	8,4%	0,6%	90,9%
	MAX	1,7%	38,2%	5,6%	54,5%
	AVG	0,1%	5,2%	0,3%	94,4%
▷ außerhalb von Workspaces	MIN	0,2%	8,4%	0,6%	90,9%
	MAX	1,7%	38,2%	5,6%	54,5%
	AVG	0,1%	5,2%	0,3%	94,3%
▷ innerhalb von Workspaces	MIN	0,2%	5,0%	0,5%	94,3%
	MAX	1,1%	3,7%	0,7%	94,5%
	AVG	0,2%	3,8%	0,4%	95,6%
COPY OBJECTS	MIN	0,2%	1,3%	0,6%	97,8%
	MAX	0,7%	0,7%	21,3%	77,3%
	AVG	0,1%	0,9%	0,4%	98,7%
DELETE OBJECTS	MIN	0,1%	1,6%	0,3%	97,9%
	MAX	0,6%	7,9%	0,8%	90,7%
	AVG	0,0%	1,3%	0,2%	98,5%
MERGE OBJECTS	MIN	0,5%	25,5%	1,9%	72,1%
	MAX	4,3%	46,7%	1,9%	47,1%
	AVG	0,2%	9,3%	0,6%	89,9%

		Parser	Reduction	Rendering	SQL-Ausführung
FREEZE	MIN	1,6%	6,8%	0,9%	90,7%
	MAX	4,4%	4,8%	2,5%	88,3%
	AVG	0,3%	2,2%	0,4%	97,1%
▷ außerhalb von Workspaces	MIN	1,6%	6,8%	0,9%	90,7%
	MAX	4,8%	5,3%	2,7%	87,2%
	AVG	0,3%	2,2%	0,4%	97,1%
▷ innerhalb von Workspaces	MIN	1,8%	8,3%	1,2%	88,6%
	MAX	0,9%	3,0%	0,3%	95,7%
	AVG	0,4%	2,5%	0,3%	96,7%
ATTACH und DETACH	MIN	0,5%	0,4%	0,3%	98,9%
	MAX	3,1%	5,6%	4,9%	86,4%
	AVG	0,2%	1,2%	0,3%	98,3%
▷ ATTACH	MIN	0,5%	0,3%	0,3%	98,9%
	MAX	2,6%	5,3%	4,9%	87,2%
	AVG	0,2%	1,1%	0,3%	98,4%
▷ DETACH	MIN	0,7%	0,4%	0,3%	98,6%
	MAX	7,0%	12,6%	0,3%	80,1%
	AVG	0,7%	2,7%	0,4%	96,3%
GET ...	MIN	1,6%	1,2%	1,7%	95,4%
	MAX	54,8%	5,0%	3,7%	36,5%
	AVG	0,6%	1,5%	0,5%	97,5%
▷ ROOT	MIN	2,7%	1,2%	1,7%	94,4%
	MAX	22,1%	3,2%	0,5%	74,2%
	AVG	0,7%	0,4%	0,3%	98,6%
▷ ALTERNATIVES	MIN	1,6%	1,9%	1,9%	94,6%
	MAX	16,8%	10,9%	0,6%	71,6%
	AVG	0,5%	0,5%	0,3%	98,8%
▷ SUCCESSORS	MIN	0,3%	0,3%	0,3%	99,1%
	MAX	77,9%	0,1%	2,2%	19,7%
	AVG	1,3%	0,3%	0,3%	98,1%
▷ PREDECESSOR	MIN	1,5%	1,8%	1,8%	94,8%
	MAX	18,9%	11,4%	12,1%	57,5%
	AVG	0,5%	0,6%	0,4%	98,5%
▷ BASEVERSION	MIN	0,5%	4,9%	1,3%	93,3%
	MAX	10,0%	10,0%	7,3%	72,7%
	AVG	0,3%	3,0%	0,7%	96,1%
GET CVC	MIN	2,3%	2,6%	1,2%	93,9%
	MAX	9,3%	3,1%	1,3%	86,3%
	AVG	0,8%	0,8%	0,4%	98,0%
PIN	MIN	2,3%	2,6%	1,2%	93,9%
	MAX	9,3%	3,1%	1,3%	86,3%
	AVG	0,8%	0,8%	0,4%	98,0%
SELECT	MIN	2,3%	2,9%	1,0%	93,9%
	MAX	4,4%	21,7%	46,7%	27,2%
	AVG	0,2%	0,5%	0,5%	98,9%
▷ außerhalb von Workspaces	MIN	2,2%	2,8%	1,0%	94,0%
	MAX	5,0%	10,9%	53,2%	31,0%
	AVG	0,2%	0,4%	0,6%	98,8%
▷▷ 1 Navigationsschritt	MIN	2,2%	2,8%	1,0%	94,0%
	MAX	7,5%	15,5%	7,0%	70,0%
	AVG	0,5%	0,6%	0,2%	98,7%
▷▷ 2 Navigationsschritte	MIN	2,1%	3,4%	1,2%	93,3%
	MAX	9,3%	20,1%	37,6%	33,0%
	AVG	0,2%	0,5%	0,4%	98,9%
▷▷ 4 Navigationsschritte	MIN	1,3%	4,0%	2,7%	91,9%
	MAX	1,7%	1,1%	61,4%	35,8%
	AVG	0,1%	0,3%	0,8%	98,8%
▷ innerhalb von Workspaces	MIN	3,5%	5,4%	1,5%	89,6%
	MAX	0,7%	42,5%	6,6%	50,3%
	AVG	0,2%	0,7%	0,2%	99,0%
▷▷ 1 Navigationsschritt	MIN	3,5%	5,4%	1,5%	89,6%
	MAX	1,2%	82,9%	1,1%	14,7%
	AVG	0,5%	3,2%	0,2%	96,1%
▷▷ 2 Navigationsschritte	MIN	3,3%	4,8%	2,1%	89,8%
	MAX	0,9%	19,6%	9,2%	70,2%
	AVG	0,2%	0,4%	0,2%	99,3%

G.3.3 Absolute Zeiten zum Überprüfen der Kardinalitäten in μs

		Summe	Reduction	Rendering	SQL-Ausführung	SQL
CREATE NEW OBJECT	SUM	813743	6372	3453	803918	400
	MIN	0	0	0	0	0
	MAX	9746	731	376	8639	4
	AVG	4648	36	19	4593	2

		Summe	Reduction	Rendering	SQL-Ausführung	SQL
	STDDEV	4070	60	31	3979	1
CREATE NEW RELATIONSHIP	SUM	196819380	4788265	357050	191674065	40400
	MIN	2144	221	14	1909	2
	MAX	41257	9330	8937	22990	4
	AVG	9791	238	17	9536	2
	STDDEV	619	106	103	410	0
▷ kein FRE	SUM	825666	40474	2878	782314	400
	MIN	2311	377	25	1909	4
	MAX	9128	590	50	8488	4
	AVG	8255	404	28	7823	4
	STDDEV	1082	32	5	1045	0
▷ ein FRE	SUM	195993714	4747791	354172	190891751	40000
	MIN	4849	221	14	4614	2
	MAX	41257	9330	8937	22990	2
	AVG	9798	237	17	9544	2
	STDDEV	595	106	104	385	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	1030238235	19005226	1676642	1009556367	160000
	MIN	7594	346	21	7227	3
	MAX	200212	24588	104797	70827	10
	AVG	39623	730	64	38829	6
	STDDEV	13267	451	818	11998	2
▷ 1 Objekt + 1 kein FRE	SUM	26160579	864568	72978	25223033	10000
	MIN	21803	776	64	20963	10
	MAX	40627	10286	891	29450	10
	AVG	26159	864	72	25223	10
	STDDEV	1212	351	45	816	0
▷ 1 Objekt + 1 zwei FRE	SUM	307873364	3740925	301334	303831105	30000
	MIN	7594	346	21	7227	3
	MAX	74222	7487	6970	59765	3
	AVG	30787	374	30	30383	3
	STDDEV	10394	206	191	9997	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	457990270	8468123	725042	448797105	70000
	MIN	23636	790	49	22797	7
	MAX	168682	24588	73416	70678	7
	AVG	45797	846	72	44879	7
	STDDEV	11334	426	767	10141	0
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	238214022	5931610	577288	231705124	50000
	MIN	32837	1110	76	31651	10
	MAX	182193	6569	104797	70827	10
	AVG	47642	1186	115	46341	10
	STDDEV	7591	305	1493	5793	0
DELETE OBJECTS	SUM	16133038	380720	29783	15722535	3002
	MIN	0	0	0	0	0
	MAX	58971	3783	4858	50330	7
	AVG	3225	76	5	3144	0
	STDDEV	4112	119	69	3924	0

G.3.4 Normierte Zeiten zum Überprüfen der Kardinalitäten

		Reduction	Rendering	SQL-Ausführung
CREATE NEW OBJECT	MIN	0,0%	0,0%	0,0%
	MAX	7,5%	3,9%	88,6%
	AVG	0,8%	0,4%	98,8%
CREATE NEW RELATIONSHIP	MIN	10,3%	0,7%	89,0%
	MAX	22,6%	21,7%	55,7%
	AVG	2,4%	0,2%	97,4%
▷ kein FRE	MIN	16,3%	1,1%	82,6%
	MAX	6,5%	0,5%	93,0%
	AVG	4,9%	0,3%	94,8%
▷ ein FRE	MIN	4,6%	0,3%	95,2%
	MAX	22,6%	21,7%	55,7%
	AVG	2,4%	0,2%	97,4%
CREATE NEW OBJECT + RELATIONSHIP	MIN	4,6%	0,3%	95,2%
	MAX	12,3%	52,3%	35,4%
	AVG	1,8%	0,2%	98,0%
▷ 1 Objekt + 1 kein FRE	MIN	3,6%	0,3%	96,1%
	MAX	25,3%	2,2%	72,5%
	AVG	3,3%	0,3%	96,4%
▷ 1 Objekt + 1 zwei FRE	MIN	4,6%	0,3%	95,2%
	MAX	10,1%	9,4%	80,5%
	AVG	1,2%	0,1%	98,7%
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	MIN	3,3%	0,2%	96,5%
	MAX	14,6%	43,5%	41,9%
	AVG	1,8%	0,2%	98,0%

		Reduction	Rendering	SQL-Ausführung
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	MIN	3,4%	0,2%	96,4%
	MAX	3,6%	57,5%	38,9%
	AVG	2,5%	0,2%	97,3%
DELETE OBJECTS	MIN	0,0%	0,0%	0,0%
	MAX	6,4%	8,2%	85,3%
	AVG	2,4%	0,2%	97,5%

G.4 Variante 4

G.4.1 Absolute Zeiten in μs

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
alle Befehle	SUM	3921152907	5187216	69897467	14304429	3831763795	953688
CREATE NEW OBJECT	SUM	1472568	20006	80184	14027	1358351	700
	MIN	7255	21	160	52	7022	4
	MAX	54421	10537	24223	1228	18433	4
	MEDIAN	8146	44	246	61	7795	4
	AVG	8415	114	458	80	7762	4
	STDDEV	3766	793	1847	108	1017	0
CREATE NEW RELATIONSHIP	SUM	511239429	866282	4247666	1595360	504530121	100100
	MIN	3943	36	109	9	3789	1
	MAX	96650	9884	9879	9758	67129	5
	MEDIAN	24532	38	176	73	24245	5
	AVG	25435	43	211	79	25101	4
	STDDEV	4592	147	196	196	4053	0
▷ kein FRE	SUM	425389	14267	13019	1895	396208	100
	MIN	3990	83	109	9	3789	1
	MAX	8067	2288	175	843	4761	1
	MEDIAN	4173	98	125	10	3940	1
	AVG	4254	143	130	19	3962	1
	STDDEV	550	274	12	83	181	0
▷ ein FRE	SUM	510814040	852015	4234647	1593465	504133913	100000
	MIN	16896	36	144	68	16648	5
	MAX	96650	9884	9879	9758	67129	5
	MEDIAN	24550	38	176	73	24263	5
	AVG	25541	43	212	80	25207	5
	STDDEV	4315	146	196	197	3777	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	1598996884	2438977	21390676	6241553	1568925678	405000
	MIN	11827	67	390	63	11307	5
	MAX	517115	15010	295166	29828	177111	26
	MEDIAN	72119	86	937	241	70855	18
	AVG	61500	94	823	240	60343	15
	STDDEV	34260	171	2394	359	31335	6
▷ 1 Objekt + 1 kein FRE	SUM	12433022	135576	448876	81290	11767280	5000
	MIN	11838	74	394	63	11307	5
	MAX	45492	9009	2463	1695	32325	5
	MEDIAN	12134	101	430	74	11529	5
	AVG	12433	136	449	81	11767	5
	STDDEV	1877	327	129	74	1347	0
▷ 1 Objekt + 1 zwei FRE	SUM	287414038	783022	4517261	1377388	280736367	90000
	MIN	27036	67	390	123	26456	9
	MAX	376214	6014	295166	10549	64485	9
	MEDIAN	28604	70	407	128	27999	9
	AVG	28741	78	452	138	28074	9
	STDDEV	4726	124	2953	211	1438	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	760408365	993743	9743647	2729295	746941680	180000
	MIN	55112	73	840	233	53966	18
	MAX	211449	15010	10394	8934	177111	18
	MEDIAN	75873	87	947	243	74596	18
	AVG	76041	99	974	273	74694	18
	STDDEV	5511	208	311	357	4634	0
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	538741459	526636	6680892	2053580	529480351	130000
	MIN	90863	81	1168	349	89265	26
	MAX	428212	5782	231480	29828	161122	26
	MEDIAN	107398	88	1208	368	105734	26
	AVG	107748	105	1336	411	105896	26
	STDDEV	8136	112	3400	519	4106	0
CREATE SUCCESSOR	SUM	460896500	506220	25817092	1763450	432809738	128167
	MIN	6861	15	667	45	6134	5

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	MAX	971103	112157	234940	115080	508926	136
	MEDIAN	18006	16	1176	57	16757	5
	AVG	22483	25	1259	86	21113	6
	STDDEV	24133	786	1865	841	20641	5
▷ außerhalb von Workspaces	SUM	446131980	484668	25292492	1700041	418654779	124076
	MIN	6861	15	667	45	6134	5
	MAX	971103	112157	234940	115080	508926	136
	MEDIAN	17997	16	1179	57	16745	5
	AVG	22307	24	1265	85	20933	6
	STDDEV	23889	795	1882	851	20361	5
▷ innerhalb von Workspaces	SUM	14764520	21552	524600	63409	14154959	4091
	MIN	12905	33	692	66	12114	6
	MAX	354744	2680	10079	2187	339798	96
	MEDIAN	21445	38	748	75	20584	6
	AVG	29529	43	1049	127	28310	8
	STDDEV	30107	118	928	203	28858	8
COPY OBJECTS	SUM	696867087	417805	9162983	2989350	684296949	197169
	MIN	8087	17	106	53	7911	4
	MAX	1694058	4363	175187	10111	1504397	338
	MEDIAN	8530	18	115	58	8339	4
	AVG	34843	21	458	149	34215	9
	STDDEV	51123	48	1406	311	49358	11
DELETE OBJECTS	SUM	420887908	74939	5651924	982072	414178973	78019
	MIN	11278	13	195	39	11031	4
	MAX	1137966	3027	12206	31190	1091543	184
	MEDIAN	48360	14	545	110	47691	9
	AVG	84178	15	1130	196	82836	15
	STDDEV	95598	43	1494	534	93528	17
MERGE OBJECTS	SUM	9249416	18990	837848	57312	8335266	2205
	MIN	4836	27	1290	97	3422	4
	MAX	55253	2830	16706	1846	33871	5
	MEDIAN	18435	32	1547	101	16755	4
	AVG	18499	38	1676	115	16671	4
	STDDEV	3062	125	826	99	2012	0
FREEZE	SUM	12180787	37822	291319	39237	11812409	4211
	MIN	1415	25	107	14	1269	2
	MAX	69768	3216	2272	1283	62997	20
	MEDIAN	6508	29	145	18	6316	2
	AVG	11073	34	265	36	10739	3
	STDDEV	8345	107	243	54	7941	2
▷ außerhalb von Workspaces	SUM	11250950	34323	267984	35069	10913574	3947
	MIN	1415	25	107	14	1269	2
	MAX	63570	3216	1988	1283	57083	20
	MEDIAN	6394	29	135	17	6213	2
	AVG	11251	34	268	35	10914	3
	STDDEV	8418	112	240	48	8018	3
▷ innerhalb von Workspaces	SUM	929837	3499	23335	4168	898835	264
	MIN	1549	32	127	21	1369	2
	MAX	66272	59	2272	944	62997	19
	MEDIAN	7280	34	152	23	7071	2
	AVG	9298	35	233	42	8988	2
	STDDEV	7253	3	271	95	6883	2
ATTACH und DETACH	SUM	46606896	157905	781175	180501	45487315	14867
	MIN	2898	17	12	9	2860	1
	MAX	192859	4309	5959	25334	157257	14
	MEDIAN	8395	26	148	26	8195	3
	AVG	8752	30	147	34	8542	2
	STDDEV	6752	102	205	351	6094	1
▷ ATTACH	SUM	44787768	142672	731775	166458	43746863	14234
	MIN	3018	17	12	9	2980	1
	MAX	192859	4309	5959	25334	157257	14
	MEDIAN	8411	26	148	26	8211	3
	AVG	8782	28	143	33	8578	2
	STDDEV	6771	77	195	356	6143	1
▷ DETACH	SUM	1819128	15233	49400	14043	1740452	633
	MIN	2906	23	14	9	2860	1
	MAX	37477	3264	4886	2027	27300	9
	MEDIAN	7078	26	225	28	6799	3
	AVG	8085	68	220	62	7735	2
	STDDEV	5669	329	367	193	4780	1
GET ...	SUM	18350556	76983	272881	88642	17912050	3500
	MIN	698	13	11	14	660	1
	MAX	24514	2859	2124	1925	17606	3
	MEDIAN	5652	26	23	17	5586	1
	AVG	7340	31	109	35	7165	1
	STDDEV	4189	110	184	78	3817	0
▷ ROOT	SUM	2306632	16376	9501	9330	2271425	500
	MIN	698	13	11	14	660	1
	MAX	8674	1665	245	913	5851	1
	MEDIAN	4986	27	22	15	4922	1
	AVG	4613	33	19	19	4543	1
	STDDEV	1436	93	13	56	1273	0
▷ ALTERNATIVES	SUM	2952441	15593	15609	10480	2910759	500
	MIN	776	14	16	16	730	1
	MAX	12951	2814	1926	970	7241	1

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	MEDIAN	5945	24	31	18	5872	1
	AVG	5905	31	31	21	5822	1
	STDDEV	1018	149	86	52	731	0
▷ SUCCESSORS	SUM	3046236	10694	9802	9084	3016656	500
	MIN	1032	14	16	16	986	1
	MAX	8667	659	43	414	7551	1
	MEDIAN	6168	16	17	17	6118	1
	AVG	6092	21	20	18	6033	1
	STDDEV	737	29	5	18	684	0
▷ PREDECESSOR	SUM	2566717	14064	14610	12425	2525618	500
	MIN	762	14	17	16	715	1
	MAX	10801	1830	979	1925	6067	1
	MEDIAN	5109	17	30	17	5045	1
	AVG	5133	28	29	25	5051	1
	STDDEV	620	112	46	102	361	0
▷ BASEVERSION	SUM	7478530	20256	223359	47323	7187592	1500
	MIN	3480	31	303	81	3065	3
	MAX	23755	2859	2124	1166	17606	3
	MEDIAN	15368	34	419	84	14831	3
	AVG	14957	41	447	95	14375	3
	STDDEV	2276	126	130	98	1922	0
GET CVC	SUM	8664890	49870	57157	27838	8530025	1500
	MIN	796	24	26	15	731	1
	MAX	17364	3543	1677	1073	11071	1
	MEDIAN	5478	28	28	16	5406	1
	AVG	5777	33	38	19	5687	1
	STDDEV	1002	106	74	44	778	0
PIN	SUM	13711103	53555	226612	38742	13392194	3000
	MIN	1744	27	131	20	1566	2
	MAX	37800	3071	2816	2405	29508	2
	MEDIAN	9030	32	141	24	8833	2
	AVG	9141	36	151	26	8928	2
	STDDEV	1442	84	119	62	1177	0
SELECT	SUM	86271311	214497	349090	129134	85578590	5250
	MIN	844	23	26	10	785	1
	MAX	103555	8256	19078	8843	67378	1
	MEDIAN	14269	35	51	17	14166	1
	AVG	16433	41	66	25	16301	1
	STDDEV	9731	139	387	145	9060	0
▷ außerhalb von Workspaces	SUM	57696101	160077	243280	84310	57208434	4000
	MIN	849	23	26	10	790	1
	MAX	96980	8256	18742	2604	67378	1
	MEDIAN	14266	30	43	17	14176	1
	AVG	14424	40	61	21	14302	1
	STDDEV	5631	158	323	78	5072	0
▷▷ 1 Navigationsschritt	SUM	3384637	22621	41209	5618	3315189	500
	MIN	849	23	26	10	790	1
	MAX	30687	3405	18742	23	8517	1
	MEDIAN	6829	27	31	11	6760	1
	AVG	6769	45	82	11	6630	1
	STDDEV	2035	167	844	1	1023	0
▷▷ 2 Navigationsschritte	SUM	35819877	90377	128828	51804	35548868	2500
	MIN	999	26	37	15	921	1
	MAX	56020	8256	4694	2604	40466	1
	MEDIAN	14651	28	42	16	14565	1
	AVG	14328	36	52	21	14220	1
	STDDEV	2614	167	128	80	2240	0
▷▷ 4 Navigationsschritte	SUM	18491587	47079	73243	26888	18344377	1000
	MIN	895	33	54	18	790	1
	MAX	74410	2544	2461	2027	67378	1
	MEDIAN	13260	35	60	22	13143	1
	AVG	18492	47	73	27	18344	1
	STDDEV	6999	130	142	91	6636	0
▷ innerhalb von Workspaces	SUM	28575210	54420	105810	44824	28370156	1250
	MIN	891	35	55	16	785	1
	MAX	92474	923	19078	8843	63630	1
	MEDIAN	14279	42	59	26	14152	1
	AVG	22860	44	85	36	22696	1
	STDDEV	15278	25	543	261	14449	0
▷▷ 1 Navigationsschritt	SUM	2231803	10751	17942	4222	2198888	250
	MIN	891	35	55	16	785	1
	MAX	13090	98	1692	24	11276	1
	MEDIAN	9304	42	59	17	9186	1
	AVG	8927	43	72	17	8796	1
	STDDEV	1969	6	132	1	1830	0
▷▷ 2 Navigationsschritte	SUM	26343407	43669	87868	40602	26171268	1000
	MIN	1131	38	56	25	1012	1
	MAX	92474	923	19078	8843	63630	1
	MEDIAN	26226	42	59	26	26099	1
	AVG	26343	44	88	41	26171	1
	STDDEV	15056	28	604	291	14134	0

G.4.2 Normierte Zeiten

		Parser	Reduction	Rendering	SQL-Ausführung
CREATE NEW OBJECT	MIN	0,3%	2,2%	0,7%	96,8%
	MAX	19,4%	44,5%	2,3%	33,9%
	AVG	1,4%	5,4%	1,0%	92,2%
CREATE NEW RELATIONSHIP	MIN	0,9%	2,8%	0,2%	96,1%
	MAX	10,2%	10,2%	10,1%	69,5%
	AVG	0,2%	0,8%	0,3%	98,7%
▷ kein FRE	MIN	2,1%	2,7%	0,2%	95,0%
	MAX	28,4%	2,2%	10,4%	59,0%
	AVG	3,4%	3,1%	0,4%	93,1%
▷ ein FRE	MIN	0,2%	0,9%	0,4%	98,5%
	MAX	10,2%	10,2%	10,1%	69,5%
	AVG	0,2%	0,8%	0,3%	98,7%
CREATE NEW OBJECT + RELATIONSHIP	MIN	0,6%	3,3%	0,5%	95,6%
	MAX	2,9%	57,1%	5,8%	34,2%
	AVG	0,2%	1,3%	0,4%	98,1%
▷ 1 Objekt + 1 kein FRE	MIN	0,6%	3,3%	0,5%	95,5%
	MAX	19,8%	5,4%	3,7%	71,1%
	AVG	1,1%	3,6%	0,7%	94,6%
▷ 1 Objekt + 1 zwei FRE	MIN	0,2%	1,4%	0,5%	97,9%
	MAX	1,6%	78,5%	2,8%	17,1%
	AVG	0,3%	1,6%	0,5%	97,7%
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	MIN	0,1%	1,5%	0,4%	97,9%
	MAX	7,1%	4,9%	4,2%	83,8%
	AVG	0,1%	1,3%	0,4%	98,2%
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	MIN	0,1%	1,3%	0,4%	98,2%
	MAX	1,4%	54,1%	7,0%	37,6%
	AVG	0,1%	1,2%	0,4%	98,3%
CREATE SUCCESSOR	MIN	0,2%	9,7%	0,7%	89,4%
	MAX	11,5%	24,2%	11,9%	52,4%
	AVG	0,1%	5,6%	0,4%	93,9%
▷ außerhalb von Workspaces	MIN	0,2%	9,7%	0,7%	89,4%
	MAX	11,5%	24,2%	11,9%	52,4%
	AVG	0,1%	5,7%	0,4%	93,8%
▷ innerhalb von Workspaces	MIN	0,3%	5,4%	0,5%	93,9%
	MAX	0,8%	2,8%	0,6%	95,8%
	AVG	0,1%	3,6%	0,4%	95,9%
COPY OBJECTS	MIN	0,2%	1,3%	0,7%	97,8%
	MAX	0,3%	10,3%	0,6%	88,8%
	AVG	0,1%	1,3%	0,4%	98,2%
DELETE OBJECTS	MIN	0,1%	1,7%	0,3%	97,8%
	MAX	0,3%	1,1%	2,7%	95,9%
	AVG	0,0%	1,3%	0,2%	98,4%
MERGE OBJECTS	MIN	0,6%	26,7%	2,0%	70,8%
	MAX	5,1%	30,2%	3,3%	61,3%
	AVG	0,2%	9,1%	0,6%	90,1%
FREEZE	MIN	1,8%	7,6%	1,0%	89,7%
	MAX	4,6%	3,3%	1,8%	90,3%
	AVG	0,3%	2,4%	0,3%	97,0%
▷ außerhalb von Workspaces	MIN	1,8%	7,6%	1,0%	89,7%
	MAX	5,1%	3,1%	2,0%	89,8%
	AVG	0,3%	2,4%	0,3%	97,0%
▷ innerhalb von Workspaces	MIN	2,1%	8,2%	1,4%	88,4%
	MAX	0,1%	3,4%	1,4%	95,1%
	AVG	0,4%	2,5%	0,4%	96,7%
ATTACH und DETACH	MIN	0,6%	0,4%	0,3%	98,7%
	MAX	2,2%	3,1%	13,1%	81,5%
	AVG	0,3%	1,7%	0,4%	97,6%
▷ ATTACH	MIN	0,6%	0,4%	0,3%	98,7%
	MAX	2,2%	3,1%	13,1%	81,5%
	AVG	0,3%	1,6%	0,4%	97,7%
▷ DETACH	MIN	0,8%	0,5%	0,3%	98,4%
	MAX	8,7%	13,0%	5,4%	72,8%
	AVG	0,8%	2,7%	0,8%	95,7%
GET ...	MIN	1,9%	1,6%	2,0%	94,6%
	MAX	11,7%	8,7%	7,9%	71,8%
	AVG	0,4%	1,5%	0,5%	97,6%
▷ ROOT	MIN	1,9%	1,6%	2,0%	94,6%
	MAX	19,2%	2,8%	10,5%	67,5%
	AVG	0,7%	0,4%	0,4%	98,5%
▷ ALTERNATIVES	MIN	1,8%	2,1%	2,1%	94,1%
	MAX	21,7%	14,9%	7,5%	55,9%
	AVG	0,5%	0,5%	0,4%	98,6%
▷ SUCCESSORS	MIN	1,4%	1,6%	1,6%	95,5%
	MAX	7,6%	0,5%	4,8%	87,1%
	AVG	0,4%	0,3%	0,3%	99,0%
▷ PREDECESSOR	MIN	1,8%	2,2%	2,1%	93,8%
	MAX	16,9%	9,1%	17,8%	56,2%
	AVG	0,5%	0,6%	0,5%	98,4%
▷ BASEVERSION	MIN	0,9%	8,7%	2,3%	88,1%
	MAX	12,0%	8,9%	4,9%	74,1%

		Parser	Reduction	Rendering	SQL-Ausführung
	AVG	0,3%	3,0%	0,6%	96,1%
GET CVC	MIN	1,5%	7,5%	1,1%	89,8%
	MAX	8,1%	7,4%	6,4%	78,1%
	AVG	0,4%	1,7%	0,3%	97,7%
PIN	MIN	1,5%	7,5%	1,1%	89,8%
	MAX	8,1%	7,4%	6,4%	78,1%
	AVG	0,4%	1,7%	0,3%	97,7%
SELECT	MIN	2,7%	3,1%	1,2%	93,0%
	MAX	8,0%	18,4%	8,5%	65,1%
	AVG	0,2%	0,4%	0,1%	99,2%
▷ außerhalb von Workspaces	MIN	2,7%	3,1%	1,2%	93,1%
	MAX	8,5%	19,3%	2,7%	69,5%
	AVG	0,3%	0,4%	0,1%	99,2%
▷▷ 1 Navigationsschritt	MIN	2,7%	3,1%	1,2%	93,1%
	MAX	11,1%	61,1%	0,1%	27,8%
	AVG	0,7%	1,2%	0,2%	97,9%
▷▷ 2 Navigationsschritte	MIN	2,6%	3,7%	1,5%	92,2%
	MAX	14,7%	8,4%	4,6%	72,2%
	AVG	0,3%	0,4%	0,1%	99,2%
▷▷ 4 Navigationsschritte	MIN	3,7%	6,0%	2,0%	88,3%
	MAX	3,4%	3,3%	2,7%	90,5%
	AVG	0,3%	0,4%	0,1%	99,2%
▷ innerhalb von Workspaces	MIN	3,9%	6,2%	1,8%	88,1%
	MAX	1,0%	20,6%	9,6%	68,8%
	AVG	0,2%	0,4%	0,2%	99,3%
▷▷ 1 Navigationsschritt	MIN	3,9%	6,2%	1,8%	88,1%
	MAX	0,7%	12,9%	0,2%	86,1%
	AVG	0,5%	0,8%	0,2%	98,5%
▷▷ 2 Navigationsschritte	MIN	3,4%	5,0%	2,2%	89,5%
	MAX	1,0%	20,6%	9,6%	68,8%
	AVG	0,2%	0,3%	0,2%	99,3%

G.4.3 Absolute Zeiten zum Überprüfen der Kardinalitäten in μs

		Summe	Reduction	Rendering	SQL-Ausführung	SQL
CREATE NEW OBJECT	SUM	795018	6641	3480	784897	400
	MIN	0	0	0	0	0
	MAX	9852	741	383	8728	4
	AVG	4541	37	19	4485	2
	STDDEV	3979	61	32	3886	1
CREATE NEW RELATIONSHIP	SUM	158860793	4548532	493239	153819022	40400
	MIN	1370	208	14	1148	2
	MAX	32069	7836	9657	14576	4
	AVG	7902	226	24	7652	2
	STDDEV	604	105	150	349	0
▷ kein FRE	SUM	787288	40557	2844	743887	400
	MIN	2082	373	25	1684	4
	MAX	9432	571	48	8813	4
	AVG	7871	405	28	7438	4
	STDDEV	1285	30	4	1251	0
▷ ein FRE	SUM	158073505	4507975	490395	153075135	40000
	MIN	1370	208	14	1148	2
	MAX	32069	7836	9657	14576	2
	AVG	7902	225	24	7653	2
	STDDEV	593	104	151	338	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	473517433	17626203	1649620	454241610	160000
	MIN	5378	310	25	5043	3
	MAX	422385	354565	24499	43321	10
	AVG	18210	677	63	17470	6
	STDDEV	10273	2306	206	7761	2
▷ 1 Objekt + 1 kein FRE	SUM	22128170	842103	75906	21210161	10000
	MIN	17761	745	66	16950	10
	MAX	59382	31411	2102	25869	10
	AVG	22127	842	75	21210	10
	STDDEV	2005	979	83	943	0
▷ 1 Objekt + 1 zwei FRE	SUM	88321336	3735239	337509	84248588	30000
	MIN	5378	310	25	5043	3
	MAX	383238	354565	9186	19487	3
	AVG	8830	373	33	8424	3
	STDDEV	4113	3622	129	362	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	213378056	7626759	717573	205033724	70000
	MIN	17737	723	58	16956	7
	MAX	115325	62835	24499	27991	7
	AVG	21336	762	71	20503	7
	STDDEV	1374	638	278	458	0

		Summe	Reduction	Rendering	SQL-Ausführung	SQL
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	149689871	5422102	518632	143749137	50000
	MIN	26435	1034	85	25316	10
	MAX	54614	6742	4551	43321	10
	AVG	29936	1084	103	28749	10
	STDDEV	932	198	169	565	0
DELETE OBJECTS	SUM	11782485	340447	30825	11411213	3002
	MIN	0	0	0	0	0
	MAX	30804	4016	56	26732	7
	AVG	2356	68	6	2282	0
	STDDEV	2913	117	7	2789	0

G.4.4 Normierte Zeiten zum Überprüfen der Kardinalitäten

		Reduction	Rendering	SQL-Ausführung
CREATE NEW OBJECT	MIN	0,0%	0,0%	0,0%
	MAX	7,5%	3,9%	88,6%
	AVG	0,8%	0,4%	98,8%
CREATE NEW RELATIONSHIP	MIN	15,2%	1,0%	83,8%
	MAX	24,4%	30,1%	45,5%
	AVG	2,9%	0,3%	96,8%
▷ kein FRE	MIN	17,9%	1,2%	80,9%
	MAX	6,1%	0,5%	93,4%
	AVG	5,1%	0,4%	94,5%
▷ ein FRE	MIN	15,2%	1,0%	83,8%
	MAX	24,4%	30,1%	45,5%
	AVG	2,8%	0,3%	96,8%
CREATE NEW OBJECT + RELATIONSHIP	MIN	5,8%	0,5%	93,8%
	MAX	83,9%	5,8%	10,3%
	AVG	3,7%	0,3%	95,9%
▷ 1 Objekt + 1 kein FRE	MIN	4,2%	0,4%	95,4%
	MAX	52,9%	3,5%	43,6%
	AVG	3,8%	0,3%	95,9%
▷ 1 Objekt + 1 zwei FRE	MIN	5,8%	0,5%	93,8%
	MAX	92,5%	2,4%	5,1%
	AVG	4,2%	0,4%	95,4%
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	MIN	4,1%	0,3%	95,6%
	MAX	54,5%	21,2%	24,3%
	AVG	3,6%	0,3%	96,1%
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	MIN	3,9%	0,3%	95,8%
	MAX	12,3%	8,3%	79,3%
	AVG	3,6%	0,3%	96,0%
DELETE OBJECTS	MIN	0,0%	0,0%	0,0%
	MAX	13,0%	0,2%	86,8%
	AVG	2,9%	0,3%	96,9%

Anhang H Diagramme zur Leistungsuntersuchung

H.1 Verhältnisse der gemessenen Zeiten

Abbildung H.1 Zusammensetzung der Gesamtausführungszeit: Alle Befehle

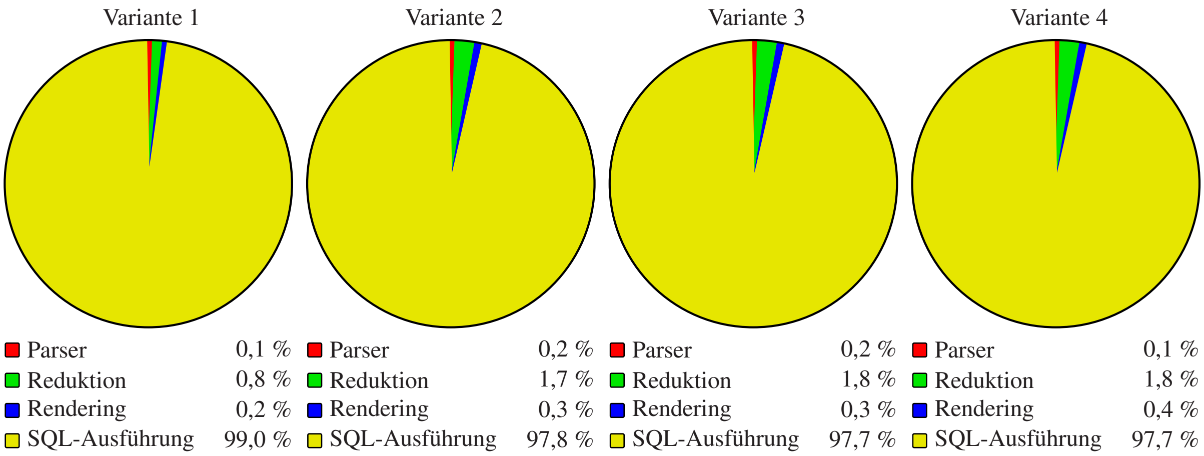


Abbildung H.2 Zusammensetzung der Gesamtausführungszeit: Update-Befehle

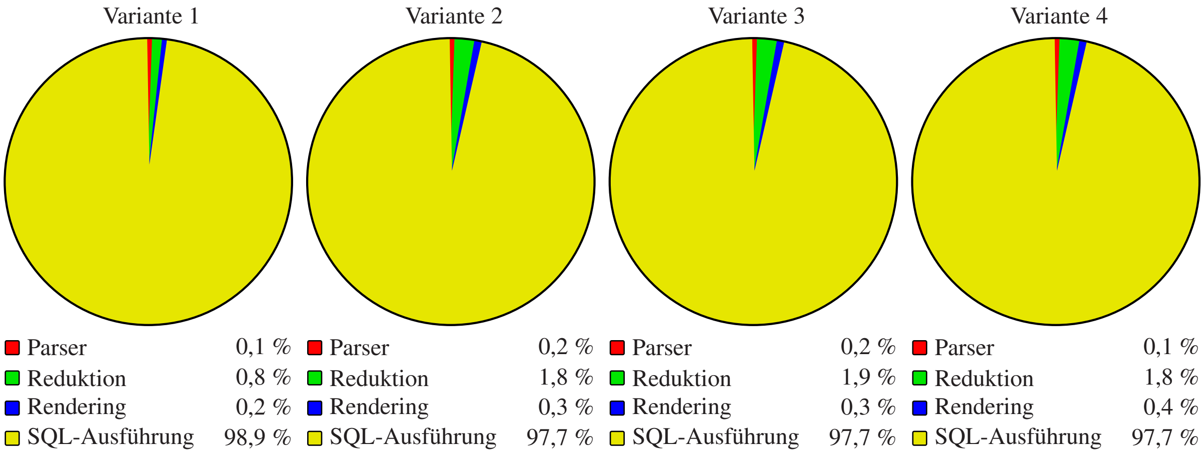


Abbildung H.3 Zusammensetzung der Gesamtausführungszeit: Select Befehle

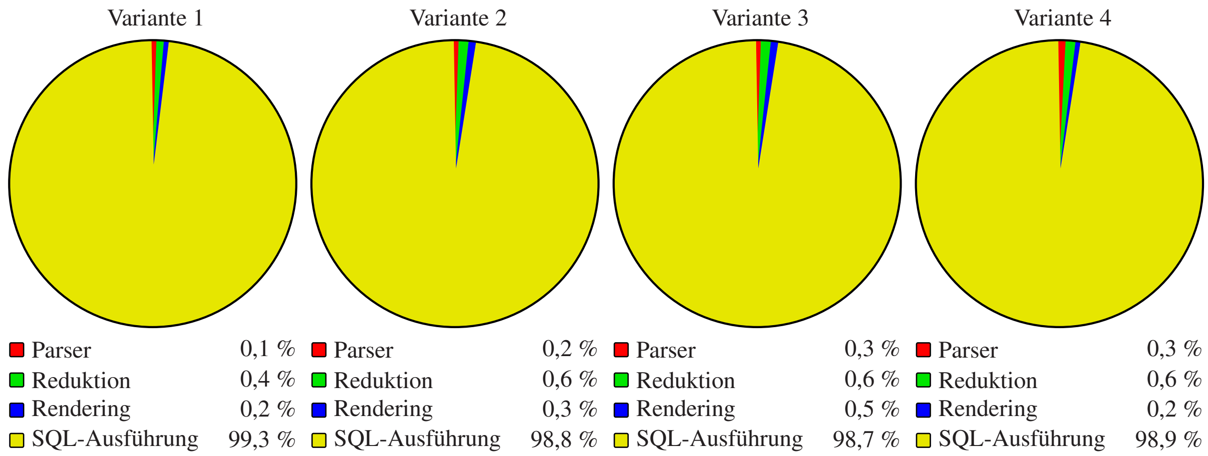


Abbildung H.4 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT

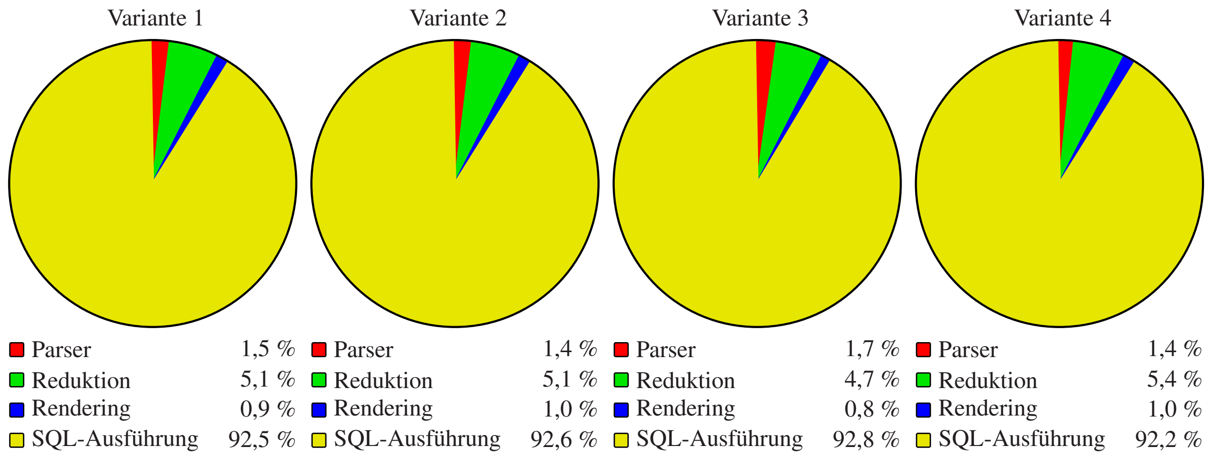


Abbildung H.5 Zusammensetzung der Gesamtausführungszeit: CREATE NEW RELATIONSHIP

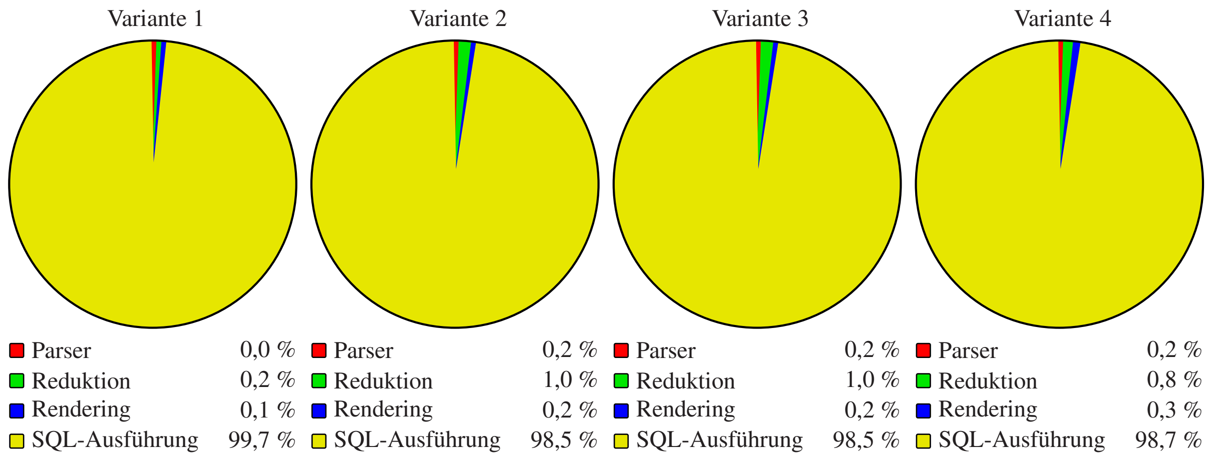


Abbildung H.6 Zusammensetzung der Gesamtausführungszeit: CREATE NEW RELATIONSHIP (kein FRE)

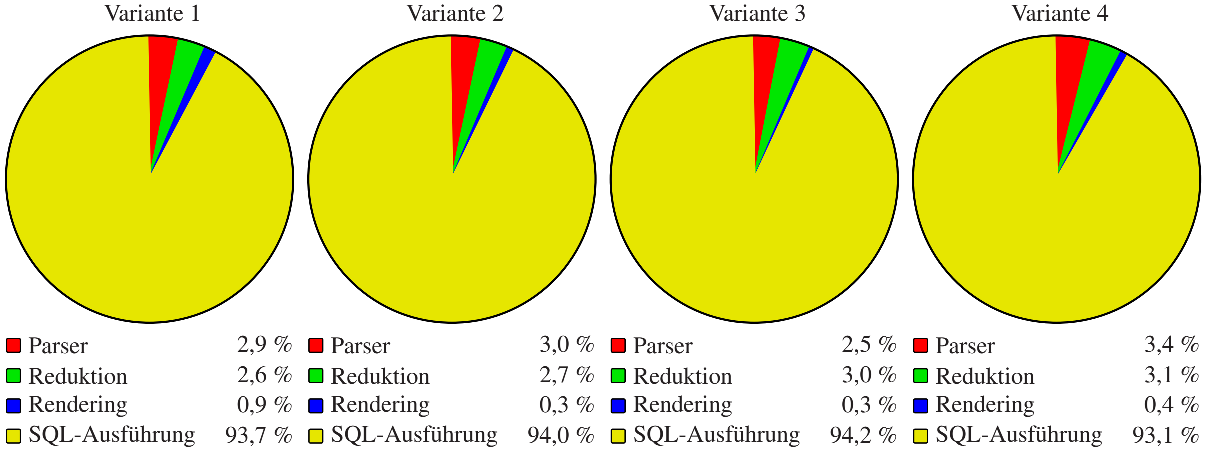


Abbildung H.7 Zusammensetzung der Gesamtausführungszeit: CREATE NEW RELATIONSHIP (ein FRE)

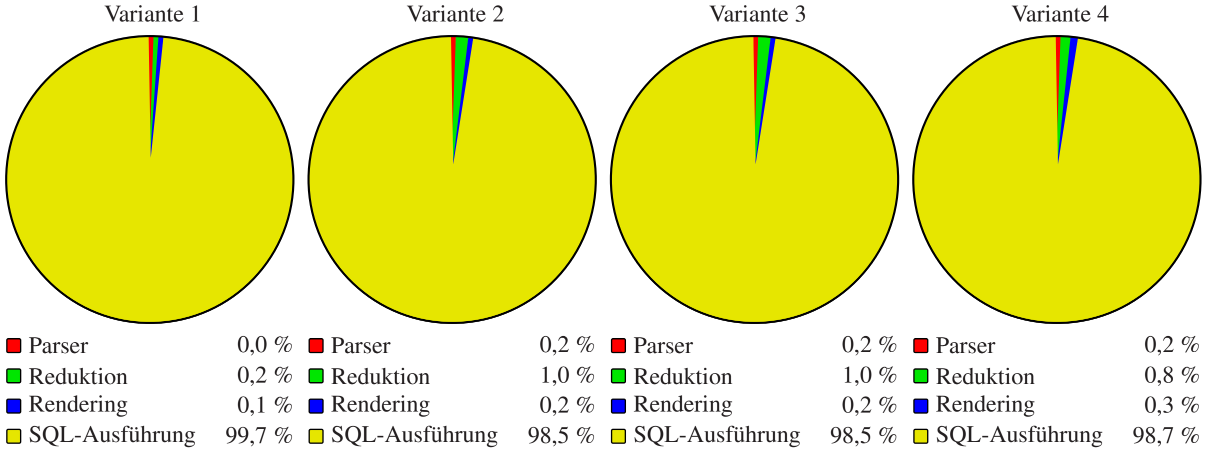


Abbildung H.8 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP

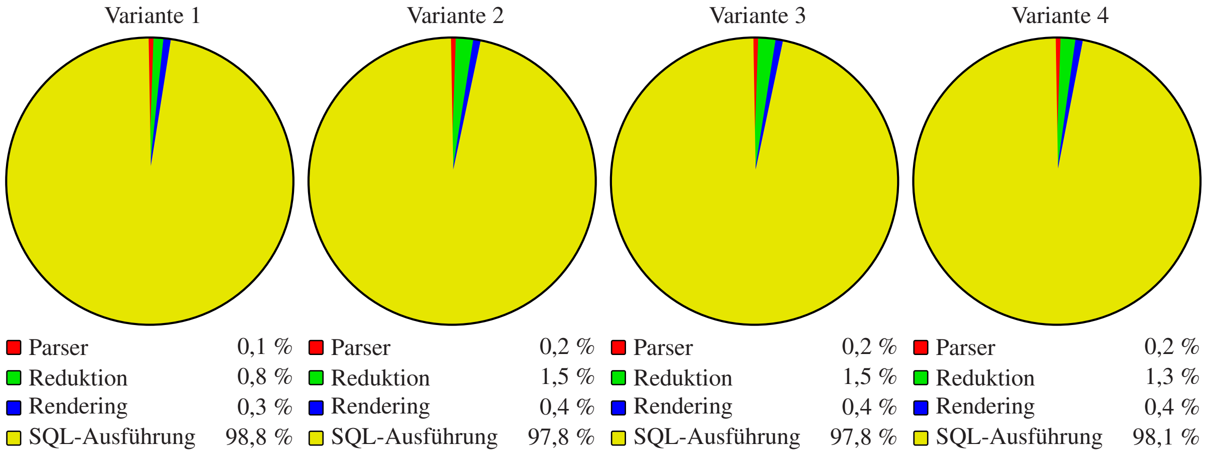


Abbildung H.9 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 kein FRE)

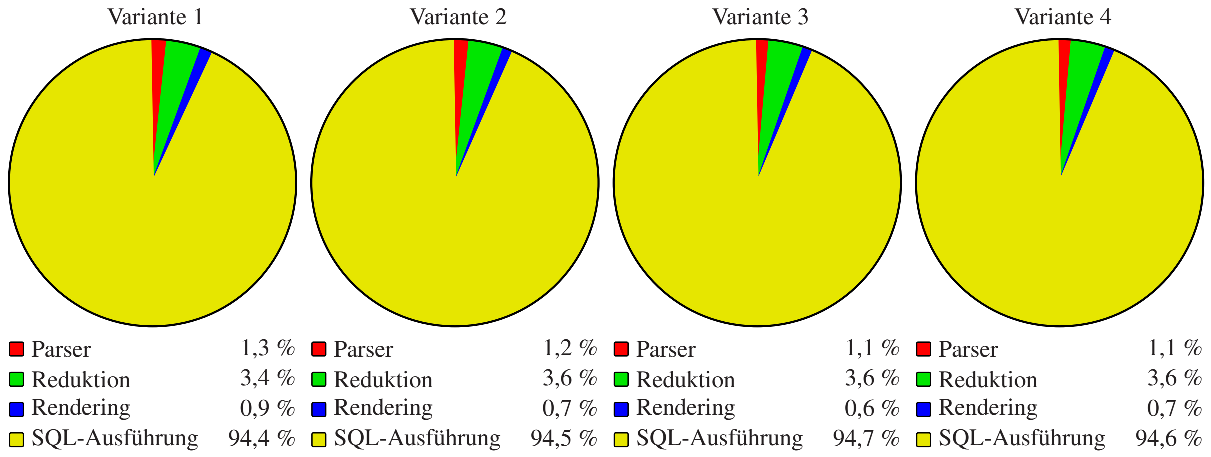


Abbildung H.10 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 zwei FRE)

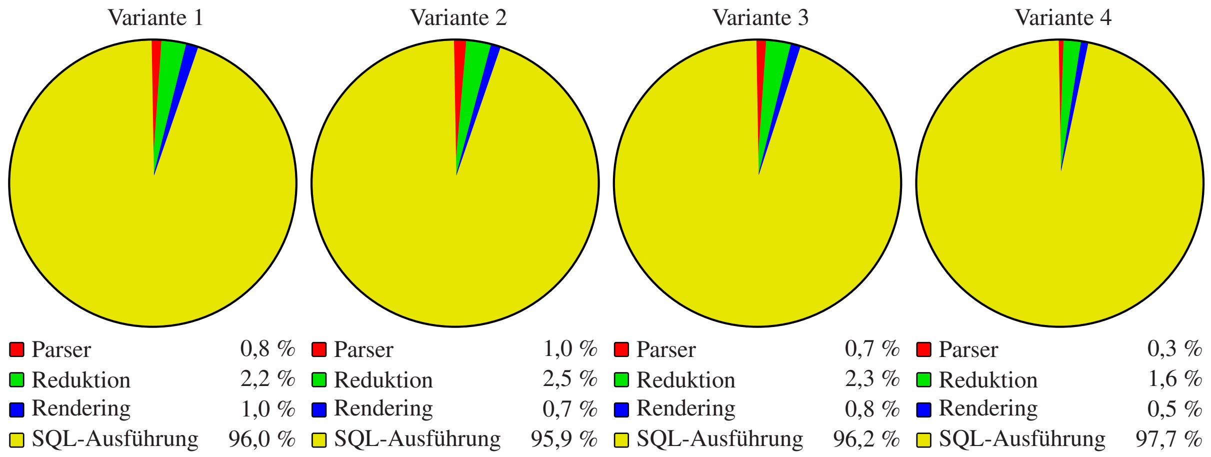


Abbildung H.11 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP (2 Objecte + 1 ein FRE + 1 zwei FRE)

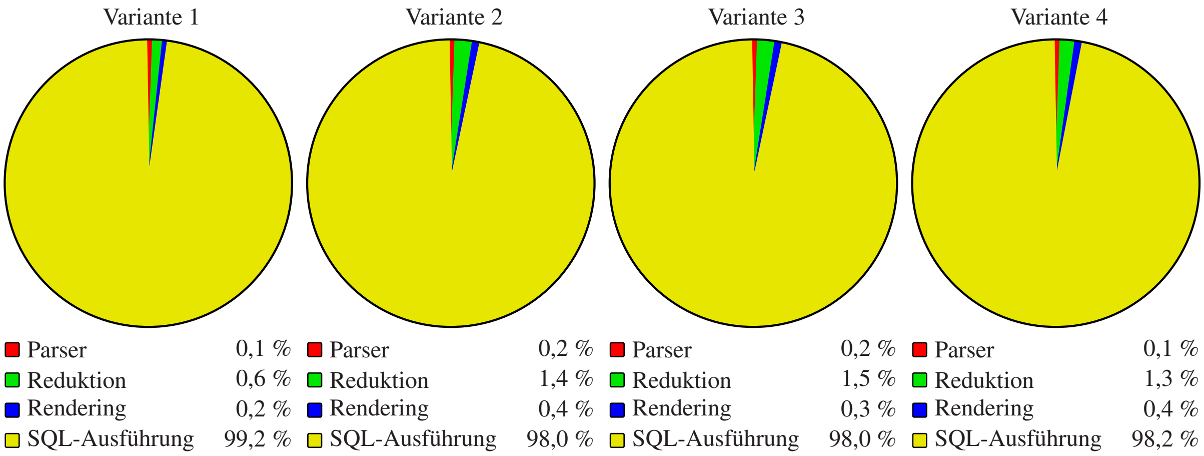


Abbildung H.12 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP (3 Objekte + 2 ein FRE + 1 zwei FRE)

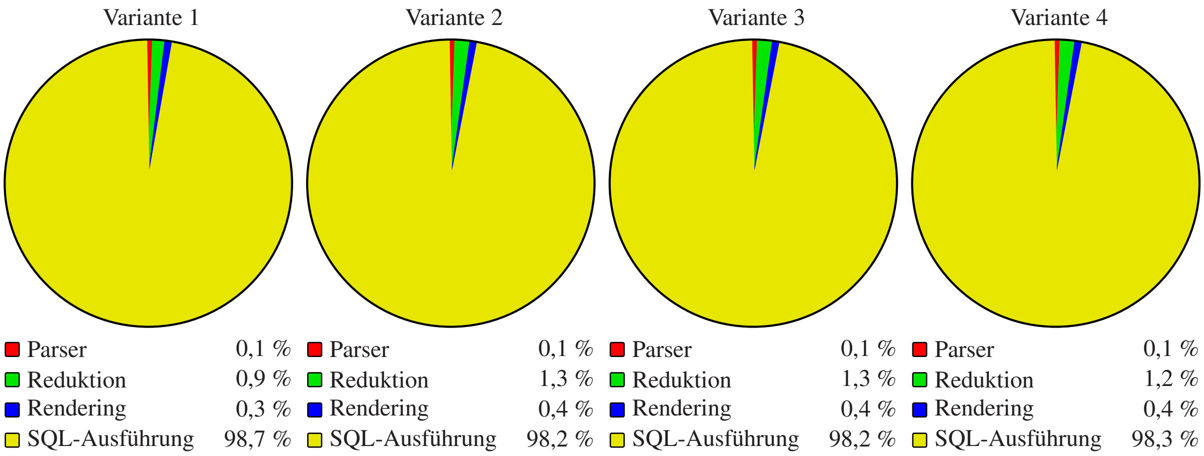


Abbildung H.13 Zusammensetzung der Gesamtausführungszeit: CREATE SUCCESSOR

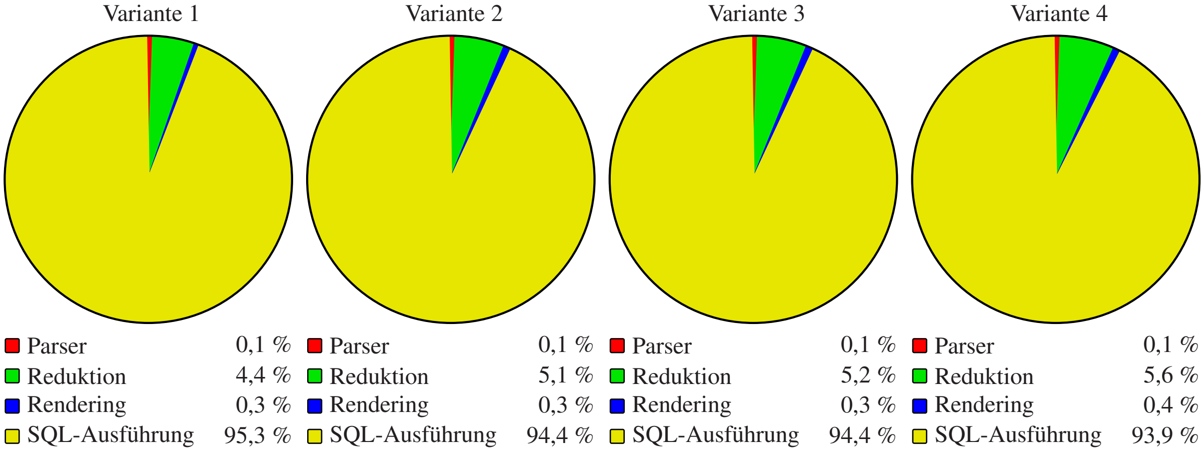


Abbildung H.14 Zusammensetzung der Gesamtausführungszeit: CREATE SUCCESSOR außerhalb von Workspaces

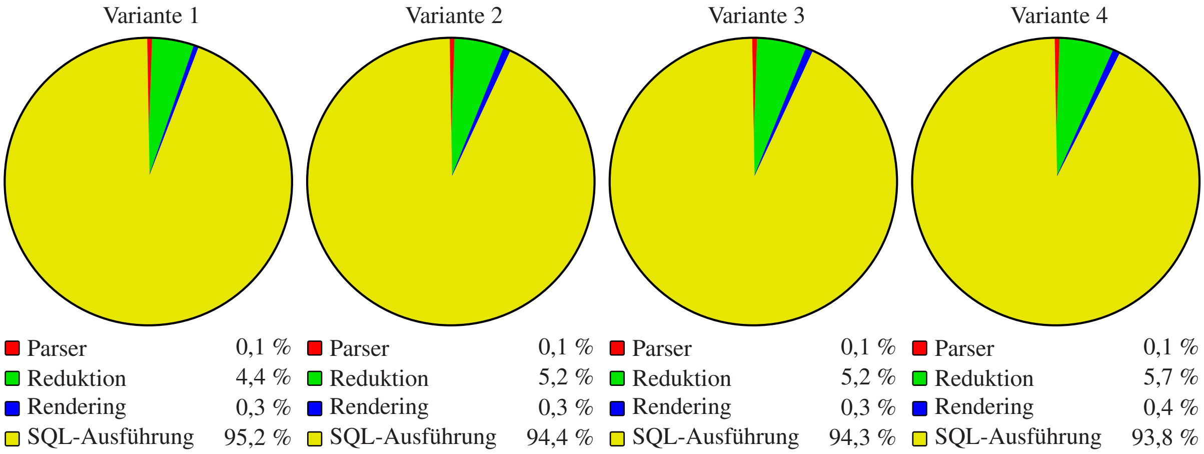


Abbildung H.15 Zusammensetzung der Gesamtausführungszeit: CREATE SUCCESSOR innerhalb von Workspaces

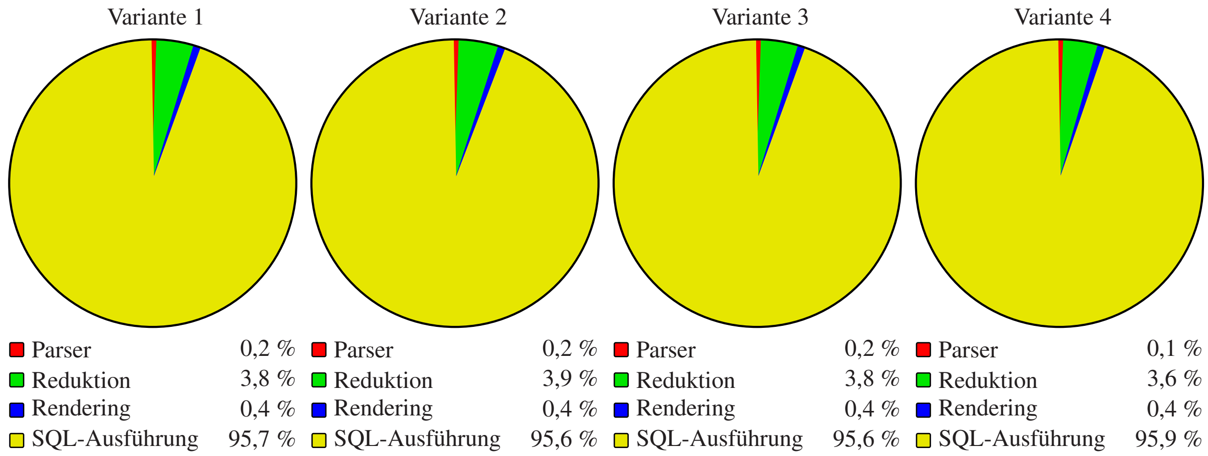


Abbildung H.16 Zusammensetzung der Gesamtausführungszeit: COPY OBJECTS

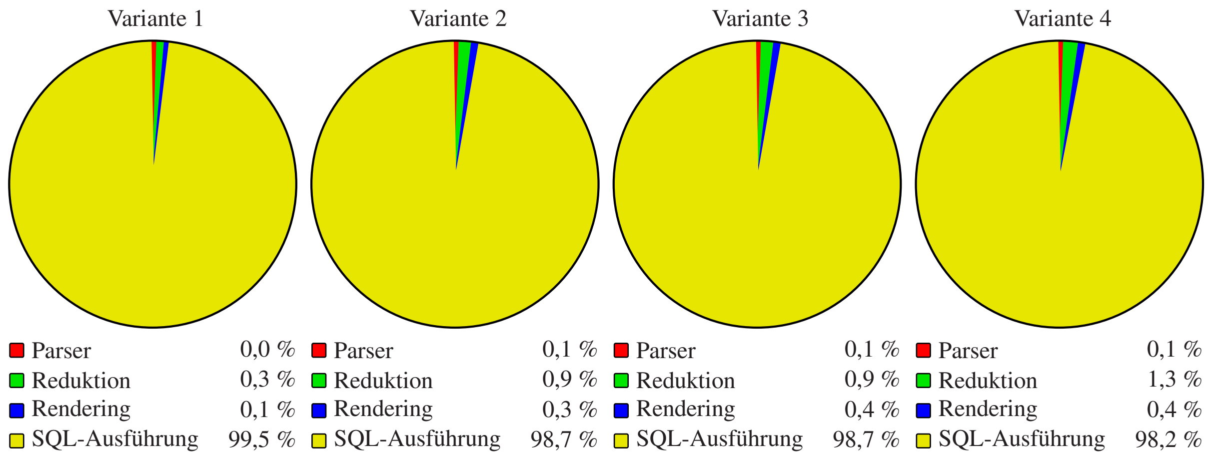


Abbildung H.17 Zusammensetzung der Gesamtausführungszeit: DELETE OBJECTS

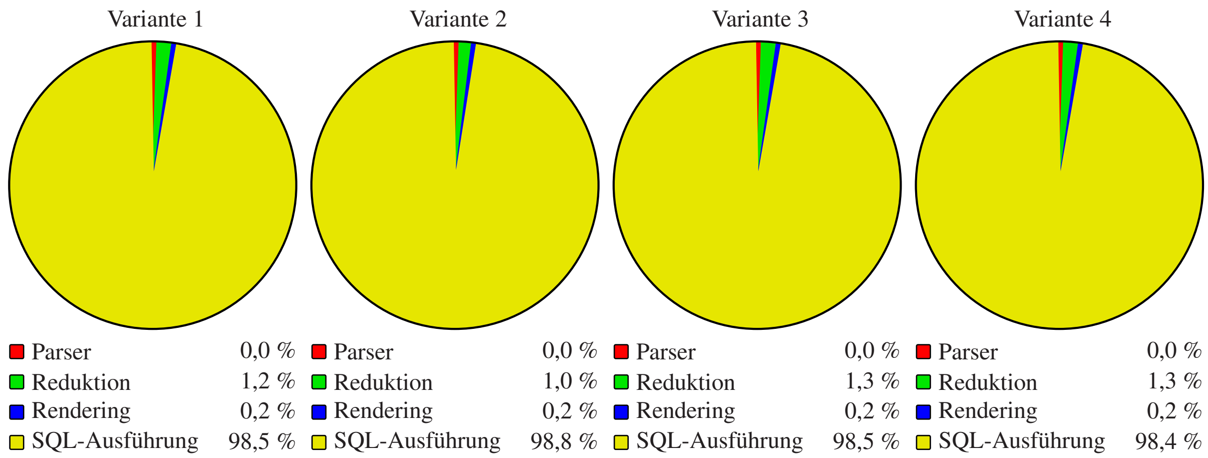


Abbildung H.18 Zusammensetzung der Gesamtausführungszeit: MERGE OBJECTS

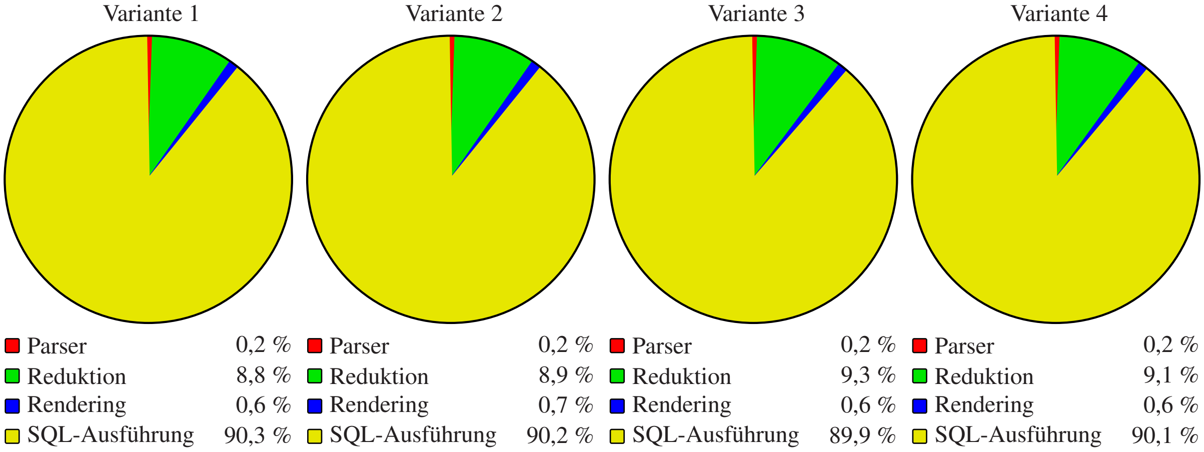


Abbildung H.19 Zusammensetzung der Gesamtausführungszeit: FREEZE

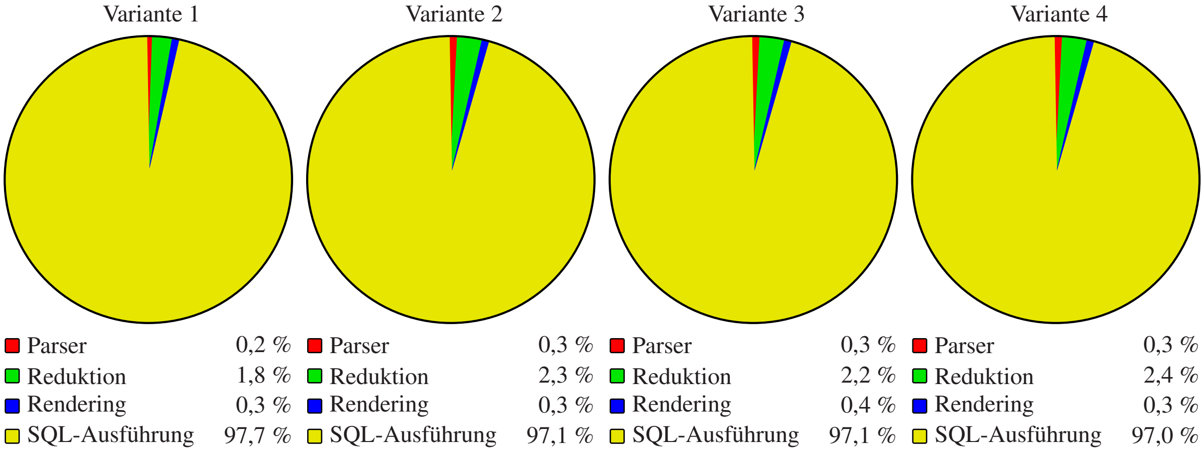


Abbildung H.20 Zusammensetzung der Gesamtausführungszeit: FREEZE außerhalb von Workspaces

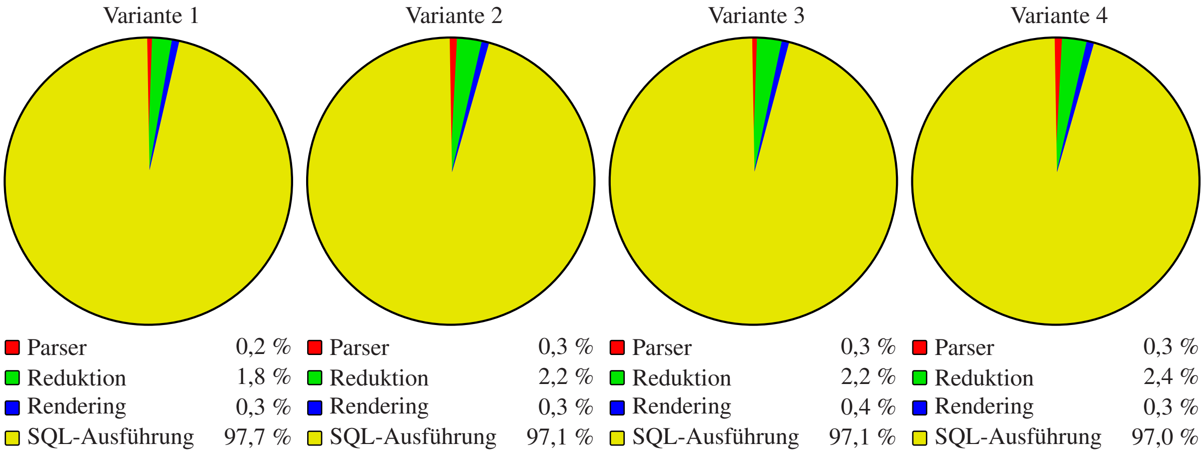


Abbildung H.21 Zusammensetzung der Gesamtausführungszeit: FREEZE innerhalb von Workspaces

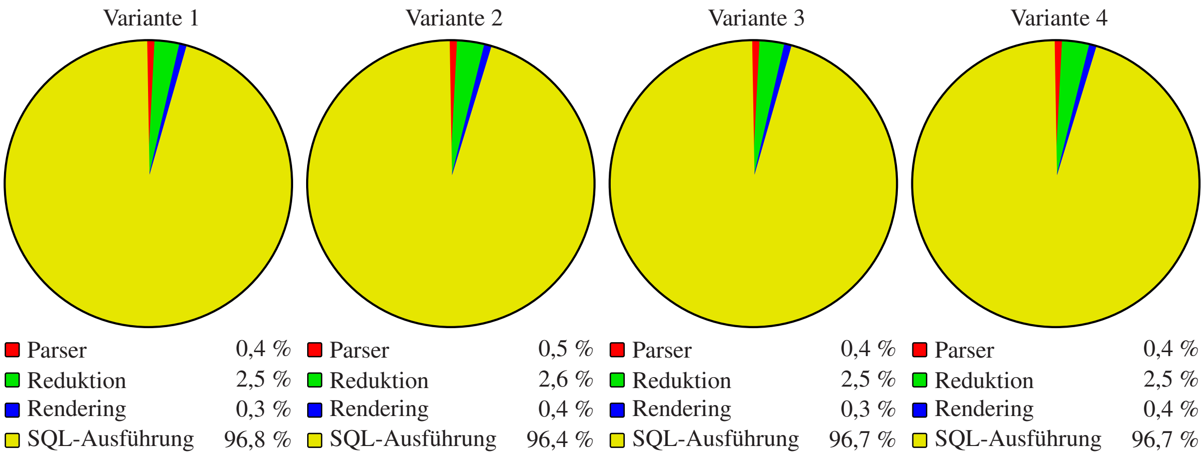


Abbildung H.22 Zusammensetzung der Gesamtausführungszeit: ATTACH und DETACH

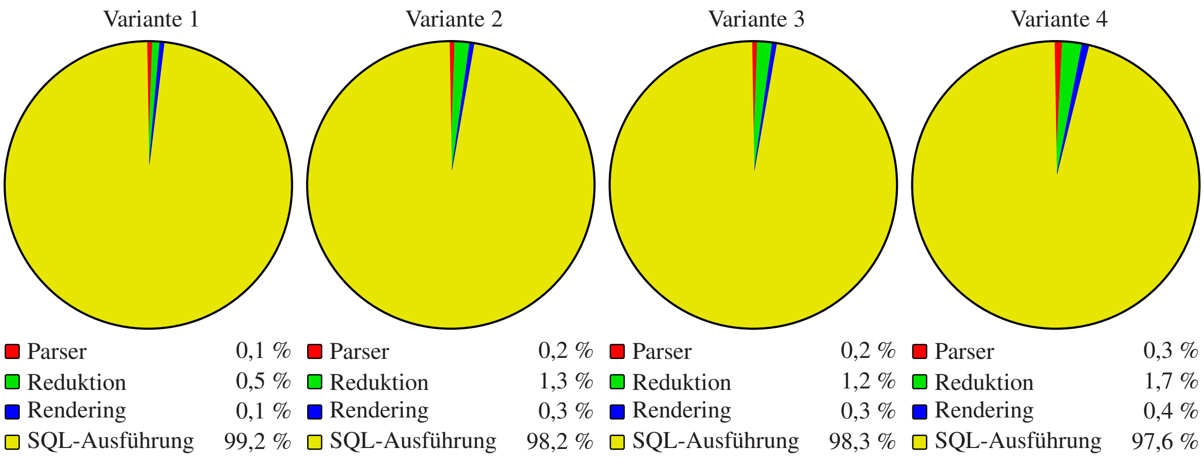


Abbildung H.23 Zusammensetzung der Gesamtausführungszeit: ATTACH

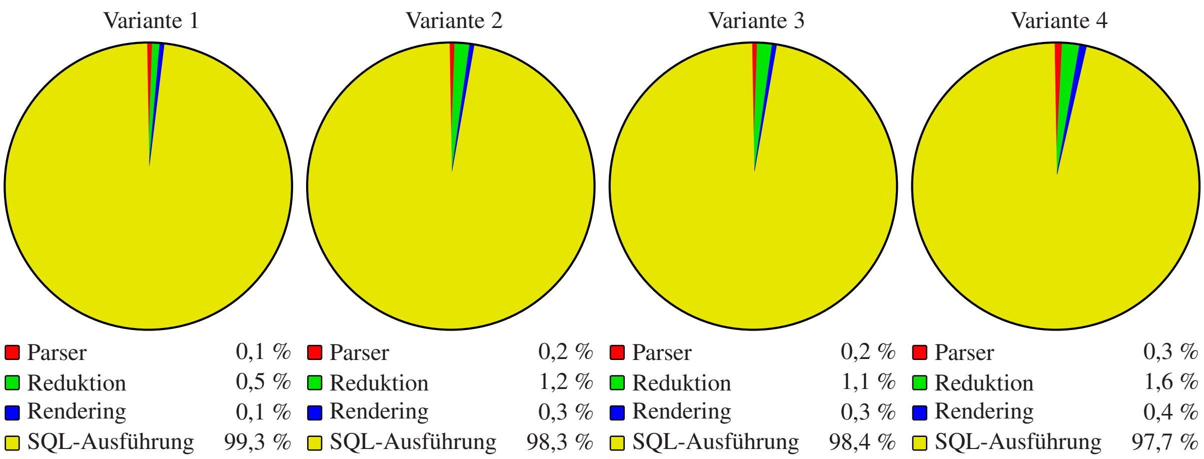


Abbildung H.24 Zusammensetzung der Gesamtausführungszeit: DETACH

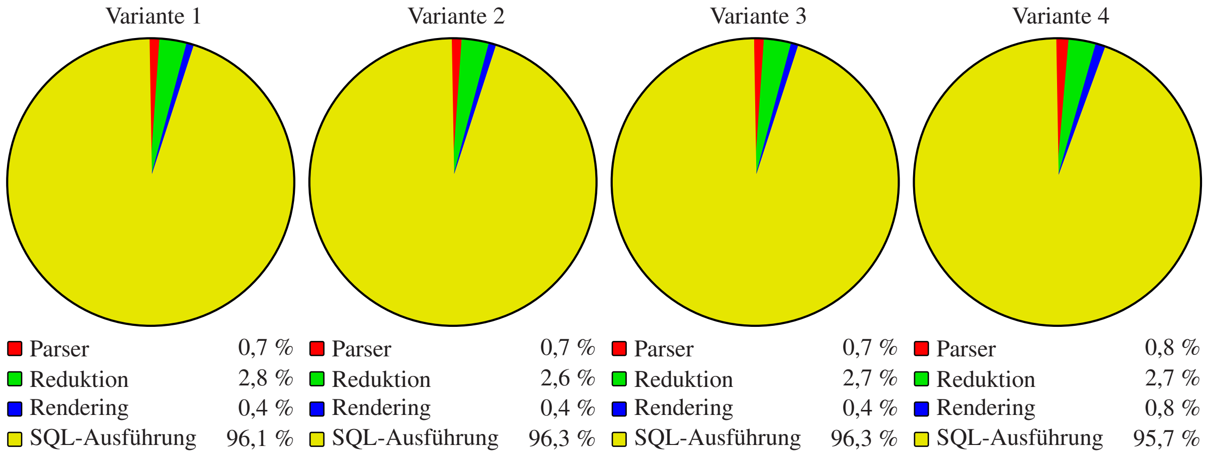


Abbildung H.25 Zusammensetzung der Gesamtausführungszeit: GET

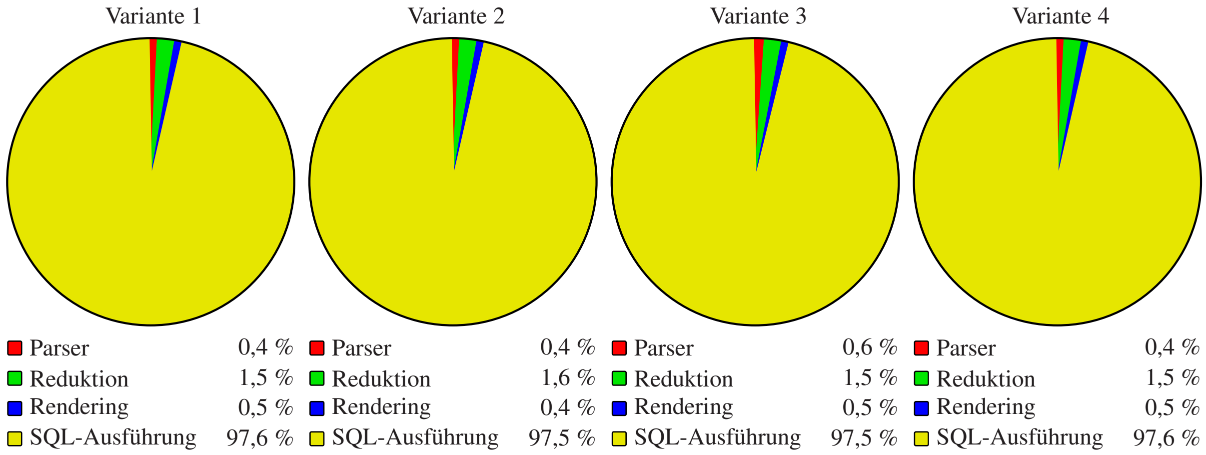


Abbildung H.26 Zusammensetzung der Gesamtausführungszeit: GET ROOT

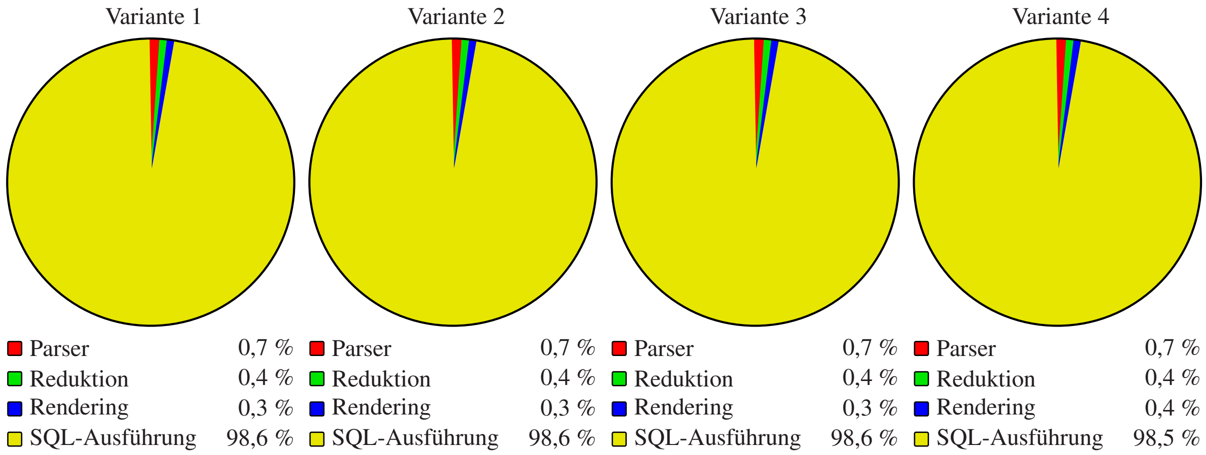


Abbildung H.27 Zusammensetzung der Gesamtausführungszeit: GET ALTERNATIVES

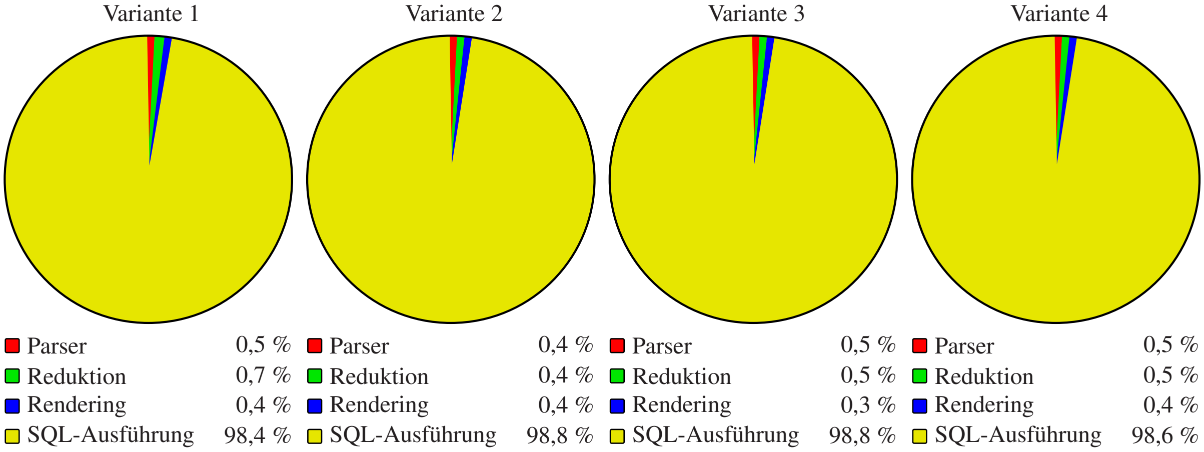


Abbildung H.28 Zusammensetzung der Gesamtausführungszeit: GET SUCCESSORS

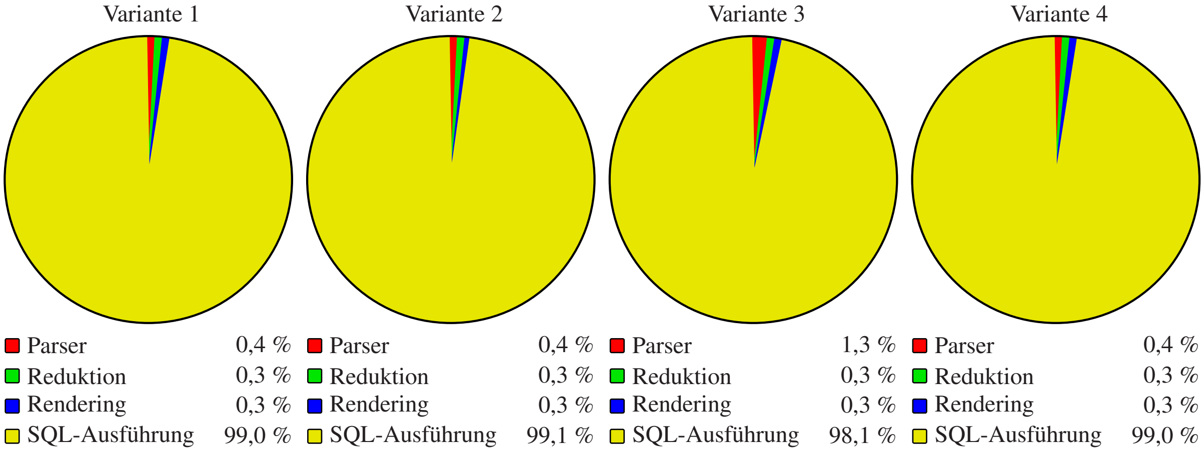


Abbildung H.29 Zusammensetzung der Gesamtausführungszeit: GET PREDECESSOR

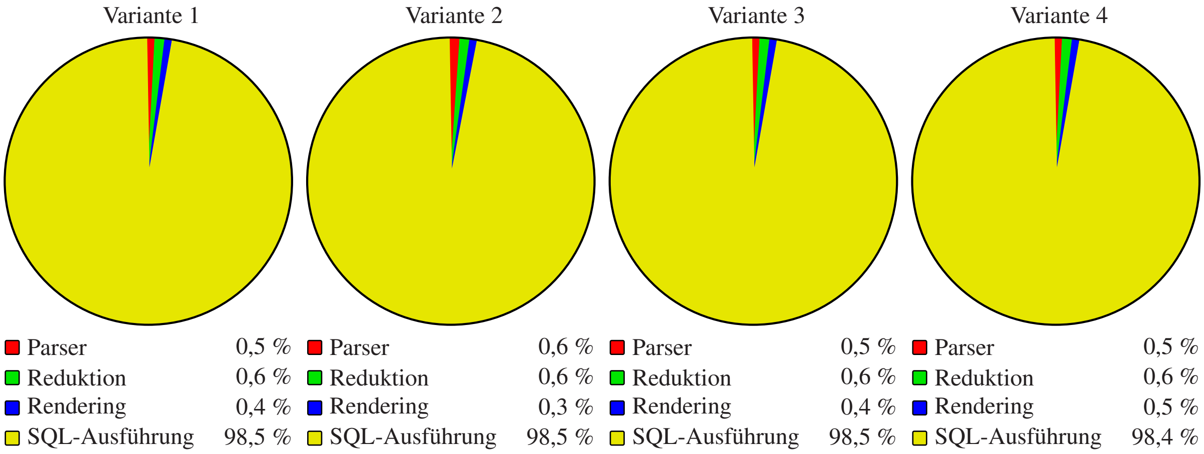


Abbildung H.30 Zusammensetzung der Gesamtausführungszeit: GET BASEVERSION

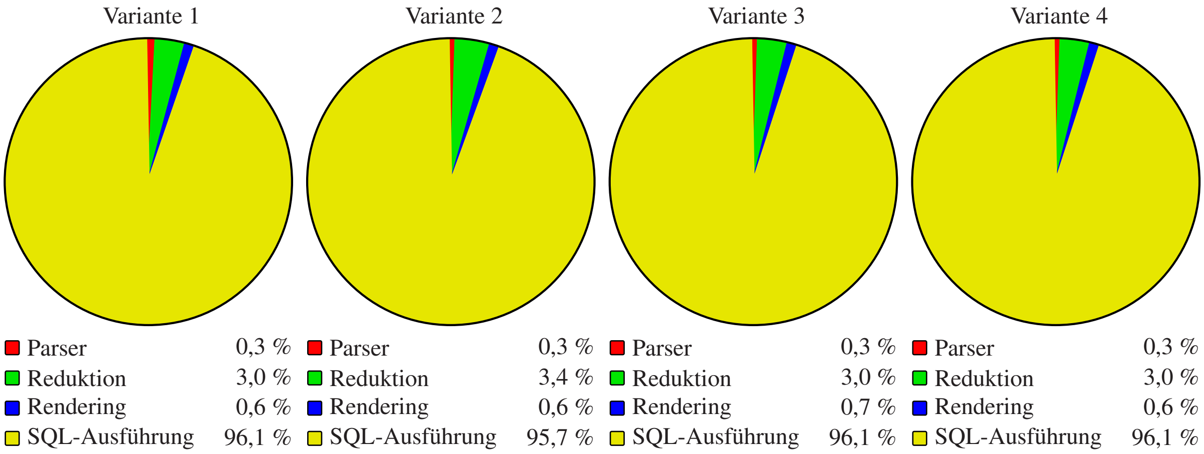


Abbildung H.31 Zusammensetzung der Gesamtausführungszeit: GET CVC

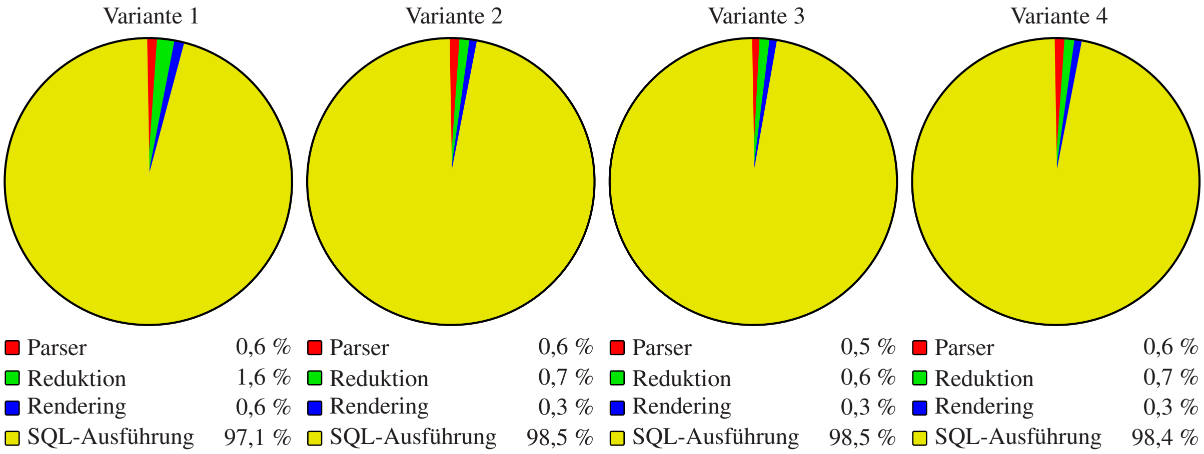


Abbildung H.32 Zusammensetzung der Gesamtausführungszeit: PIN

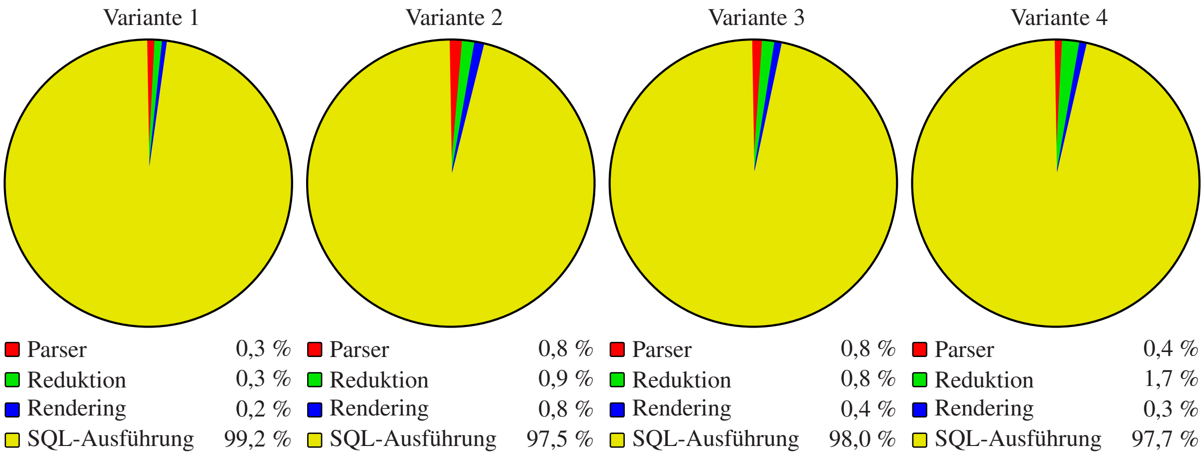


Abbildung H.33 Zusammensetzung der Gesamtausführungszeit: SELECT

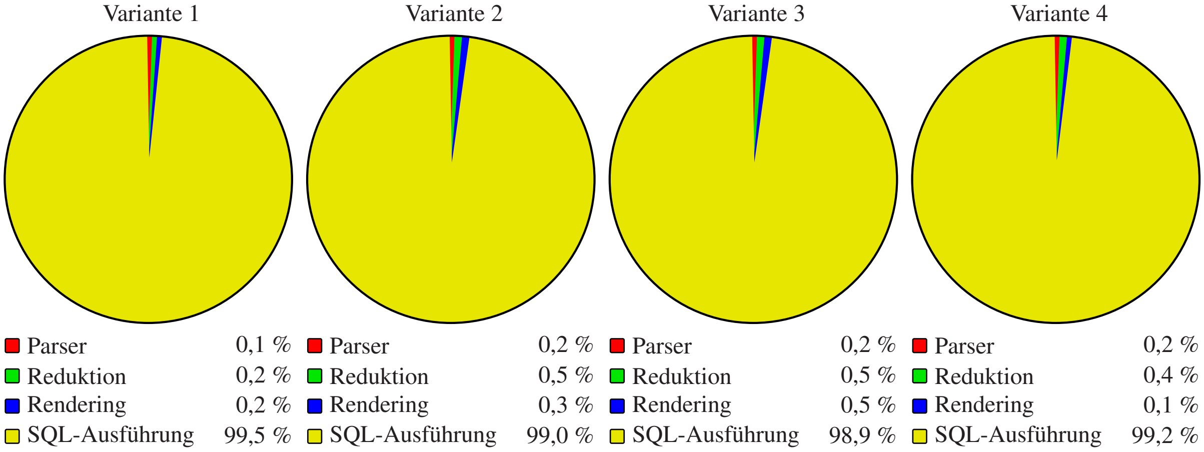


Abbildung H.34 Zusammensetzung der Gesamtausführungszeit: SELECT außerhalb von Workspaces

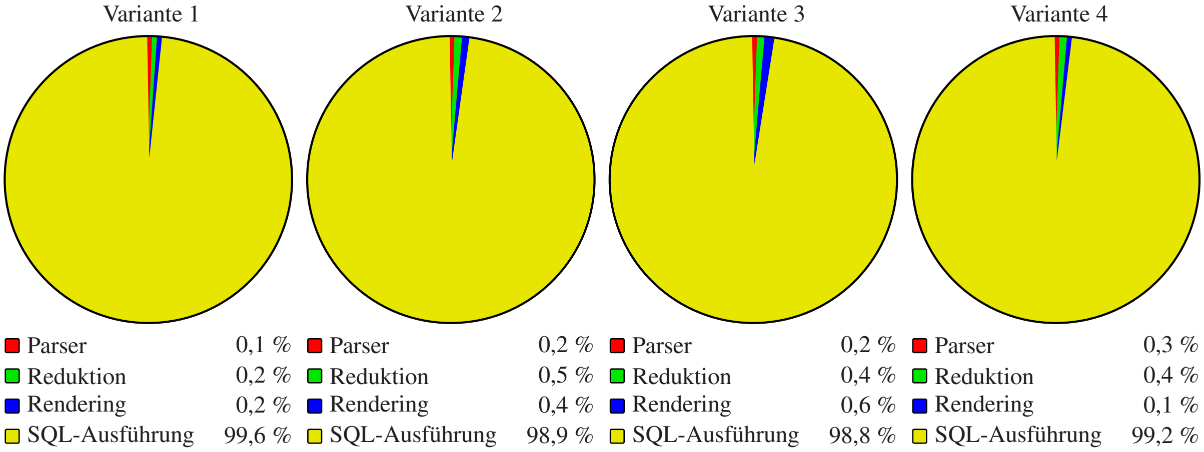
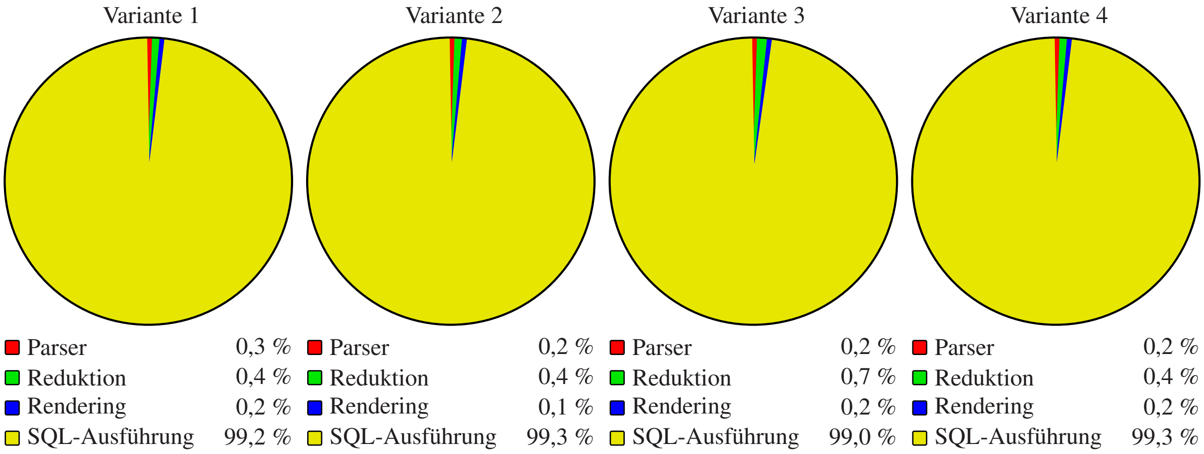


Abbildung H.35 Zusammensetzung der Gesamtausführungszeit: SELECT innerhalb von Workspaces



H.1.1 Vergleich der gemessenen Zeiten

Abbildung H.36 Zeitvergleich: Summe aller Befehle

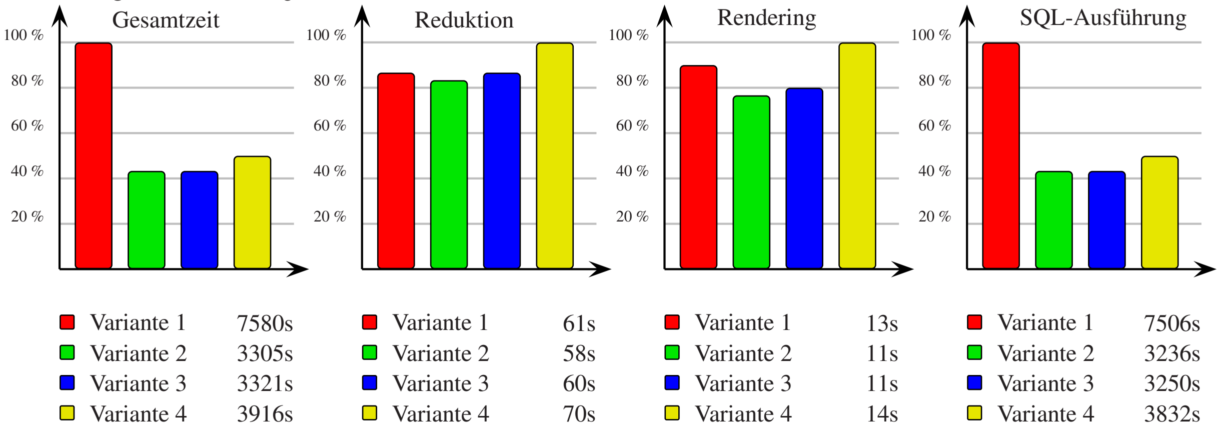


Abbildung H.37 Zeitvergleich: CREATE NEW OBJECT

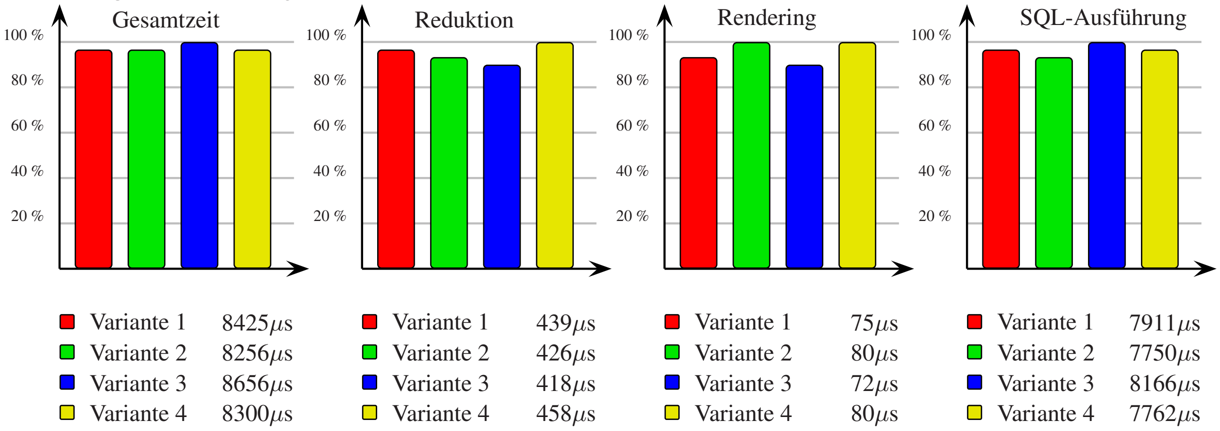


Abbildung H.38 Zeitvergleich: CREATE NEW RELATIONSHIP

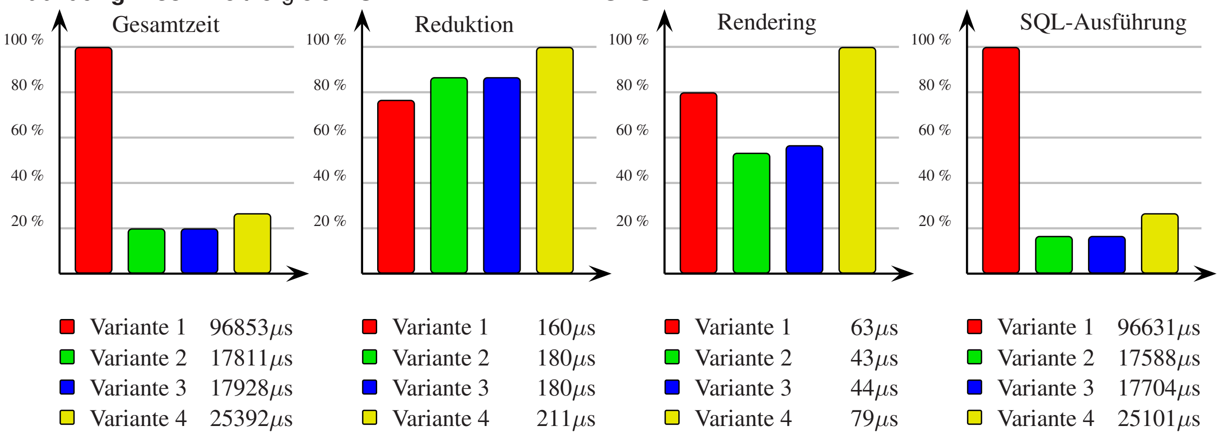


Abbildung H.39 Zeitvergleich: CREATE NEW RELATIONSHIP (kein FRE)

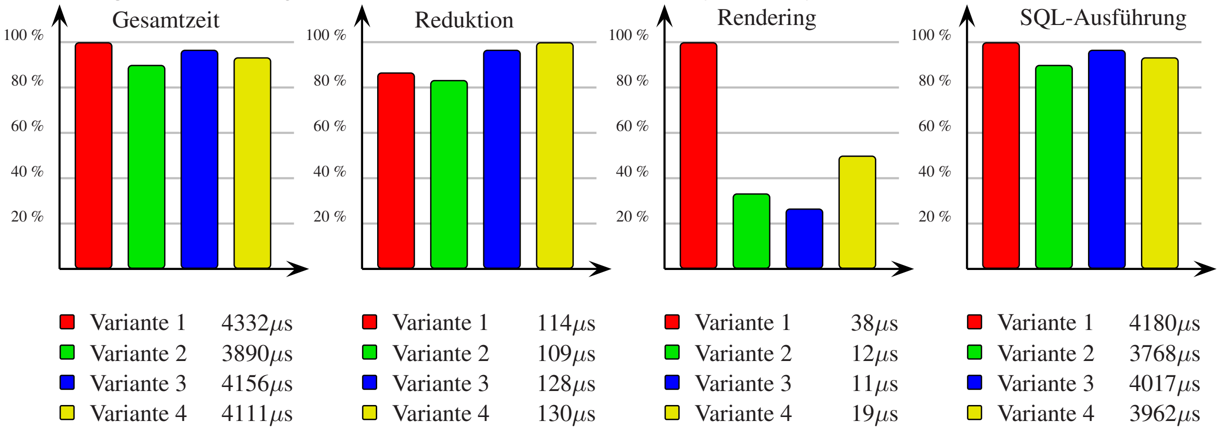


Abbildung H.40 Zeitvergleich: CREATE NEW RELATIONSHIP (ein FRE)

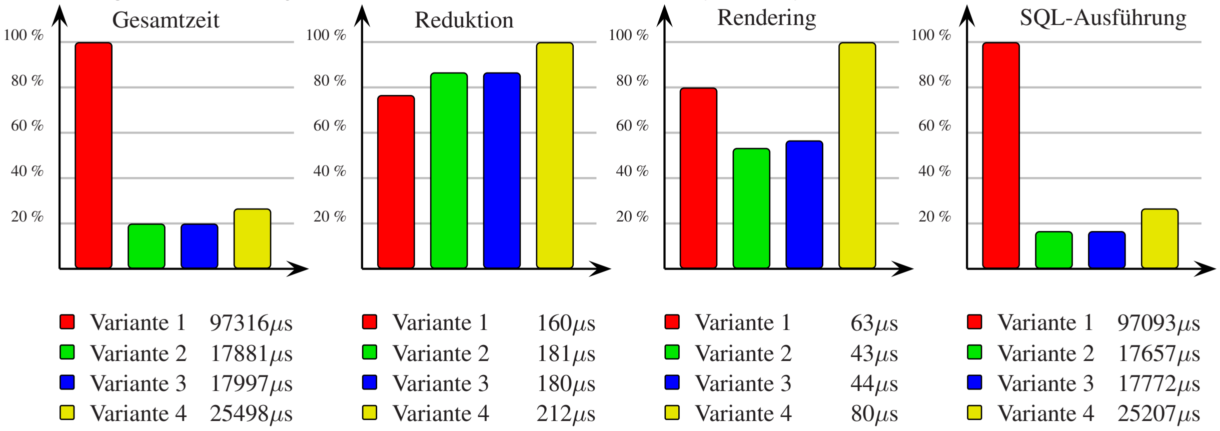


Abbildung H.41 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP

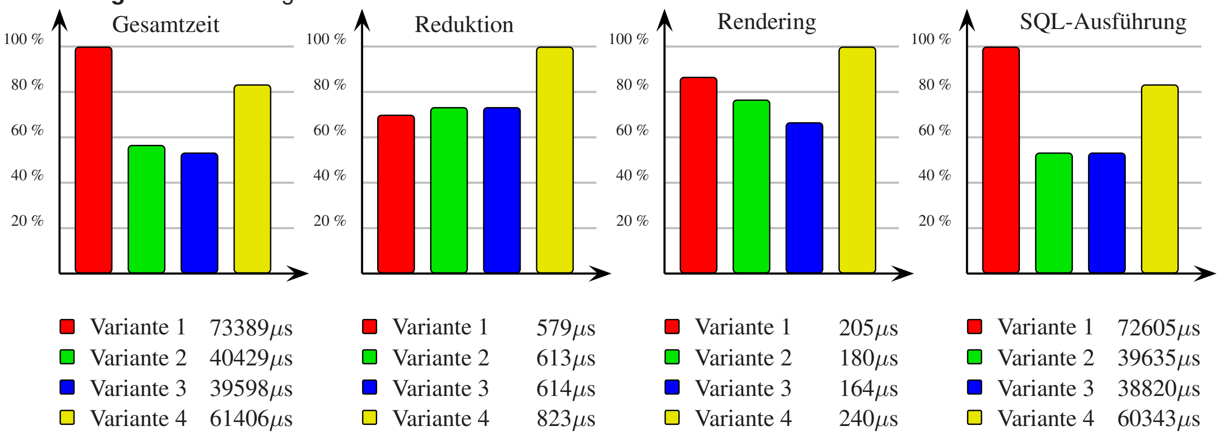


Abbildung H.42 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 kein FRE)

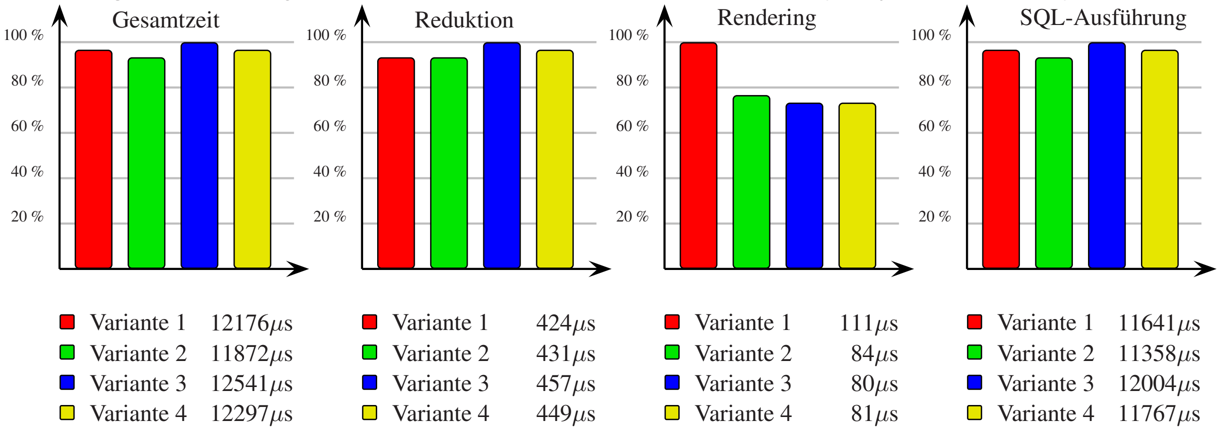


Abbildung H.43 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 zwei FRE)

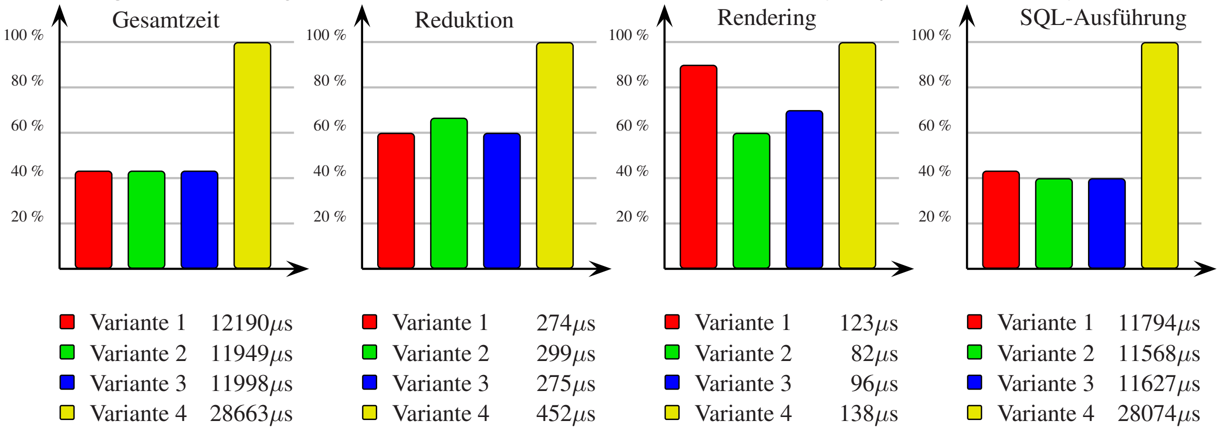


Abbildung H.44 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (2 Objecte + 1 ein FRE + 1 zwei FRE)

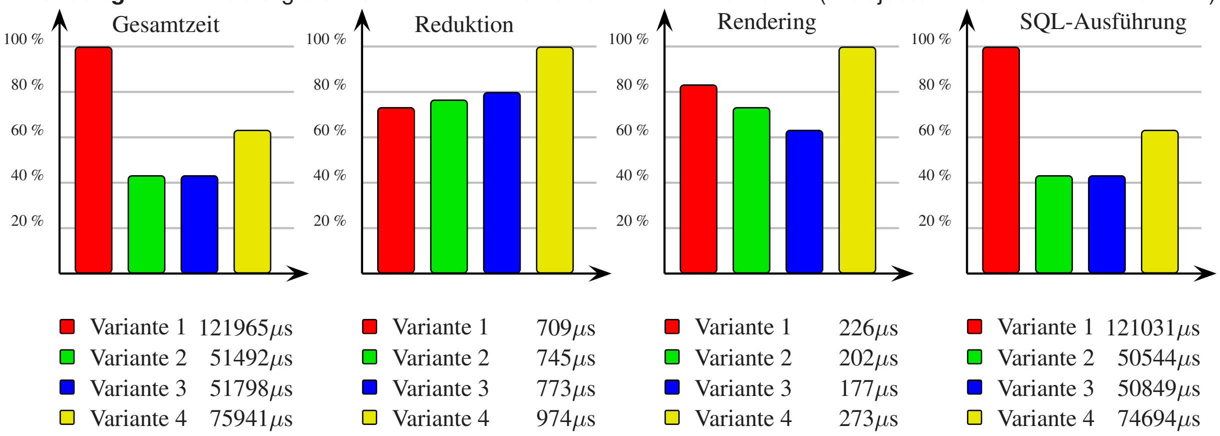


Abbildung H.45 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (3 Objekte + 2 ein FRE + 1 zwei FRE)

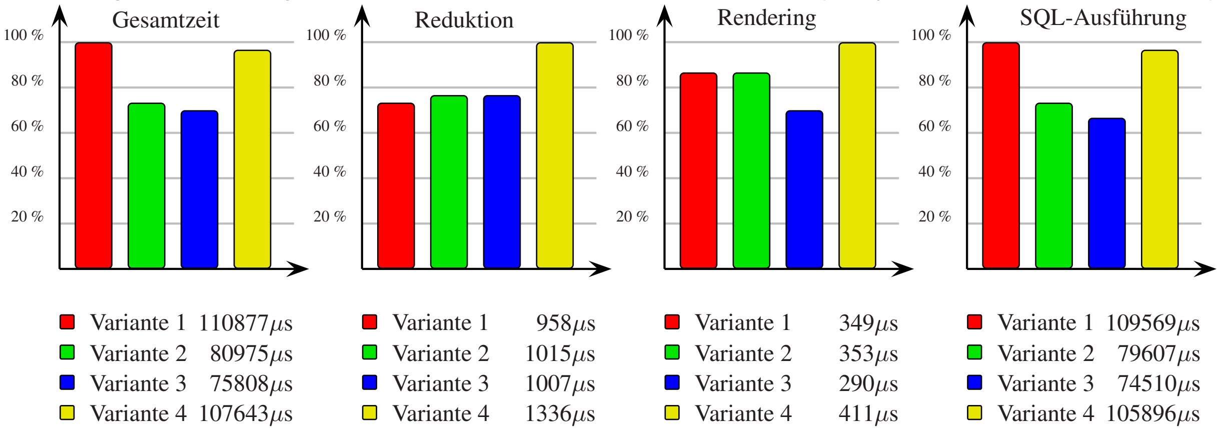


Abbildung H.46 Zeitvergleich: CREATE SUCCESSOR

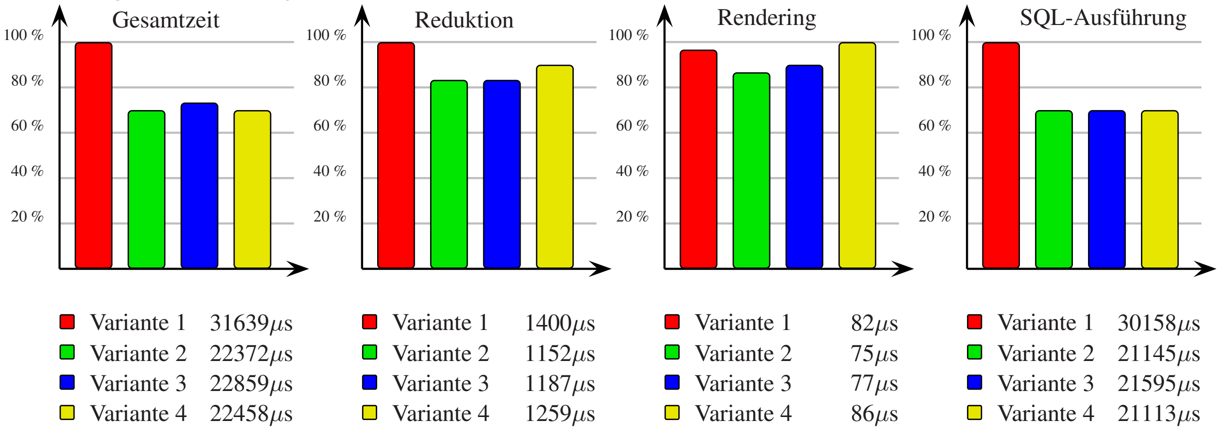


Abbildung H.47 Zeitvergleich: CREATE SUCCESSOR außerhalb von Workspaces

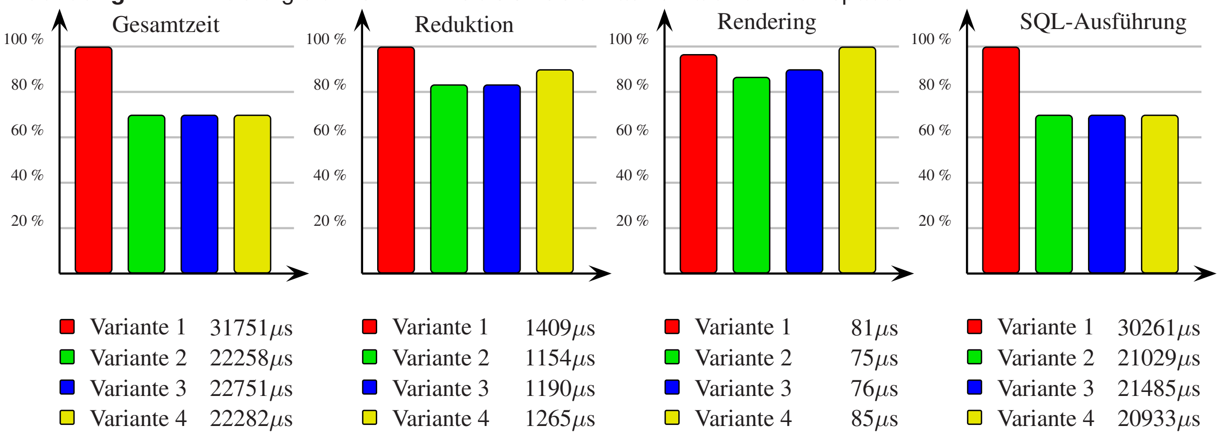


Abbildung H.48 Zeitvergleich: CREATE SUCCESSOR innerhalb von Workspaces

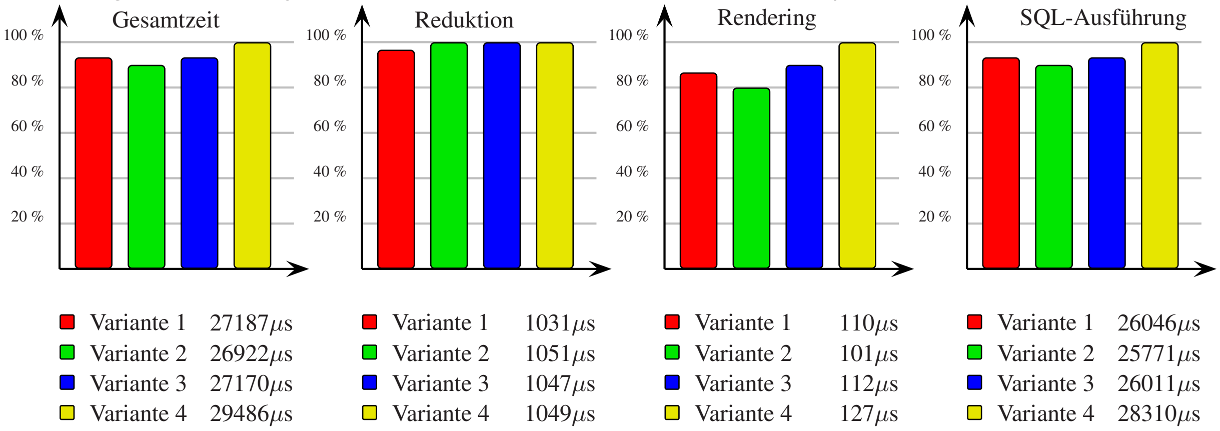


Abbildung H.49 Zeitvergleich: COPY OBJECTS

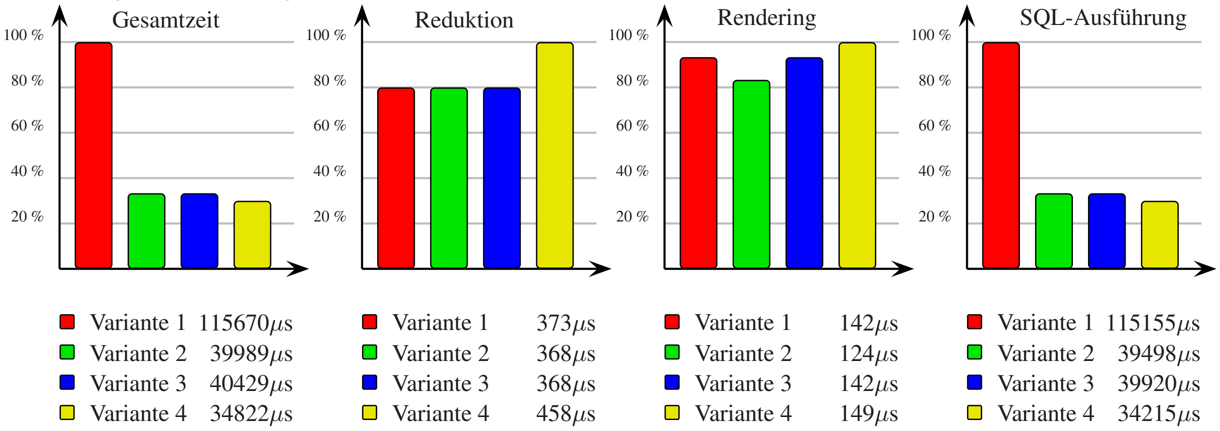


Abbildung H.50 Zeitvergleich: DELETE OBJECTS

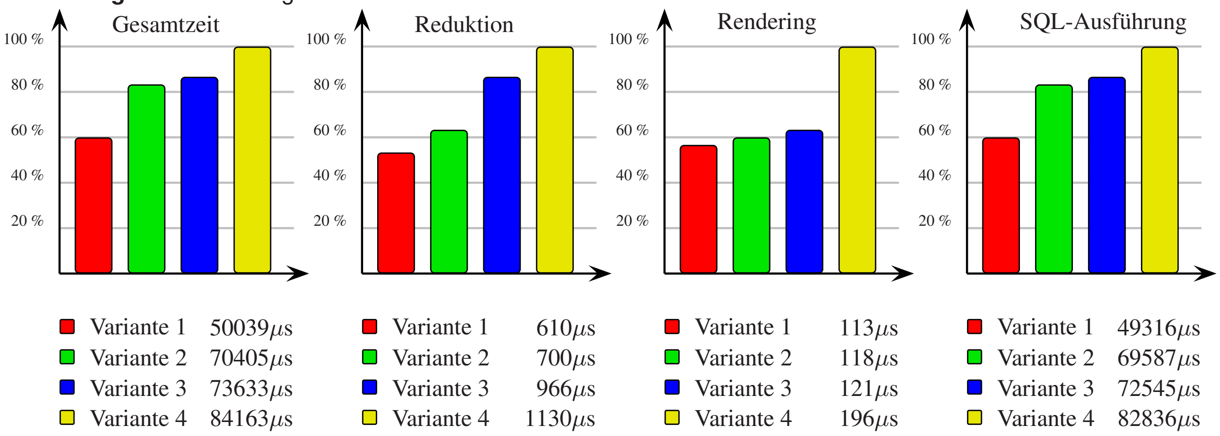


Abbildung H.51 Zeitvergleich: MERGE OBJECTS

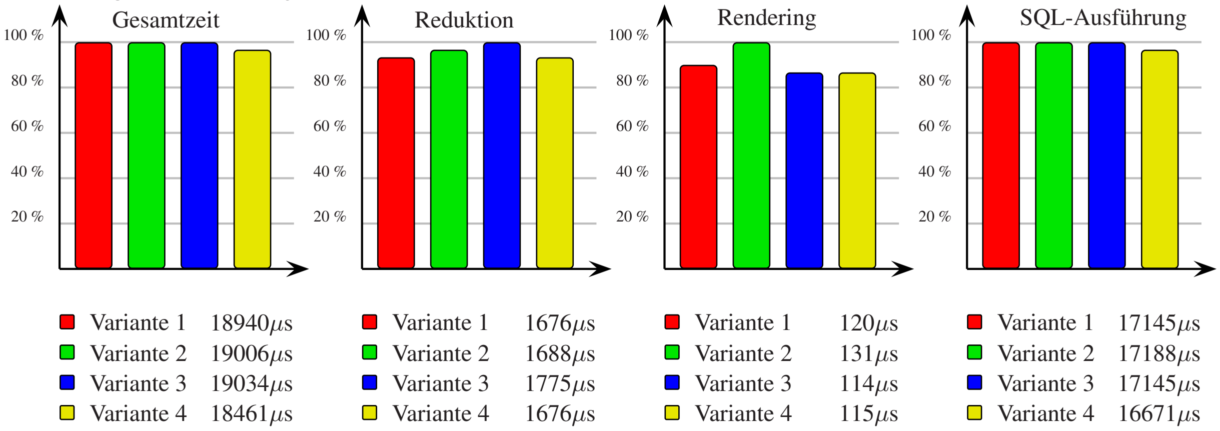


Abbildung H.52 Zeitvergleich: FREEZE

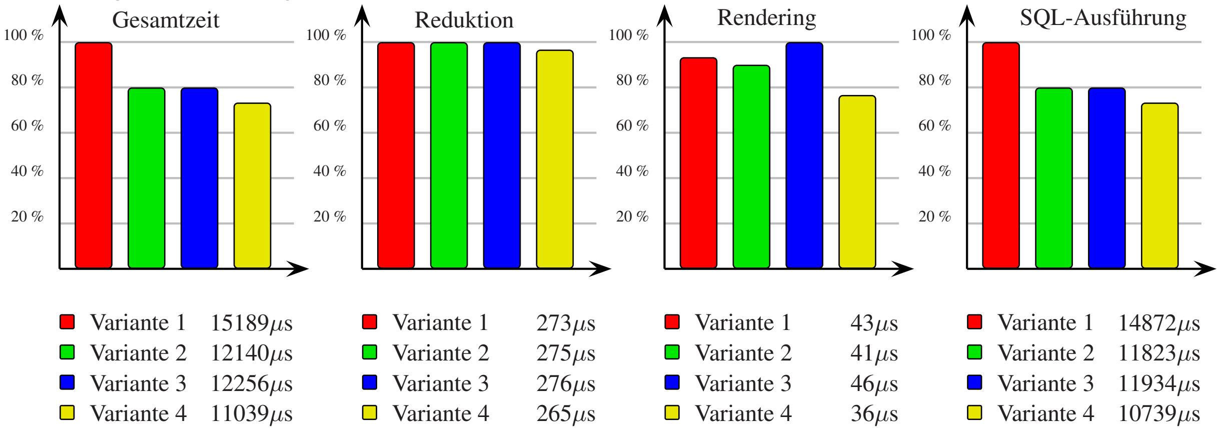


Abbildung H.53 Zeitvergleich: FREEZE außerhalb von Workspaces

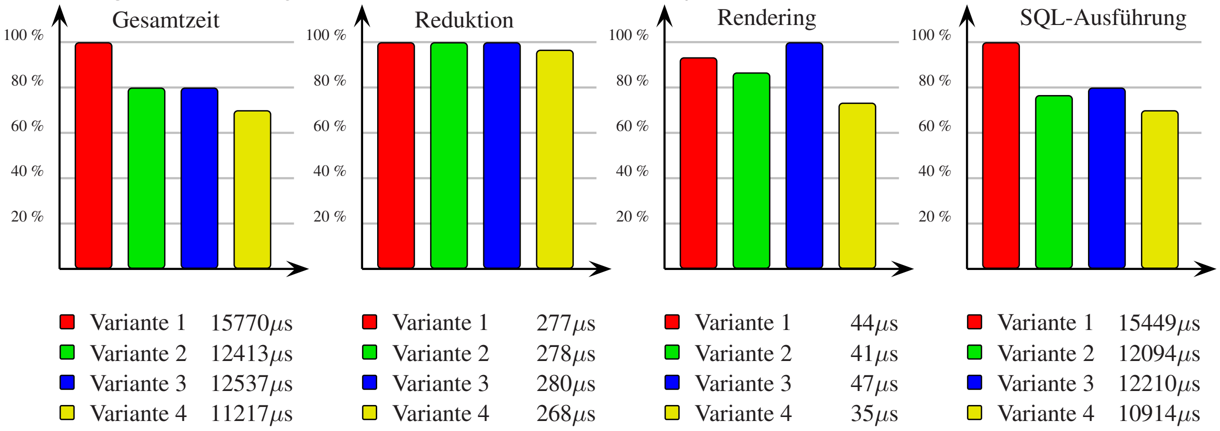


Abbildung H.54 Zeitvergleich: FREEZE innerhalb von Workspaces

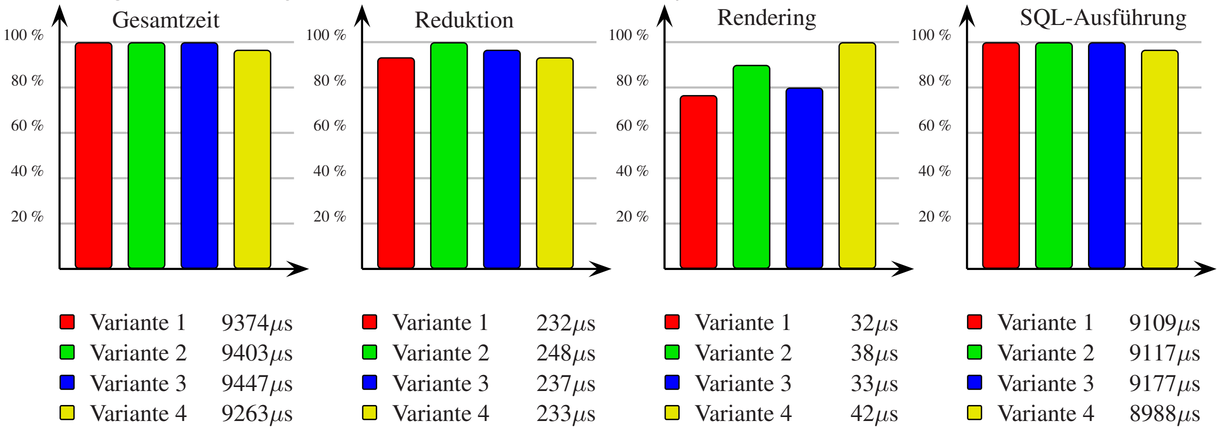


Abbildung H.55 Zeitvergleich: ATTACH und DETACH

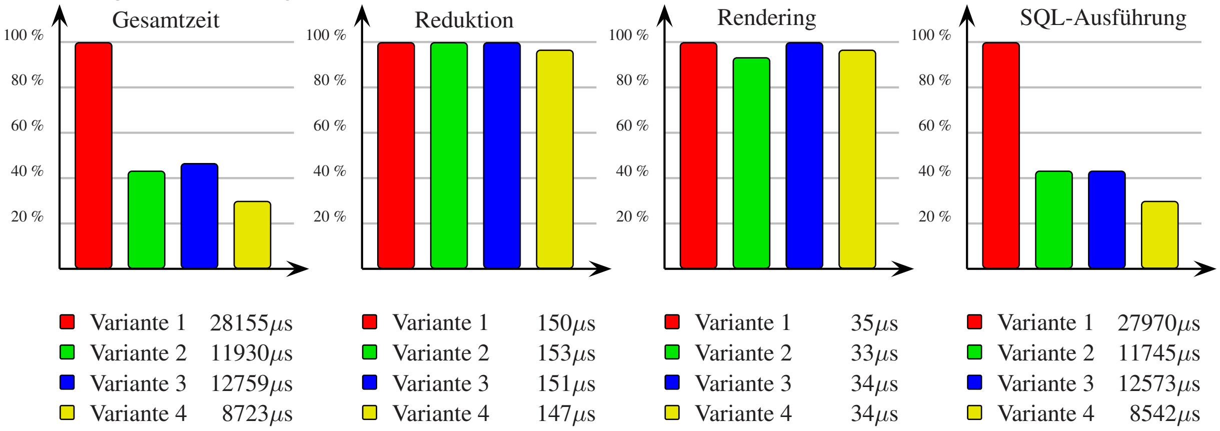


Abbildung H.56 Zeitvergleich: ATTACH

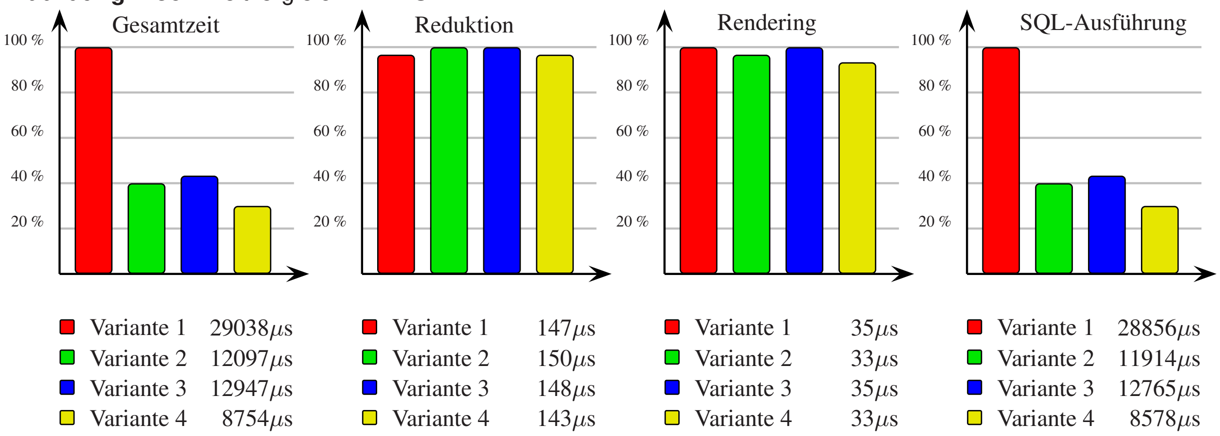


Abbildung H.57 Zeitvergleich: DETACH

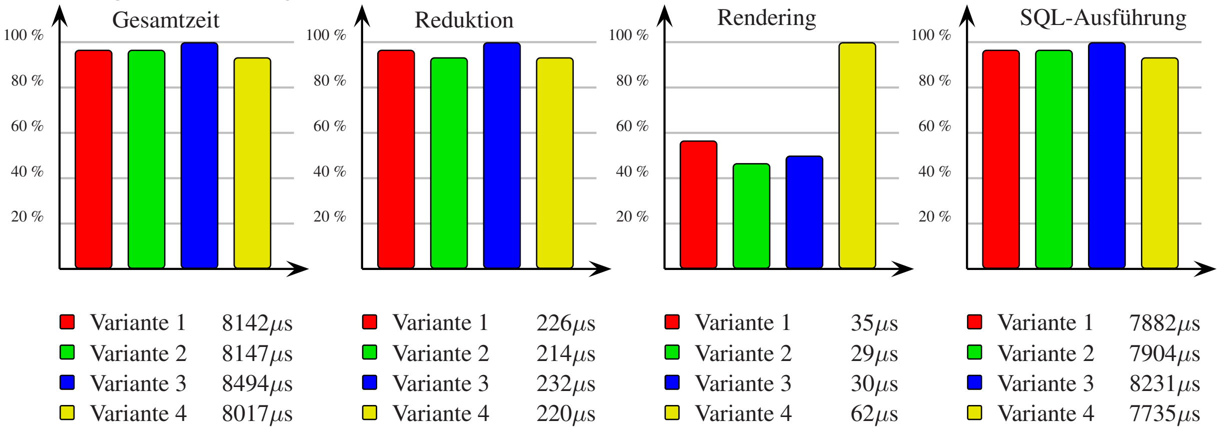


Abbildung H.58 Zeitvergleich: GET

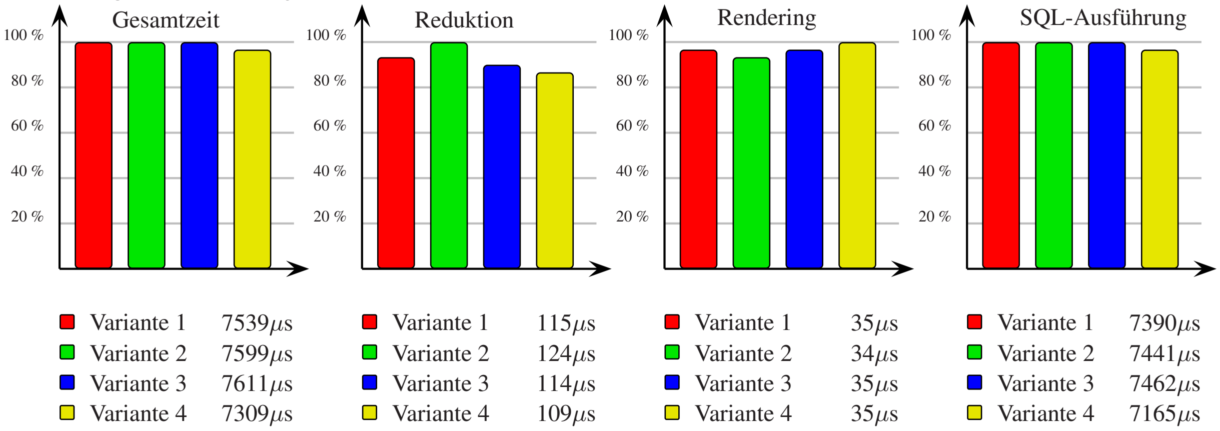


Abbildung H.59 Zeitvergleich: GET ROOT

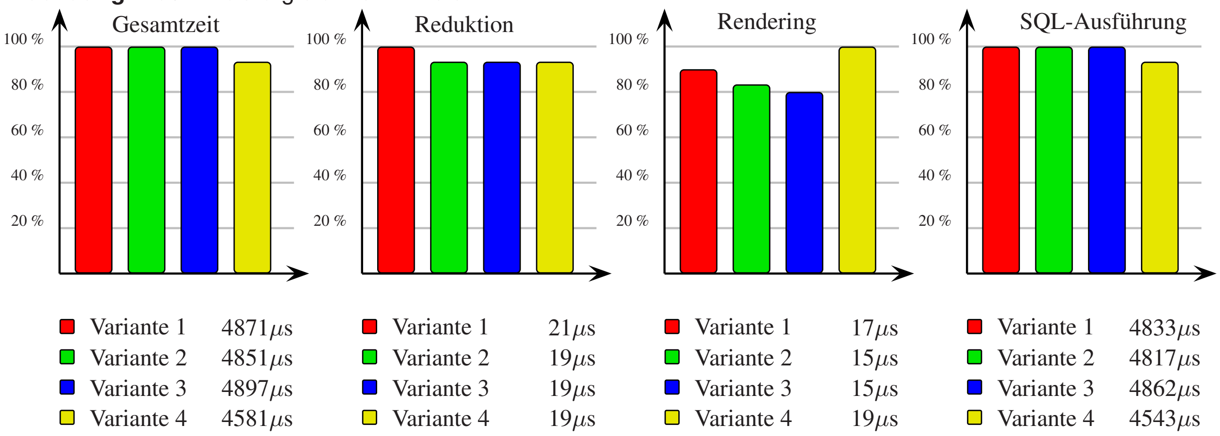


Abbildung H.60 Zeitvergleich: GET ALTERNATIVES

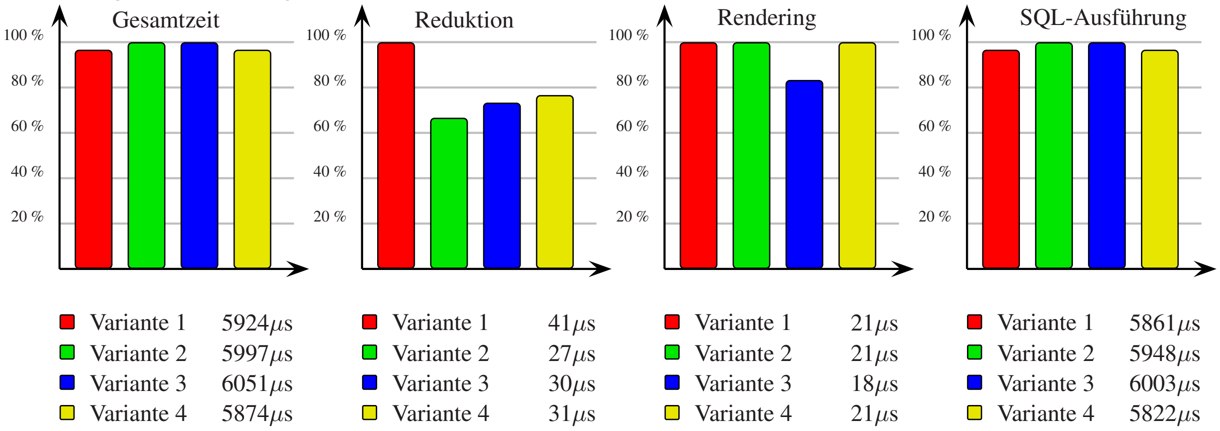


Abbildung H.61 Zeitvergleich: GET SUCCESSORS

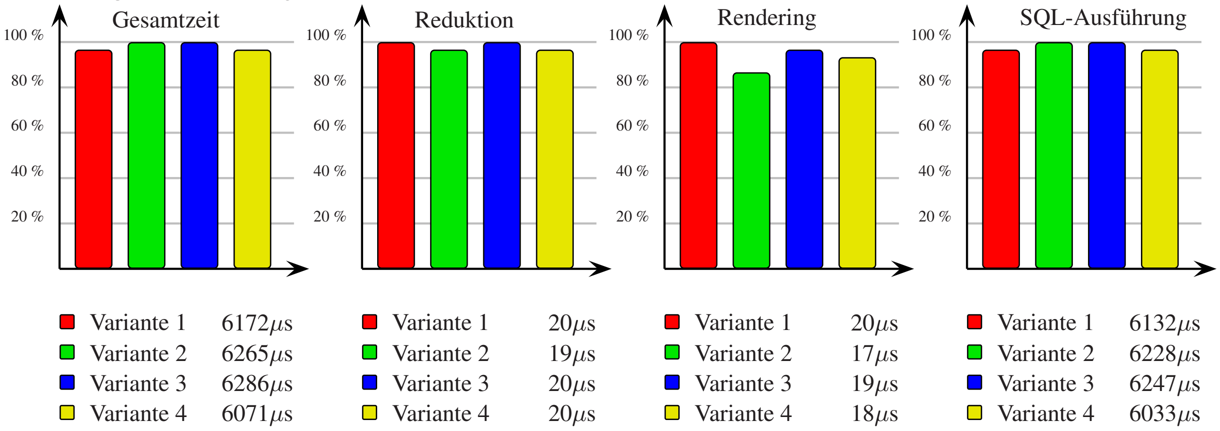


Abbildung H.62 Zeitvergleich: GET PREDECESSOR

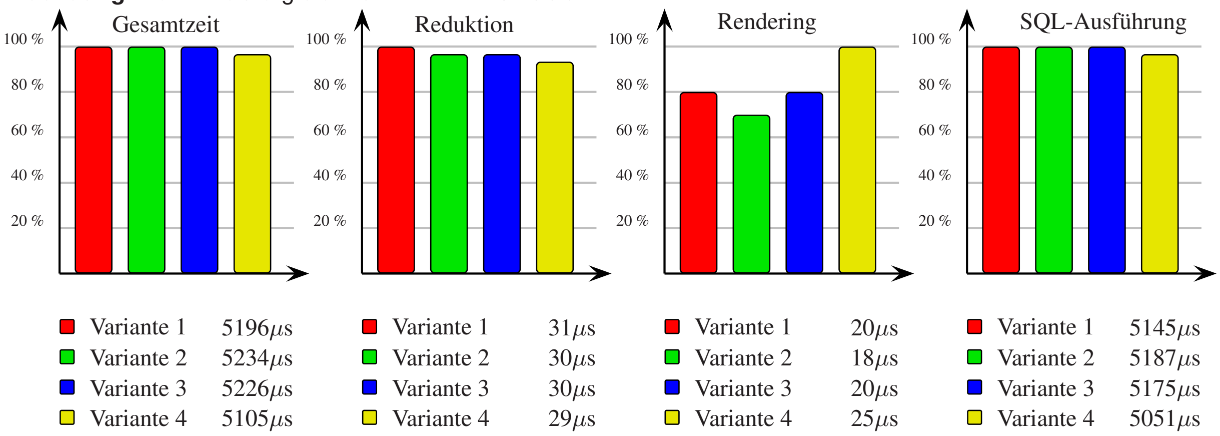


Abbildung H.63 Zeitvergleich: GET BASEVERSION

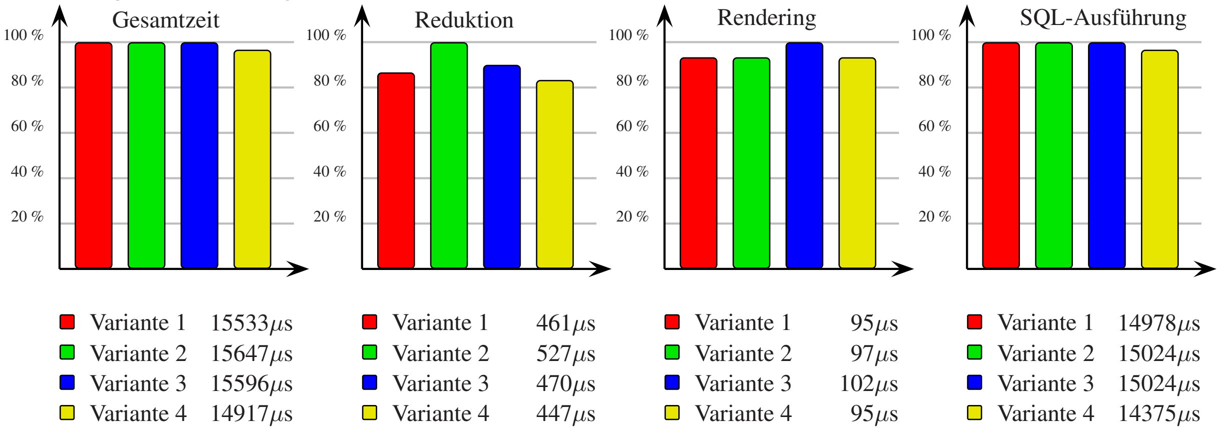


Abbildung H.64 Zeitvergleich: GET CVC

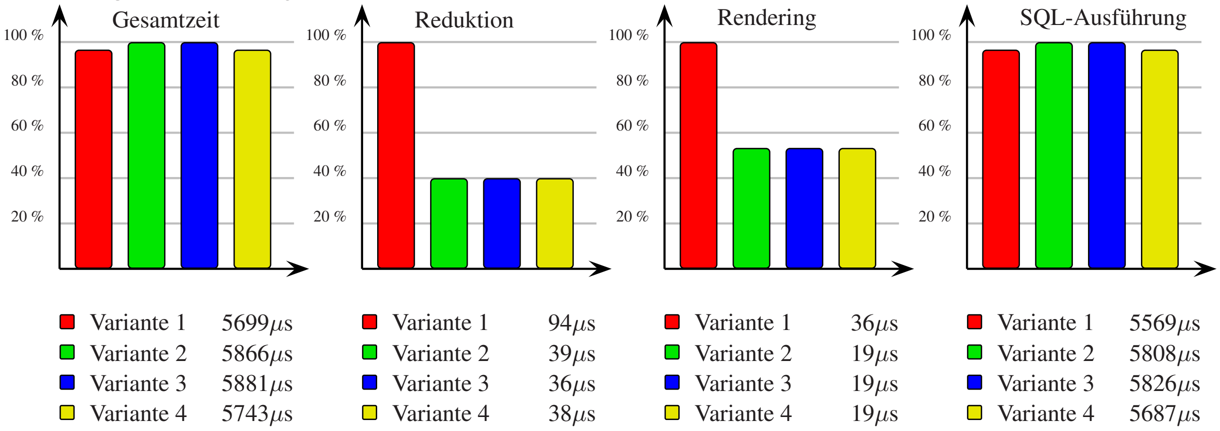


Abbildung H.65 Zeitvergleich: PIN

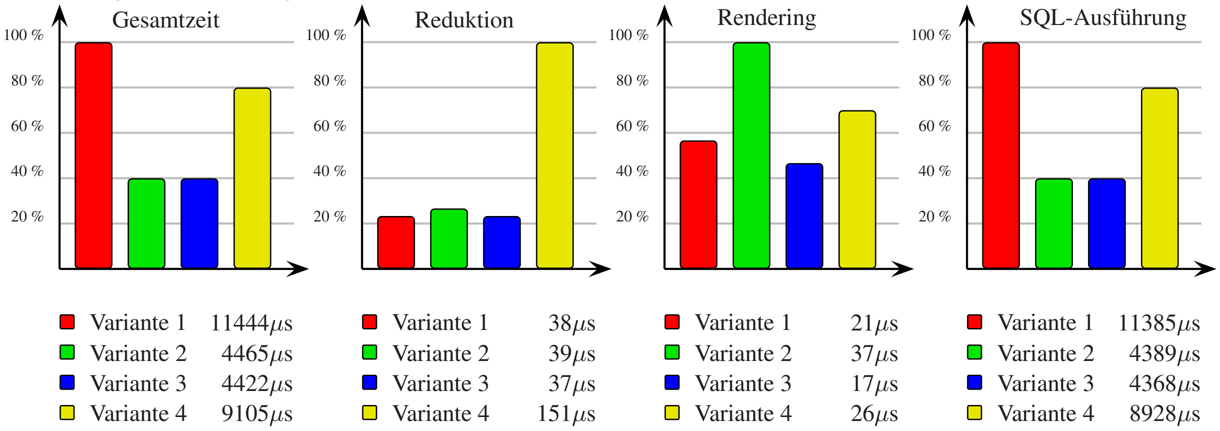


Abbildung H.66 Zeitvergleich: SELECT

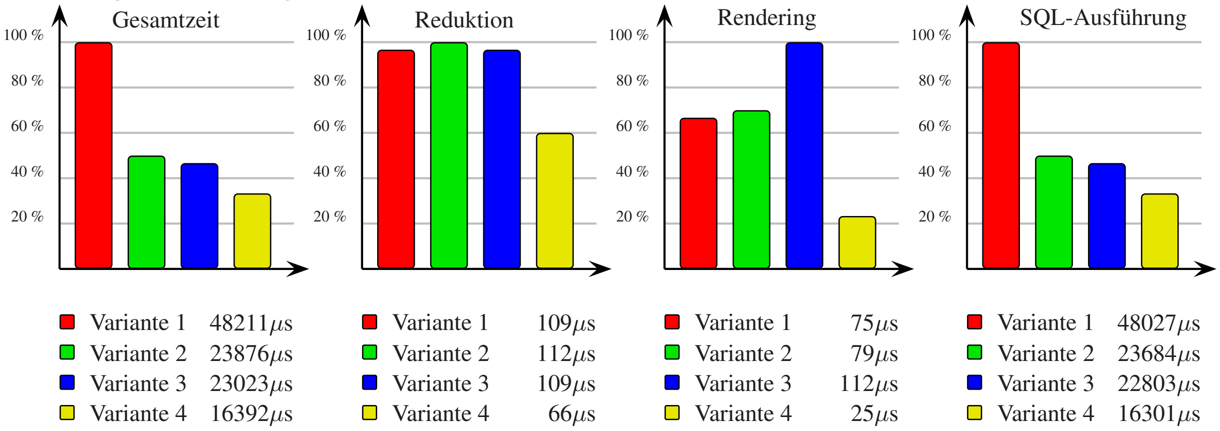


Abbildung H.67 Zeitvergleich: SELECT außerhalb von Workspaces

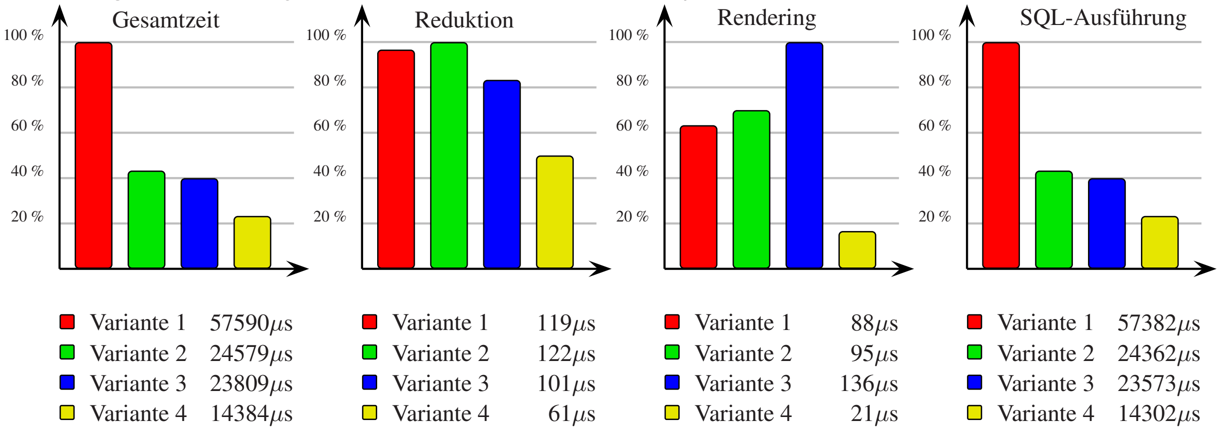


Abbildung H.68 Zeitvergleich: SELECT a.v.W. 1 Navigationsschritt

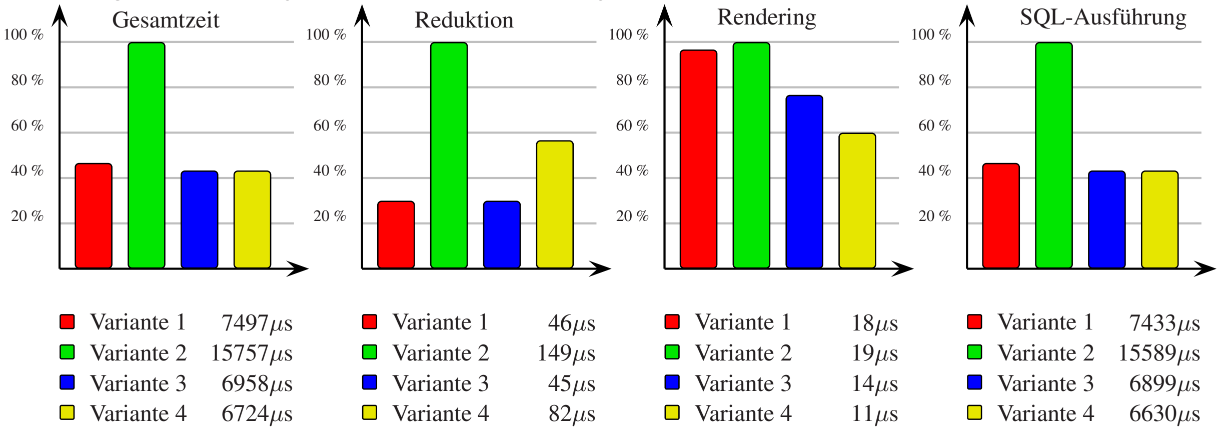


Abbildung H.69 Zeitvergleich: SELECT a.v.W. 2 Navigationsschritte

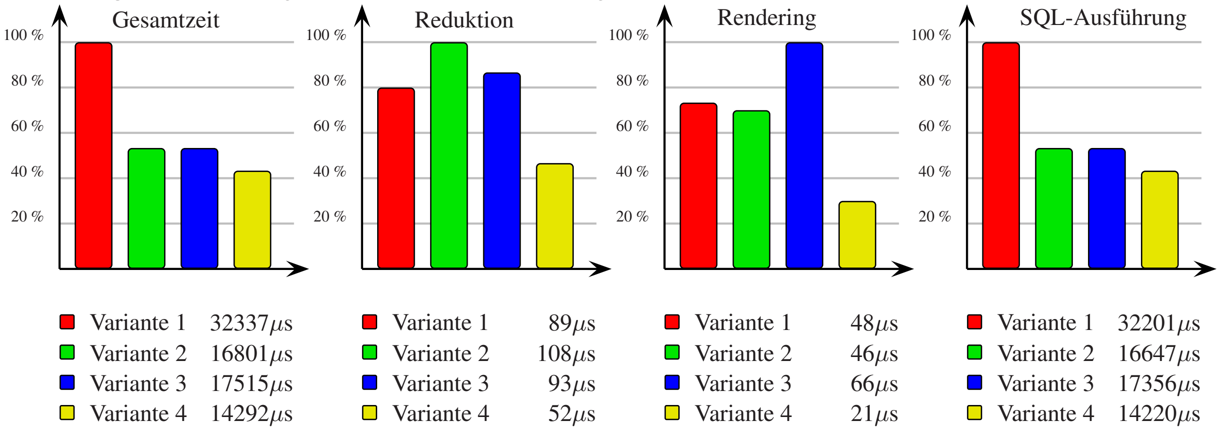


Abbildung H.70 Zeitvergleich: SELECT a.v.W. 4 Navigationsschritte

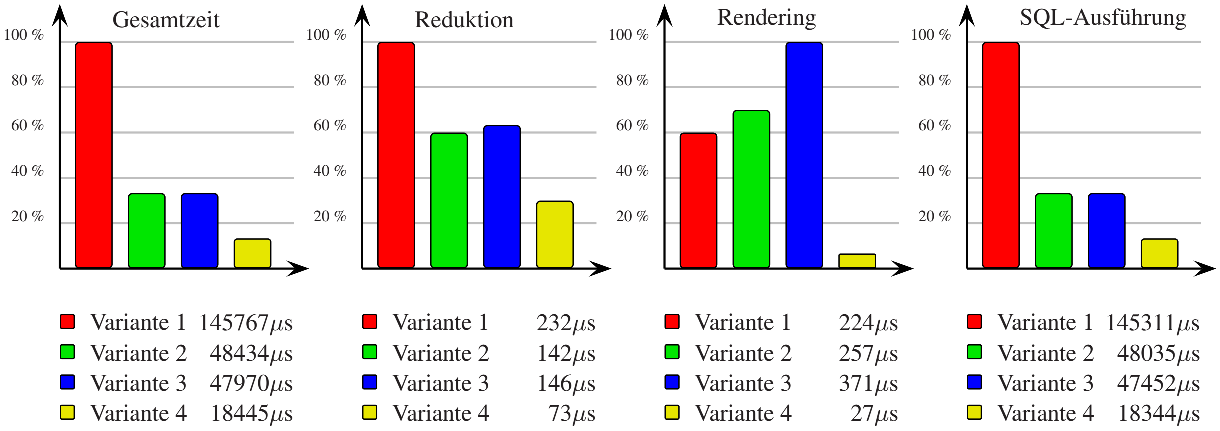


Abbildung H.71 Zeitvergleich: SELECT innerhalb von Workspaces

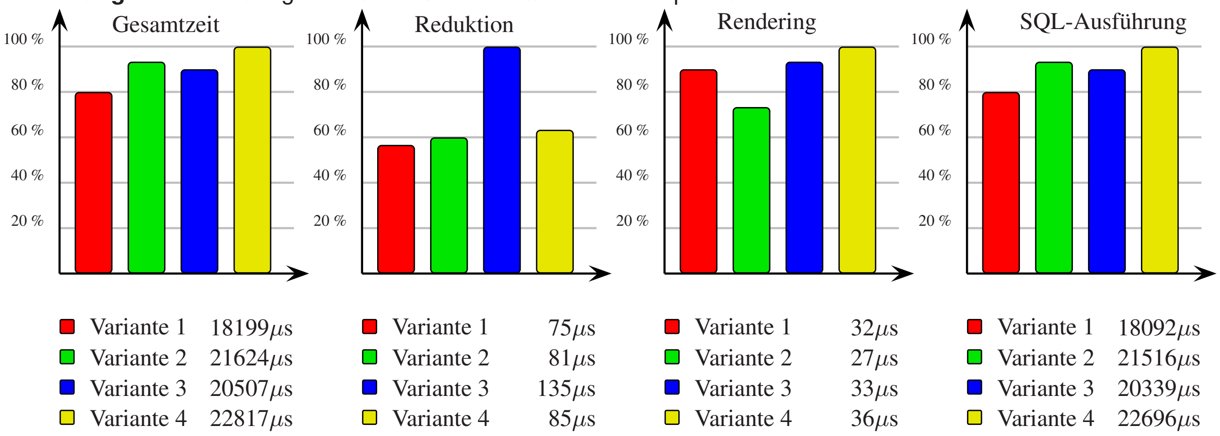


Abbildung H.72 Zeitvergleich: SELECT i.v.W. 1 Navigationsschritt

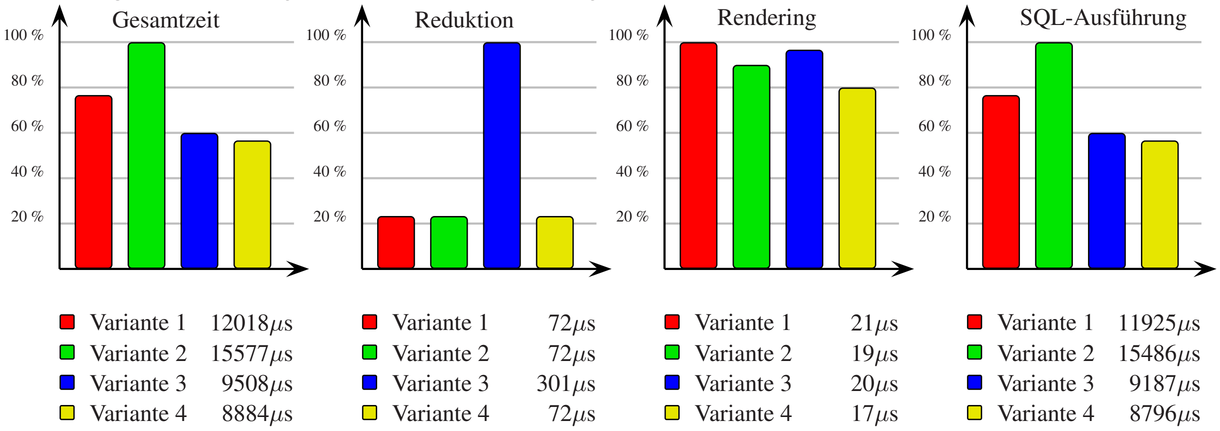
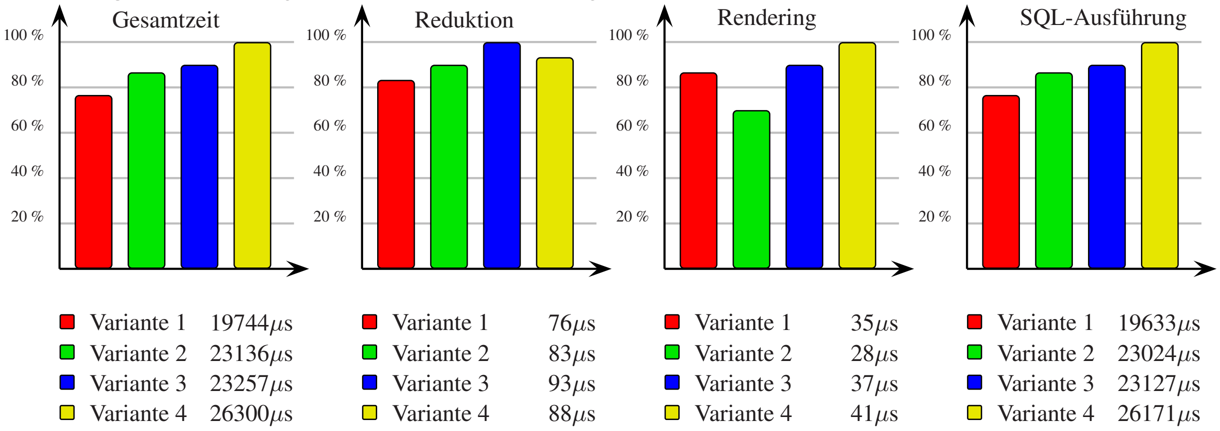


Abbildung H.73 Zeitvergleich: SELECT i.v.W. 2 Navigationsschritte



H.1.2 Beschleunigung

Abbildung H.74 Beschleunigung Variante 2:1 - Reduktion + Rendering

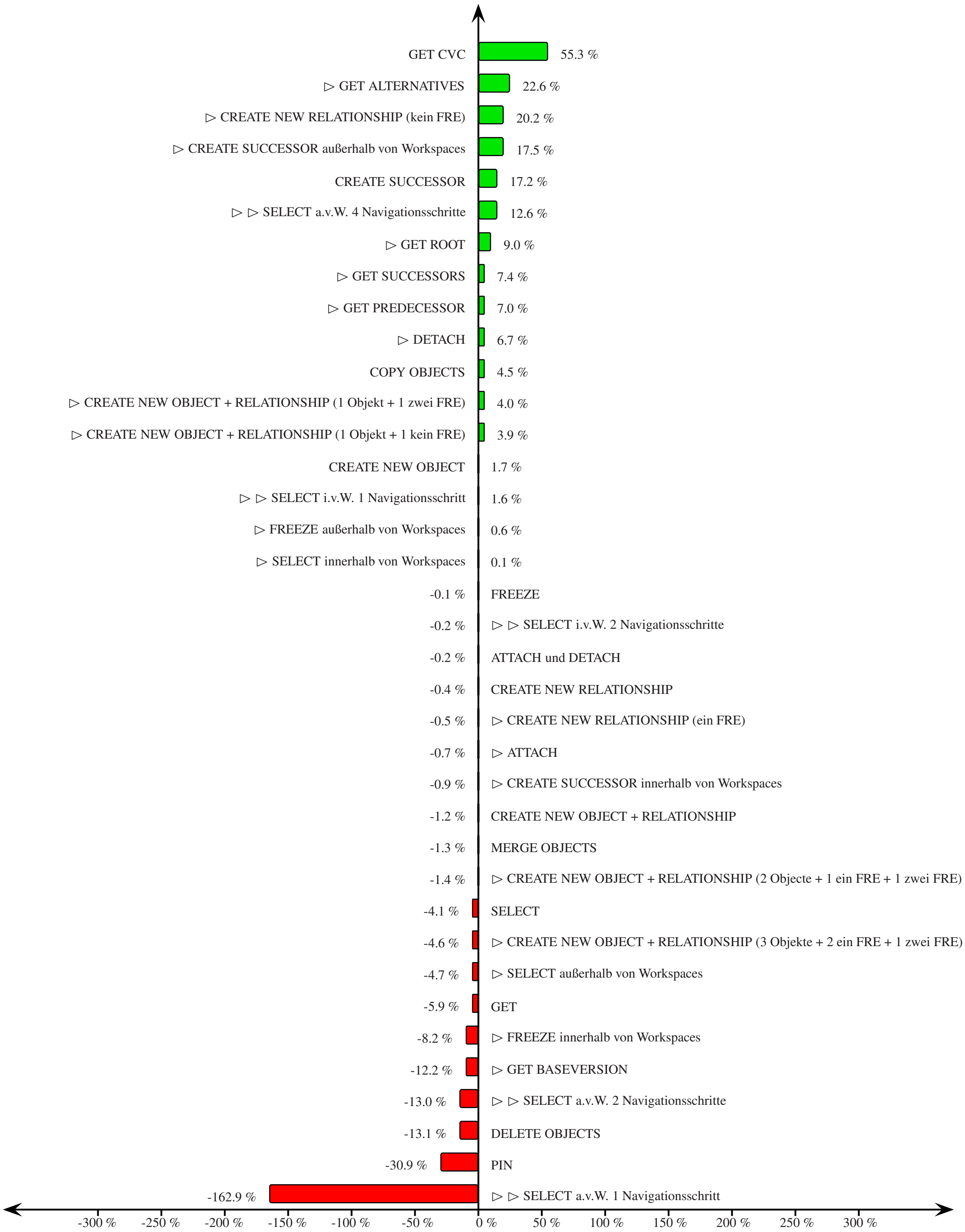


Abbildung H.75 Beschleunigung Variante 2:1 - SQL-Ausführung

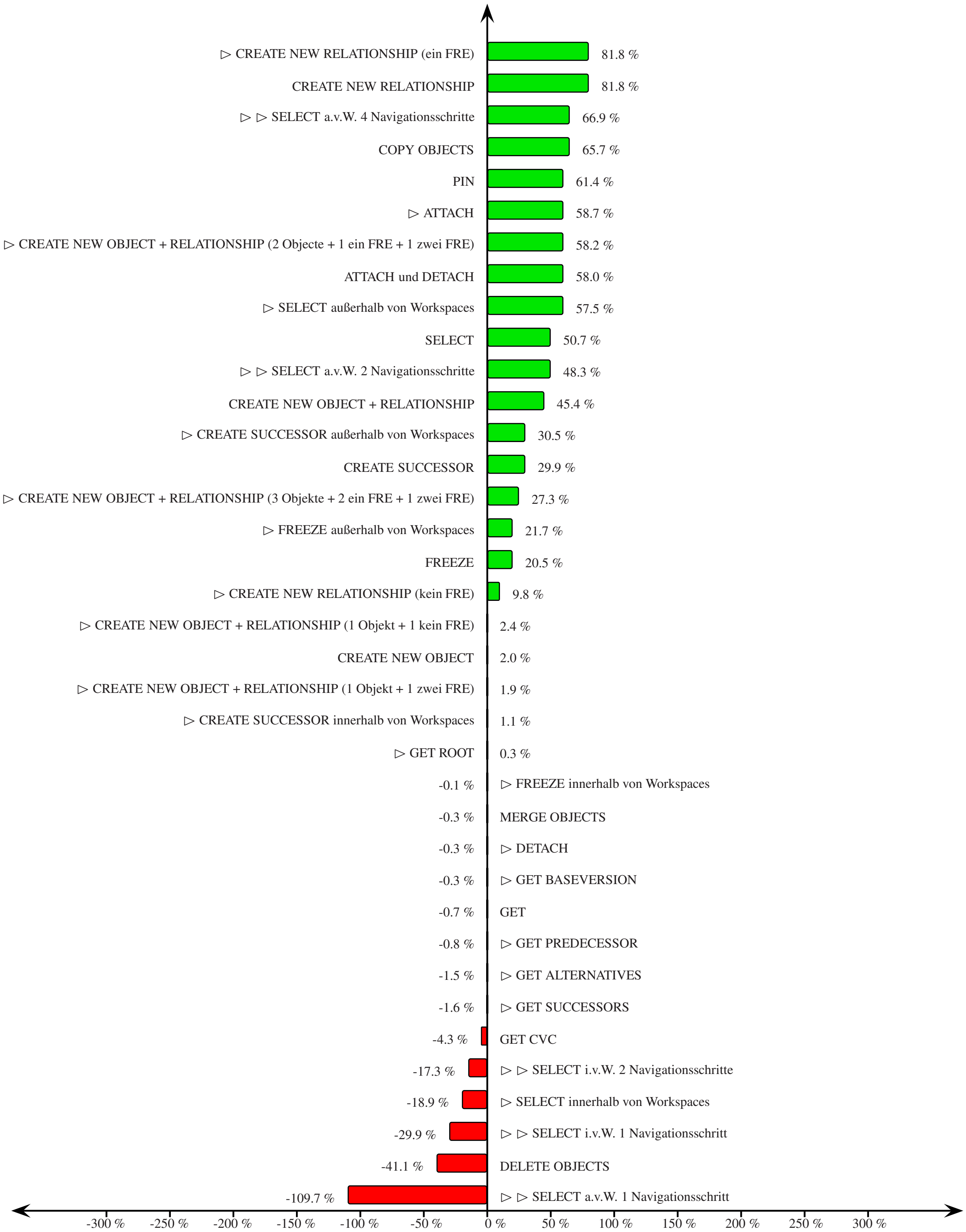


Abbildung H.76 Beschleunigung Variante 2:1 - Reduktion + Rendering + SQL-Ausführung

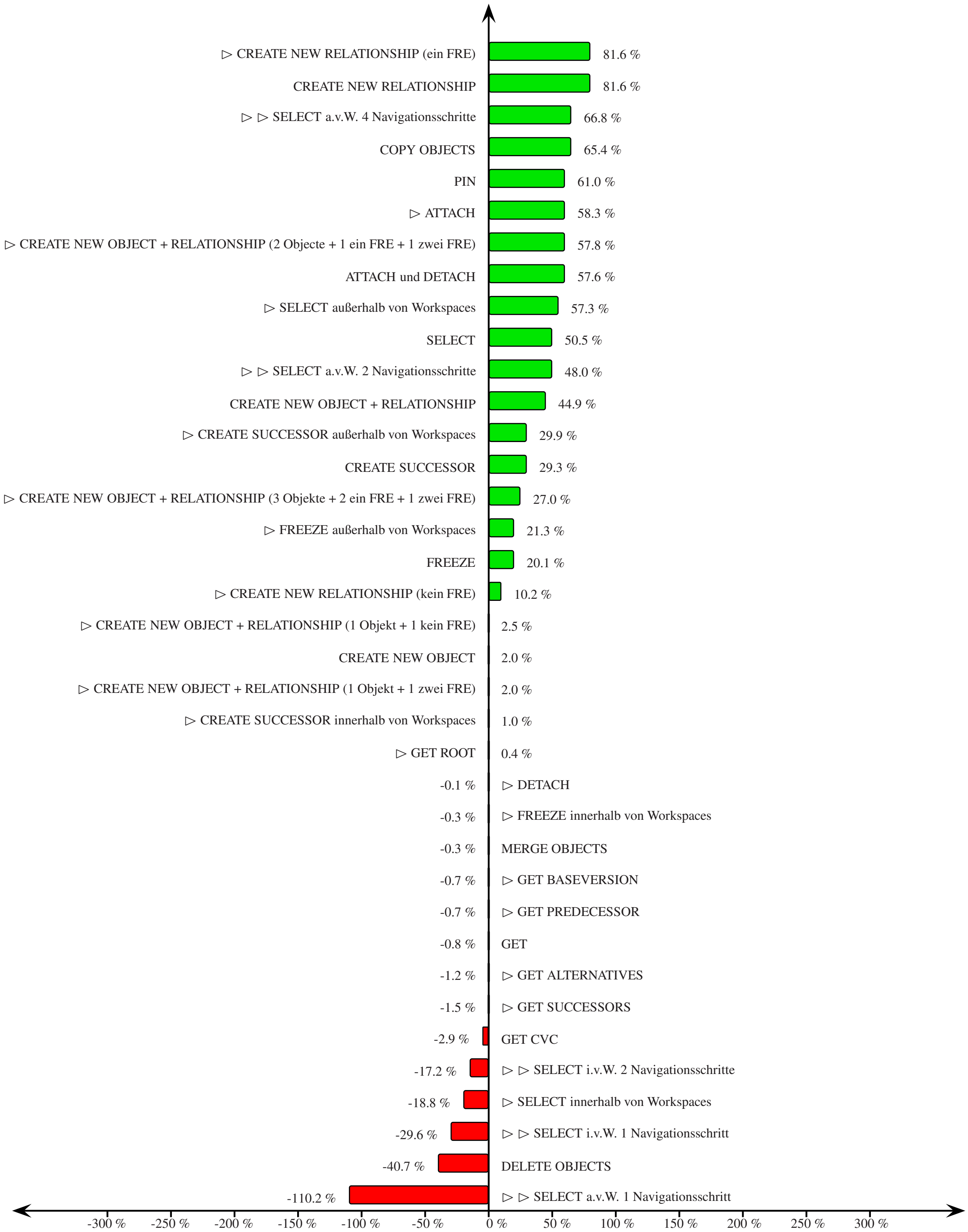


Abbildung H.77 Beschleunigung Variante 3:1 - Reduktion + Rendering

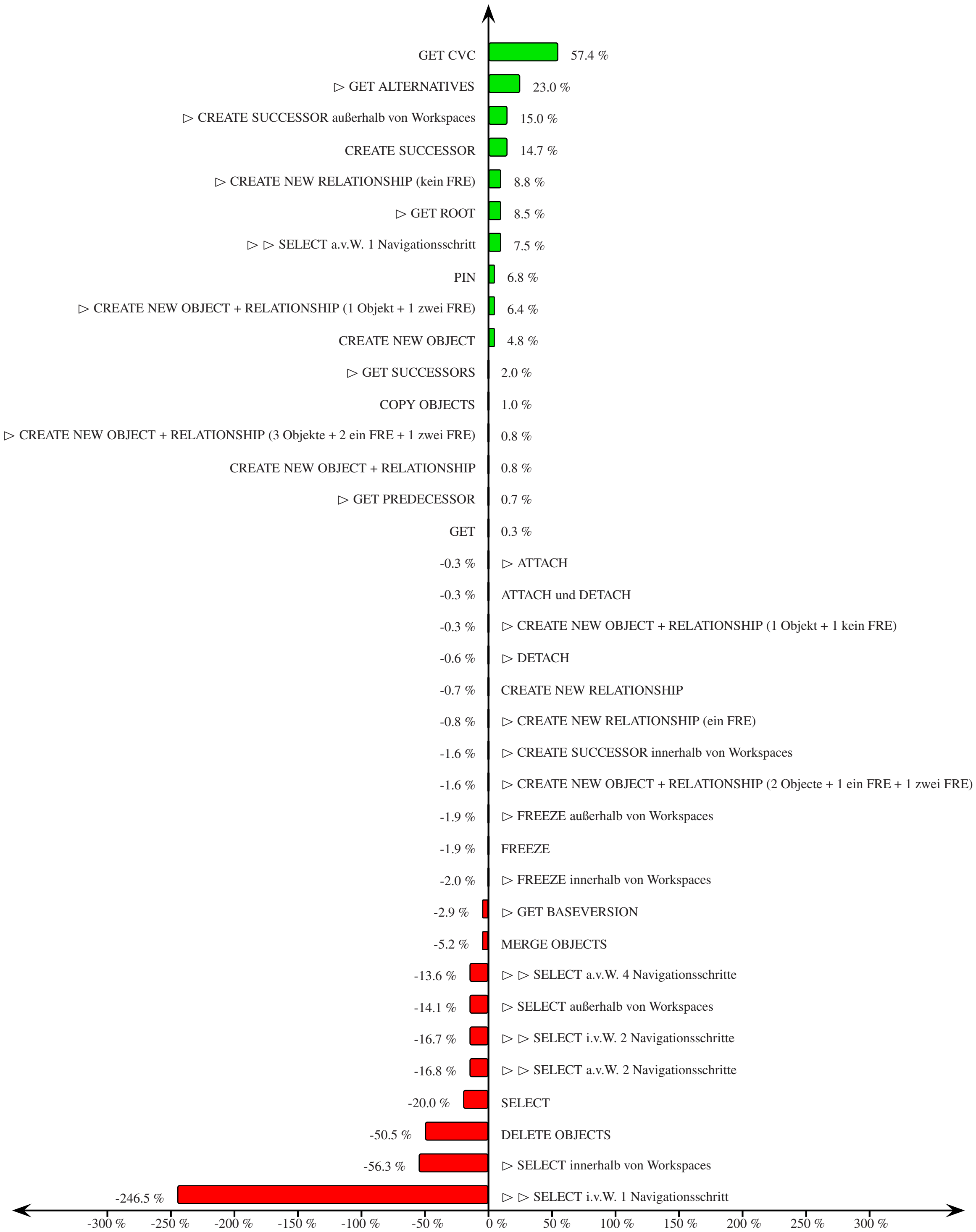


Abbildung H.78 Beschleunigung Variante 3:1 - SQL-Ausführung

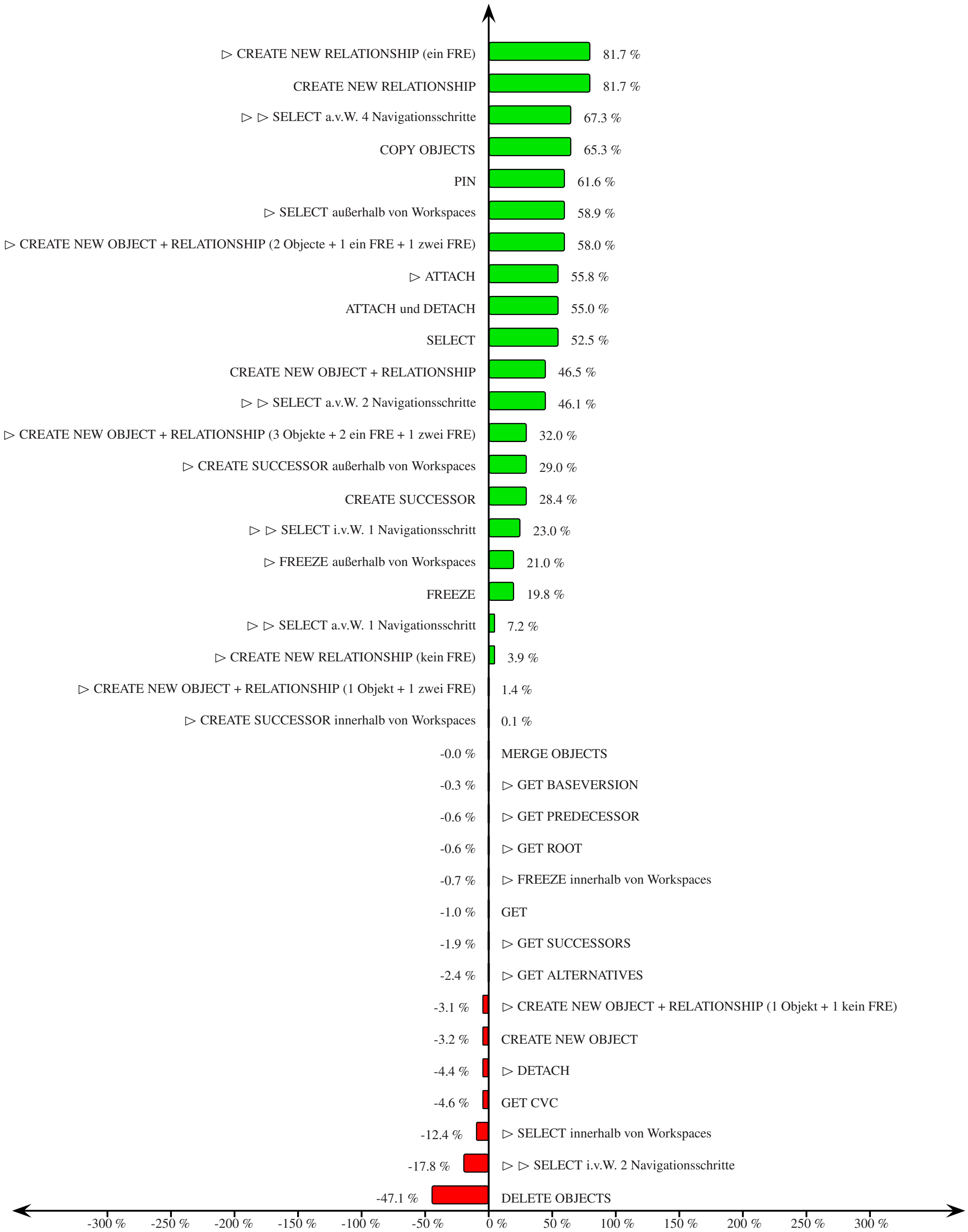


Abbildung H.79 Beschleunigung Variante 3:1 - Reduktion + Rendering + SQL-Ausführung

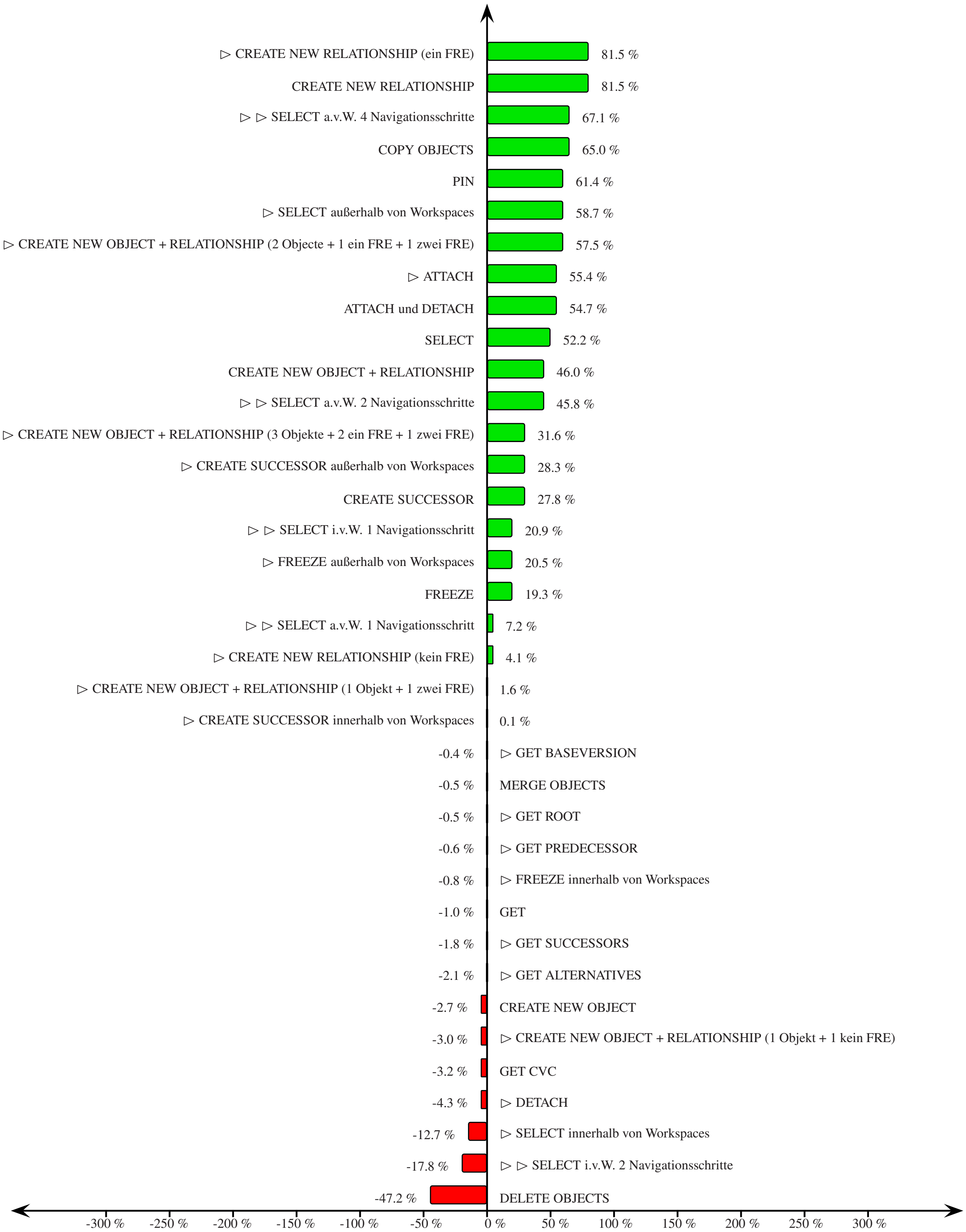


Abbildung H.80 Beschleunigung Variante 4:1 - Reduktion + Rendering

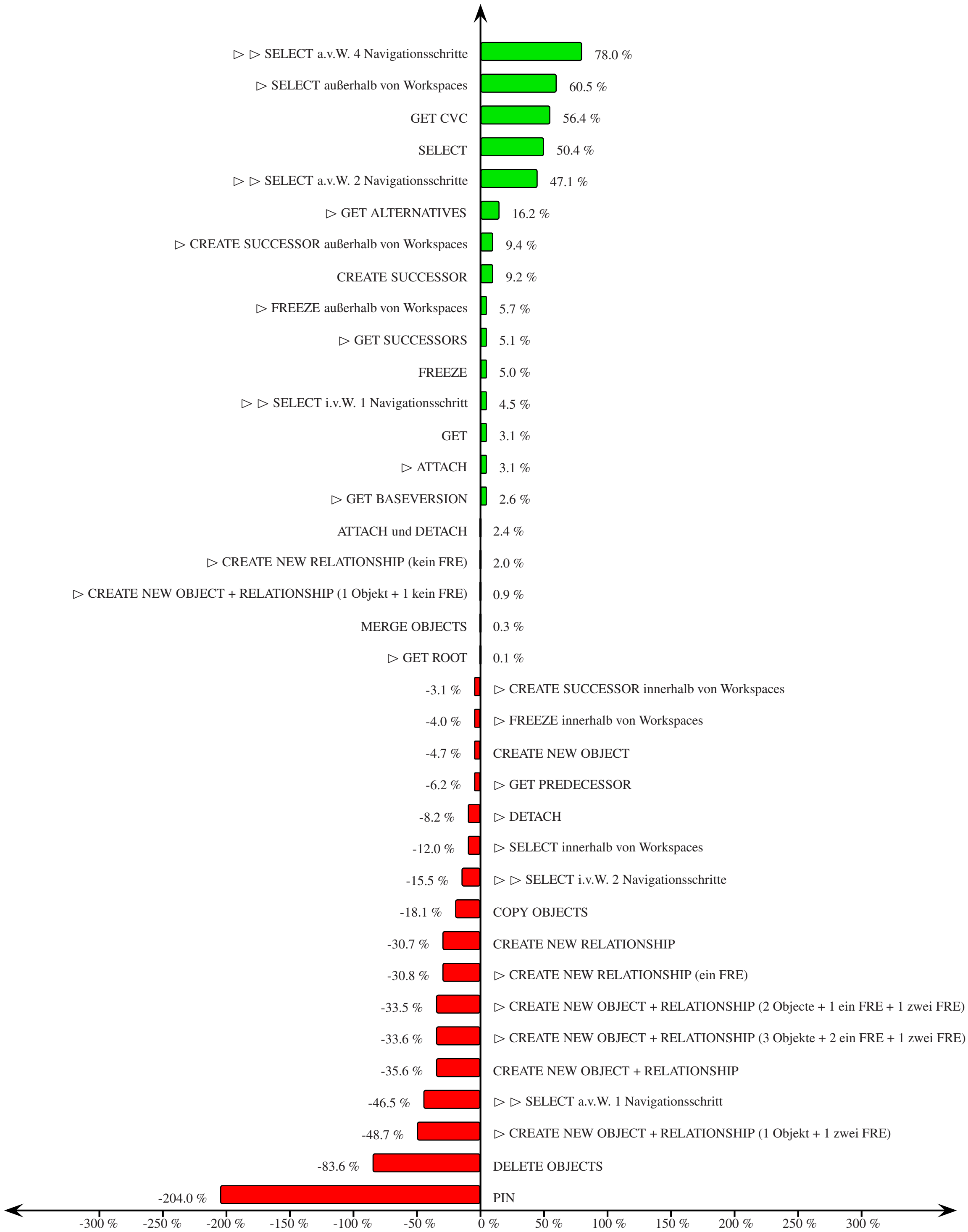


Abbildung H.81 Beschleunigung Variante 4:1 - SQL-Ausführung

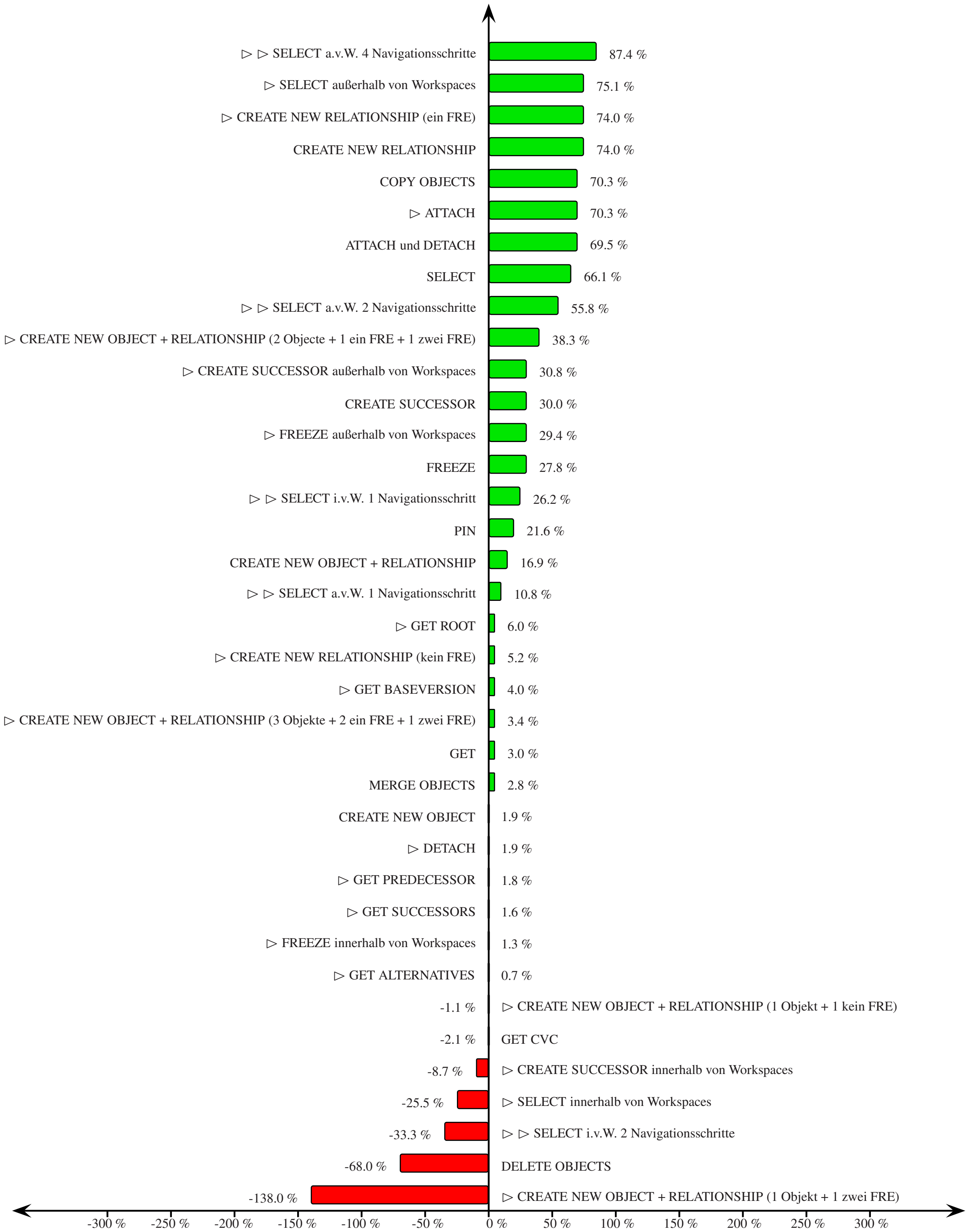


Abbildung H.82 Beschleunigung Variante 4:1 - Reduktion + Rendering + SQL-Ausführung

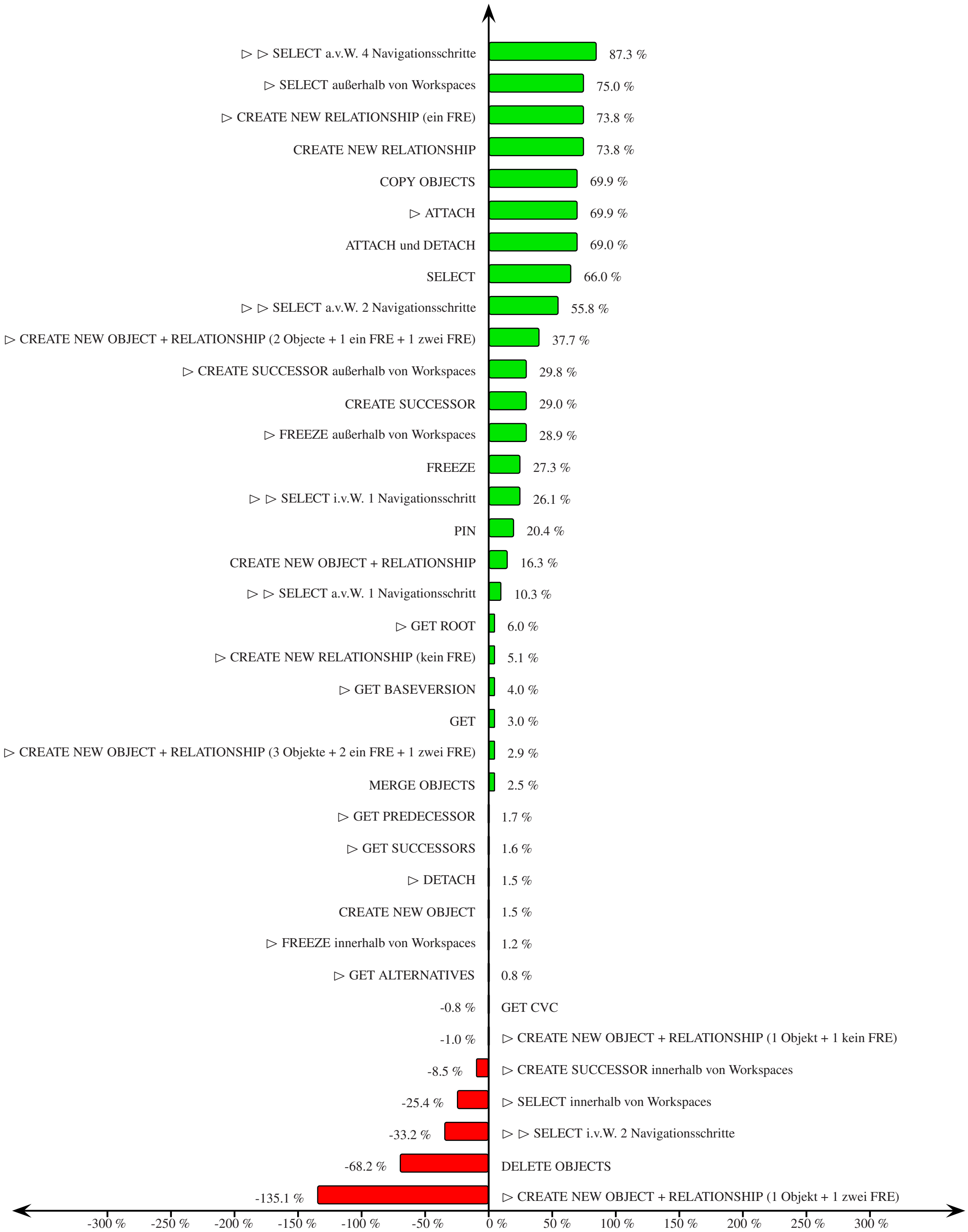


Abbildung H.83 Beschleunigung Variante 3:2 - Reduktion + Rendering

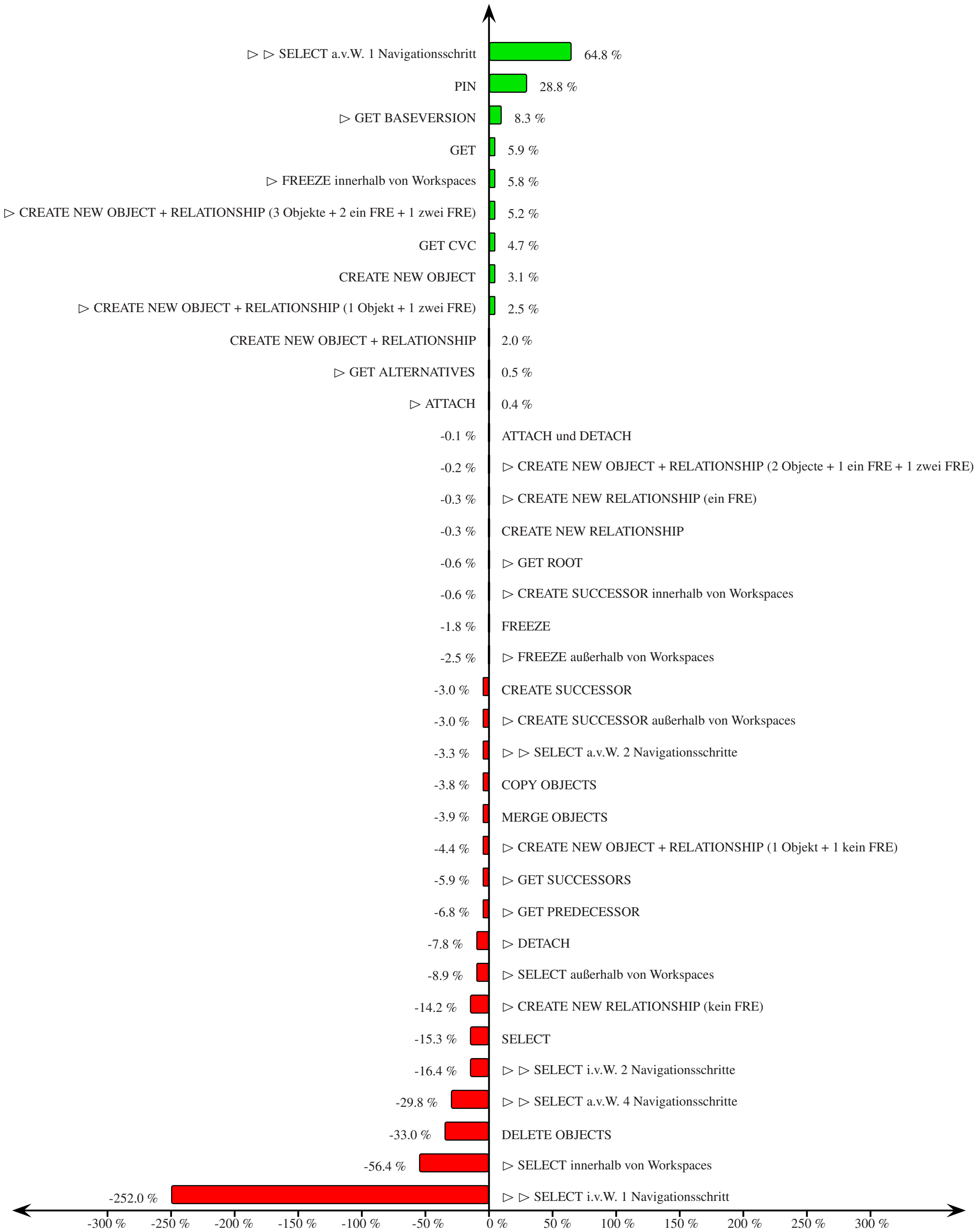


Abbildung H.84 Beschleunigung Variante 3:2 - SQL-Ausführung

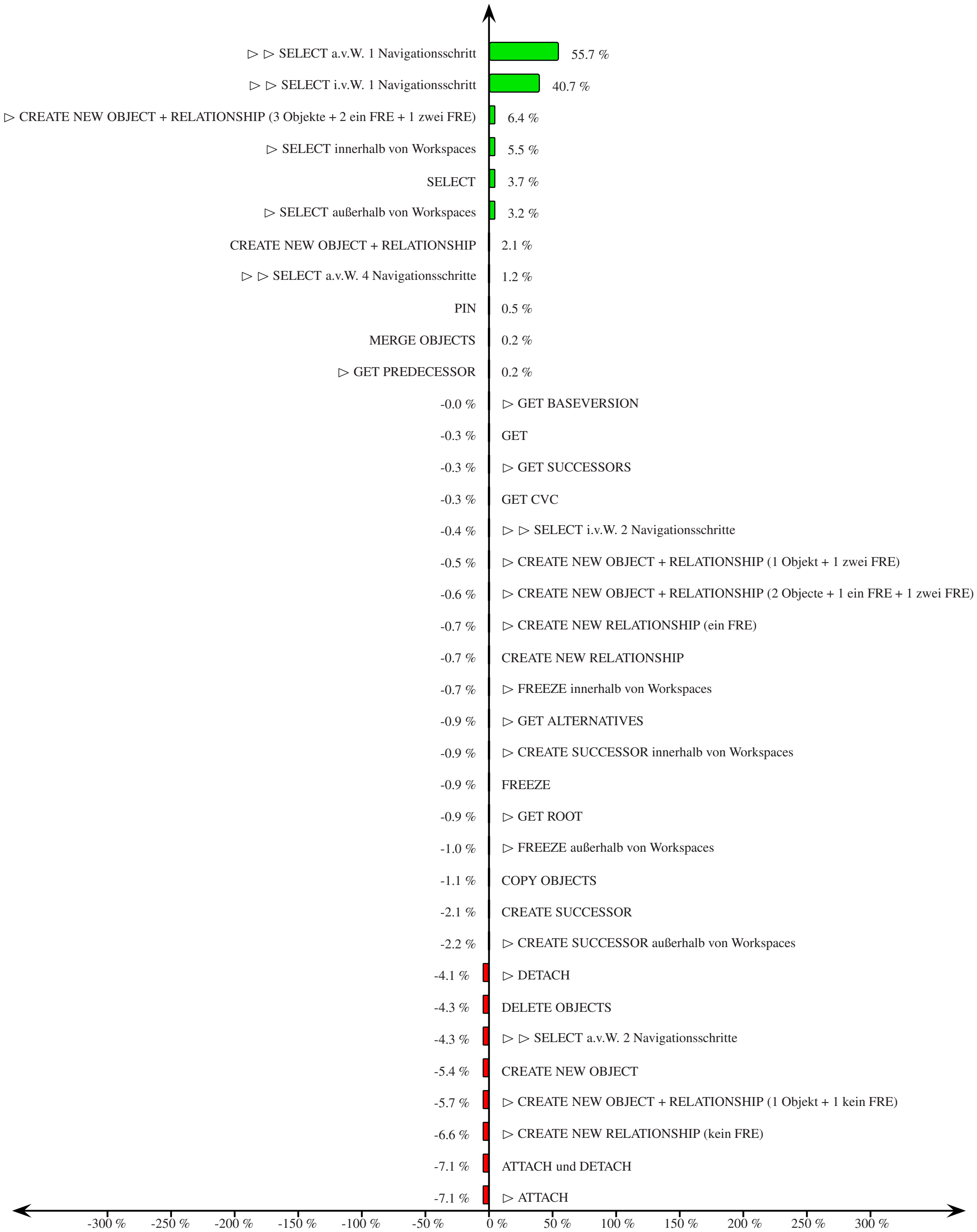


Abbildung H.85 Beschleunigung Variante 3:2 - Reduktion + Rendering + SQL-Ausführung

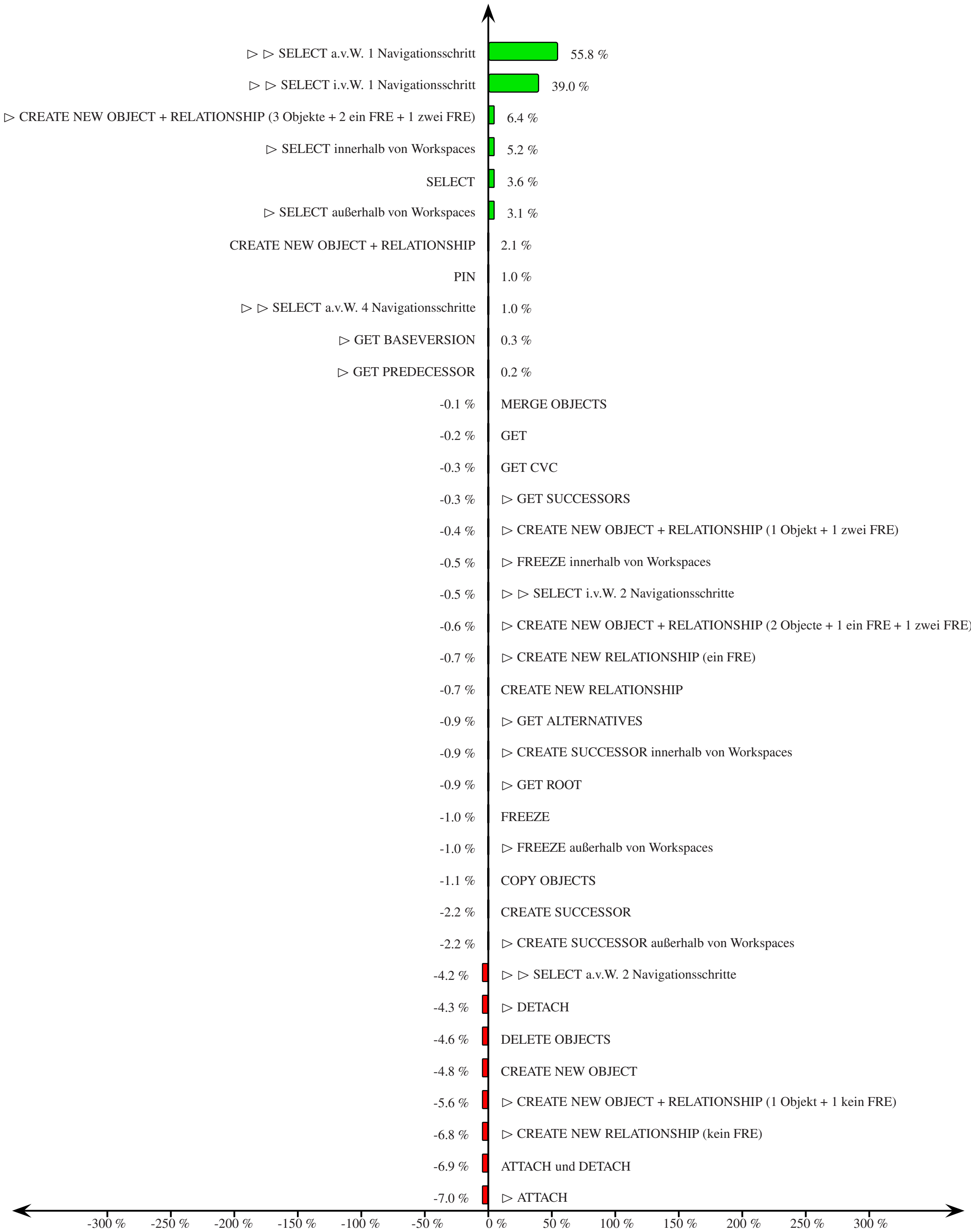


Abbildung H.86 Beschleunigung Variante 4:2 - Reduktion + Rendering

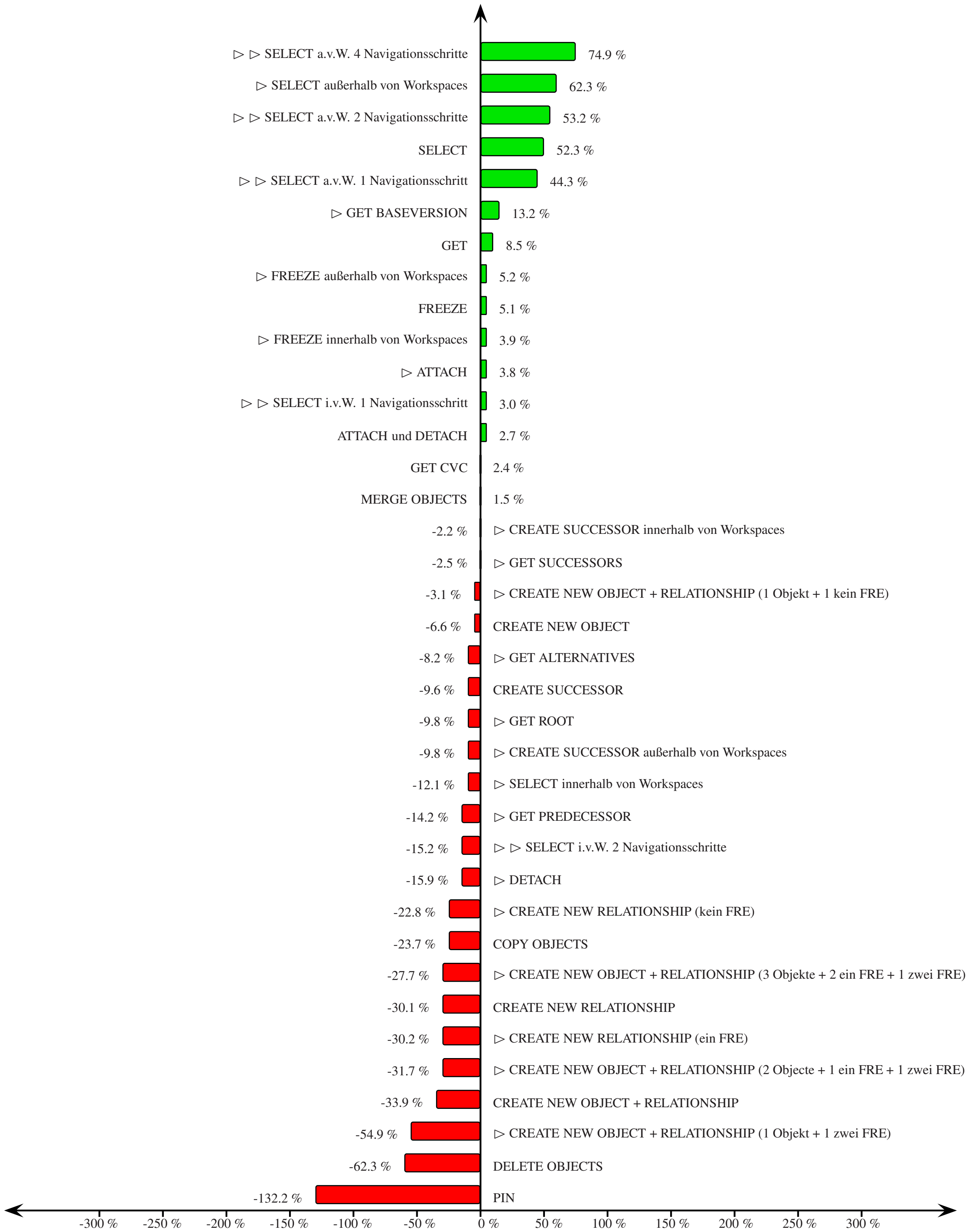


Abbildung H.87 Beschleunigung Variante 4:2 - SQL-Ausführung

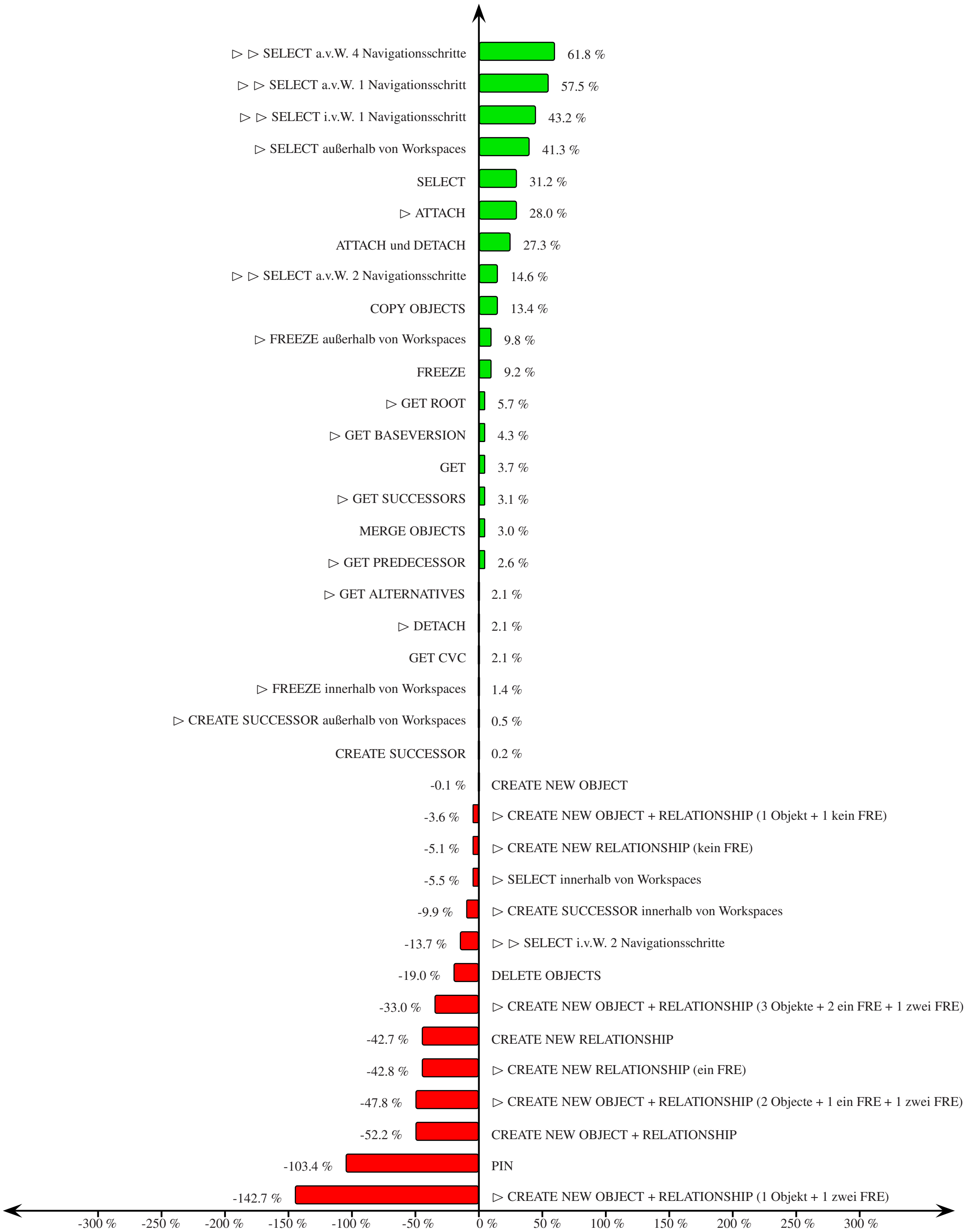


Abbildung H.88 Beschleunigung Variante 4:2 - Reduktion + Rendering + SQL-Ausführung

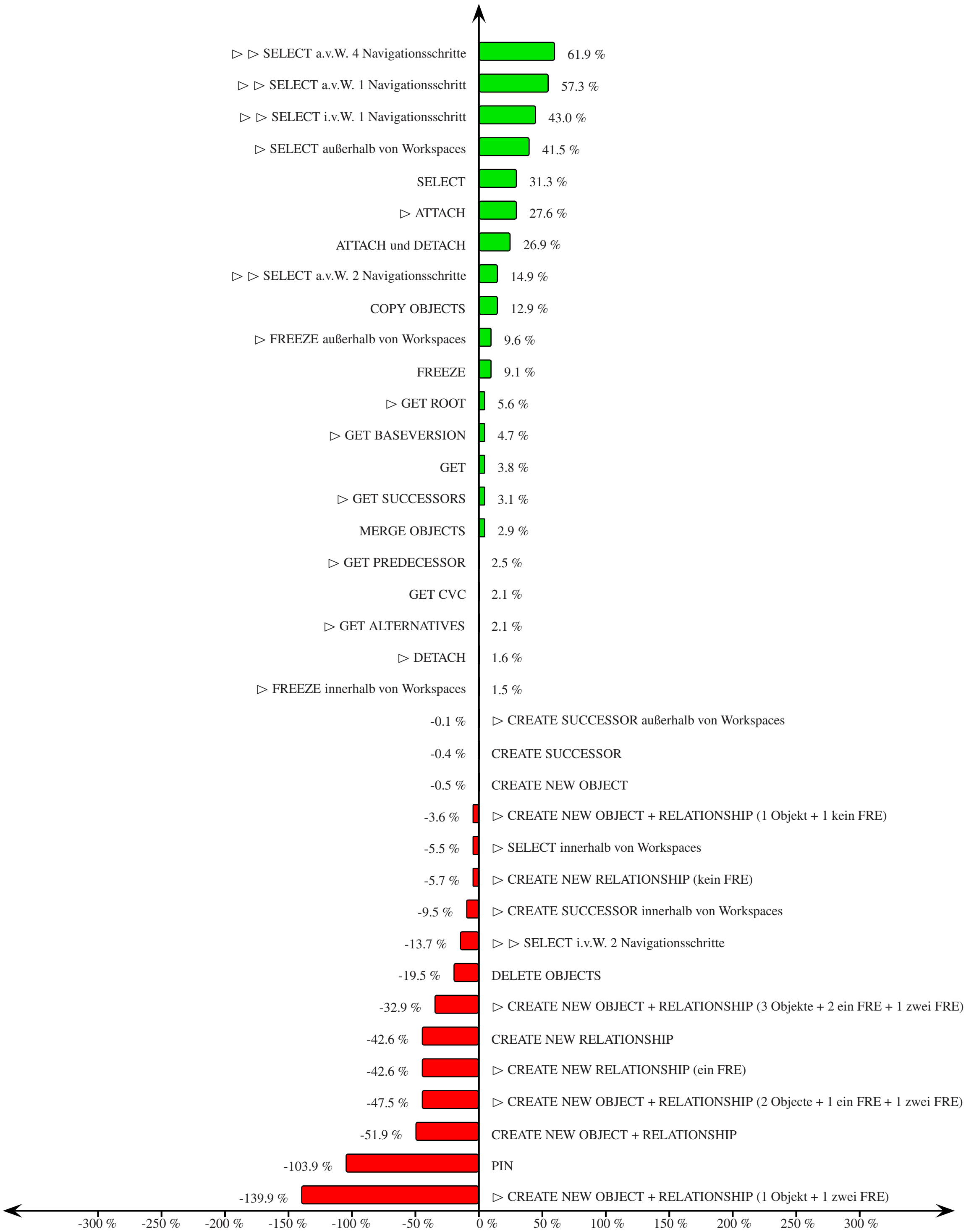


Abbildung H.89 Beschleunigung Variante 4:3 - Reduktion + Rendering

Abbildung H.90 Beschleunigung Variante 4:3 - SQL-Ausführung

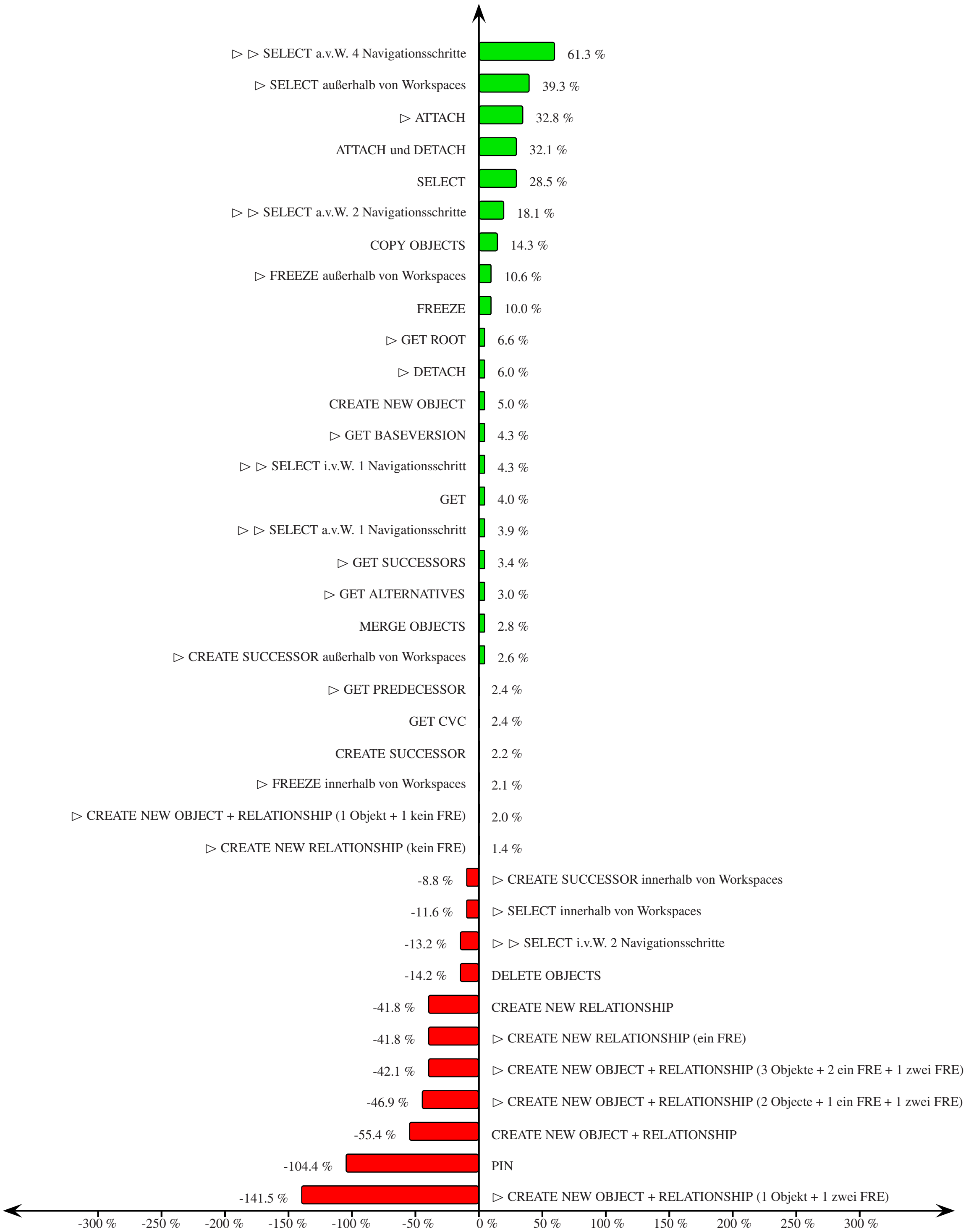
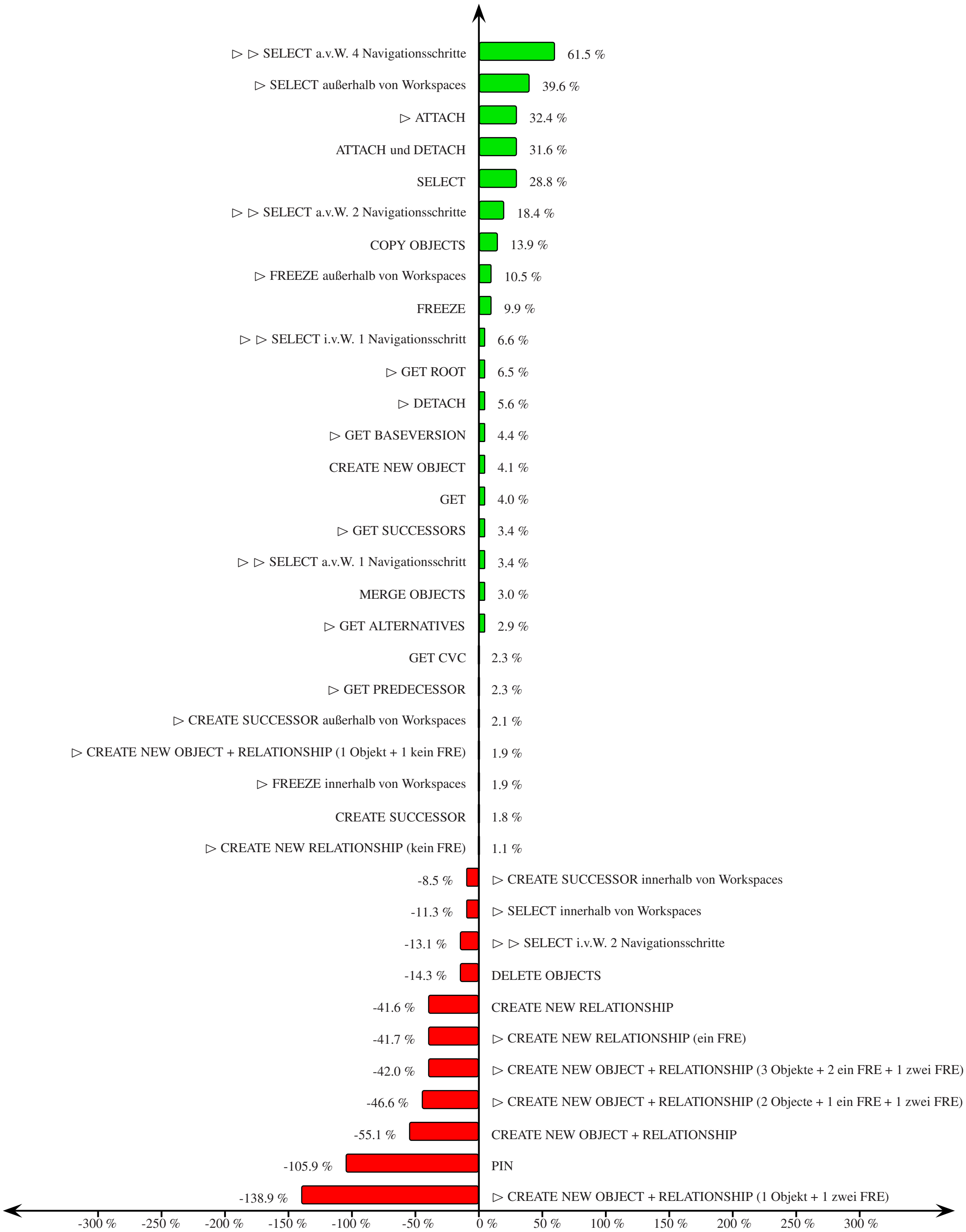


Abbildung H.91 Beschleunigung Variante 4:3 - Reduktion + Rendering + SQL-Ausführung



H.1.3 Kosten der einzelnen Befehle

Abbildung H.92 Schema 1 - Reduktion + Rendering

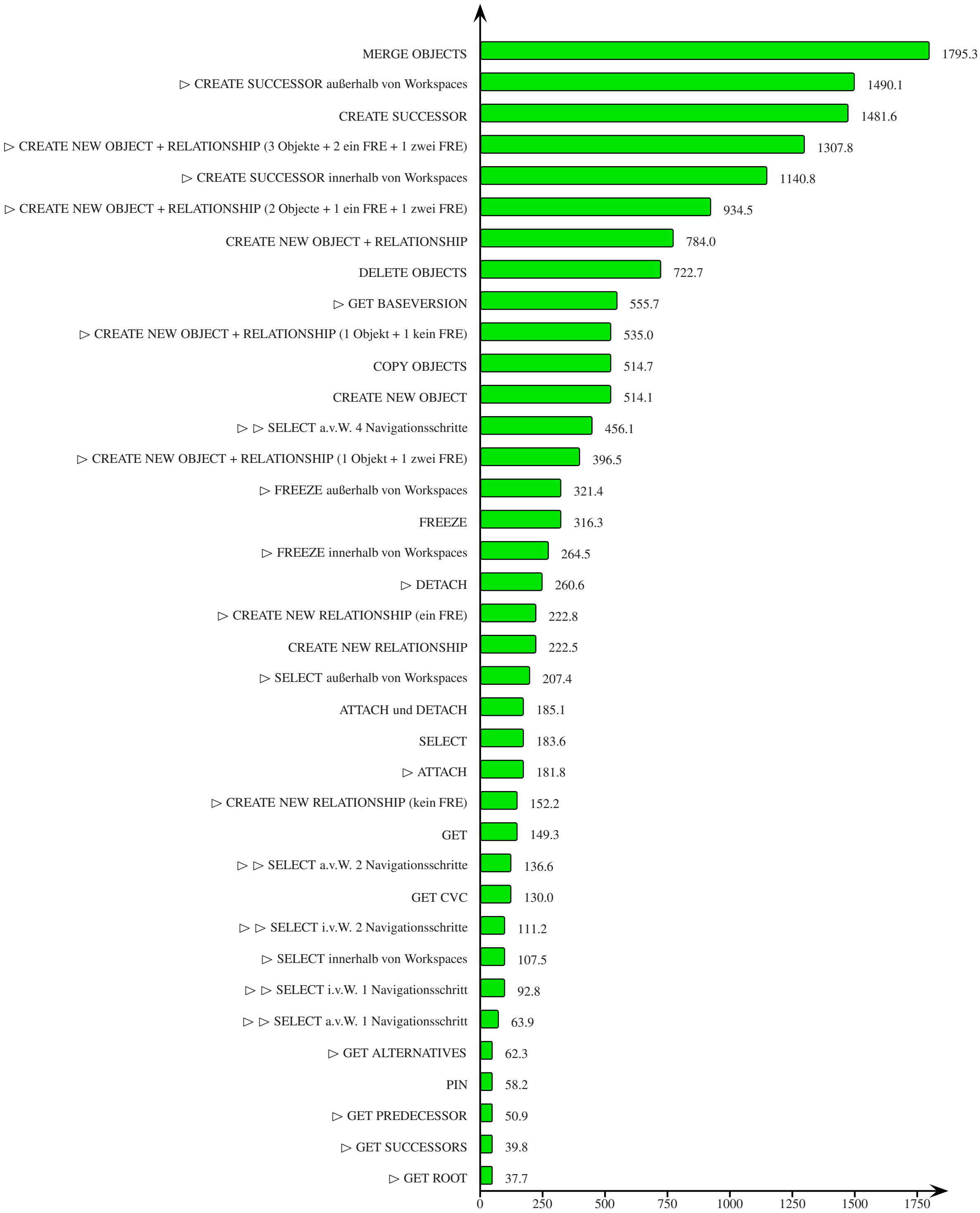


Abbildung H.93 Schema 1 - SQL-Ausführung

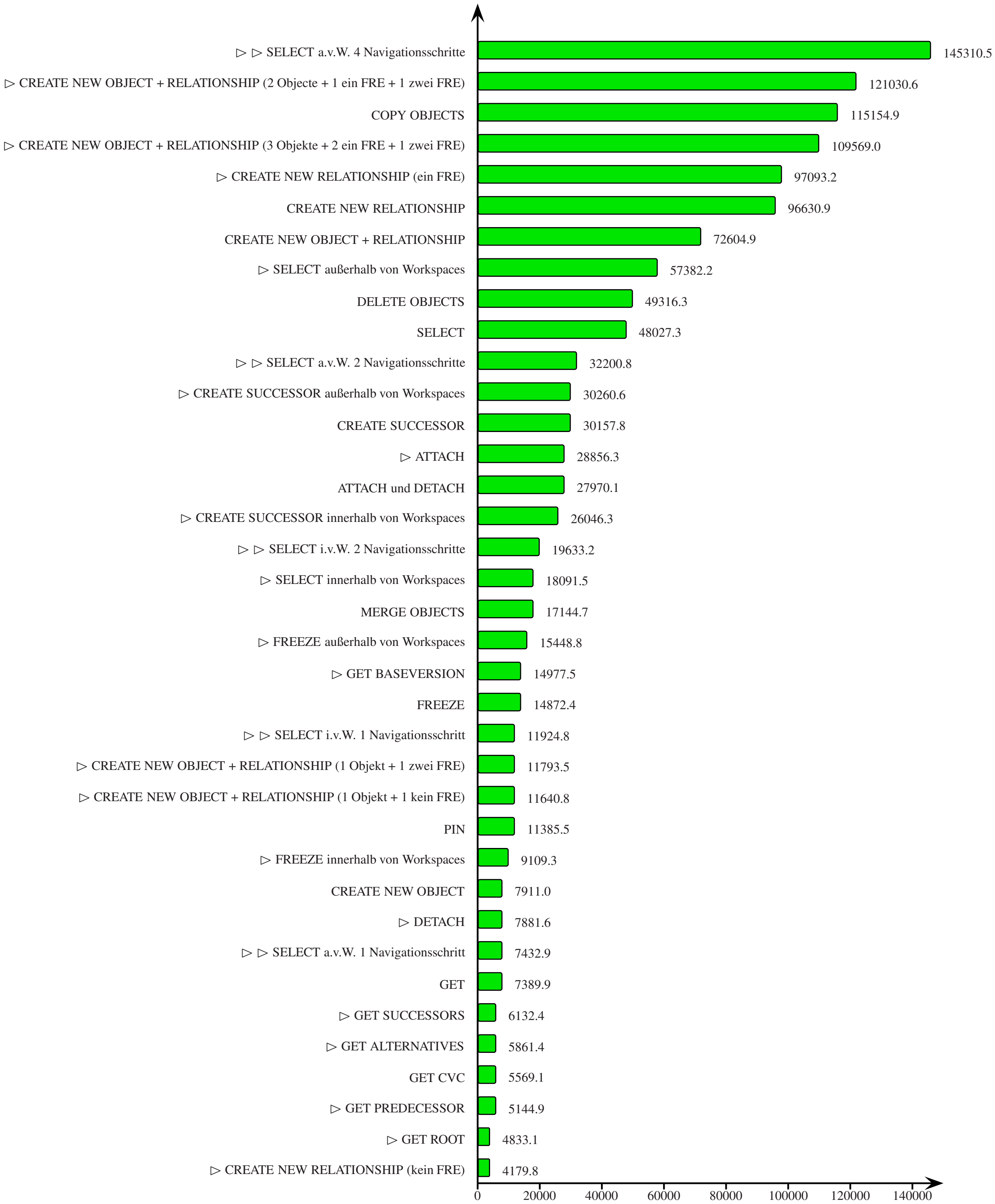


Abbildung H.94 Schema 1 - Reduktion + Rendering + SQL-Ausführung

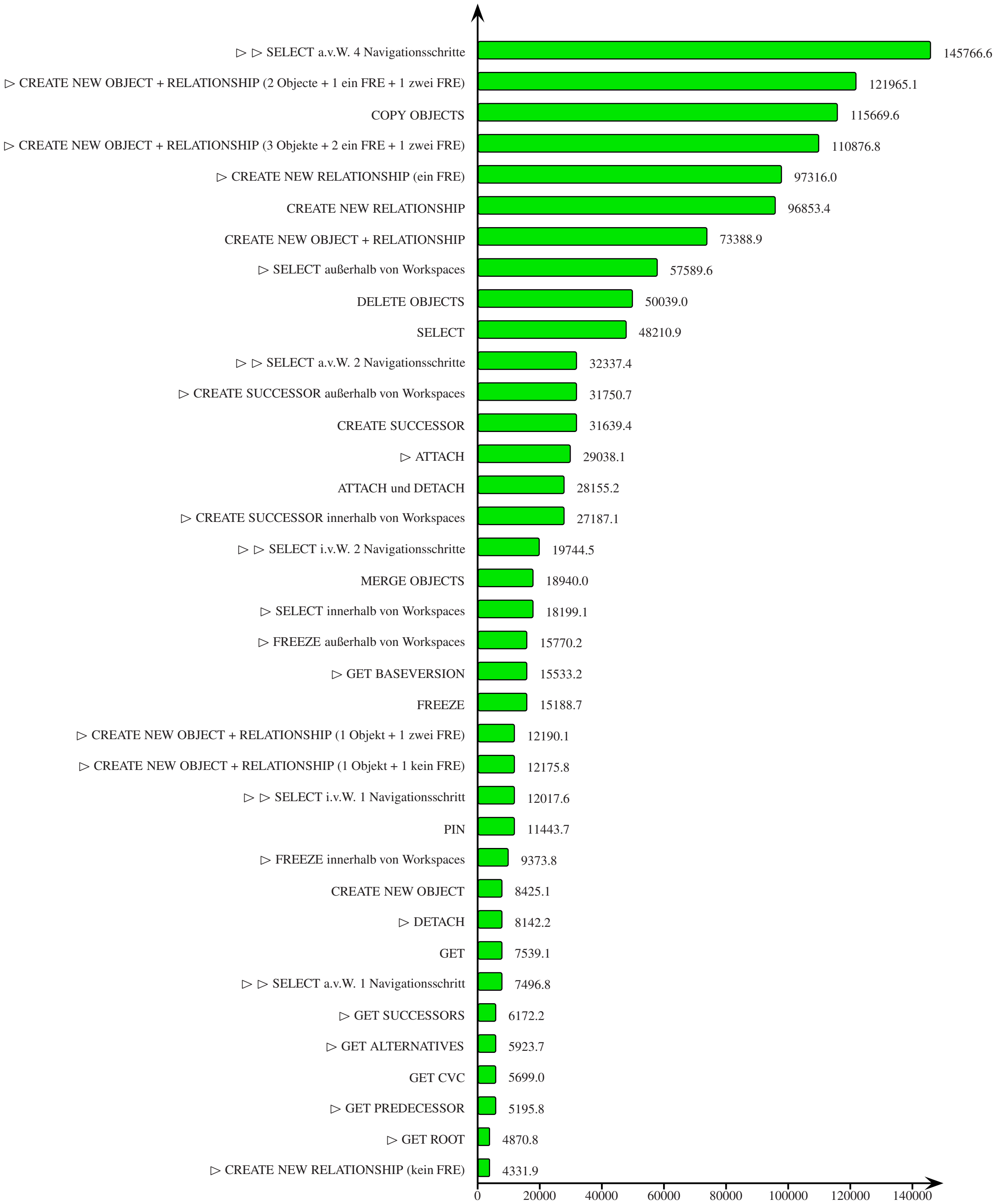


Abbildung H.95 Schema 2 - Reduktion + Rendering

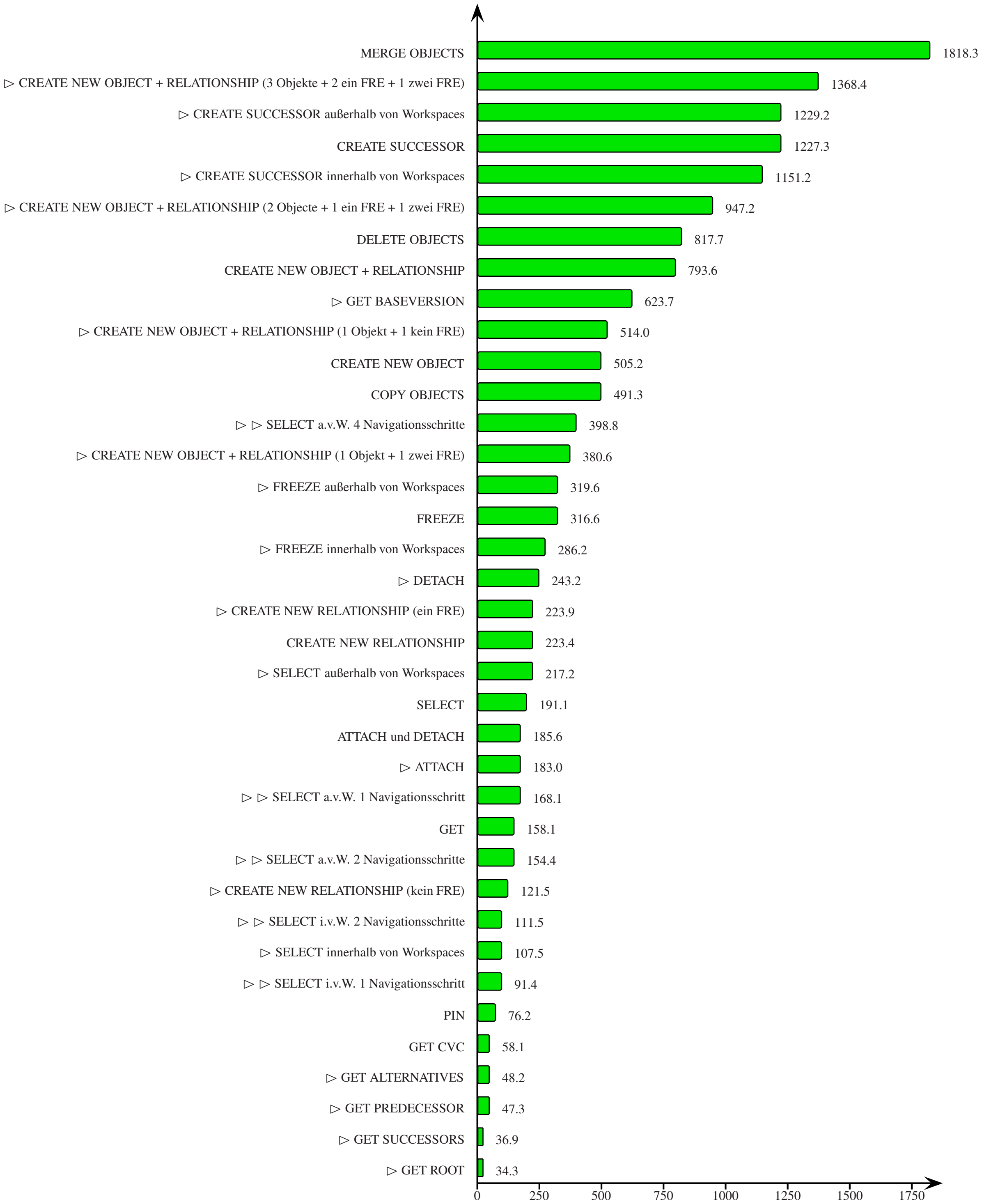


Abbildung H.96 Schema 2 - SQL-Ausführung

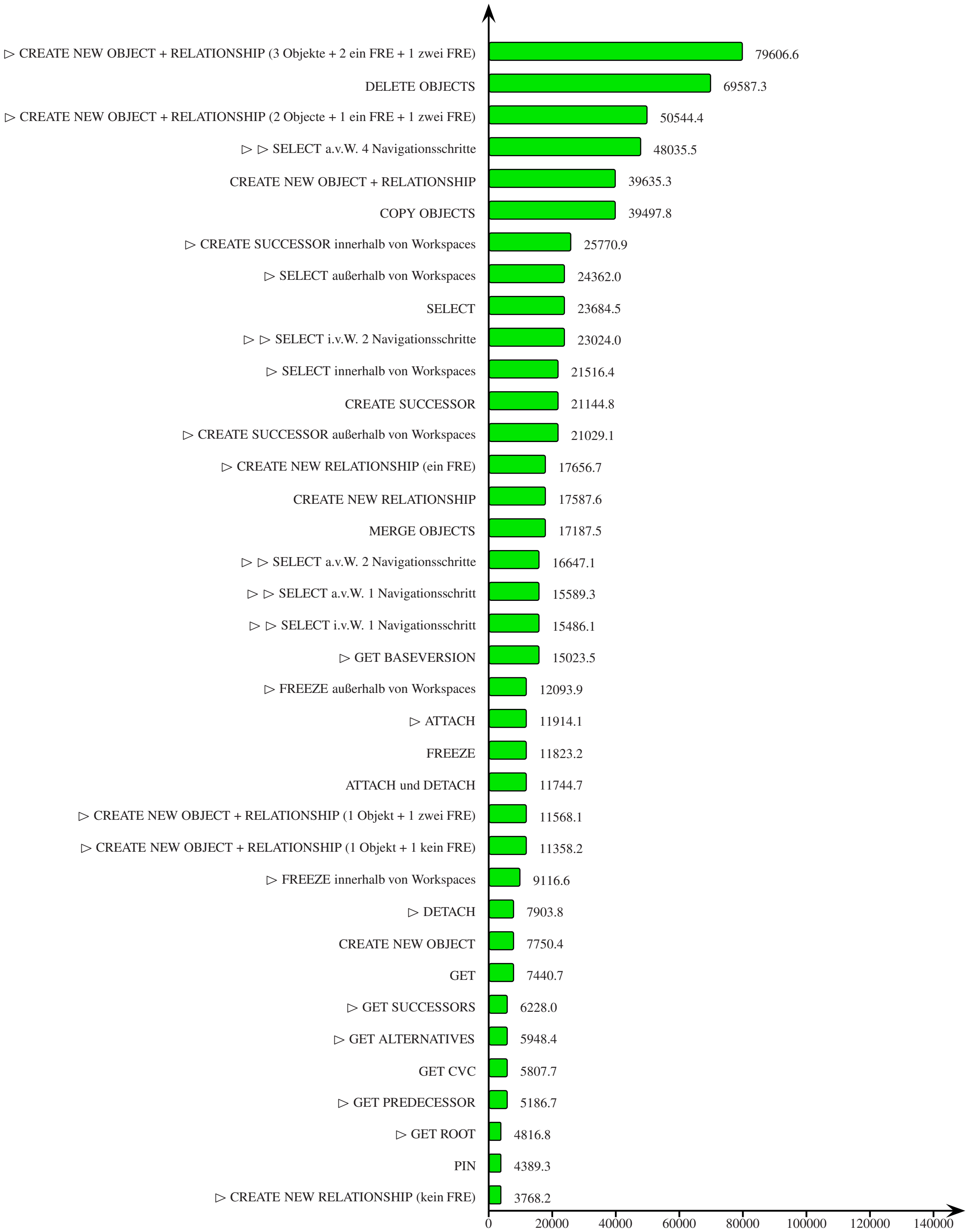


Abbildung H.97 Schema 2 - Reduktion + Rendering + SQL-Ausführung

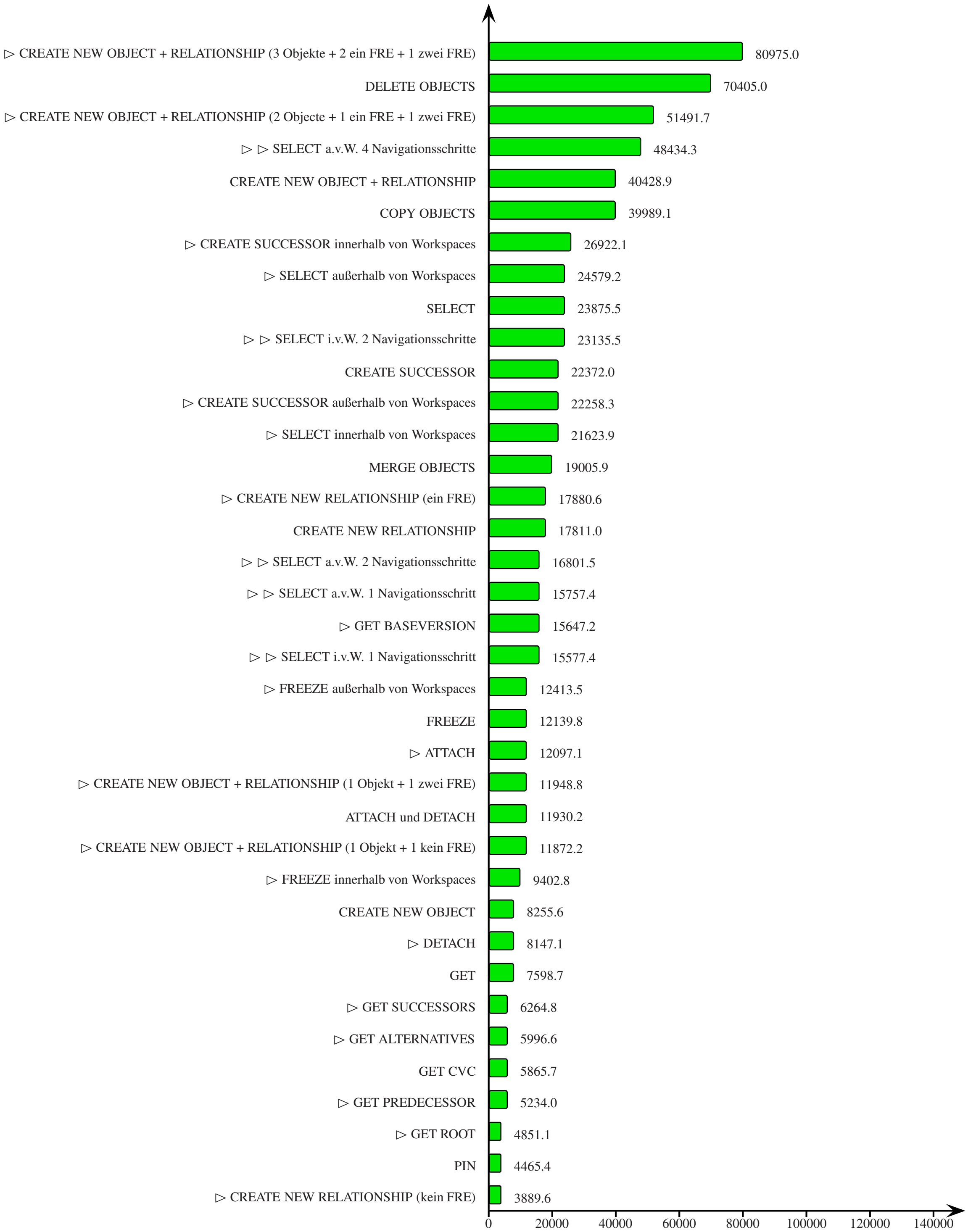


Abbildung H.98 Schema 3 - Reduktion + Rendering

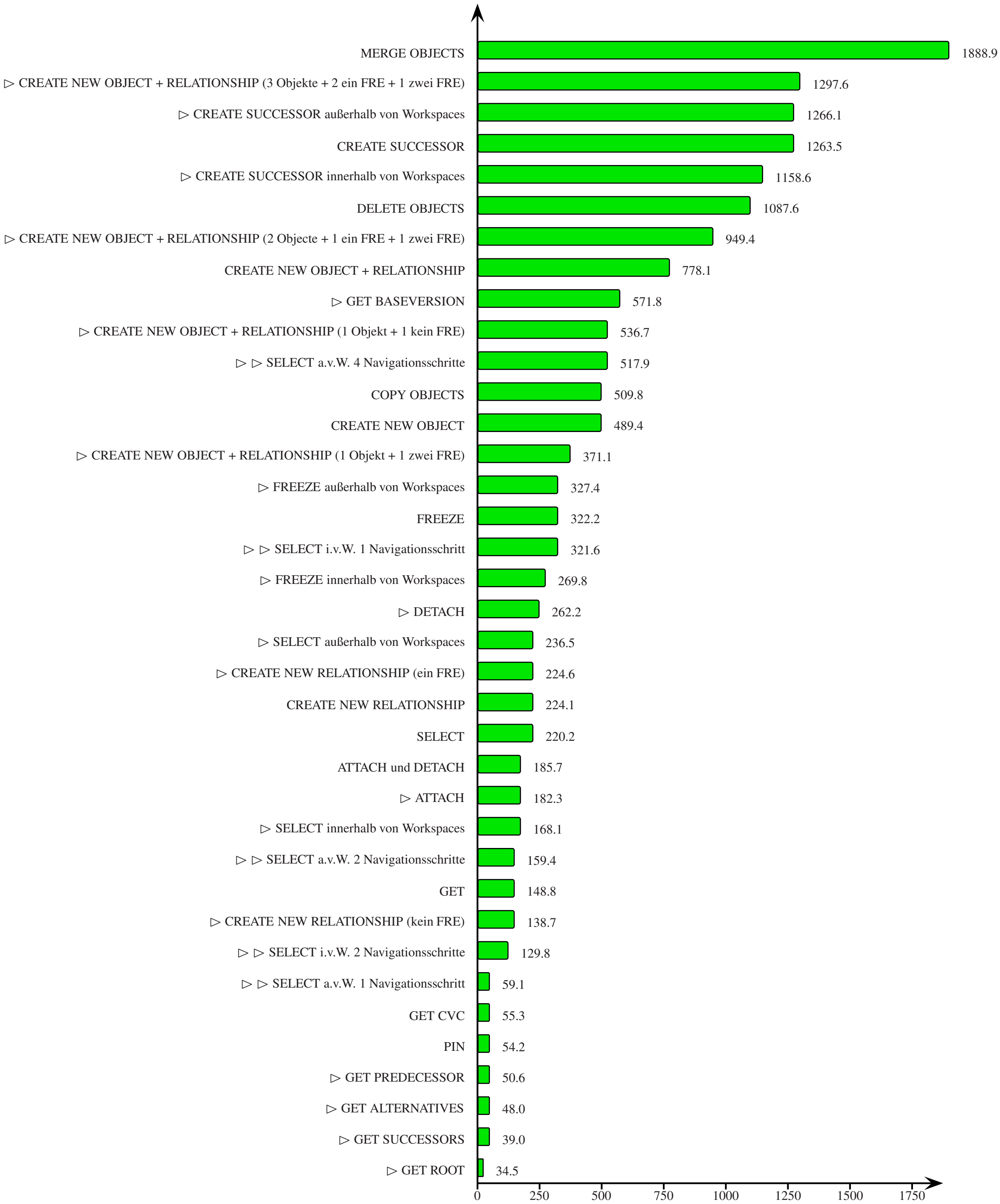


Abbildung H.99 Schema 3 - SQL-Ausführung

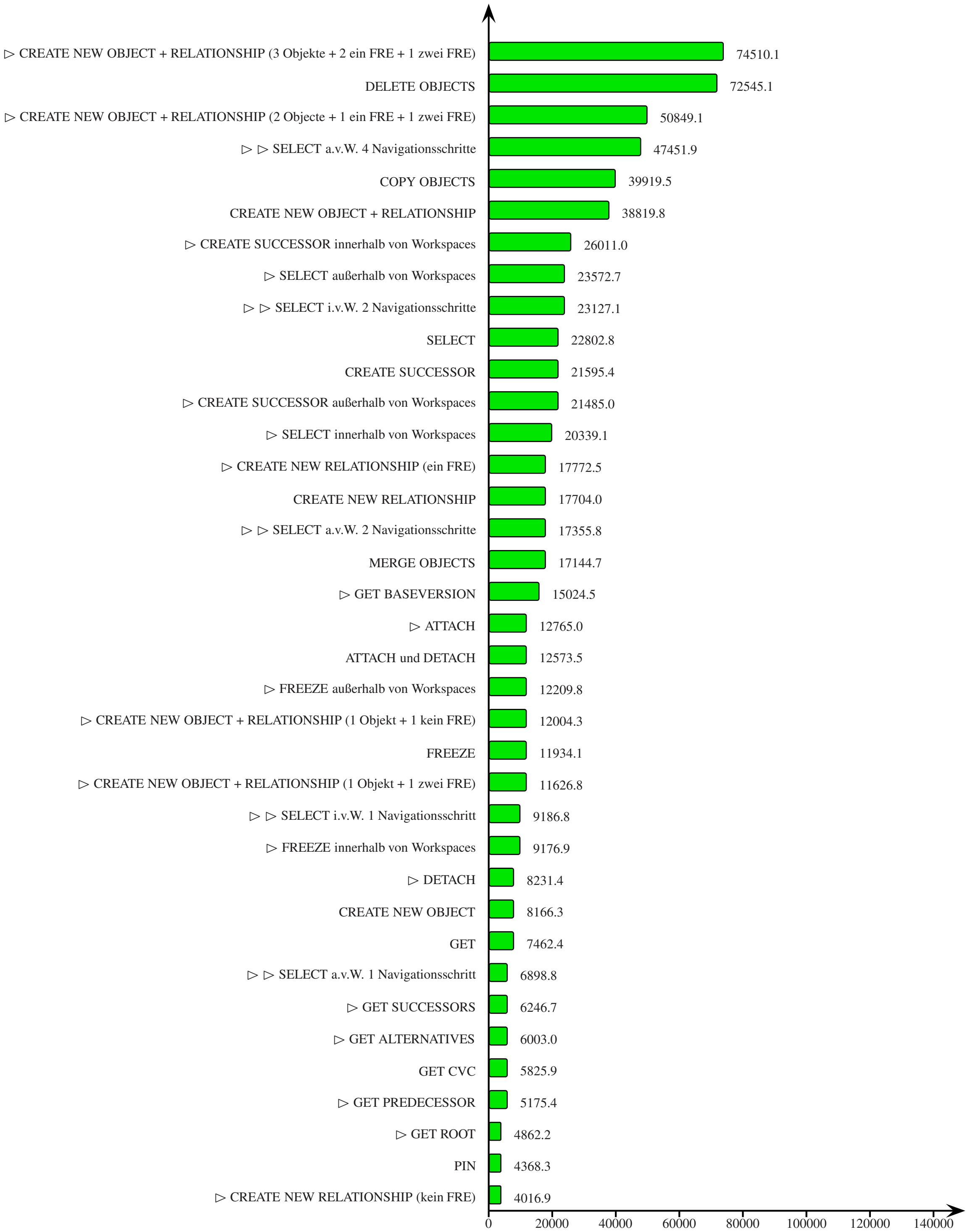


Abbildung H.100 Schema 3 - Reduktion + Rendering + SQL-Ausführung

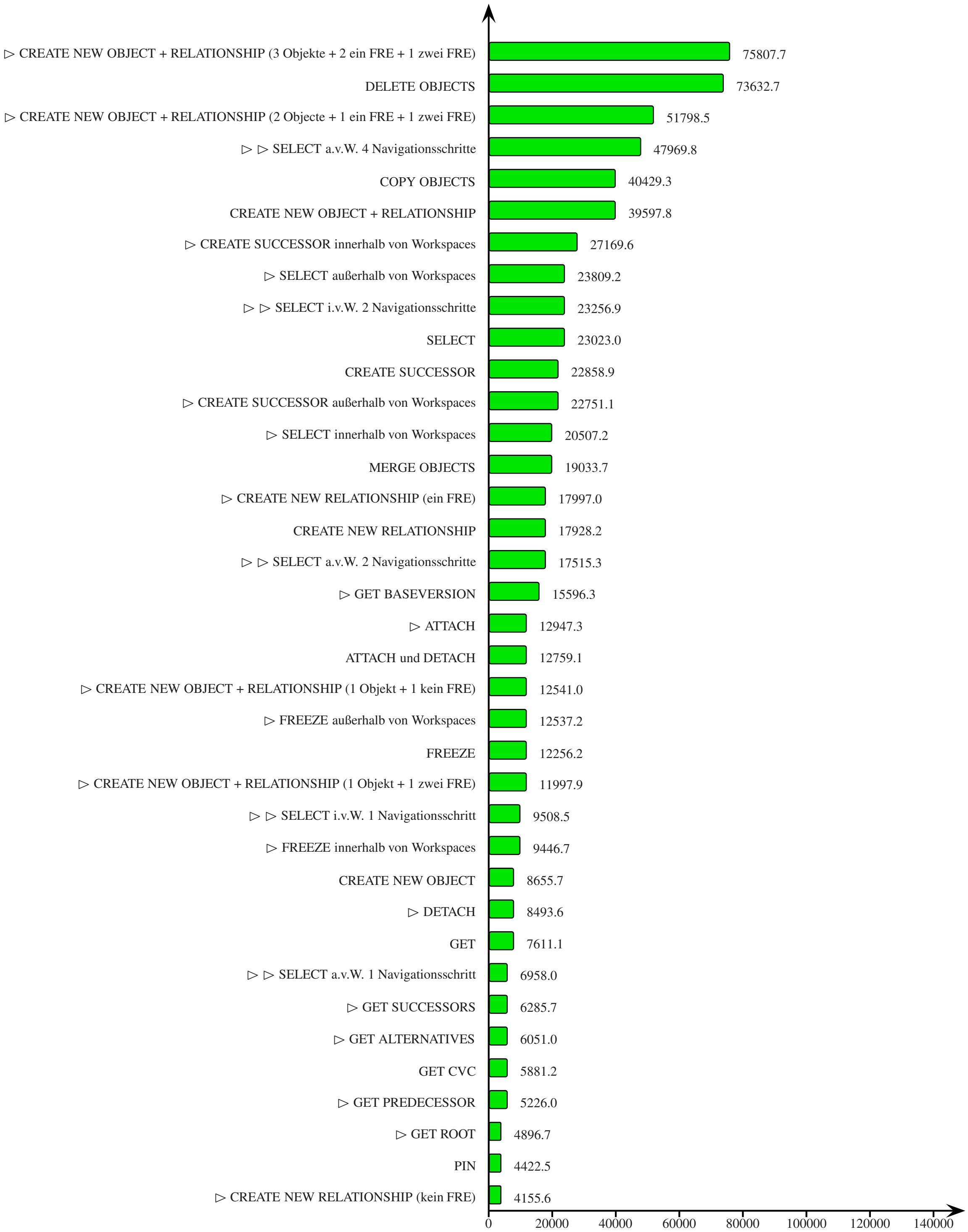


Abbildung H.101 Schema 4 - Reduktion + Rendering

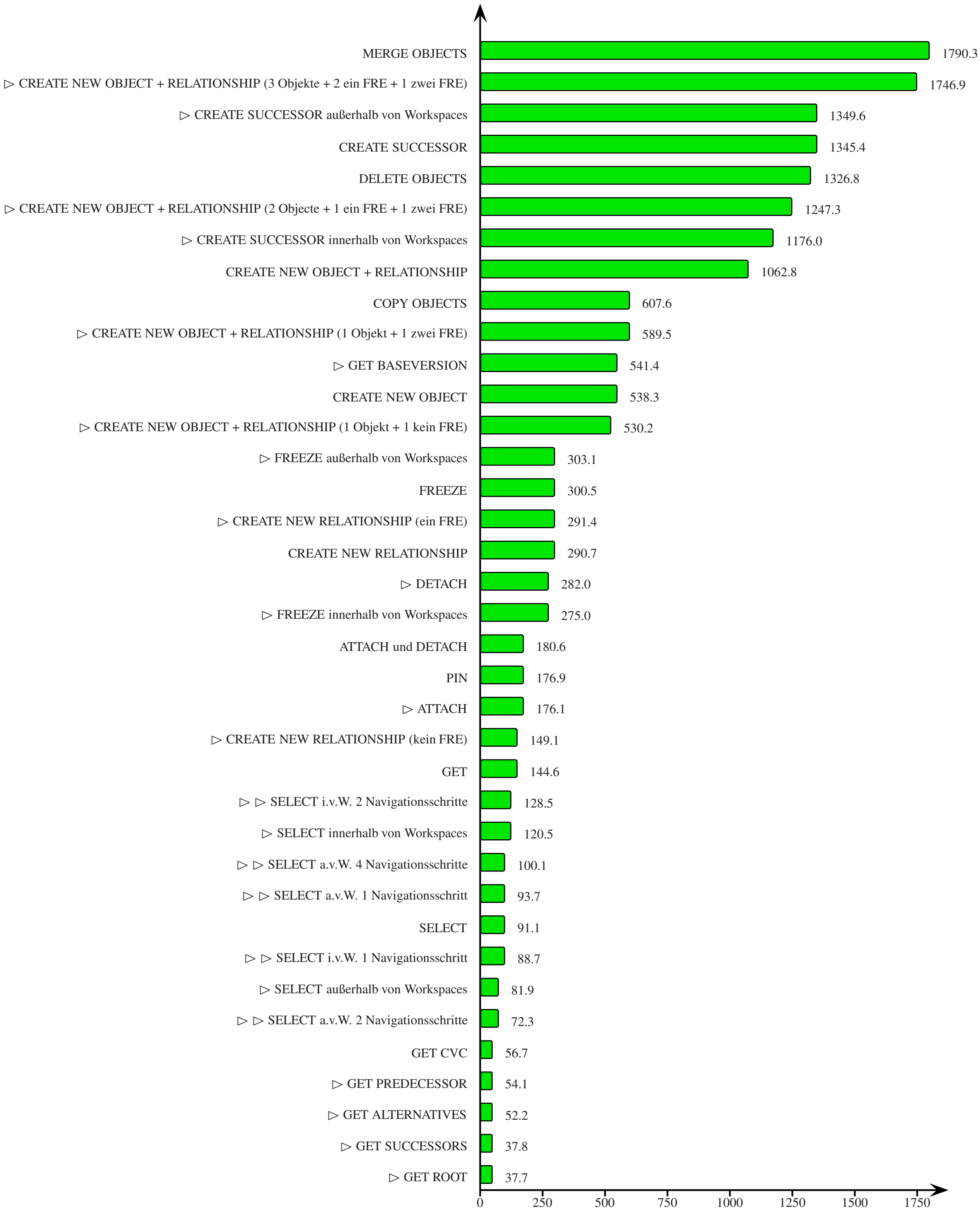


Abbildung H.102 Schema 4 - SQL-Ausführung

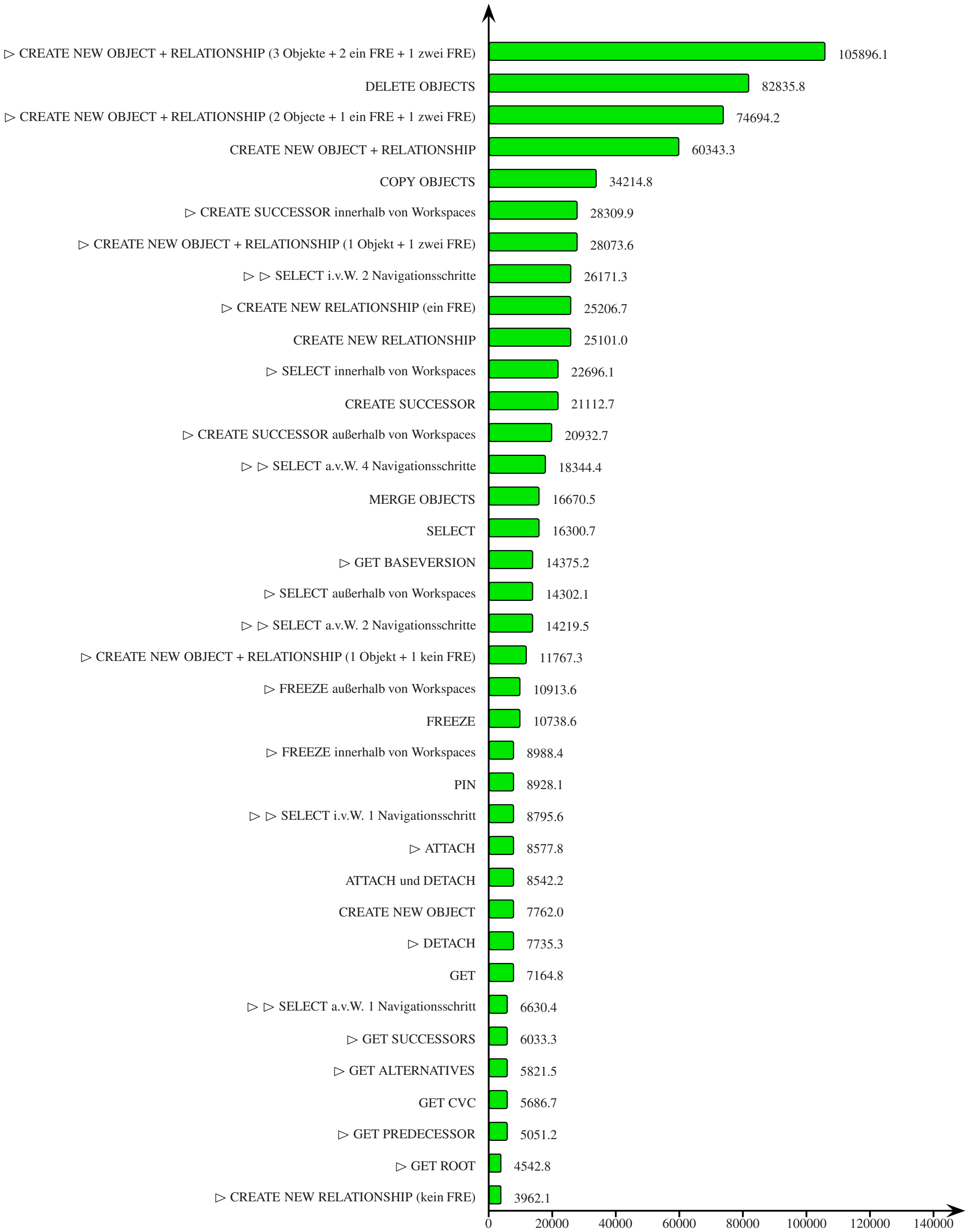
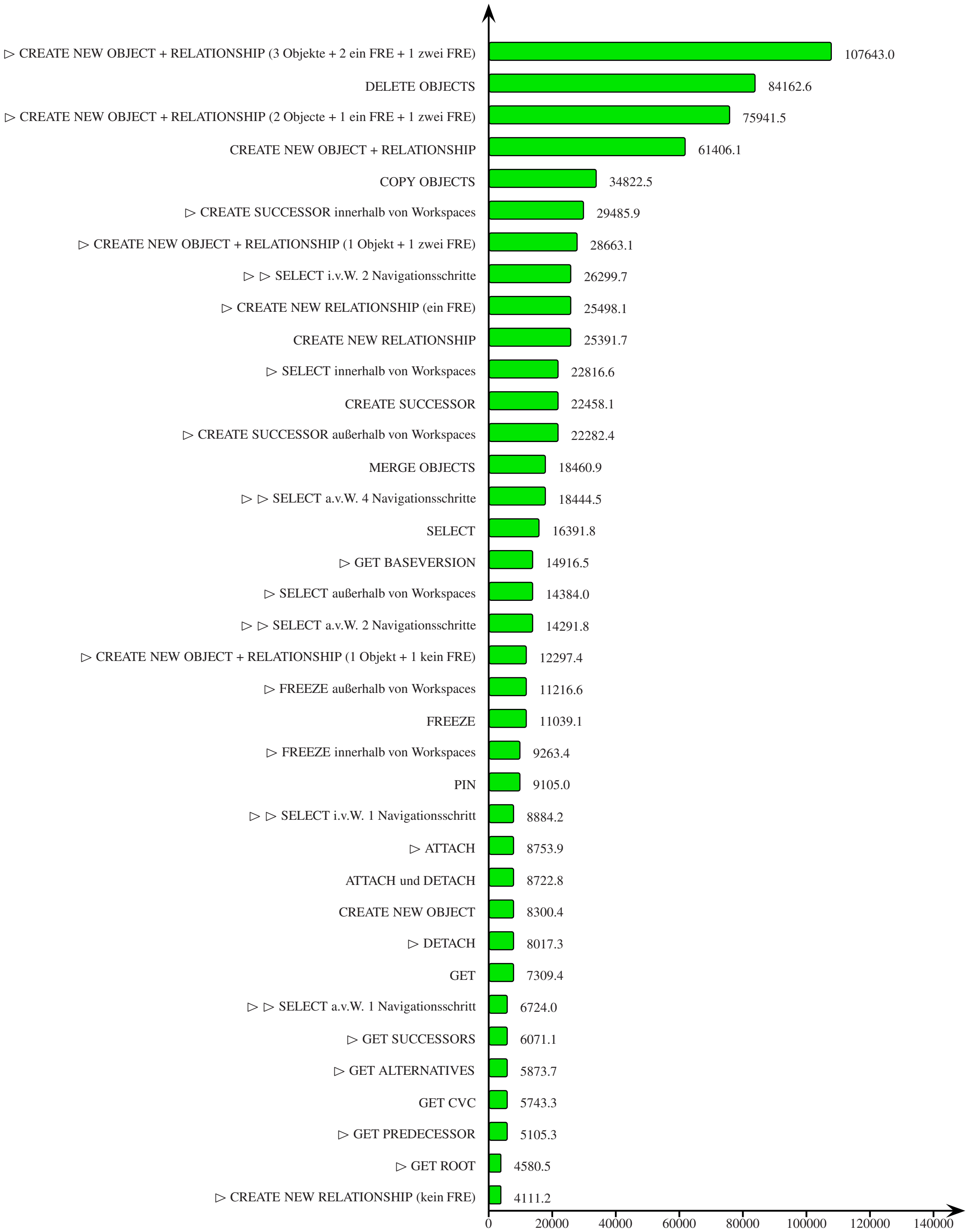


Abbildung H.103 Schema 4 - Reduktion + Rendering + SQL-Ausführung



H.1.4 Vergeich der Kosten für das überprüfen der Kardinalitäten

Abbildung H.104 Zeitvergleich: Summe aller Befehle

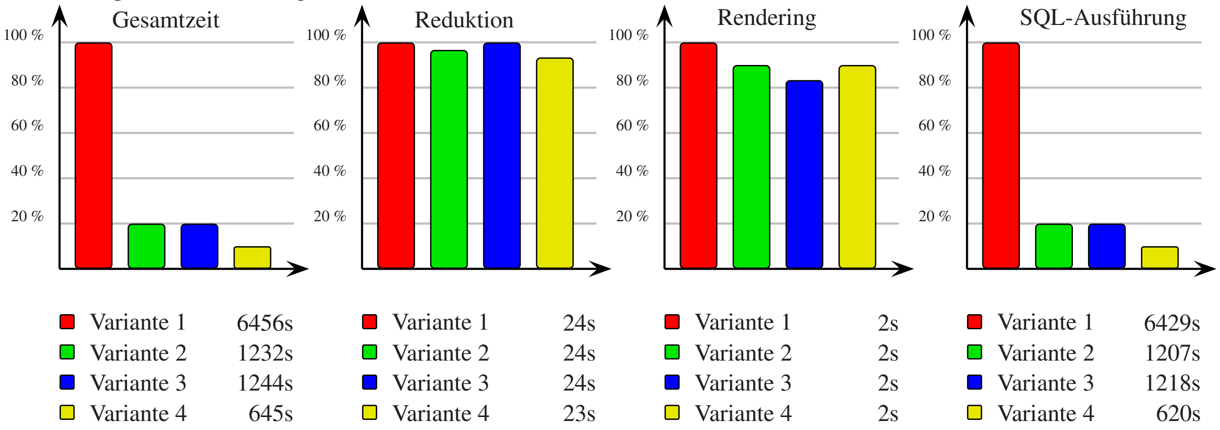


Abbildung H.105 Zeitvergleich: CREATE NEW OBJECT

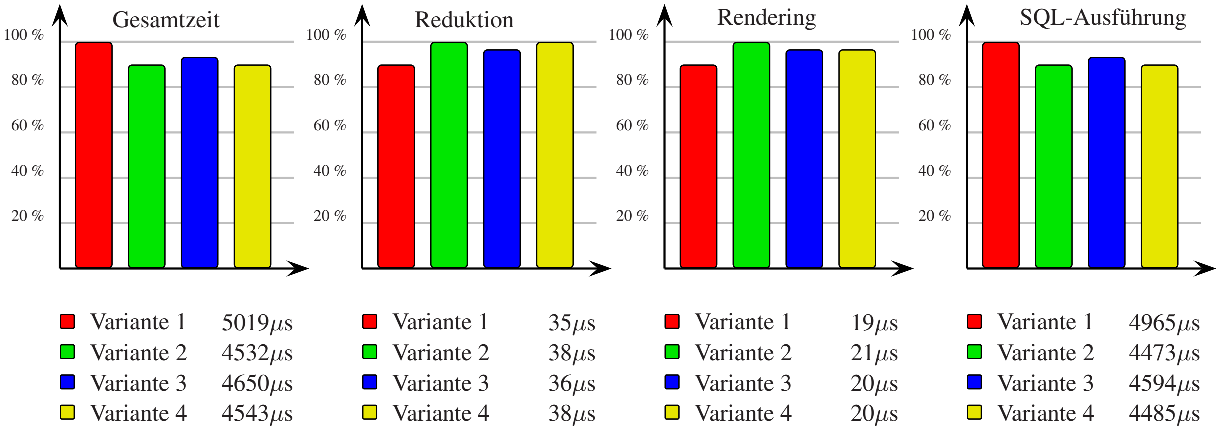


Abbildung H.106 Zeitvergleich: CREATE NEW RELATIONSHIP

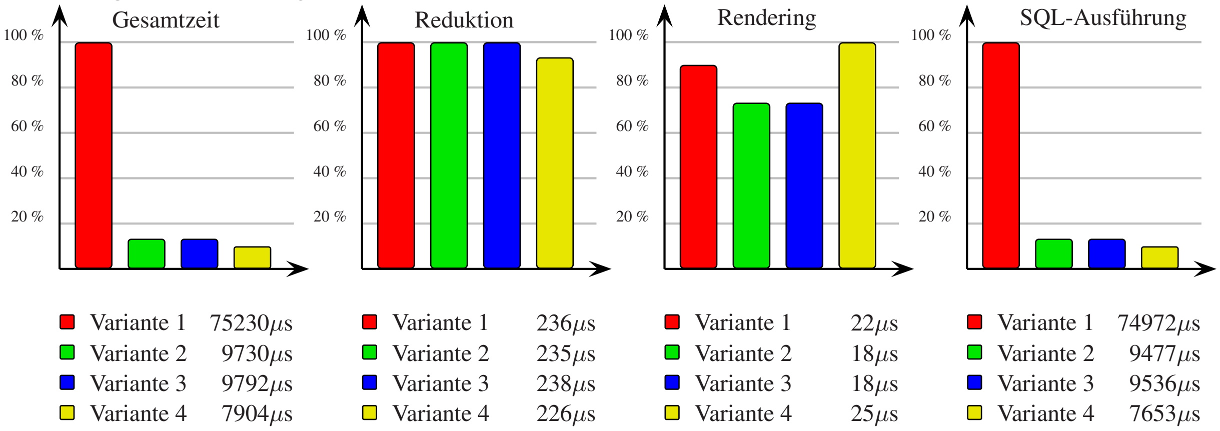


Abbildung H.107 Zeitvergleich: CREATE NEW RELATIONSHIP (kein FRE)

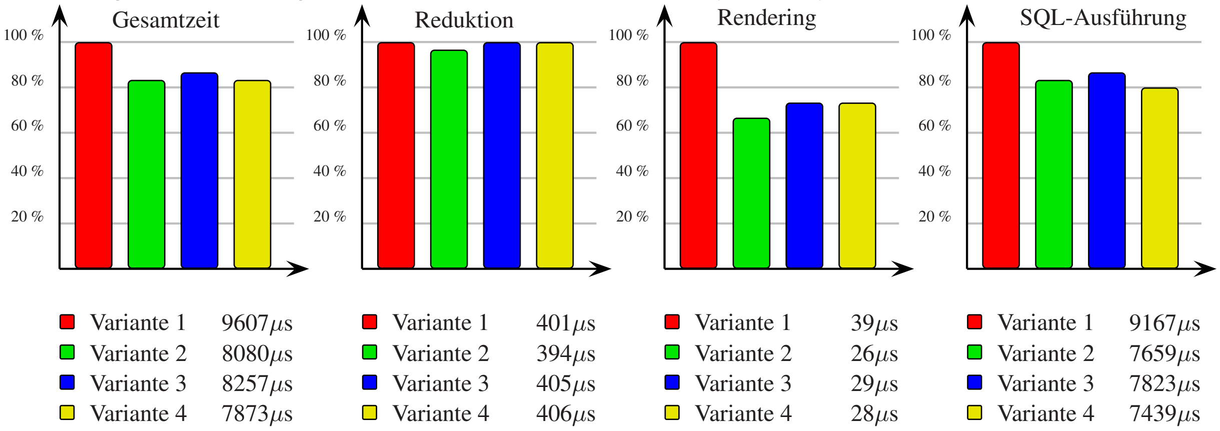


Abbildung H.108 Zeitvergleich: CREATE NEW RELATIONSHIP (ein FRE)

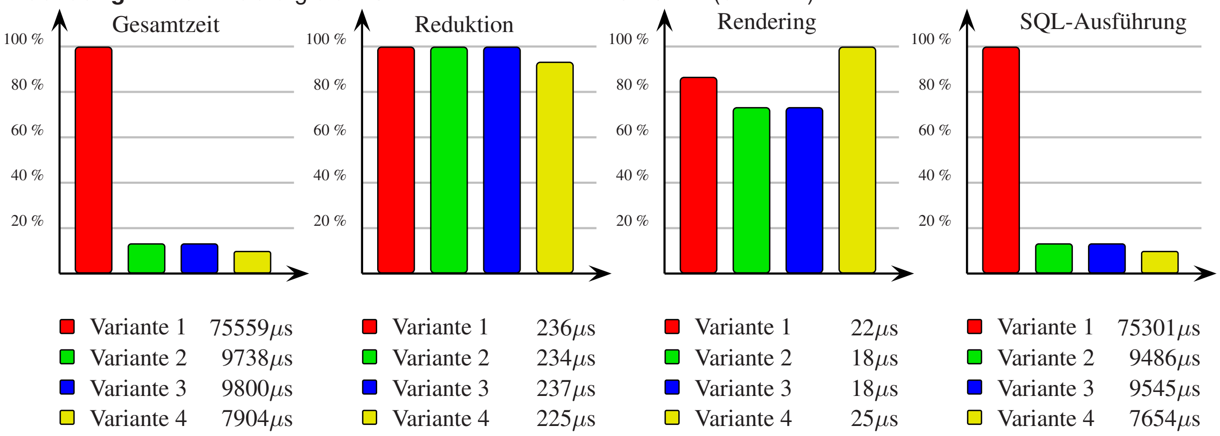


Abbildung H.109 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP

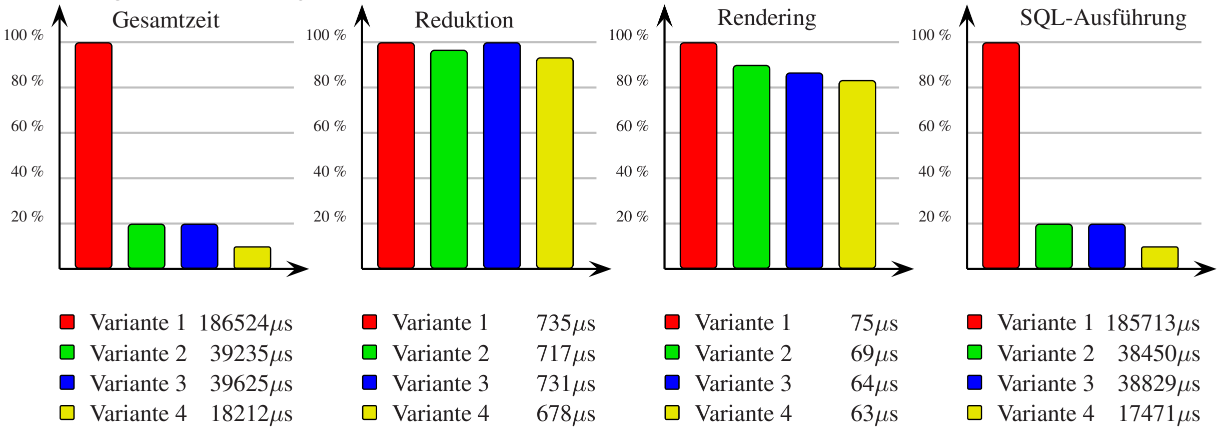


Abbildung H.110 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 kein FRE)

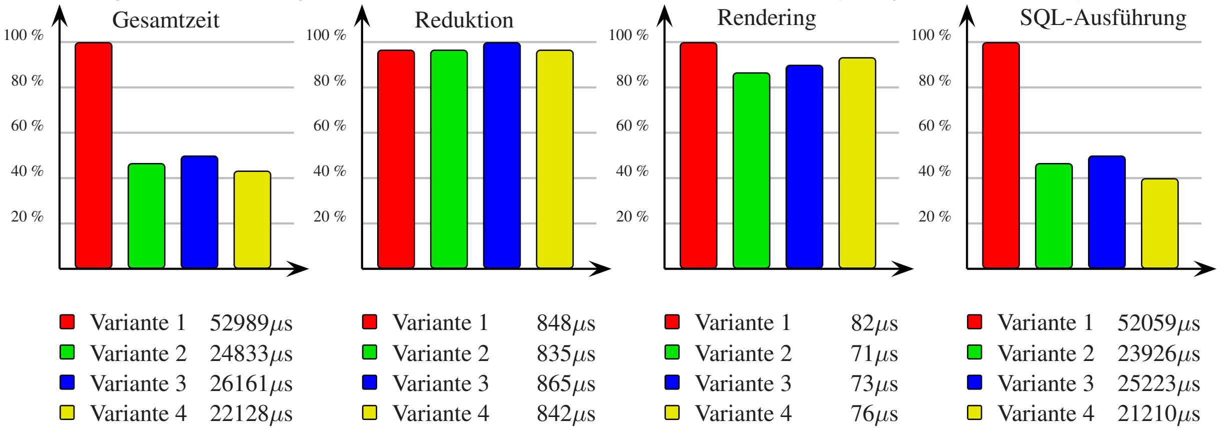


Abbildung H.111 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 zwei FRE)

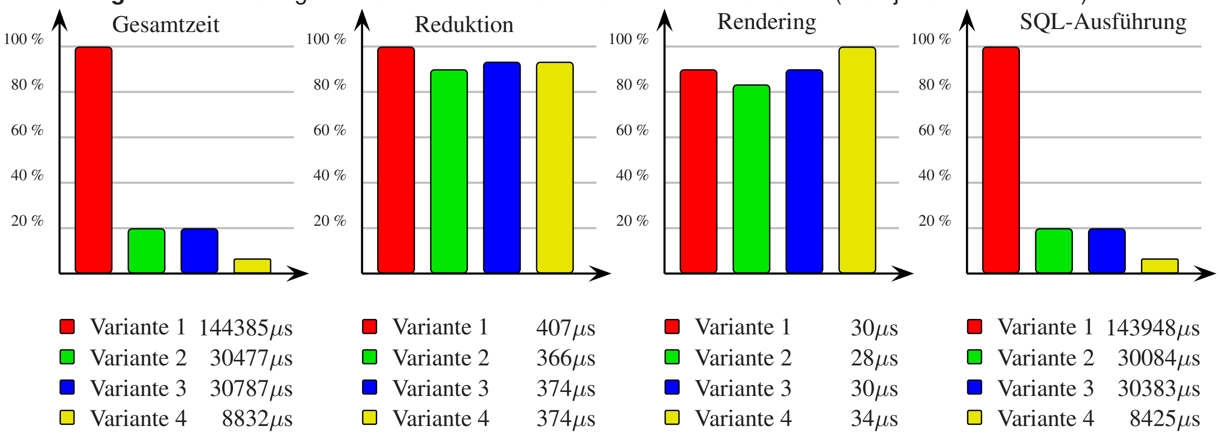


Abbildung H.112 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (2 Objecte + 1 ein FRE + 1 zwei FRE)

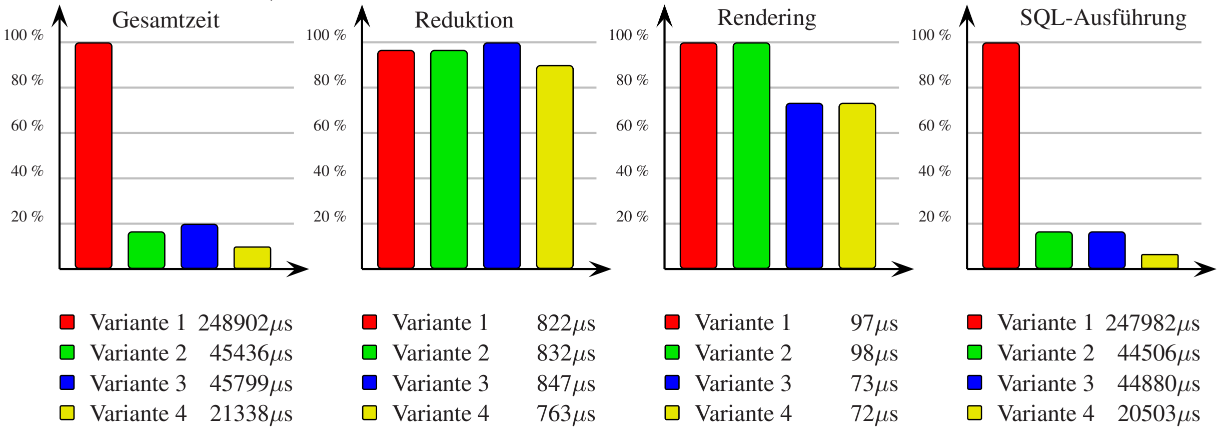


Abbildung H.113 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (3 Objekte + 2 ein FRE + 1 zwei FRE)

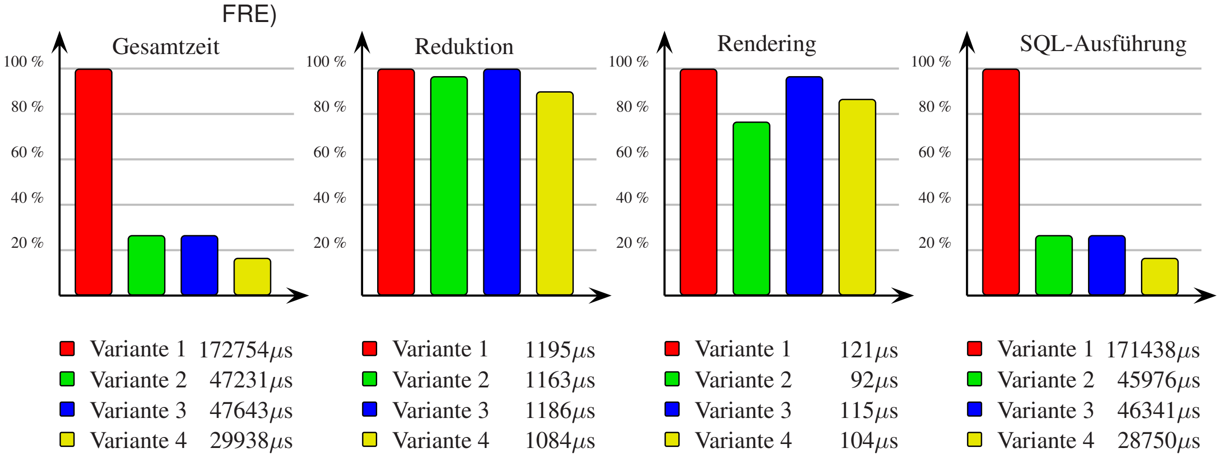
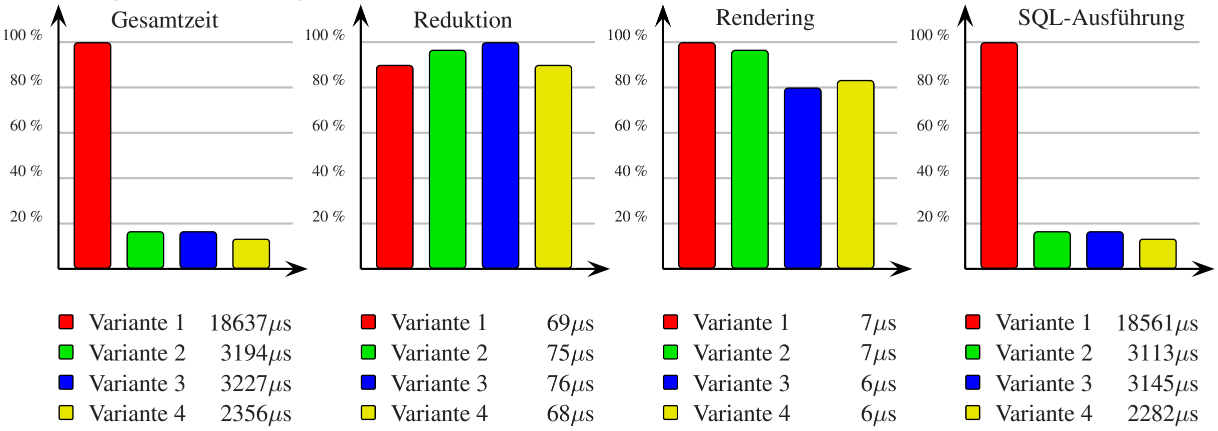


Abbildung H.114 Zeitvergleich: DELETE OBJECTS



H.1.5 Verhältnis der Kosten für die Befehlsausführung und die Überprüfung von Kardinalitäten

Abbildung H.115 Kostenanteile: Summe aller Befehle

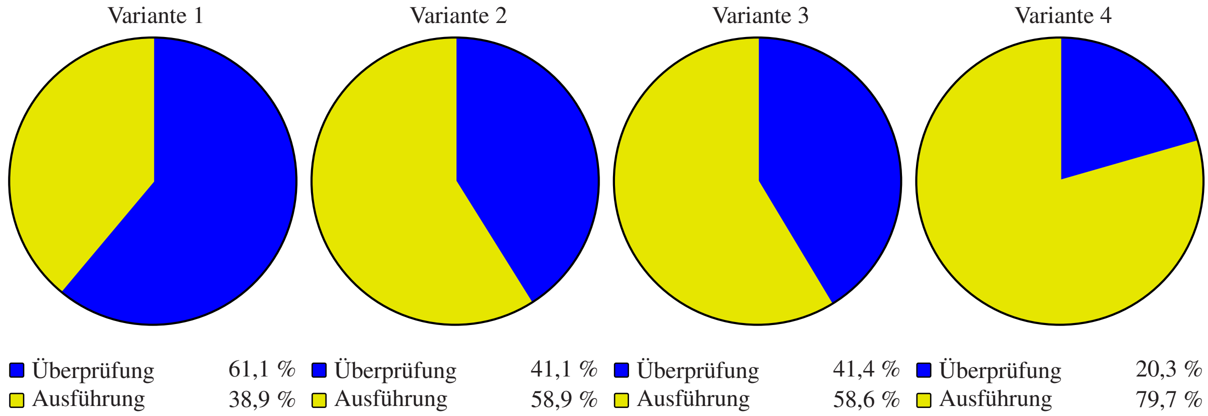


Abbildung H.116 Kostenanteile: CREATE NEW OBJECT

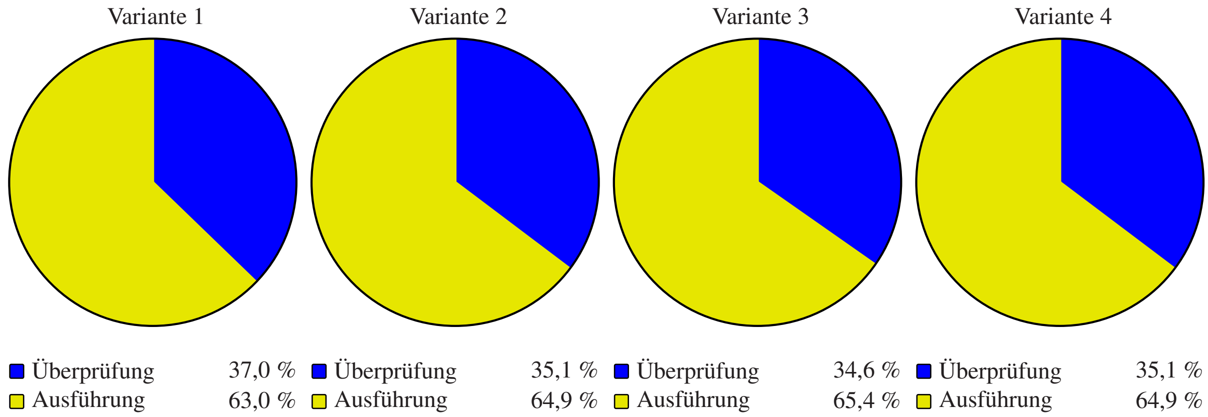


Abbildung H.117 Kostenanteile: CREATE NEW RELATIONSHIP

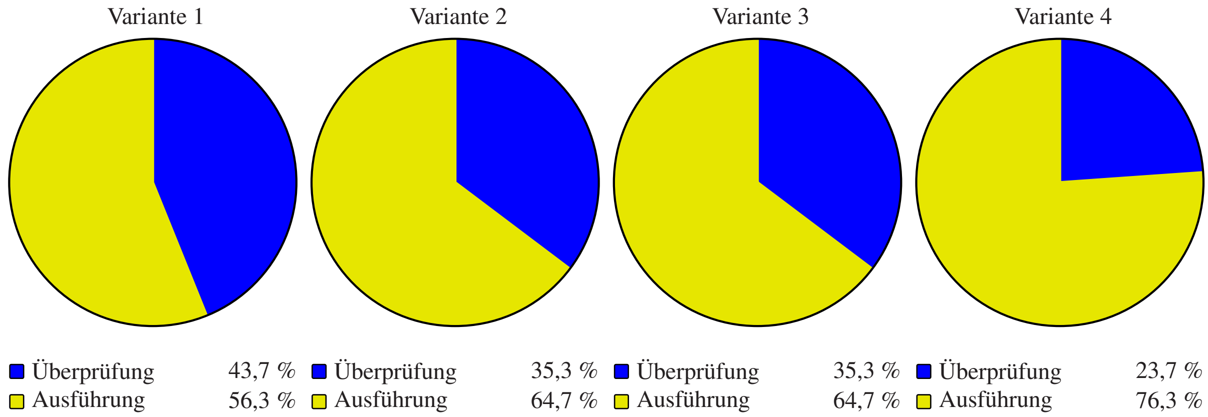


Abbildung H.118 Kostenanteile: CREATE NEW RELATIONSHIP (kein FRE)

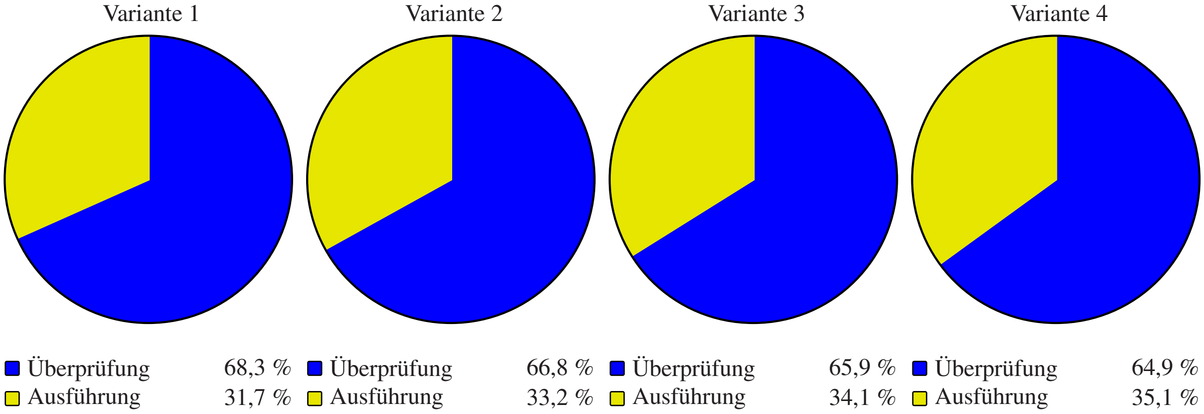


Abbildung H.119 Kostenanteile: CREATE NEW RELATIONSHIP (ein FRE)

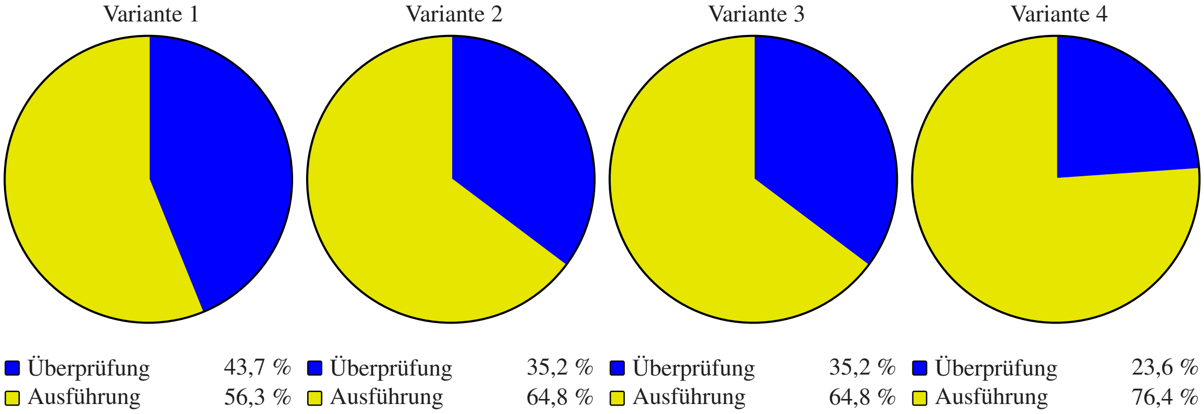


Abbildung H.120 Kostenanteile: CREATE NEW OBJECT + RELATIONSHIP

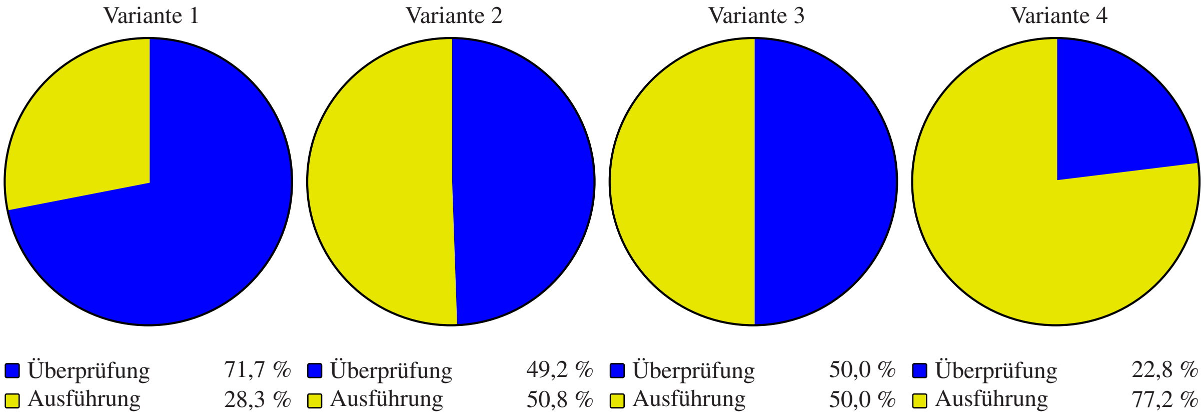


Abbildung H.121 Kostenanteile: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 kein FRE)

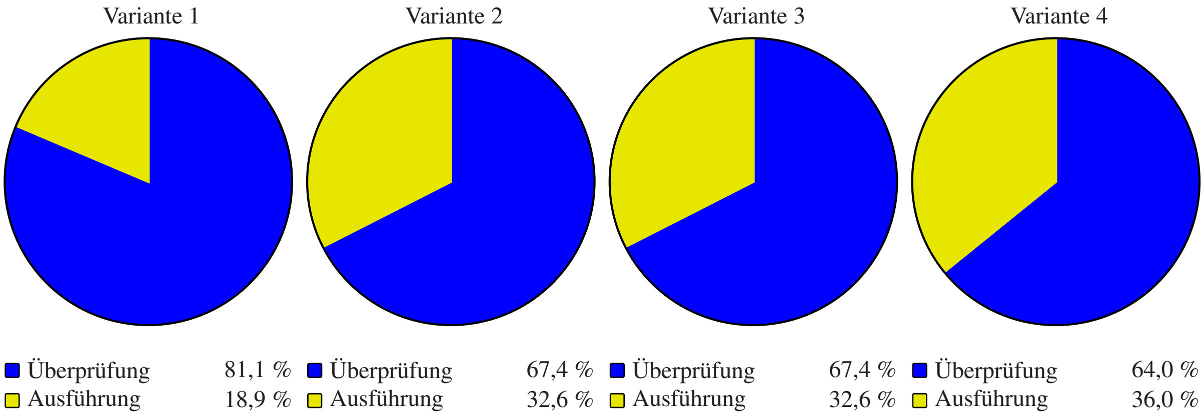


Abbildung H.122 Kostenanteile: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 zwei FRE)

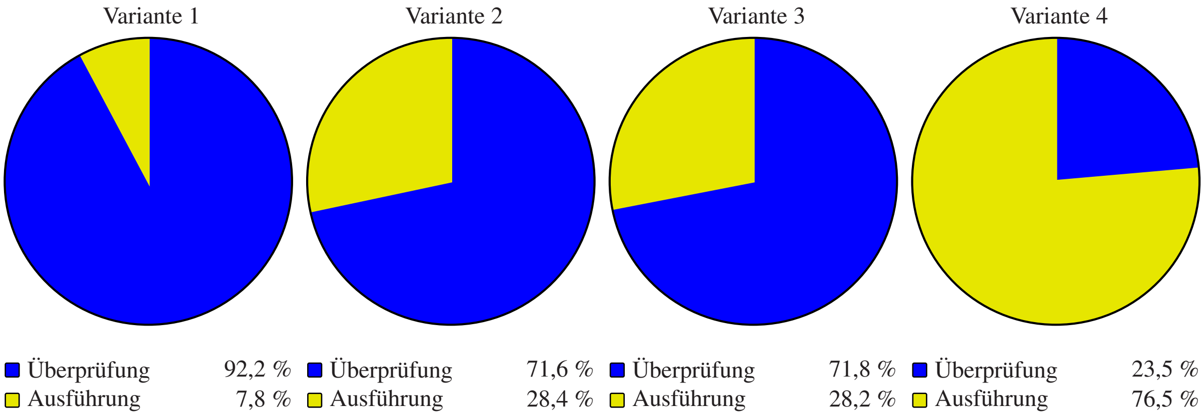


Abbildung H.123 Kostenanteile: CREATE NEW OBJECT + RELATIONSHIP (2 Objecte + 1 ein FRE + 1 zwei FRE)

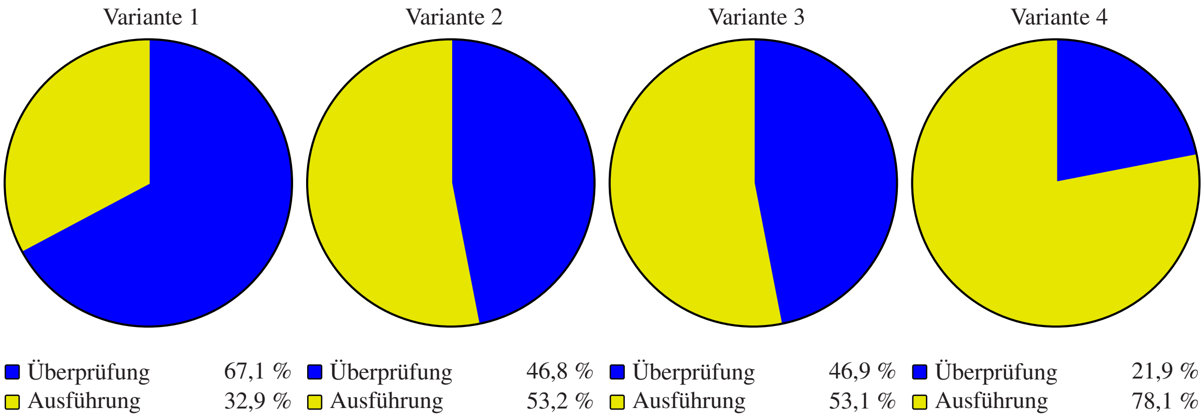


Abbildung H.124 Kostenanteile: CREATE NEW OBJECT + RELATIONSHIP (3 Objekte + 2 ein FRE + 1 zwei FRE)

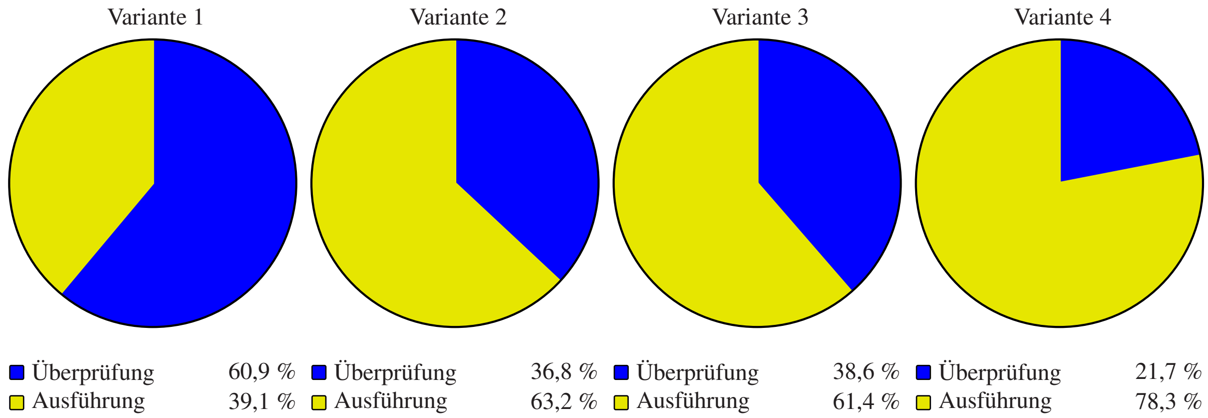
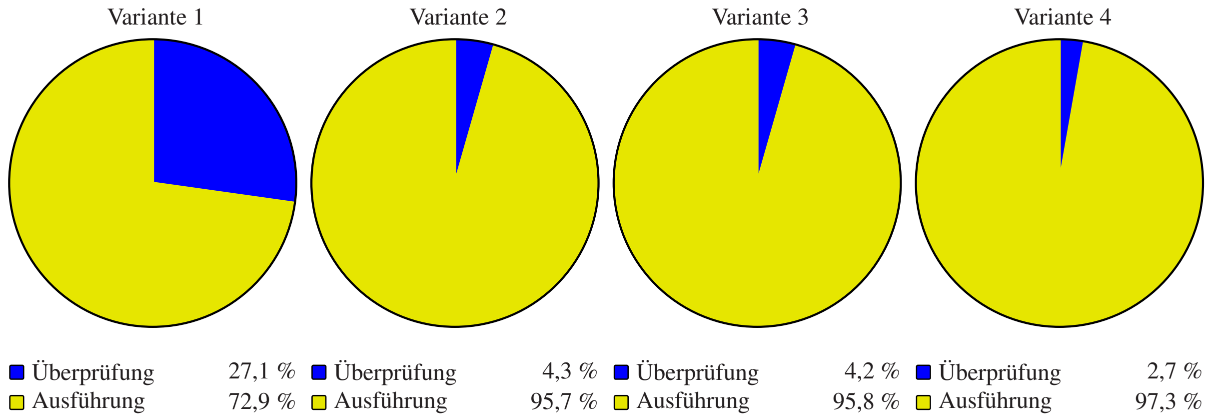


Abbildung H.125 Kostenanteile: DELETE OBJECTS



Anhang I Untersuchungen mit gespeicherten Prozeduren

I.1 Messwerte

I.1.1 Variante 4

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
alle Befehle	SUM	1012804497	1651361	18278269	4581195	988293672	264138
CREATE NEW RELATIONSHIP	SUM	137904	3698	5920	374	127912	30
	MIN	3871	98	141	11	3621	1
	MAX	15570	720	1434	15	13401	1
	MEDIAN	4242	101	157	13	3971	1
	AVG	4597	123	197	12	4264	1
	STDDEV	2042	111	230	1	1700	0
CREATE NEW OBJECT + RELATIONSHIP	SUM	524418138	956870	7559926	2388536	513512806	134500
	MIN	12041	69	395	69	11508	5
	MAX	413261	14604	23702	62157	312798	26
	MEDIAN	71095	99	1016	249	69731	18
	AVG	63183	115	911	288	61869	16
	STDDEV	33868	243	497	1122	32006	6
▷ 1 Objekt + 1 kein FRE	SUM	3858649	85129	148973	25802	3598745	1500
	MIN	12072	93	402	69	11508	5
	MAX	39777	14604	2559	1372	21242	5
	MEDIAN	12396	140	458	75	11723	5
	AVG	12862	284	497	86	11996	5
	STDDEV	2494	1105	204	93	1092	0
▷ 1 Objekt + 1 zwei FRE	SUM	85125517	281405	1395072	447466	83001574	27000
	MIN	27272	69	395	124	26684	9
	MAX	133719	2487	2742	2334	126156	9
	MEDIAN	27989	75	454	136	27324	9
	AVG	28375	94	465	149	27667	9
	STDDEV	2731	123	121	115	2373	0
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	220652770	347261	3199448	1004227	216101834	54000
	MIN	62559	80	868	232	61379	18
	MAX	402450	3793	23702	62157	312798	18
	MEDIAN	73385	104	1038	258	71985	18
	AVG	73551	116	1066	335	72034	18
	STDDEV	8064	132	459	1639	5834	0
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	214781202	243075	2816433	911041	210810653	52000
	MIN	99796	88	1199	353	98156	26
	MAX	190918	2197	4894	38237	145590	26
	MEDIAN	107118	112	1342	380	105284	26
	AVG	107391	122	1408	456	105405	26
	STDDEV	5073	93	266	1054	3661	0
CREATE SUCCESSOR	SUM	148637076	168609	6126001	635647	141706819	41595
	MIN	7668	15	709	50	6894	5
	MAX	1091871	6233	31232	72641	981765	270
	MEDIAN	17450	25	775	59	16591	5
	AVG	24773	28	1021	106	23618	6
	STDDEV	33618	115	1066	952	31484	8
COPY OBJECTS	SUM	210290567	130344	2919089	979118	206262016	59493
	MIN	8092	13	103	51	7925	4
	MAX	604920	5209	6736	32589	560386	131
	MEDIAN	8723	22	132	64	8505	4
	AVG	35048	22	487	163	34377	9

		Summe	Parser	Reduction	Rendering	SQL-Ausführung	SQL
	STDDEV	47319	79	626	470	46144	10
UPDATE OBJECTS	SUM	7108386	69759	166678	30705	6841244	2000
	MIN	3793	46	141	26	3580	2
	MAX	28308	6495	1741	885	19187	2
	MEDIAN	7005	50	156	28	6771	2
	AVG	7108	70	167	31	6841	2
	STDDEV	972	235	100	46	591	0
GET ...	SUM	3531985	24907	100472	18884	3387722	700
	MIN	746	23	22	14	687	1
	MAX	45121	2854	23927	1091	17249	3
	MEDIAN	5275	26	31	18	5200	1
	AVG	7064	50	201	38	6775	1
	STDDEV	5760	204	1141	76	4339	0
▷ ROOT	SUM	429681	5049	2472	2001	420159	100
	MIN	746	23	22	14	687	1
	MAX	7634	1645	154	456	5379	1
	MEDIAN	5030	25	23	15	4967	1
	AVG	4297	50	25	20	4202	1
	STDDEV	1929	182	13	44	1689	0
▷ ALTERNATIVES	SUM	537193	4452	3280	2823	526638	100
	MIN	875	24	30	16	805	1
	MAX	8934	1809	164	994	5967	1
	MEDIAN	5451	26	31	18	5376	1
	AVG	5372	45	33	28	5266	1
	STDDEV	1078	177	13	97	790	0
▷ SUCCESSORS	SUM	501760	4803	27262	1848	467847	100
	MIN	817	23	30	17	747	1
	MAX	31917	1804	23927	32	6154	1
	MEDIAN	5277	26	32	18	5201	1
	AVG	5018	48	273	18	4678	1
	STDDEV	4074	180	2377	2	1514	0
▷ PREDECESSOR	SUM	482660	4507	3985	2661	471507	100
	MIN	802	24	30	16	732	1
	MAX	8852	1817	640	474	5921	1
	MEDIAN	5197	26	31	18	5122	1
	AVG	4827	45	40	27	4715	1
	STDDEV	1622	178	63	58	1323	0
▷ BASEVERSION	SUM	1580691	6096	63473	9551	1501571	300
	MIN	11289	31	351	81	10826	3
	MAX	28729	2854	7535	1091	17249	3
	MEDIAN	15797	32	503	84	15178	3
	AVG	15807	61	635	96	15016	3
	STDDEV	2220	281	759	101	1080	0
GET CVC	SUM	1710001	11988	15845	5572	1676596	300
	MIN	874	24	43	16	791	1
	MAX	11434	2920	1354	276	6884	1
	MEDIAN	5631	27	46	17	5541	1
	AVG	5700	40	53	19	5589	1
	STDDEV	790	176	79	15	520	0
SELECT	SUM	3098230	23603	21196	6377	3047054	250
	MIN	929	42	42	10	835	1
	MAX	34159	10667	1500	968	21024	1
	MEDIAN	12574	49	60	17	12448	1
	AVG	12393	94	85	26	12188	1
	STDDEV	5749	670	144	84	4851	0
▷ 1 Navigationsschritt	SUM	318258	12829	2511	1533	301385	50
	MIN	929	42	42	10	835	1
	MAX	19155	10667	364	968	7156	1
	MEDIAN	6308	44	44	11	6209	1
	AVG	6365	257	50	31	6028	1
	STDDEV	2732	1487	45	134	1066	0
▷ 2 Navigationsschritte	SUM	1192003	4882	8233	1728	1177160	100
	MIN	1020	46	56	16	902	1
	MAX	17031	72	1225	30	15704	1
	MEDIAN	11879	48	58	17	11756	1
	AVG	11920	49	82	17	11772	1
	STDDEV	2746	3	161	2	2580	0
▷ 4 Navigationsschritte	SUM	1587969	5892	10452	3116	1568509	100
	MIN	1229	54	77	18	1080	1
	MAX	23588	122	1500	942	21024	1
	MEDIAN	12931	56	84	22	12769	1
	AVG	15880	59	105	31	15685	1
	STDDEV	4756	10	155	92	4499	0

I.1.2 Variante 5

		Summe	Parser	Reduction	SQL-Ausführung
alle Befehle	SUM	149993454	1710159	5700549	142582746
CREATE NEW RELATIONSHIP	SUM	65267	3093	5129	57045
	MIN	1795	97	148	1550
	MAX	4585	143	571	3871
	MEDIAN	2074	100	158	1816
	AVG	2176	103	171	1902
	STDDEV	460	9	75	376
CREATE NEW OBJECT + RELATIONSHIP	SUM	67772608	1019802	2875096	63877710
	MIN	3787	68	291	3428
	MAX	259705	68554	32538	158613
	MEDIAN	8188	74	304	7810
	AVG	8165	123	346	7696
	STDDEV	6902	1051	498	5353
▷ 1 Objekt + 1 kein FRE	SUM	1399442	85717	156009	1157716
	MIN	3845	88	329	3428
	MAX	56791	30438	11207	15146
	MEDIAN	4051	94	406	3551
	AVG	4665	286	520	3859
	STDDEV	3844	1801	769	1274
▷ 1 Objekt + 1 zwei FRE	SUM	16254004	301227	1000789	14951988
	MIN	4441	68	291	4082
	MAX	151521	7430	9147	134944
	MEDIAN	4619	71	299	4249
	AVG	5418	100	334	4984
	STDDEV	5037	322	316	4398
▷ 2 Objecte + 1 ein FRE + 1 zwei FRE	SUM	29484241	366008	1026149	28092084
	MIN	6325	70	299	5956
	MAX	190514	40452	11371	138691
	MEDIAN	9225	74	306	8845
	AVG	9828	122	342	9364
	STDDEV	6571	1003	378	5189
▷ 3 Objekte + 2 ein FRE + 1 zwei FRE	SUM	20634921	266850	692149	19675922
	MIN	8031	71	293	7667
	MAX	259705	68554	32538	158613
	MEDIAN	9374	74	299	9001
	AVG	10317	133	346	9838
	STDDEV	7535	1559	756	5220
CREATE SUCCESSOR	SUM	26789570	138509	802195	25848866
	MIN	2061	15	117	1929
	MAX	167590	7316	8964	151310
	MEDIAN	2232	16	122	2094
	AVG	4465	23	134	4308
	STDDEV	6895	136	200	6558
COPY OBJECTS	SUM	29985050	111293	776053	29097704
	MIN	2104	13	116	1975
	MAX	195904	3907	4747	187250
	MEDIAN	2336	14	121	2201
	AVG	4998	19	129	4850
	STDDEV	7376	74	90	7213
UPDATE OBJECTS	SUM	2005072	71656	309747	1623669
	MIN	1710	41	197	1472
	MAX	54144	6431	34316	13397
	MEDIAN	1807	47	206	1554
	AVG	2005	72	310	1624
	STDDEV	2100	299	1198	603
GET ...	SUM	603555	24295	82388	496872
	MIN	707	21	92	594
	MAX	54136	1886	26607	25643
	MEDIAN	860	25	103	732
	AVG	1207	49	165	994
	STDDEV	2632	171	1186	1275
▷ ROOT	SUM	89183	4753	11148	73282
	MIN	732	22	92	618
	MAX	5732	1436	489	3807
	MEDIAN	801	24	103	674
	AVG	892	48	111	733
	STDDEV	612	165	43	404
▷ ALTERNATIVES	SUM	129505	3961	11806	113738
	MIN	753	22	92	639
	MAX	28572	1421	1508	25643
	MEDIAN	950	25	96	829
	AVG	1295	40	118	1137
	STDDEV	2759	139	141	2479
▷ SUCCESSORS	SUM	93295	4421	10810	78064
	MIN	729	22	92	615
	MAX	4975	1484	247	3244
	MEDIAN	882	25	104	753
	AVG	933	44	108	781
	STDDEV	422	149	19	254
▷ PREDECESSOR	SUM	113429	6152	37767	69510
	MIN	707	21	92	594
	MAX	31197	1457	26607	3133

		Summe	Parser	Reduction	SQL-Ausführung
	MEDIAN	790	25	105	660
	AVG	1134	62	378	695
	STDDEV	3103	206	2636	261
▷ BASEVERSION	SUM	178143	5008	10857	162278
	MIN	1437	28	99	1310
	MAX	13608	1886	247	11475
	MEDIAN	1626	31	103	1492
	AVG	1781	50	109	1623
	STDDEV	1237	185	17	1036
GET CVC	SUM	294262	12235	32859	249168
	MIN	716	23	91	602
	MAX	8957	1770	1125	6062
	MEDIAN	804	26	97	681
	AVG	981	41	110	831
	STDDEV	651	152	69	430
SELECT	SUM	470453	26552	64745	379156
	MIN	1345	41	119	1185
	MAX	21545	10864	6508	4173
	MEDIAN	1589	49	163	1377
	AVG	1882	106	259	1517
	STDDEV	1757	696	604	458
▷ 1 Navigationsschritt	SUM	91014	15147	8113	67754
	MIN	1384	41	141	1202
	MAX	15432	10864	422	4146
	MEDIAN	1490	43	155	1292
	AVG	1820	303	162	1355
	STDDEV	1981	1538	40	402
▷ 2 Navigationsschritte	SUM	186929	4972	32712	149245
	MIN	1395	45	119	1231
	MAX	10673	72	6508	4093
	MEDIAN	1638	48	165	1425
	AVG	1869	50	327	1492
	STDDEV	1219	5	838	376
▷ 4 Navigationsschritte	SUM	192510	6433	23920	162157
	MIN	1362	53	124	1185
	MAX	8206	569	3464	4173
	MEDIAN	1669	56	165	1448
	AVG	1925	64	239	1622
	STDDEV	1023	52	445	526

I.2 Verhältnisse der gemessenen Zeiten

Abbildung I.1 Zusammensetzung der Gesamtausführungszeit: Alle Befehle

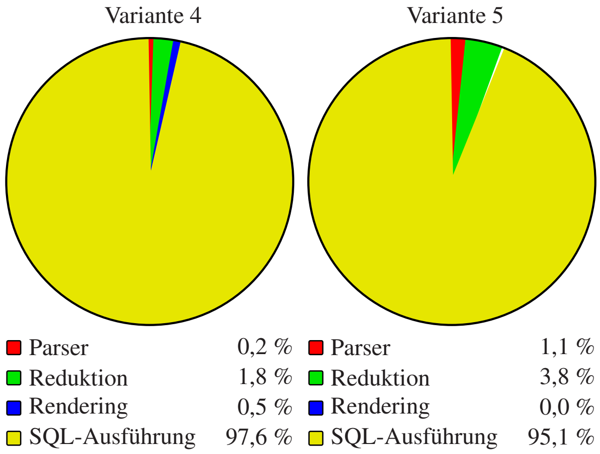


Abbildung I.2 Zusammensetzung der Gesamtausführungszeit: CREATE NEW RELATIONSHIP (kein FRE)

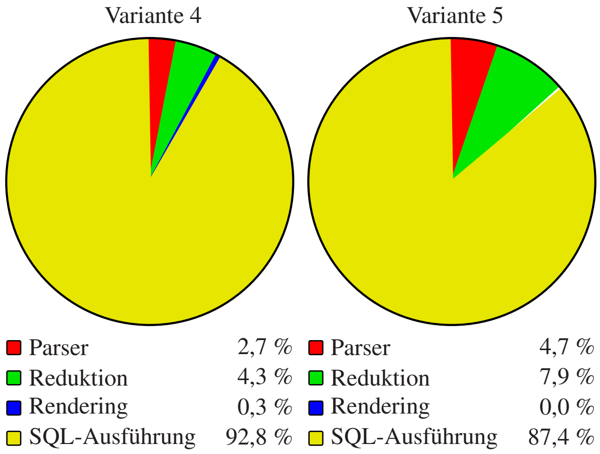


Abbildung I.3 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP

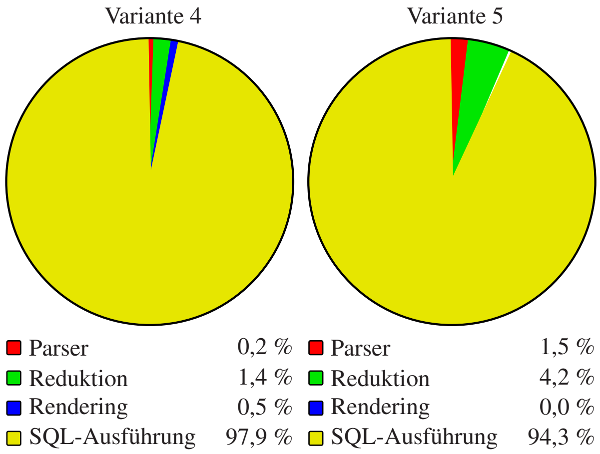


Abbildung I.4 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 kein FRE)

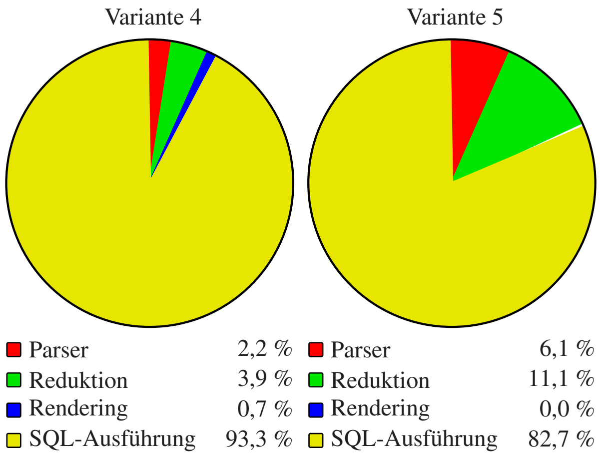


Abbildung I.5 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 zwei FRE)

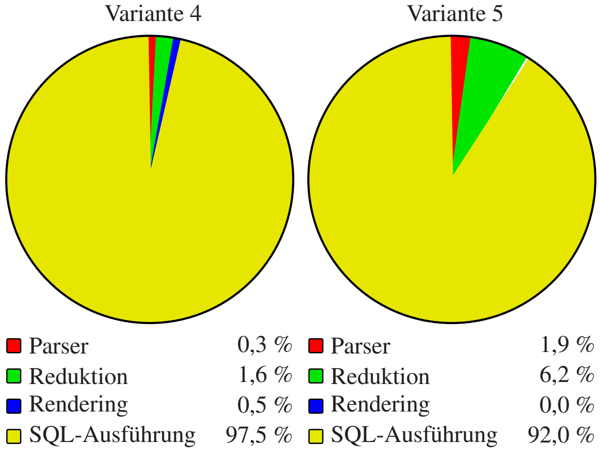


Abbildung I.6 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP (2 Objecte + 1 ein FRE + 1 zwei FRE)

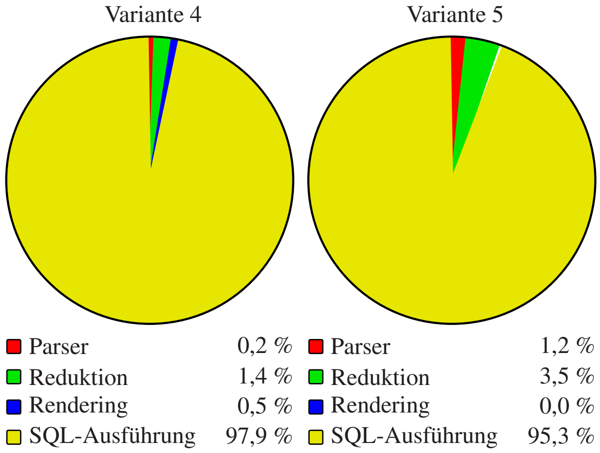


Abbildung I.7 Zusammensetzung der Gesamtausführungszeit: CREATE NEW OBJECT + RELATIONSHIP (3 Objekte + 2 ein FRE + 1 zwei FRE)

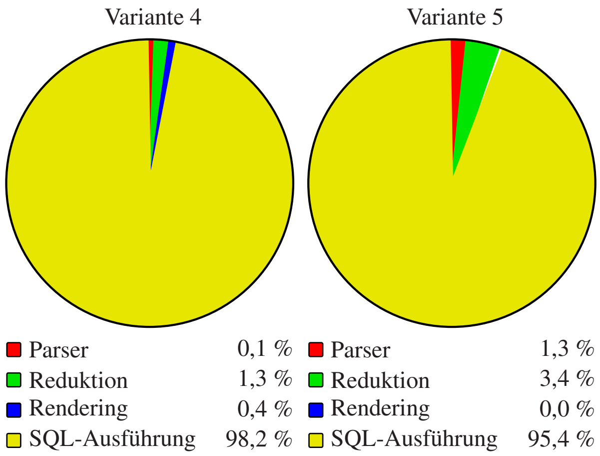


Abbildung I.8 Zusammensetzung der Gesamtausführungszeit: CREATE SUCCESSOR

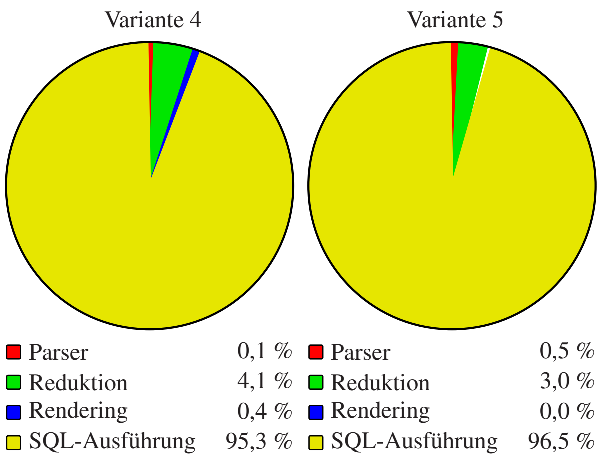


Abbildung I.9 Zusammensetzung der Gesamtausführungszeit: COPY OBJECTS

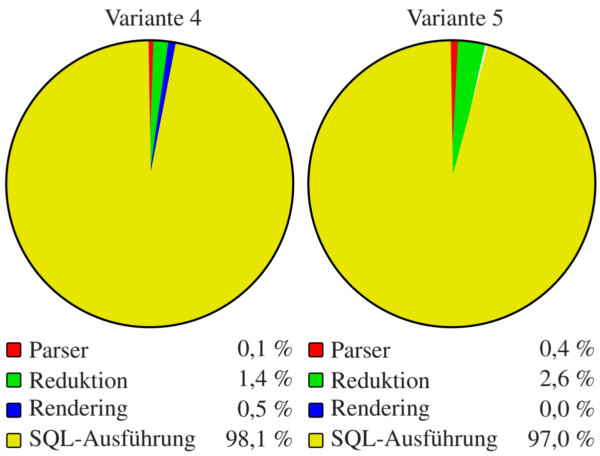


Abbildung I.10 Zusammensetzung der Gesamtausführungszeit: UPDATE OBJECT

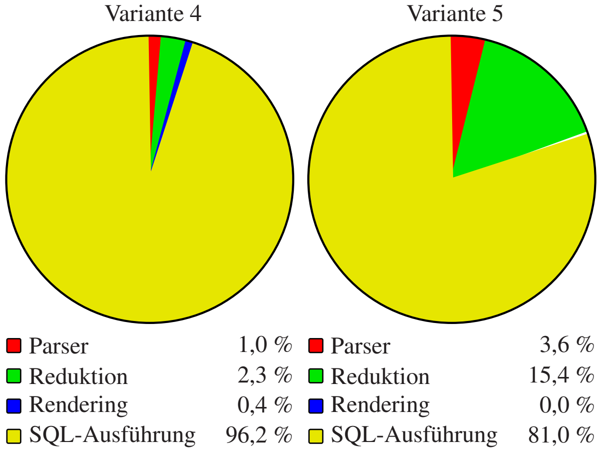


Abbildung I.11 Zusammensetzung der Gesamtausführungszeit: GET

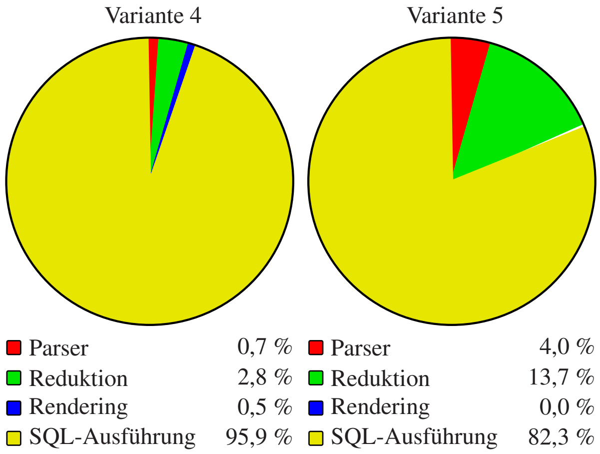


Abbildung I.12 Zusammensetzung der Gesamtausführungszeit: GET ROOT

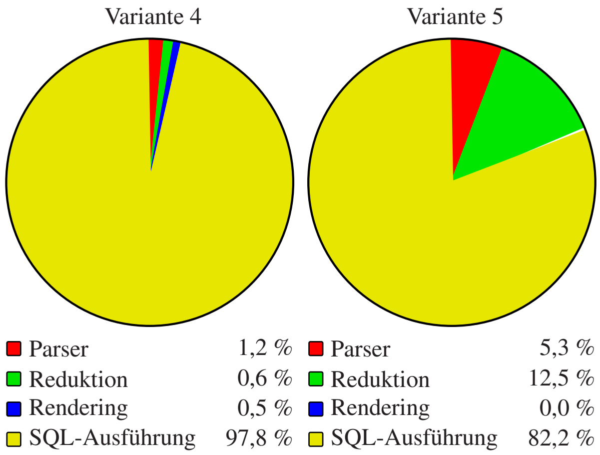


Abbildung I.13 Zusammensetzung der Gesamtausführungszeit: GET ALTERNATIVES

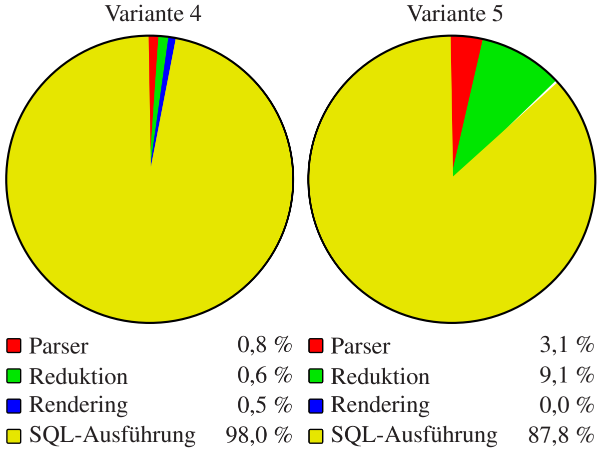


Abbildung I.14 Zusammensetzung der Gesamtausführungszeit: GET SUCCESSORS

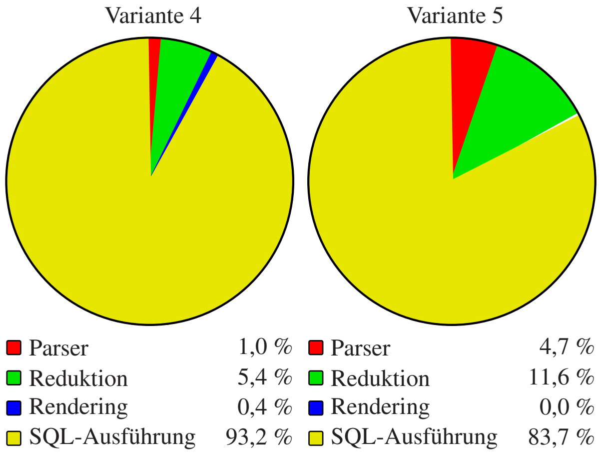


Abbildung I.15 Zusammensetzung der Gesamtausführungszeit: GET PREDECESSOR

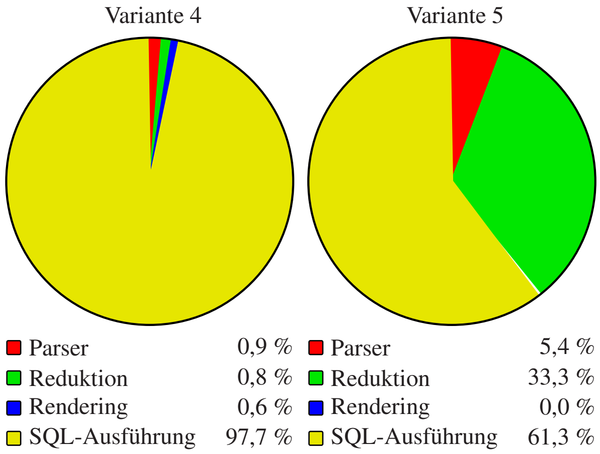


Abbildung I.16 Zusammensetzung der Gesamtausführungszeit: GET BASEVERSION

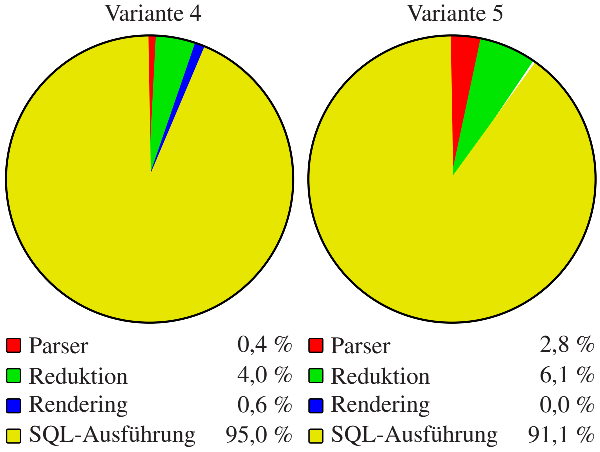


Abbildung I.17 Zusammensetzung der Gesamtausführungszeit: GET CVC

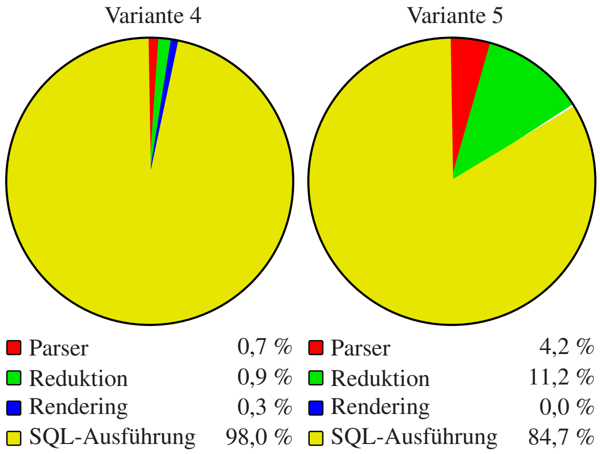


Abbildung I.18 Zusammensetzung der Gesamtausführungszeit: SELECT

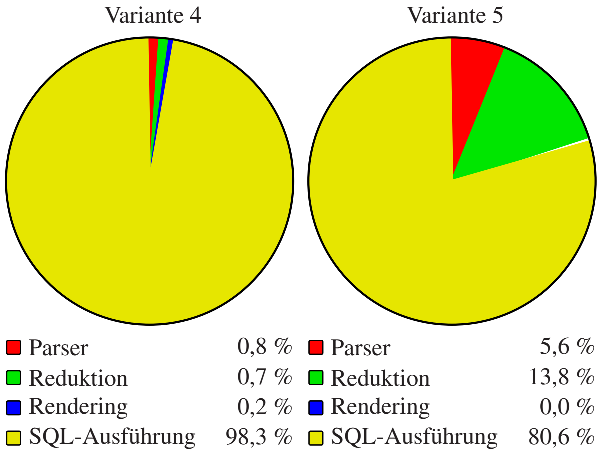
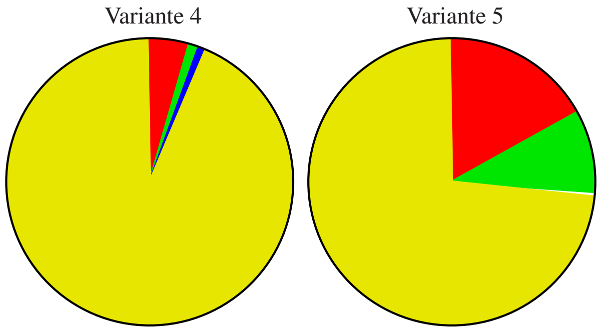
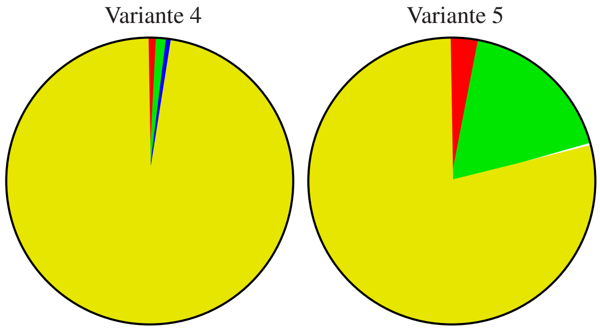


Abbildung I.19 Zusammensetzung der Gesamtausführungszeit: SELECT 1 Navigationsschritt



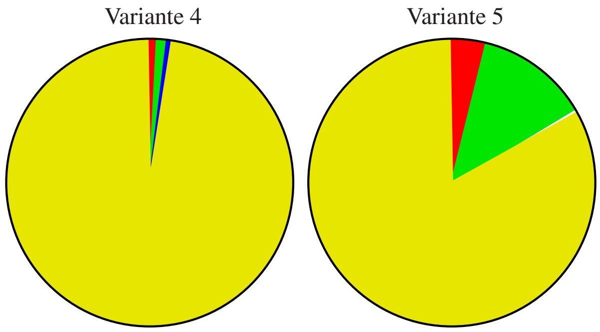
Parser	4,0 %	Parser	16,6 %
Reduktion	0,8 %	Reduktion	8,9 %
Rendering	0,5 %	Rendering	0,0 %
SQL-Ausführung	94,7 %	SQL-Ausführung	74,4 %

Abbildung I.20 Zusammensetzung der Gesamtausführungszeit: SELECT 2 Navigationsschritte



Parser	0,4 %	Parser	2,7 %
Reduktion	0,7 %	Reduktion	17,5 %
Rendering	0,1 %	Rendering	0,0 %
SQL-Ausführung	98,8 %	SQL-Ausführung	79,8 %

Abbildung I.21 Zusammensetzung der Gesamtausführungszeit: SELECT 4 Navigationsschritte



Parser	0,4 %	Parser	3,3 %
Reduktion	0,7 %	Reduktion	12,4 %
Rendering	0,2 %	Rendering	0,0 %
SQL-Ausführung	98,8 %	SQL-Ausführung	84,2 %

I.3 Vergleich der gemessenen Zeiten

Abbildung I.22 Zeitvergleich: Summe aller Befehle

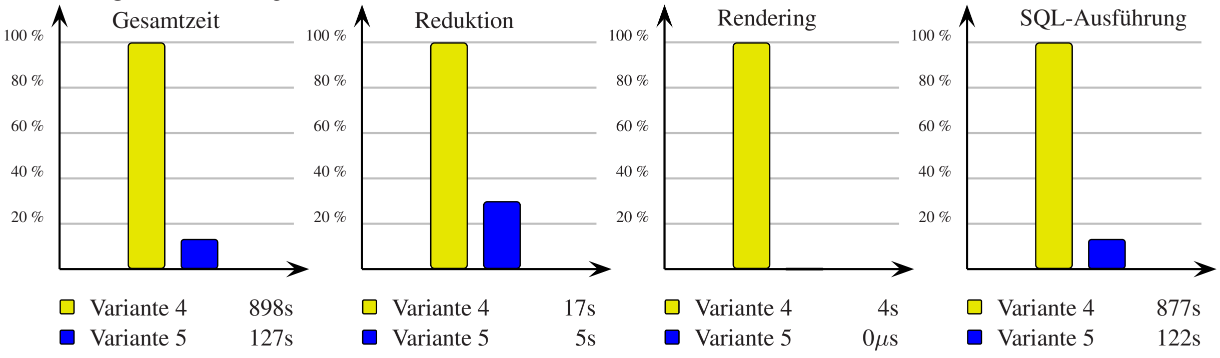


Abbildung I.23 Zeitvergleich: CREATE NEW RELATIONSHIP (kein FRE)

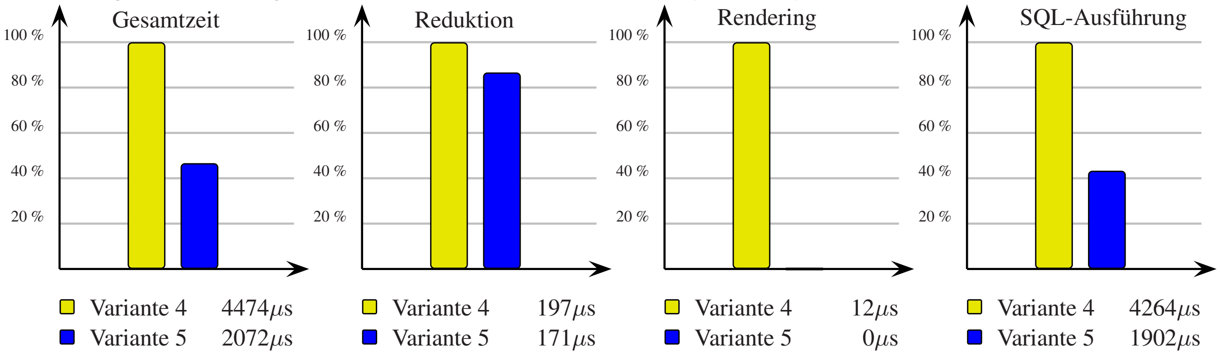


Abbildung I.24 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP

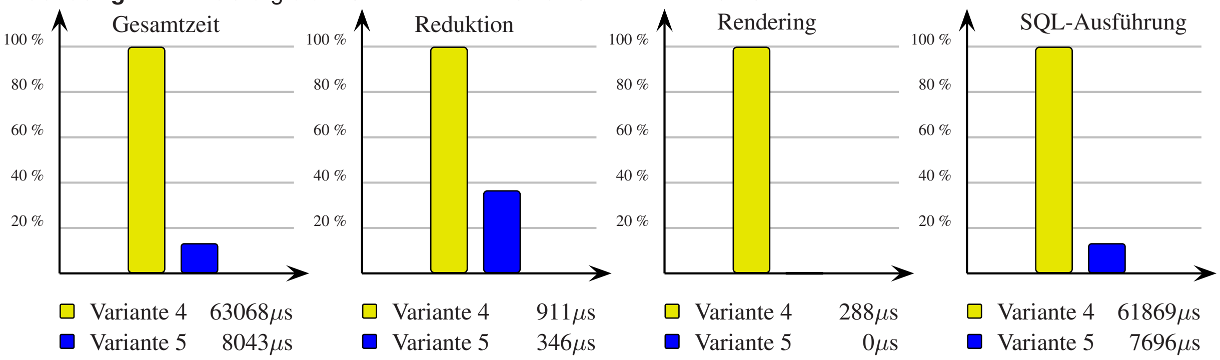


Abbildung I.25 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 kein FRE)

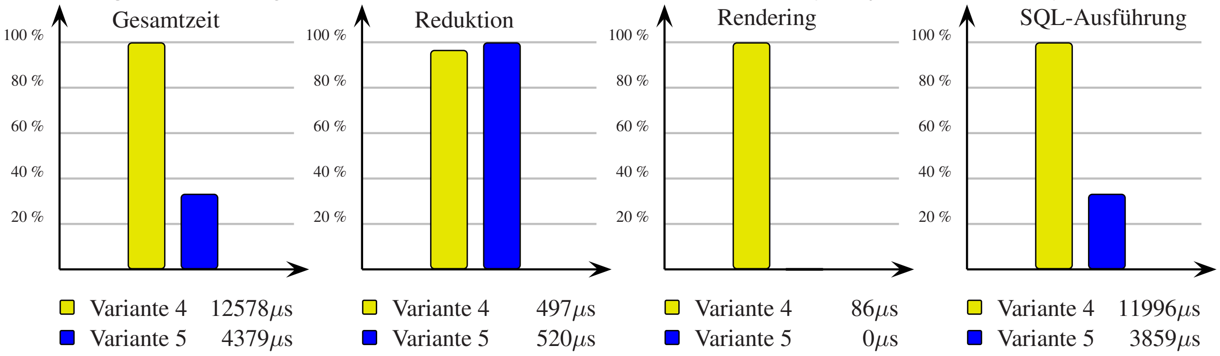


Abbildung I.26 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (1 Objekt + 1 zwei FRE)

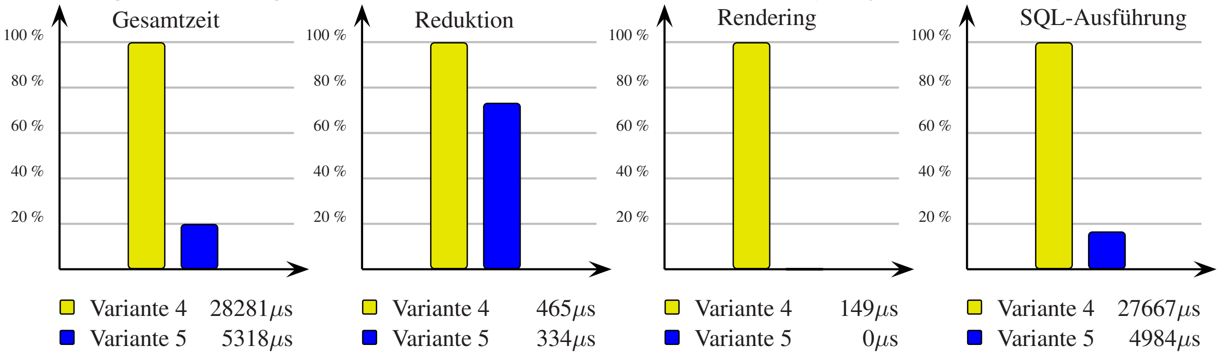


Abbildung I.27 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (2 Objecte + 1 ein FRE + 1 zwei FRE)

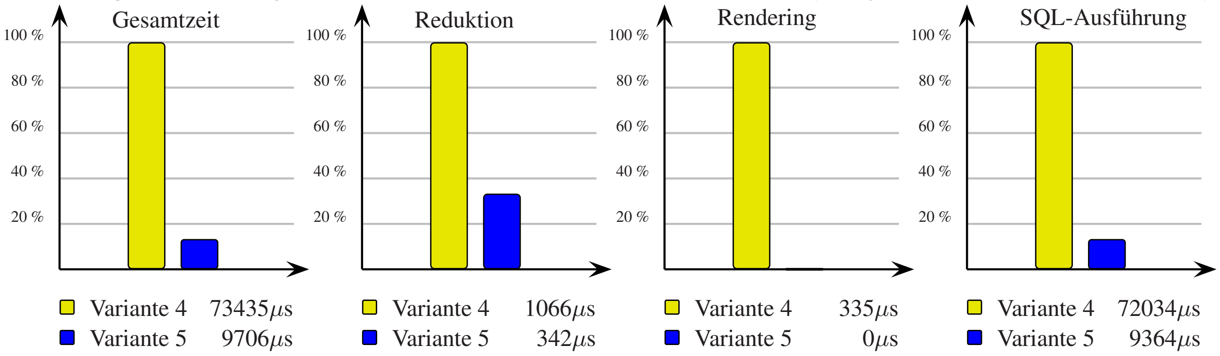


Abbildung I.28 Zeitvergleich: CREATE NEW OBJECT + RELATIONSHIP (3 Objekte + 2 ein FRE + 1 zwei FRE)

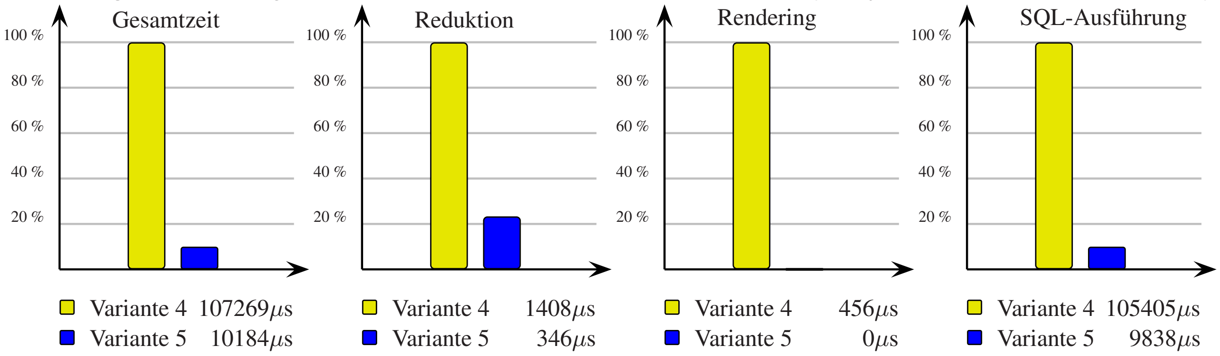


Abbildung I.29 Zeitvergleich: CREATE SUCCESSOR

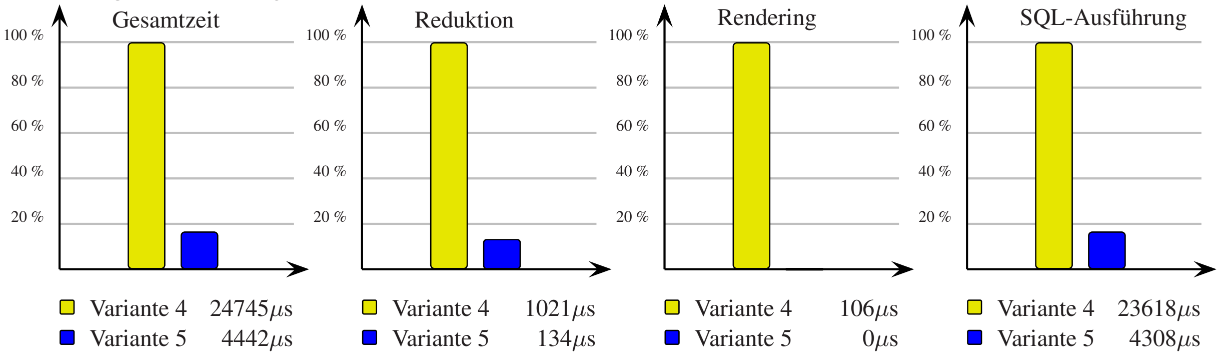


Abbildung I.30 Zeitvergleich: COPY OBJECTS

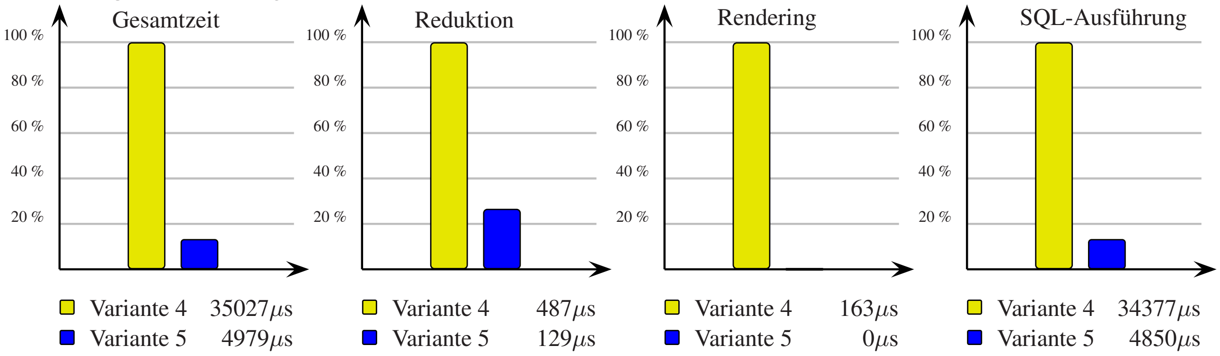


Abbildung I.31 Zeitvergleich: UPDATE OBJECT

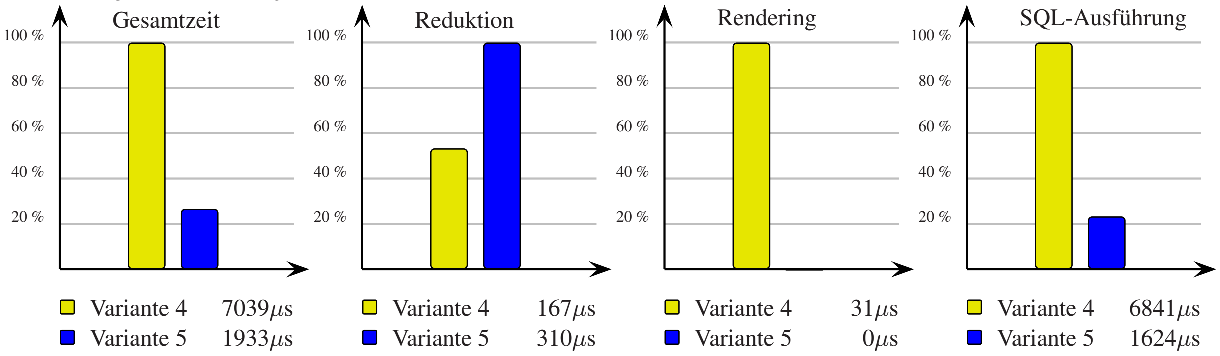


Abbildung I.32 Zeitvergleich: GET

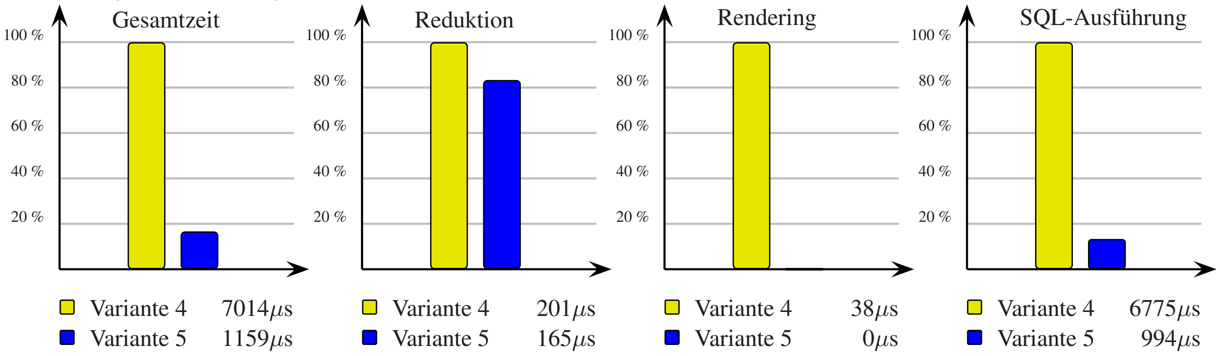


Abbildung I.33 Zeitvergleich: GET ROOT

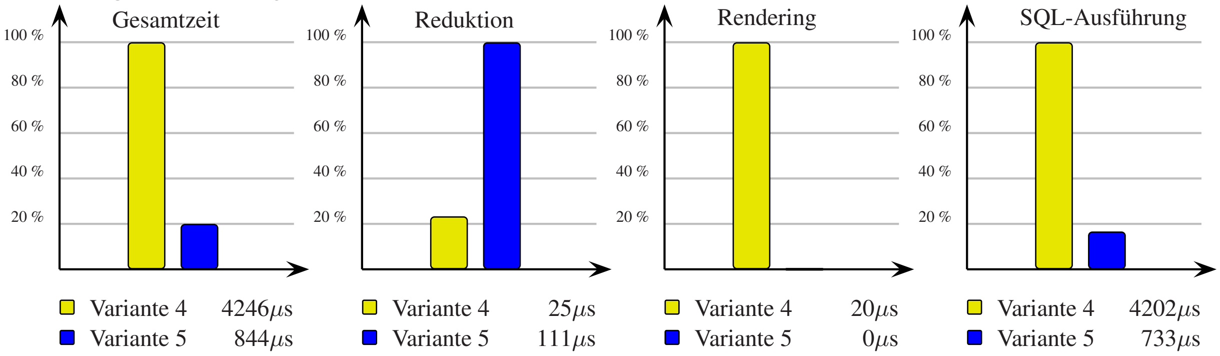


Abbildung I.34 Zeitvergleich: GET ALTERNATIVES

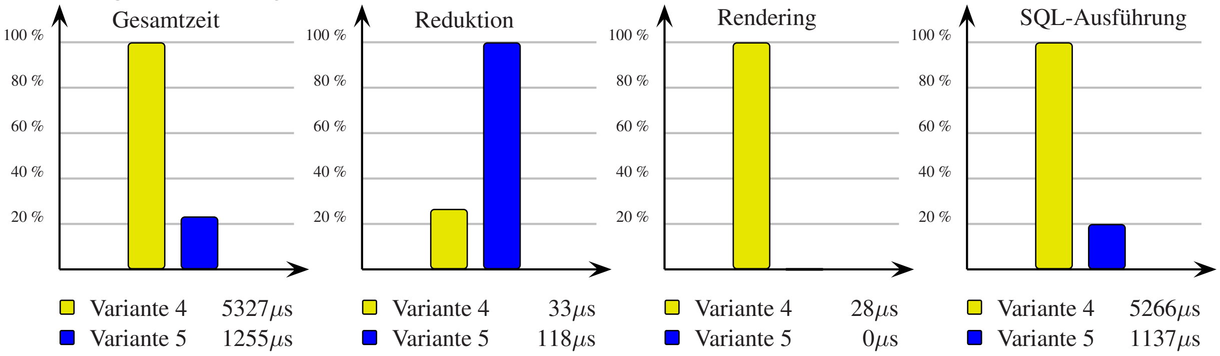


Abbildung I.35 Zeitvergleich: GET SUCCESSORS

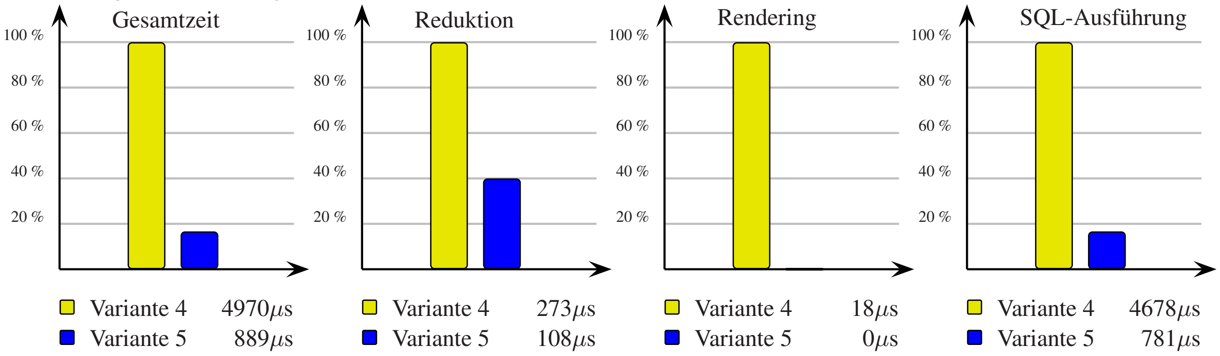


Abbildung I.36 Zeitvergleich: GET PREDECESSOR

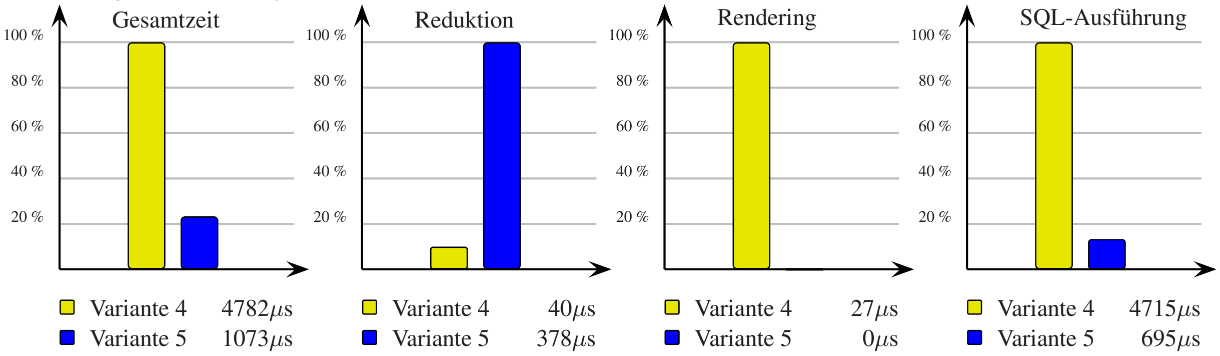


Abbildung I.37 Zeitvergleich: GET BASEVERSION

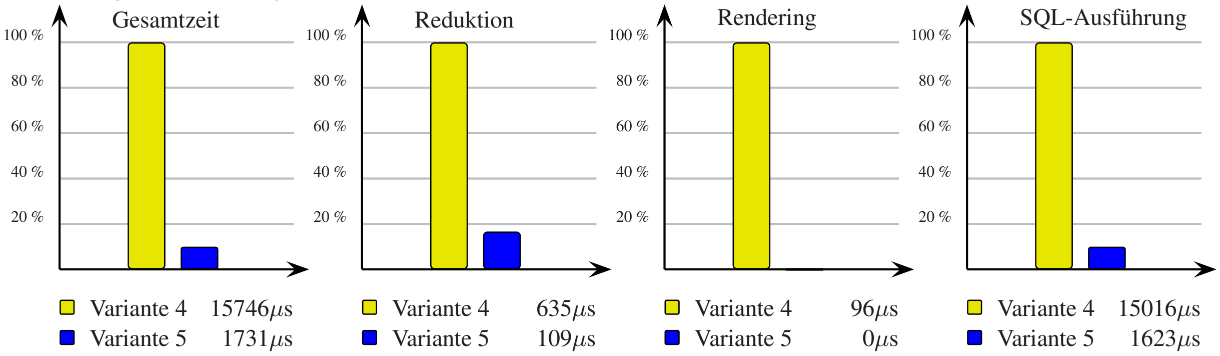


Abbildung I.38 Zeitvergleich: GET CVC

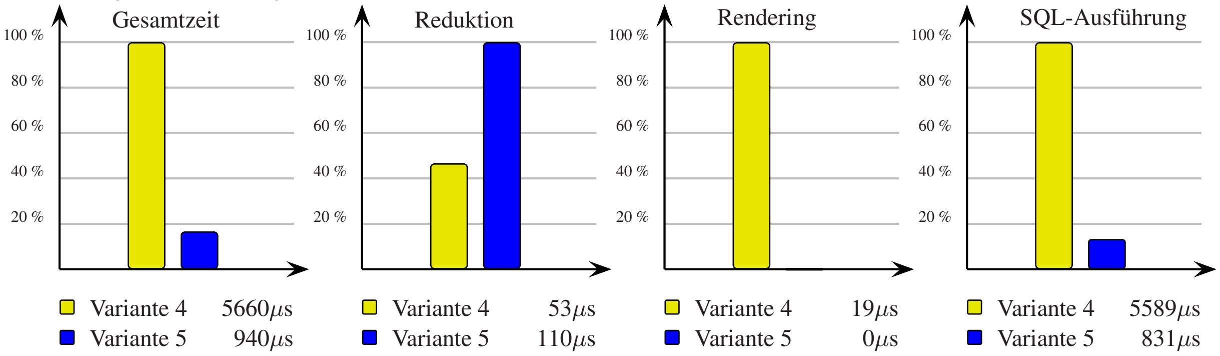


Abbildung I.39 Zeitvergleich: SELECT

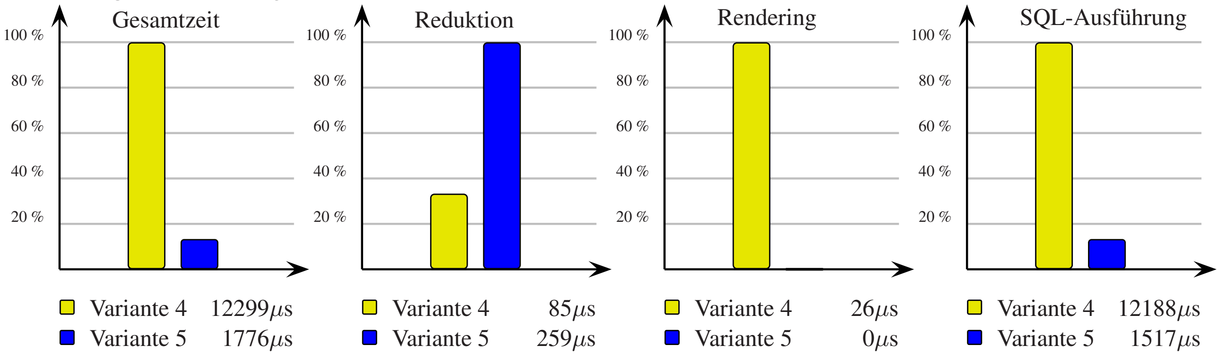


Abbildung I.40 Zeitvergleich: SELECT 1 Navigationsschritt

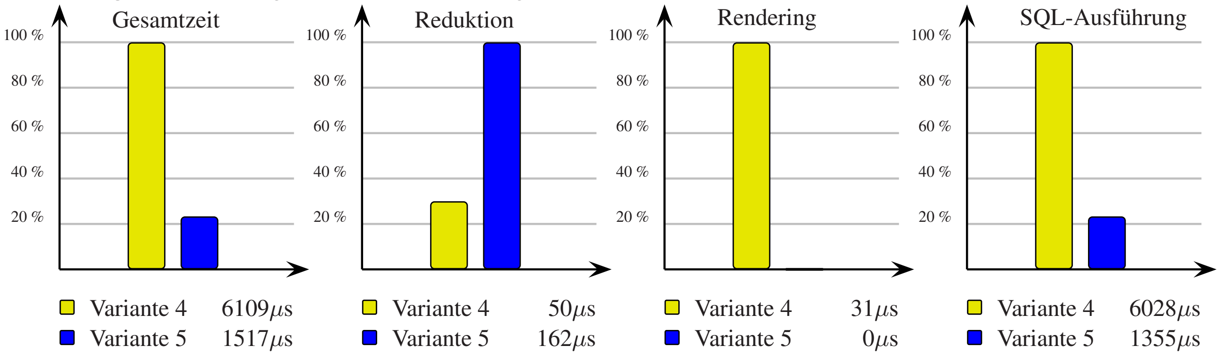


Abbildung I.41 Zeitvergleich: SELECT 2 Navigationsschritte

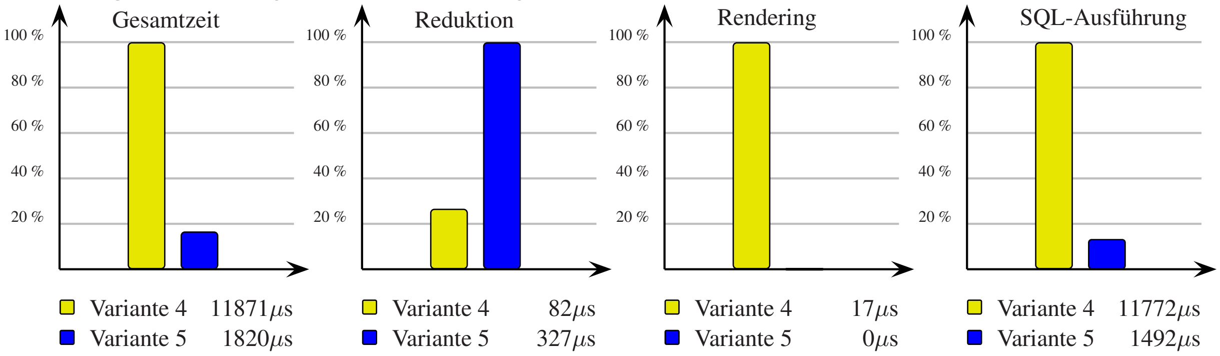
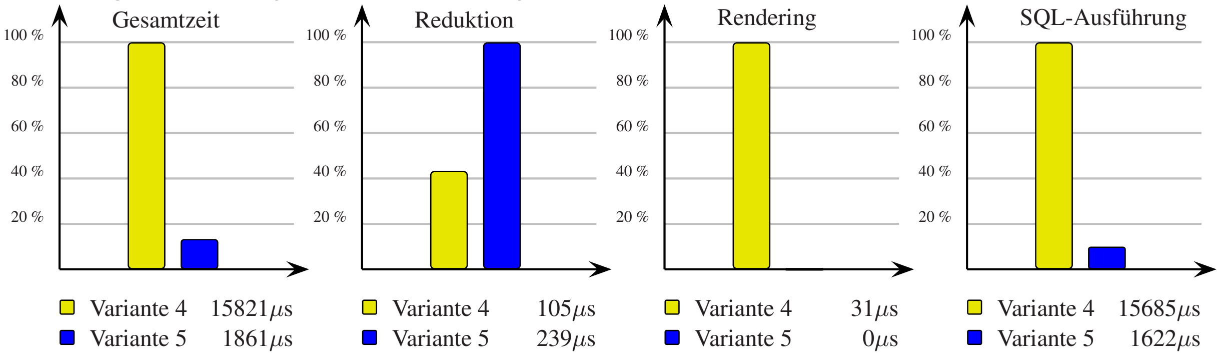


Abbildung I.42 Zeitvergleich: SELECT 4 Navigationsschritte



Abkürzungsverzeichnis

AOP Aspect-Oriented Programming

$CORBA$ Common Object Request Broker Architecture

DDL Data Definition Language

DML Data Manipulation Language

DSL Domain-Specific Language

$EBNF$ Erweiterte Backus-Naur-Form

IP Intentional Programming

LOC Lines of Code

MDA Model Driven Architecture

MOF Meta-Object Facility

OMG Object Management Group

PIM Platform Independent Model

PSM Platform Specific Model

UML Unified Modeling Language

XMI XML Metadata Interchange

Literaturverzeichnis

[Asp03] *AspectJ Project*, 2003. `http://eclipse.org/aspectj/`.

[BD94] Philip A. Bernstein and Umeshwar Dayal. An Overview of Repository Technology. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 705–713. Morgan Kaufmann, 1994.

[Ber96] Philip A. Bernstein. Repository System Engineering. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*. ACM Press, 1996. `http://www.acm.org/sigmod/dl/SIGMOD96/P542.PDF`.

[Ber97] Philip A. Bernstein. Repositories and Object Oriented Databases. In Klaus R. Dittrich and Andreas Geppert, editors, *Proceedings BTW '97*, pages 34–46, Ulm, Germany, 1997. Springer-Verlag.

[Bos00] Jan Bosch. *Design and Use of Software Architectures: Adopting and evolving a product-line approach*. Addison-Wesley, 2000.

[CE00] Krzysztof Czarnecki and Ulrich Eisenecker. *Generative programming: methods, tools and applications*. Addison-Wesley, 2000.

[Cza01] Krzysztof Czarnecki. *Generative Programmierung und die Entwicklung von Softwaresystemfamilien*, 2001. `http://www.tu-chemnitz.de/informatik/osg/IFKOLL/2001/docs/k53.pdf`.

[Fra03] David S. Frankel. *Model Driven Architecture:Applying MDA to Enterprise Computing*. Wiley Publishing, 2003.

[Fra04] Fraunhofer Gesellschaft zur Förderung der angewandten Forschung e.V. *Virtuelles Software- Engineering- Kompetenzzentrum*, 2004. `http://www.visek.de/`.

[Hal77] Maurice H. Halstead Halstead. *Elements of Software Science*, 1977.

[Hus03] Martin Husemann. Konzeption und Realisierung eines intuitiven CWM-basierten SQL-Editors. Diplomarbeit, Universität Kaiserslautern, August 2003.

[KH03a] Jernej Kovse and Theo Härder. DSL-DIA - An Environment for Domain-Specific Languages for Database-Intensive Applications. In *Proceedings of 9th Int. Conf. on Object-Oriented Information Systems (OOIS'03)*, pages 304–310, Heidelberg, DE, 2003. Springer Verlag. `http://wwwdvs.informatik.uni-kl.de/pubs/papers/KH03.OOIS.pdf`.

[KH03b] Jernej Kovse and Theo Härder. V-Grid - A Versioning Services Framework for the Grid. In *Proceedings of 3rd Int. Workshop Web Datenbanken, Berliner XML Tage*, pages 140–154, 2003. http://wwwdvs.informatik.uni-kl.de/pubs/papers/KH03.WebDB.pdf.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997. http://www.parc.xerox.com/csl/groups/sda/publications/papers/Kiczales-E%COOP97/for-web.pdf.

[Koc02] Thomas Koch. *An Introduction to Model Driven Architecture*. Interactive Obejcts Software GmbH, 2002. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/reposi%t/htm/reconversion_management.asp.

[Lic01] Horst Lichter. Softwareproduktfamielen. Vorlesungsskript, RWTH Aachen, 2001.

[McC76] Thomas J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.

[McI68] D McIlroy. Mass-produced software components. In P. Naur and B. Randell, editors, *Software Engineering, NATO Science Committe report*, pages 138–155, 1968.

[Mic03a] Microsoft Corporation. *Microsoft .NET*, 2003. http://www.microsoft.com/net/.

[Mic03b] Microsoft Corporation. *MSDN Library - Version Management*, Oktober 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/reposi%t/htm/reconversion_management.asp.

[OMG02a] Object Management Group. *Meta-Object Facility*, 2002. http://www.omg.org/technology/documents/formal/mof.htm.

[OMG02b] Object Management Group. *Model Driven Architecture*, 2002. http://www.omg.org/mda/.

[OMG03a] Object Management Group. *Common Object Request Broker Architecture*, 2003. http://www.corba.org/.

[OMG03b] Object Management Group. *Unified Modeling Language*, 2003. http://www.uml.org/.

[OMG03c] Object Management Group. *XML Metadata Interchange*, 2003. http://www.omg.org/technology/documents/formal/xmi.htm.

[Sim95] Charles Simonyi. *The death of computer languages, the birth of Intentional Programming*. Microsoft Research, 1995.

[Sim96] Charles Simonyi. *Intentional programming: Innovation in the legacy age*. Microsoft Research, 1996.

[Sun04a] Sun Microsystems, Inc. *Java 2 Platform, Enterprise Edition (J2EE)*, 2004. http://java.sun.com/j2ee/.

[Sun04b] Sun Microsystems, Inc. *Java Technology*, 2004. `http://java.sun.com/`.

[Sun04c] Sun Microsystems, Inc. *JDBC Technology*, 2004. `http://java.sun.com/products/jdbc/`.

[Vir03] Virtual Machinery. *JHawk*, 2003. `http://www.virtualmachinery.com/jhawkmetrics.htm`.