BACHELORARBEIT

zur Erlangung des akademischen Grades

„Bachelor of Science in Engineering“ im Studiengang Informatik/Computer Science

**Vergleich von Performance und Anwendung verschiedener ORM-Frameworks**

Ausgeführt von: Lukas Schweinberger

Personenkennzeichen: 1910257205

1. BegutachterIn: Mag. DI Bernhard Löwenstein

Neusiedl an der Zaya, Datum

Eidesstattliche Erklärung

„Ich, als Autor / als Autorin und Urheber / Urheberin der vorliegenden Arbeit, bestätige mit meiner Unterschrift die Kenntnisnahme der einschlägigen urheber- und hochschulrechtlichen Bestimmungen (vgl. Urheberrechtsgesetz idgF sowie Satzungsteil Studienrechtliche Bestimmungen / Prüfungsordnung der FH Technikum Wien idgF).

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt und Gedankengut jeglicher Art aus fremden sowie selbst verfassten Quellen zur Gänze zitiert habe. Ich bin mir bei Nachweis fehlender Eigen- und Selbstständigkeit sowie dem Nachweis eines Vorsatzes zur Erschleichung einer positiven Beurteilung dieser Arbeit der Konsequenzen bewusst, die von der Studiengangsleitung ausgesprochen werden können (vgl. Satzungsteil Studienrechtliche Bestimmungen / Prüfungsordnung der FH Technikum Wien idgF).

Weiters bestätige ich, dass ich die vorliegende Arbeit bis dato nicht veröffentlicht und weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt habe. Ich versichere, dass die abgegebene Version jener im Uploadtool entspricht.“

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Ort, Datum |  | Unterschrift |

Kurzfassung

Diese Arbeit befasst sich mit der Analyse von neu entwickelten ORM Frameworks durch Studierende. In diesem Zusammenhang wurden zwei Forschungsfragen entwickelt. Welches der Frameworks bietet die beste Ausführungszeit für Create, Read, Update und Delete Funktionalität? Sowie, wie unterscheiden sich die Frameworks in der Anwendung?

Ziel dieser Arbeit ist es, einen Vergleich der Frameworks anhand von vier funktionalen Merkmalen sowie einem Merkmal zur Anwendung der Frameworks durchzuführen. Die funktionalen Merkmale sind in die Teilbereiche Create, Read, Update und Delete eingeteilt. Besonderes Augenmerk wird auf Referenz und relationale Beziehungstypen gelenkt. Des Weiteren werden die neu entwickelten Framewoks mit Entity Framework Core verglichen, um einen Bezugspunkt zu weiteren bestehenden Arbeiten zu schaffen.

Die Ergebnisse zeigen, dass die neu entwickelten Frameworks in Teilbereichen des CRUD-Patterns ähnliche oder sogar eine kürzere Programmlaufzeit als Entity Framework Core besitzen. In spezifischen Bereichen ist eine Auswahl der neu entwickelten Frameworks um einen vielfachen Faktor schlechter als Entity Framework Core. Die Analyse der Anwendung hat ergeben, dass nur eins der neu entwickelten Frameworks alle Beziehungstypen unterstützt. Eines wiederum unterstützt eine Vielzahl von Beziehungen und Operationen nicht ein anderes unterstützt alle Beziehungen bis auf Read im Bereich Viele-zu-Viele. Hierbei kommt es mir fortlaufender Iterationsanzahl zu starken Performanceeinbrüchen.

Abstract

**Schlagwörter:** ORM-Frameworks, Entity Framework Core, Laufzeitvergleich, Studierenden Frameworks, Anwendung von ORM Frameworks

This thesis analysis ORM frameworks developed by students. In this context, two research questions were developed. Which of the frameworks provides the best execution time for Create, Read, Update and Delete functionality? As well as, how do the frameworks differ in application?

The goal of this paper is to compare the frameworks based on four functional characteristics as well as one characteristic on the application of the frameworks. The functional features are divided into the subareas Create, Read, Update and Delete. Special attention is paid to reference and relational relationship types. Furthermore, the newly developed frameworks are compared to Entity Framework Core to provide a reference point to other existing work.

The results show that the newly developed frameworks have similar or even shorter program runtime than Entity Framework Core in subareas of the CRUD pattern. In specific areas, a selection of the newly developed frameworks are many times worse than Entity Framework Core. Application analysis revealed that only one of the newly developed frameworks supports all relationship types. One in turn does not support a variety of relationships and operations another supports all relationships except Read in the many-to-many range. This leads to severe performance degradation as the number of iterations increases.

Danksagung

**Keywords:** ORM frameworks, Entity Framework Core, runtime comparison, student frameworks, application of ORM frameworks.

Ich bedanke mich bei Bernhard Löwenstein für die Unterstützung bei der Verfassung der Bachelorarbeit. Ich bedanke mich bei meinen Kommilitonen\*innen Elisabeth Glatz, Lam Nguyen, Samuel Fiedorowicz für die langjährige Unterstützung.

Inhaltsverzeichnis

[1 Einleitung 9](#_Toc102908720)

[1.1 Ziel und Motivation 9](#_Toc102908721)

[1.2 Prognose 10](#_Toc102908722)

[1.2.1 Vergleichbare Arbeit 10](#_Toc102908723)

[1.3 Repository 10](#_Toc102908724)

[2 Forschungsstand 11](#_Toc102908725)

[2.1 Was ist ein OR-Mapper 11](#_Toc102908726)

[2.1.1 Abgrenzung von ORM zu ODM 11](#_Toc102908727)

[2.2 Vorteile & Nachteile von ORM Frameworks gegenüber Eigen-Implementationen eines SQL-Layers 12](#_Toc102908728)

[2.3 Entity Framework Core 12](#_Toc102908729)

[2.3.1 Nutzung Statistiken 12](#_Toc102908730)

[2.4 Entity Framework Core im Detail 13](#_Toc102908731)

[2.4.1 Bugs in Entity Framework Core 13](#_Toc102908732)

[2.4.2 Entity Framework Core Tool, scaffold 13](#_Toc102908733)

[2.5 Objekt Beziehungen und CRUD 14](#_Toc102908734)

[2.5.1 Beziehungen 14](#_Toc102908735)

[2.5.2 Eager- und Lazy-Loading 16](#_Toc102908736)

[2.5.3 Caching 16](#_Toc102908737)

[2.5.4 CRUD-Operationen 17](#_Toc102908738)

[2.5.5 Anwendung CRUD 17](#_Toc102908739)

[2.6 Performance von APIs 18](#_Toc102908740)

[2.6.1 Performance von Software 18](#_Toc102908741)

[3 Forschungsfragen und Methodik 20](#_Toc102908742)

[3.1 Forschungsfragen 20](#_Toc102908743)

[3.2 Methodik 20](#_Toc102908744)

[3.2.1 Definitionen 20](#_Toc102908745)

[3.3 Auswahl der ORM Frameworks 20](#_Toc102908746)

[3.4 Entity Framework Core 20](#_Toc102908747)

[3.5 Vorherrschen von PostgresSQL in den neuen ORM-Frameworks 21](#_Toc102908748)

[3.6 Einschränkungen 21](#_Toc102908749)

[4 Vergleichsmerkmal Definition 22](#_Toc102908750)

[4.1 Vergleichsmerkmal Unterschiede Anwendung 22](#_Toc102908751)

[4.2 Vergleichsmerkmal Performance Select 22](#_Toc102908752)

[4.3 Vergleichsmerkmal Performance Insert 22](#_Toc102908753)

[4.4 Vergleichsmerkmal Performance Update 22](#_Toc102908754)

[4.5 Vergleichsmerkmal Performance Delete 22](#_Toc102908755)

[4.6 Test Kriterien 22](#_Toc102908756)

[4.6.1 Konfiguration Benchmark.net 22](#_Toc102908757)

[4.6.2 Initial Zustände 23](#_Toc102908758)

[4.6.3 Tools 23](#_Toc102908759)

[4.6.4 Test-Entitäten 23](#_Toc102908760)

[4.6.5 Daten der Tests 23](#_Toc102908761)

[4.6.6 Kompilierung und Ausführung 23](#_Toc102908762)

[4.6.7 Hardware & Software 24](#_Toc102908763)

[4.6.8 Test Datenbankmangementsystem 24](#_Toc102908764)

[4.6.9 Rahmenbedingungen für die erstellten Tests 24](#_Toc102908765)

[4.6.10 Testaufbau im Detail 26](#_Toc102908766)

[4.6.11 Datenbankschema 26](#_Toc102908767)

[4.7 SQL DDL Create Tables 27](#_Toc102908768)

[4.8 Source Code Manipulation 27](#_Toc102908769)

[4.8.1 Änderungen Console.Writeline; 27](#_Toc102908770)

[4.8.2 Debug 28](#_Toc102908771)

[4.8.3 Cache 28](#_Toc102908772)

[5 Ergebnisse Unterschiede in der Anwendung der ORM-Frameworks anhand der Erstellten Tests 29](#_Toc102908773)

[5.1.1 F0 Entity Framework Core 29](#_Toc102908774)

[5.2 Framework F1 29](#_Toc102908775)

[5.3 Framework F2 30](#_Toc102908776)

[5.4 Framework F3 31](#_Toc102908777)

[5.5 Unterschiede Entitys 31](#_Toc102908778)

[5.5.1 F0 Entity Framework Core 31](#_Toc102908779)

[5.5.2 F1 Framework 31](#_Toc102908780)

[5.5.3 F2 Framework 32](#_Toc102908781)

[5.5.4 F3 Framework 32](#_Toc102908782)

[5.6 Ergebnisse Performance 32](#_Toc102908783)

[5.6.1 Insert Eins-zu-Eins 32](#_Toc102908784)

[5.6.2 Insert NoRelation 33](#_Toc102908785)

[5.6.3 Insert Eins-zu-Viele 33](#_Toc102908786)

[5.6.4 Insert Viele-zu-Viele 33](#_Toc102908787)

[5.6.5 Select Eins-zu-Eins 33](#_Toc102908788)

[5.6.6 Select NoRelation 33](#_Toc102908789)

[5.6.7 Select Eins-zu-Viele 34](#_Toc102908790)

[5.6.8 Select Viele-zu-Viele 34](#_Toc102908791)

[5.6.9 Update Eins-zu-Eins 34](#_Toc102908792)

[5.6.10 Update NoRelation 34](#_Toc102908793)

[5.6.11 Update Eins-zu-Viele 34](#_Toc102908794)

[5.6.12 Update Viele-zu-Viele 35](#_Toc102908795)

[5.6.13 Delete Eins-zu-Eins 35](#_Toc102908796)

[5.6.14 Delete NoRelation 35](#_Toc102908797)

[5.6.15 Delete Eins-zu-Viele 35](#_Toc102908798)

[5.6.16 Delete Viele-zu-Viele 35](#_Toc102908799)

[5.7 Weitere Statistiken 36](#_Toc102908800)

[5.8 Fehlende Werte 36](#_Toc102908801)

[6 Analyse 37](#_Toc102908802)

[7 Conclusio 38](#_Toc102908803)

[7.1 Ausblick 38](#_Toc102908804)

[Literaturverzeichnis 39](#_Toc102908805)

[Abbildungsverzeichnis 41](#_Toc102908806)

[Tabellenverzeichnis 42](#_Toc102908807)

[Abkürzungsverzeichnis 43](#_Toc102908808)

# Einleitung

Im Rahmen des Computer Science / Informatik Bachelor Studiums der Fachhochschule Technikum Wien hatten Studierende die Möglichkeit, Software Engineering 3 als Wahlpflichtfach auszuwählen. Ziel dieser Lernveranstaltung war es, ein eigenes ORM Framework zu entwickeln. Den Studierenden wurde bei der Entwicklung freie Auswahl bezüglich Programmiersprache und Datenbankmanagementsystem gegeben. Die Vorgaben beschränkten sich auf funktionale Anforderungen sowie Dokumentation und Testbarkeit. In ORM-Frameworks bilden eine Verbindung von objektorientierter Programmierung zu relationalen Datenbanken durch die Einführung einer Abstraktionsebene. Durch diesen Abstraktionslayer ist es möglich, dass Softwareentwickler\*innen sich auf andere Komponenten der Software vermehrt konzentrieren. Es müssen keine Datenbankzugriffsklassen geschrieben und getestet werden, des Weiteren verschwinden Hard-Coded SQL Statements aus dem Code. Doch welche Unterschiede gibt es bei zwischen den neu entwickelten ORM-Frameworks?

In diesem Zusammenhang wurden zwei Forschungsfragen entwickelt:

1. Welches der Frameworks bietet die beste Ausführungszeit für Create, Read, Update und Delete Funktionalität?
2. Wie unterscheiden sich die Frameworks in der Anwendung?

Zur Beantwortung der Fragen wurden 4 Performance Merkmale verglichen sowie ein Vergleich der Anwendung der Frameworks durchgeführt.

## Ziel und Motivation

Der Kerngedanke bzw. der Ursprungsgedanke war die Problematik, dass es für Studierenden schwer ist, die von ihnen entwickelten ORM-Frameworks untereinander zu vergleichen. Die einzigen Vergleichsmerkmale sind die Endpunkte für das Projekt sowie die Endnote. Die Note kann jedoch verschieden entstehen, zum Beispiel: ein\*e Student\*in A bekomm 20 Punkte Abzug für Code Dokumentation und erreicht 80 Punkte, Student\*in B bekommt 20 Punkte Abzug für funktionale Mängel in der Software und erreicht ebenfalls 80 Punkte. Beide erhalten gleich viele Punkte und die gleiche Note bei verschiedenen Problemstellen. Es ist zu erwähnen, dass im Rahmen der Fachhochschule eine Einsicht in die Note angefordert werden kann. Doch auch dann kann es Unterschiede bei zum Beispiel dem Bewertungskriterium Codequalität oder auch Dokumentation geben, da es sich dabei um ein subjektives Vergleichsmerkmal handelt.

Ziel dieser Arbeit ist es, eine Auswahl der neu entwickelten ORM-Frameworks miteinander zu vergleichen. Zusätzlich wird ein bestehendes verfügbares Framework mit einbezogen, näheres in Kapitel 3.4. Es ist nicht Ziel der Arbeit, die Leistungsbewertung der Studierenden nachträglich zu beeinflussen.

## Prognose

Es ist zu erwarten, das verfügbare Framework Entity Framework Core gleich schnell in der Ausführung der CRUD-Operationen[1] zu den neu entwickelten Frameworks ist. Da an dem bestehenden Framework bereits unter anderem seit dem Jahr 2016 entwickelt wird, näheres in Kapitel 5.3. Im Vergleich dazu gab es für die Entwicklung der neuen ORM Frameworks im Rahmen der Lehrveranstaltung „Software Engineering 3“ der Fachhochschule Technikum Wien 3 ECTS als Gegenwert. Wobei ein ECTS laut der Website dem österreichischen Ministerium für Bildung und Wissenschaft und Forschung [2] 25 Stunden entspricht. Hierzu sind noch Kursanwesenheit, Selbststudium und Prüfungsvorbereitung einzuberechnen.

Da die verfügbare Entwicklungszeit stark abweicht, ist zu erwarten, dass der Code der neuen Frameworks weniger optimiert ist.

In den Performancekategorien kann es ein eindeutiges besser oder schlechter Vergleichsergebnis geben. Da hierdurch Vergleichszahlen Rückschlüsse gezogen werden, kann.

### Vergleichbare Arbeit

Es gibt bereits eine vergleichbare Arbeit [3] die ORM Frameworks vergleicht. Die verwendeten Frameworks sind:

* Entity Framework Core
* NHibernate
* Dapper

Diese Frameworks sind zum Zeitpunkt 18.04.2022 als Open-Source Projekt auf GitHub zu finden

## Repository

Der Sourcecode dieser Arbeit bezieht sich auf Programmcode der in einem GIT Repository [4] Bereitgestellt wurde. In nachfolgenden Kapiteln beziehen sich Klassen, Namespaces, yaml-Datein und Projektnamen auf dieses Repository.

# Forschungsstand

Seit der Erschaffung von ORM Frameworks gab es bereits mehrere Arbeiten über Performance und Features von Mapping Frameworks und ein entsprechender Vergleich dieser. Unter anderem liefert eine Google-Suche nach „OR Mapper Performance test paper“ bereit ein Paper [5] und einen Blogeintrag [6] zu Performance Tests. Der Blog-Beitrag verwendet eine Version des Tools benchmark.net [7] , welches für das automatische Benchmarken von .NET Methoden verwendet werden kann.

Im Zuge einer erweiterten Suche nach Statistiken konnten ausschließlich Daten zu etablierten Frameworks gefunden werden. Eine suche auf Google-Scholar mit dem Suchparameter „object relational mapping“ ergab etwa 1.170.000 Ergebnisse mit gerundet 17.000 Resultaten seit Jahresbeginn 2022 (Datum zum Zeitpunkt der Recherche 12.04.2022). Es handelt sich daher noch immer um ein aktuelles Thema an dem aktiv geforscht wird.

## Was ist ein OR-Mapper

„Unter einem [OR Mapper](http://www.google.de/search?sourceid=navclient&hl=de&ie=UTF-8&rlz=1T4SUNC_deDE393&q=or_mapper) versteht man Middleware, die in einer objektorientierten Programmiersprache geschriebener Anwendungscode benötigt, um über ein relationales DBMS verwaltete Daten objektorientiert dargestellt sehen zu können: ganz so als würden sie nur in programmlokalen Variablen existieren. [8]“

„Objektrelationale Abbildung … ist eine Technik der [Softwareentwicklung](https://de-academic.com/dic.nsf/dewiki/1300154), mit der ein in

einer [objektorientierten Programmiersprache](https://de-academic.com/dic.nsf/dewiki/1040544) geschriebenes [Anwendungsprogramm](https://de-academic.com/dic.nsf/dewiki/90194) seine

[Objekte](https://de-academic.com/dic.nsf/dewiki/1040458) in einer [relationalen Datenbank](https://de-academic.com/dic.nsf/dewiki/1172293) ablegen kann. Dem Programm erscheint die

Datenbank dann als [objektorientierte Datenbank](https://de-academic.com/dic.nsf/dewiki/1040542), was die Programmierung erleichtert. [9]“

Bei dem Vergleich der beiden angeführten Definitionen ist ersichtlich, dass das Abstrahieren des Datenbankzugriffs als objektorientierte Klasse/Datenbank eine der wesentlichen Aufgaben eines OR-Mappers ist. Der OR-Mapper soll das Programmieren von Software vereinfachen, in dem er in die Software eine Schnittstelle zu einer Datenbank einführt.

### Abgrenzung von ORM zu ODM

Nach [10] liegt Unterschied zwischen Object Relational Mapping (ORM) zu Object Document Mapping (ODM) liegt in der Natur der Datenbank und dem Datensatz. Bei dem Object Relational Mapping gibt es einzelne Tabelleneinträge, die als Objekte geladen werden. Im Vergleich dazu gibt es bei ODM Einträge in Form von Dokumenten im zum Beispiel JSON Format. Object Relational Mapping bezieht sich speziell auf relationale Datenbanken. Object Document Mapping bezieht sich auf nicht relationale Datenbanken.

In kurz, die Aufgabe beider Frameworks/Mapper ist die gleiche bei unterschiedlichen Daten und Datenbankmanagementsystemen.

#### Beispiele für relationale Datenbanken

* PostgreSQL
* Microsoft SQL-Server

#### Beispiele für dokumentorientierte Datenbanken

* Amazon DocuemntDB
* MongoDB

## Vorteile & Nachteile von ORM Frameworks gegenüber Eigen-Implementationen eines SQL-Layers

Ein Querschnitt aus [11][12][13] hat Folgende Informationen erbracht.

Vorteile:

* Keine SQL-Statements im eigenen Code
* Bessere Auswechselbarkeit des Datenbankmanagement-Systems
* SQL-Injektion muss weniger beachtet werden
* Softwareentwickler\*innen benötigen keine SQL-Kenntnisse

Nachteile:

* Ein ORM-Framework ist nur so gut wie die Person die es entwickelt hat.
* Query-Optimierung bei Komplexen Anfragen kann schlecht sein
* Overhead durch das Framework und eventuell zusätzlicher Aufwand durch Konvertierung in listen anstelle von plane Arrays

## Entity Framework Core

In diesem Kapitel wird ein genauerer Blick auf eine Auswahl von ORM Frameworks genommen und ein Nutzungsstatistikenvergleich durchgeführt.

### Nutzung Statistiken

Dieses Kapitel liefert einen Einblick auf die Nutzungsstatistiken für die Frameworks der im Einleitungskapitel genannten vergleichenden Arbeit

#### Was ist NuGet

Laut Microsoft [14] ist NuGet ein essenzieller Teil der modernen Software-Entwicklung. Im gleichen Artikel wird gesagt, das NuGet diese Relevanz hat ein einfacher Softwareaustausch in sogenannten Packages zwischen Programmierer\*innen möglich ist.

#### NuGet Statistiken

|  |  |  |  |
| --- | --- | --- | --- |
| Framework Name | Total NuGet Downloads | Per day average | Source |
| Dapper | 138.100.000 | 34.300 | [15] |
| Entity Framework Core | 400.000.000 | 177.600 | [16] |
| NHibernate | 20.700.000 | 5000 | [17] |

Tabelle 1: Download Statistiken ORM Frameworks nach NuGet

Entity Framework Core ist im Gesamten etwa drei Mal so groß wie der nächste Nachfolger und somit der Spitzenreiter der Auswahl. Ein noch größerer Schritt erfolgt zwischen der Nummer zwei NHibernate und dem Nachfolger, hier hat Dapper etwa 6,5 so weniger Downloads wie der direkte NHibernate. Auch in den täglichen durchschnittlichen Downloads ist Entity Framework Core führender und hat etwa fünffach so viele tägliche Downloads wie der Nachfolger. Wenn man die relativen Verhältnisse von Gesamt Downloads zu täglichen Downloads betrachtet, ist es möglich, zusätzlich zu erkennen

## Entity Framework Core im Detail

Laut [18][19] ist Entity Framework Core am 27.Juni 2016 als Version 1.0 erschienen und es ist der Nachfolger zu Entity Framework. Hierbei ist zu beachten, dass es sich um eine neue Implementierung handelt und nicht auf den Vorgänger aufbaut. In beiden Artikeln wird Bedeutung auf den gleichzeitigen Release von .NET Core 1.0 gelenkt. Hintergrund von der Entwicklung des Frameworks .Net Core war das loslösen von der Abhängigkeit zum Windows Betriebssystem [20]. Durch die Loslösung von Windows war es ab diesem Zeitpunkt auch möglich, Webapplikationen für Linux Server in C# zu Programmieren.

### Bugs in Entity Framework Core

Bei Entity Framework, laut Trendanalyse aktueller Marktführer der Auswahl, gibt es zum Zeitpunkt 25.04.2022 1571 offene und 16041 geschlossene Ticket auf GitHub. Durch Open-Source ist es jeder Person theoretisch möglich, den Sourcecode einzusehen und bei ausreichend Zeitaufwendung den Fehler selbst zu beheben. Des Öfteren werden Tickets erstellt und die entsprechenden Bugs angesprochen und von Entwickler\*innen behandelt. Bugs können Spezialfälle oder auch kritische Bereiche wie Sicherheit bzw. Hauptfunktionalität betreffen. Es ist daher empfehlenswert, immer die aktuell stabilste Version des Frameworks zu verwenden.

### Entity Framework Core Tool, scaffold

Das Tool scaffold [21] von Microsoft ermöglicht das automatische Generieren von DbContext und Entity Klassen aus einer bestehenden Datenbank. Es ist hierbei möglich, Code-Klassen zu der Datenbank in vollem Umfang oder auch nur zu einzelnen Klassen zu generieren.

Es sind diverse Konfigurationsmöglichkeiten wie zum Beispiel ob die Originalnamen beibehalten werden sollen oder auch, ob Annotations anstelle einer Modellkonfiguration verwendet werden sollen.

Sollte es in der Datenbank Änderungen geben, müssen diese Änderungen in das Codemodell und in die Entitäten übernommen werden. Hierbei ist aufzupassen, dass beim Überschreiben der alten Files eventuelle Anpassungen, die an den Klassen vorgenommen wurden, überschrieben und somit verloren gehen können.

## Objekt Beziehungen und CRUD

In diese Kapitel werden die Beziehungen zwischen Objekten näher erklärt sowie die CRUD-Operationen.

### Beziehungen

Es gibt drei Arten von Beziehungen zwischen Tabellen. Hierbei ausgelassen ist die Art der Beziehung, bei der eine Tabelle keine Verbindung zu einer weiteren Tabelle hat [22]. Es ist zu beachten, dass ein Objekt eine Vielzahl von Beziehungen besitzen kann. Das Entity-Relationship Model wurde 1976 von Chen beschrieben [22].

#### Beziehung NoRelation

Bei dieser Art der Beziehung gibt es für jeden Tabelleneintrag in keiner anderen Tabellen einen entsprechenden Referenzeintrag. Es handelt sich daher um die von in dieser Arbeit beschrieben simpelste Beziehungsart ohne Abhängigkeiten.

Ein Beispiel für diese Art der Beziehung in der objektorientierten Programmierung wäre ein Objekt ohne Abhängigkeiten zu anderen Objekten.

#### Beziehung Eins-zu-Eins

Bei dieser Art der Beziehung gibt es für jeden Tabelleneintrag in einer Tabelle einen Eintrag in einer weiteren Tabelle geben kann [23].

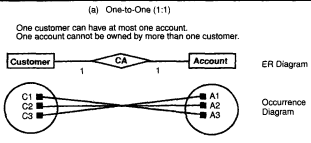


Abbildung 1: Eins zu Eins Beziehung [23]

Ein Beispiel für diese Art der Beziehung in der objektorientierten Programmierung wäre zwei Objekttypen A und B, die in einer 1 zu 1 Beziehung stehen. Objekt A hat dabei als Parameter bzw. Variable eine Referenz zu einem Objekt von Typ B. Die Objekte vom Typ B enthalten wiederum einen Parameter mit einer Referenz zu Objekt A.

#### Beziehung Eins-zu-Viele

Bei dieser Art der Beziehung gibt es für jeden Tabelleneintrag in einer Tabelle mehrere Einträge in einer anderen Tabelle geben kann [23].

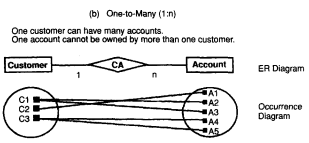


Abbildung 2: Eins zu Viele Beziehung [23]

Ein Beispiel für diese Art der Beziehung in der objektorientierten Programmierung sind zwei Objekttypen A und B die in einer 1 zu n Beziehung stehen. Objekt A hat dabei als Parameter bzw. Variable eine List, die Objekte vom Typ B enthält. Die Objekte vom Typ B enthalten wiederum einen Parameter mit einer Referenz zu Objekt A. Die Liste ist hierbei symbolhaft, es können verschiedene Collections oder auch ein Array sein.

#### Beziehung Viele-zu-Viele

Bei dieser Art der Beziehung gibt es für jeden Tabellen Eintrag in Tabelle A kann es mehrere Einträge in Tabelle B geben [23]. Für jeden Eintrag in Tabelle B kann es mehrere Einträge in Tabelle A geben. Ein Problem hierbei ist es, die Beziehungen effizient zu speichern. Eine Lösung ist das Erstellen einer Zwischentabelle C. Durch diese Tabelle wird von Tabelle A zu Tabelle C eine Eins-zu-Viele Beziehung aufgebaut und gleichzeitig wird eine Eins-zu-Viele Beziehung zwischen Tabelle B und C aufgebaut.

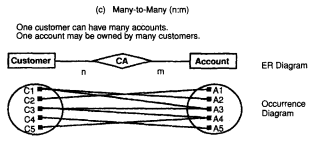


Abbildung 3: Viele zu Viele Beziehung [23]

Ein Beispiel für diese Art der Beziehung in der objektorientierten Programmierung sind zwei Objekttypen A und B, die in einer n zu m Beziehung stehen. Objekt A hat dabei als Parameter bzw. Variable eine Liste, die Objekte vom Typ B enthalten kann. Die Objekte vom Typ B enthalten wiederum einen Parameter mit einer Referenz zu einer Liste für Objekte vom Typ A. Die Listen sind hierbei symbolhaft, es können verschiedene Collections oder auch ein Array sein.

### Eager- und Lazy-Loading

Eager-Loading beschreibt das Laden der Daten bei dem Ausführen der Select/Read Operation [24]. Lazy Loading beschreibt das Laden der Daten zu einem späteren Zeitpunkt bei Aufruf der entsprechenden Daten im Code ohne vorherigen expliziten Ladeaufruf [24].

Daraus ergeben sich Vor- und Nachteile. Eager-Loading hat den Vorteil, alle Daten sind zu jeder nachfolgenden Zeit im Programmcode vorhanden. Sollte zum Beispiel die Datenbank ausfallen, so kann das Programm weiterhin seine Funktion ausführen. Ein Nachteil ist es, durch das sofortige Laden der Daten ist der Zustand der Daten unter Umständen veraltet im Vergleich zur Datenbank Version. Bei Lazy-Loading sind diese zwei verhalten genau umgekehrt.

### Caching

Nach AWS[25] handelt es sich bei einem Cache um einen temporären Speicher, der Daten für eine spätere Verwendung Zwischenspeichern soll. Dies soll eine effizientere Wiederverwendung der Daten ermöglichen. Die Daten können auf Hardwareebene oder auch Softwareebene gecached werden.

Unter Umständen kann es vorkommen, dass es mehrere Queries über dieselben Daten innerhalb des Codes gibt. Durch Caching wird eine bessere Performance erreicht.

Eine Verwendung von Caches kann bei einer falschen Implementierung zu Problemen führen. Ein mögliches Problem ist es, das Daten veraltet sein Können. Eine Lösung hierfür ist es, ein Zeitstempel zu verwenden und ein daraus resultierender Time-out der Daten.

Ein anders Beispiel für ein Problem, das durch Caching ausgelöst wurde ist ein Bug, der im Jahr 2015 auf der Spieleplattform Steam aufgetreten ist. Laut Steam-Blogpost[26] wurde eine DoS Attacke während der Weihnachtszeit auf Steam ausgeführt. Dieser Angriff hat einen Sicherheitsmechanismus ausgelöst, dieser Mechanismus verwendet unter anderem Caching um mehr Nutzeranfragen beantworten zu können. Daraus resultierte das Problem, das Nutzer Teile der persönlichen Daten von anderen Nutzern einsehen konnten. Bei dem Aufrufen des Profils wurde nicht das eigene Profil angezeigt, sondern das Profil, das sich aktuell im Cache des Servers befindet.

### CRUD-Operationen

Das CRUD-Pattern beschreibt 4 Funktionalitäten [1].

1. Create beschreibt das Erstellen eines Datensatzes
2. Read beschreibt das Lesen eines Datensatzes
3. Update beschreibt das Aktualisieren eines Datensatzes
4. Delete beschreibt das Löschen eines Datensatzes.

Das Akronym ergibt sich durch den ersten Buchstaben jeder Aufgabe. Als Synonym kann man die Key-Wörter, die in SQL verwendet werden, betrachten. Im Detail ist Insert, Select, Update und Delete eine Anwendung der CRUD-Operationen.

Da dieser Bereich das Interface zwischen Datenbank und Programm-Code ist, ist es relevant, diese Code-Bereiche möglichst performant umzusetzen (Kapitel 7.1).

### Anwendung CRUD

Ein Beispiel für die Anwendung von CRUD ist im Representational State Transfer, kurz REST, zu finden. REST beschreibt ein Design-Pattern für Programmierschnittstellen, das sich an das Word Web richtet [27]. Ursprünglich wurde der Begriff von Roy Fielding geprägt [27].

Rest ist weder ein Protokoll noch ein Standard [28], jede API kann REST in einer anderen Form umsetzen. Zumeist bedienen sich die APIs laut [27] an HTTP und JSON sowie HTML und XML.

Nach [29] findet Rest ebenso Anwendung in der Mikroservice Architektur, hierbei wird Code-Funktionalität eines Code-Monolithen aufgeteilt in kleinere Bausteine. Diese Bausteine sind über das HTTP Protokoll und Rest-APIs verbunden. Dies ermöglicht es selbst Legacy Komponenten durch die Erschaffung eines Webinterfaces anzusprechen. Des Weiteren wird die Wartbarkeit und Skalierbarkeit der Komponenten erhöht.

## Performance von APIs

Ein Problem in der Software Entwicklung ist der Begriff Economy of Scale [30] Das Modell lässt sich konkreter auf die Software-Entwicklung anwenden und mit folgenden Punkten beschreiben:

* Umso besser die Latenz der API bzw. der Operationen umso mehr Anfragen kann der Server/API-Benutzer in gleicher Zeit bearbeiten.
* Eine bessere Performance kann zu weniger verbrauchten Server Hardware Ressourcen führen.
* Server bzw. CPU-Zeit sind Fixkosten, die beim Betreiben der API entstehen.

Als Beispiel dient eine Web-API, die Anfragen zum aktuellen Wetter beantwortet. Ein Client sendet eine Anfrage an die API und bezahlt einen Betrag X für die Benutzung der pro Anfrage. Durch eine bessere Performance ist es dem\*der Betreiber\*in der API möglich, mehr Anfragen zu beantworten und mehr Geld bei gleichbleibender Hardware zu verdienen. Wenn die Performance konstant bleibt, so müsste der\*die Betreiber\*in neue Hardware beziehungsweise Ressourcen kaufen oder mieten, um mehr Anfragen in gleicher Zeit beantworten zu können.

Ein weiteres Beispiel liefert Netflix auf dem hauseignen Dev-Blog. Als Beispiel dient der Blog-Eintrag zu Optimizing data warehouse storage [31]. In diesem Blog wird von einer Situation geschildert, bei dem Netflix mehrere Petabytes an Daten in AWS S3 Maschinen speichert. Angeführt werden verschiedene Strategien zum Verringern der Datenmengen durch Mergen, Sortierung und Komprimierung dieser.

Wie anhand der Beispiele ersichtlich ist, ist die Optimierung im kommerziellen Bereich von größerer Relevanz.

### Performance von Software

Die Performance von Software lässt sich verschieden Definieren. Wie bereits im Kapitel 8 ersichtlich gibt es unter anderem den Unterschied zwischen Latenz und Speicherverbrauch. Nach [32] handelt es sich dabei um traditionelle Attribute von Software Performance. Es gibt auch noch weitere Attribute, die nicht bedingt durch Zahlen messbar sind wie Wartbarkeit und Erweiterbarkeit.

#### Messbarkeit

Die Ausführzeit von Software lässt sich messen. Als Beispiel wie viel Zeit vergeht von Beginn der Anfrage bis zu der Antwort auf diese durch die API. Aufgrund des Prozess-Scheduling [33] des Betriebssystems sowie andere Hintergrundprozesse, die auf der gleichen Hardware laufen, ist es relevant, mehrere Messungen zu Betreiben und einen Median bzw. Mittelwert zu erheben.

# Forschungsfragen und Methodik

In diesem Kapitel werden die Forschungsfragen und die Methodik beschrieben.

## Forschungsfragen

Es ergeben sich folgende Forschungsfragen:

1. Welches der Frameworks bietet die beste Ausführungszeit für Create, Read, Update und Delete Funktionalität?
2. Wie unterscheiden sich die Frameworks in der Anwendung?

## Methodik

Mithilfe von Vergleichsmerkmalen werden die ORM Frameworks analysiert und verglichen.

### Definitionen

Im weiteren Verlauf der Arbeit ist der Begriff Performance mit der Programmlaufzeit des gegebenen Frameworks gleichgesetzt.

## Auswahl der ORM Frameworks

Die Auswahl der Frameworks erfolgte nach dem Prinzip der freiwilligen Bereitstellung. Insgesamt wurden 4 ORM Frameworks mit folgenden Charakteristiken zur Verfügung gestellt:

1. C# mit Datenbanksystem PostgreSQL
2. C# mit Datenbanksystem PostgreSQL
3. C# mit Datenbanksystem PostgreSQL
4. JAVA mit Datenbanksystem PostgreSQL

In weiteren Teilen der Arbeit werden die ORM Frameworks nachfolgendem Muster benannt Fx wobei x eine fortlaufende Zahl ist. F0 entspricht dabei Entity Framework Core.

## Entity Framework Core

Aufgrund der aktuellen Trendlage der Auswahl wird der Fokus im weiteren Verlauf der Arbeit auf einen Vergleich von Entity Framework Core zu den neu entwickelten Frameworks gelenkt. Entity Framework Core wurde als Framework F0 herangezogen. In Folge einer Sourcecode Analyse erkennt man bei dem Framework F2 eine Ähnlichkeit zu Entity Framework, es werden ebenfalls DBSets benutzt.

## Vorherrschen von PostgresSQL in den neuen ORM-Frameworks

Die Spezialisierung von den ORM Frameworks zu PostgreSQL Datenbanken lässt sich durch die Entwicklungsvorgaben von Lehrveranstaltungen im Semester drei und vier erklären. Hier wurde als Vorgabe PostgreSQL genommen. Zu erwähnen ist das es im fünften Semester und somit an die OR-Mapper keine Vorgabe an das DBMS gab.

## Einschränkungen

Aufgrund der zur Verfügung stehenden ORM Frameworks wird F4 ausgeschlossen. Eine Unterscheidung und Vergleich von C# und Java ORM Frameworks wäre nur bei einer größeren Auswahl an JAVA ORM Frameworks möglich.

# Vergleichsmerkmal Definition

In diesem Kapitel werden die Vergleichsmerkmale näher definiert. Vergleichsmerkmal 1-4 orientieren sich dabei an dem CRUD design Schema.

## Vergleichsmerkmal Unterschiede Anwendung

Das Vergleichsmerkmal beschreibt Unterschiede in der Anwendung der ORM Frameworks zueinander.

## Vergleichsmerkmal Performance Select

Das Vergleichsmerkmal Performance Select, beschreibt die Programmlaufzeit des ORM Frameworks bei dem Abrufen von Daten aus der Datenbank.

## Vergleichsmerkmal Performance Insert

Das Vergleichsmerkmal Performance Select, beschreibt die Programmlaufzeit des ORM Frameworks bei dem Einfügen von Daten in die Datenbank.

## Vergleichsmerkmal Performance Update

Das Vergleichsmerkmal Performance Select, beschreibt die Programmlaufzeit des ORM Frameworks bei dem Updaten von Daten in die Datenbank.

## Vergleichsmerkmal Performance Delete

Das Vergleichsmerkmal Performance Select, beschreibt die Programmlaufzeit des ORM Frameworks bei dem Löschen von Daten aus der Datenbank.

## Test Kriterien

In diesem Kapitel werden die Ausführung und Durchführung der einzelnen Merkmale beschrieben.

### Konfiguration Benchmark.net

Für alle Vergleichsmerkmale gilt, die Ergebnisse werden automatisch von Benchmark.net berechnet. Für alle Tests wurde 25 Warm-Ups und 25 Invokations ausgeführt. Es wurde konfiguriert, dass Ausreiser bei der Median Berechnung mit einbezogen werden. Als Messmethode wurde Throughput ausgewählt.

### Initial Zustände

Alle Tests sollen einen Einheitlichen Initial Zustand für alle Tests besitzen. Dies soll erreicht werden durch eine Entsprechende IterationSetup und IterationCleanup Methode in allen Testklassen. Hintergedanke ist es einen gleichen Datenbankzustand für alle Durchläufe zu haben. Dies inkludiert ein zurücksetzen eventuell vorhandener Caches.

### Tools

Es wurde Docker in der Version 4.3.2 verwendet sowie R Statistics in der Version 4.1.3. Es wurde Benchmark.net in Version 0.13.1 verwendet, um die Tests auszuführen und zu messen.

### Test-Entitäten

In diesem Kapitel werden die Test-Objekte beschrieben. Die Objekte resultieren aus dem in Kapitel 13.9 angeführten Datenbankschema. Bei der Umsetzung der Objekte gibt es Unterschiede bei der Namensgebung der Objekt-Properties. Diese Properties haben einen verschiedenen Namen, da es bei den neu entwickelten Frameworks teilweise durch das Framework erzwungene Namensgebung gibt. Der spezifische Aufbau der relevanten Klassen mit den Entitäten sind in den folgenden Projekten zu finden ORM0Entites, ORM1Entities, ORM2Entites und ORM3Entities.

### Daten der Tests

Die Objekte der Listen werden mittels des Tools Nbuilder zufällig mit entsprechenden Daten befüllt. Die Klassen, die das Befüllen übernehmen, sind in den gleichen Projekten wie die Test-Entitäten zu finden und dem Namen NBuilderHelper.cs.

#### Häufigkeit bei Beziehungstypen

Objekte mit einer Beziehung vom Typ Eins-zu-Viele und Viele-zu-Viele können wenig bis viele Objekte in der Liste Referenz besitzen. Die Konfiguration der Häufigkeit ist in der Klasse GlobalConfig.cs im Projekt GlobalConfigBase zu finden.

Zusammengefasst:

* Jedes Buch hat zwei Kapitel, jedes Kapitel hat zwei Seiten.
* Es gibt zwei Artikel, jede Rechnung hat zwei Artikel und somit hat auch jeder Artikel t alle Rechnungen als Referenz.

### Kompilierung und Ausführung

Das Testprogramm wurde mittels der IDE Rider 2021.3.1 Kompiliert. Als Parameter wurde Release und AnyCpu gewählt. Die Ausführung des Programms erfolgte mittels MingW-Shell für Windows. Mittels eines Shell-Scripts wurde das Programm mit einem Befehlszeilenargument aufgerufen, dies ruft wiederum einen der Tests auf. Dies soll bewirken, dass für jeden Test das Programm neu gestartet wird und eine Abhängigkeit zu vorherigen Tests vermindert wird.

### Hardware & Software

Für die Tests wurde folgende Hardware und Software in folgender Version verwendet.

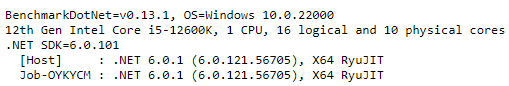


Abbildung 4: Screenshot Hardware und Software

Zusätzlich wurde als Speichermedium eine NVME-SSD mit der Modellbezeichnung, Samsung 980 1 TB PCIe 3.0 (MZ-V8V1T0BW) bei 50% Speicherfüllung verwendet.

### Test Datenbankmangementsystem

Bei der Datenbank, die für die Tests verwendet wird, handelt es sich um eine lokale Docker-Instanz einer Postgres Datenbank in Dockercontainer-Version 14.2 verwendet. Start und Konfiguration erfolgt mittels einer dockercompose Datei. Zu finden im Ordner Docker unter dem Namen docker-compose.yaml

### Rahmenbedingungen für die erstellten Tests

Alle Tests werden mit Entitys, die 3(4 mit NoRelation) Beziehungsarten repräsentieren, durchgeführt: NoRelation, Eins-zu-Eins, Eins-zu-Viele und Viele-zu-Viele

Mithilfe des Frameworks NBuilder wurde zufällige Objekte generiert und mit Daten gefüllt. Durch die Zufälligkeit der Datensätze soll bewirkt werden, dass sich die Objekte durch mehr als die Id unterscheiden. Dies soll bewirken, dass sich der UseCase an eine reale Verwendung annähert.

#### Select Spezifisch

Für alle Beziehungsvarianten wird eine Liste Z von Größe 500 außerhalb der Messung erstellt und in die Datenbank eingefügt. Die Daten werden anschließend nicht neu geladen, sondern die gleiche Liste wird wiederverwendet zum Laden der Daten. Die Objekt-Id wird zum Laden benötigt. Innerhalb der Messung werden mittels Eager-Loading alle Objekte wieder aus der Datenbank in eine Liste geladen. Zu Beginn der Messung dürfen sich keine Objekte im ORM Framework Cache befinden.

#### Insert Spezifisch

Für alle Beziehungsvarianten wird eine Liste Z von Größe 500 außerhalb der Messung erstellt. Innerhalb der Messung werden die Datensätze der Liste gespeichert. Spätestens nachdem ein Datensatz der Liste Z von einer CRUD-Operation behandelt wurde, soll ein entsprechender Speicheraufruf in der Datenbank stattfinden. Hierbei ist es nicht relevant, ob es durch das Framework oder manuell durch einen Methodenaufruf stattfindet. Zu Beginn der Messung dürfen sich keine Objekte im ORM Framework Cache befinden.

#### Update Spezifisch

Für alle Beziehungsvarianten wird eine Liste Z von Größe 500 außerhalb der Messung erstellt und in die Datenbank eingefügt. Die Daten werden anschließend nicht neu geladen, sondern die gleiche Liste wird wiederverwendet zum Aktualisieren der Daten.

Zu Beginn der Messung dürfen sich keine Objekte im ORM Framework Cache befinden. Nachdem für jeden Datensatz der Liste Z folgenden Regeln angewendet wurden, soll der aktualisierte Zustand in der Datenbank gespeichert werden.

Aktualisierungsregeln:

* Beziehungstyp NoRelation
  + Der Parameter Email wird durch „aaaaaaaaa“ ersetzt.
  + Der Parameter Customerlikescolorgreen wird durch seine Negierung ersetzt
  + Der Parameter Lastname wird durch „aaaaaaaaa“ ersetzt.
* Beziehungstyp Eins-zu-Eins
  + Der Parameter name wird durch „aaaaaaaaa“ ersetzt
  + Der Parameter weaponname wird durch „aaaaaaaaa“ ersetzt
  + Der Parameter damage wird ersetzt durch den aktuellen Schleifenzähler mal zwei
* Beziehungstyp Eins-zu-Viele
  + Der Buchname wird ersetzt mit „aaaaaaaaa“
  + Der Parameter Chaptername des ersten Kapitels wird ersetzt mit „aaaaaaaaa“
  + Der Text der ersten Seite des ersten Kapitels wird ersetzt mit „aaaaaaaaa“
* Beziehungstype Viele-zu-Viele
  + Der Kaufpreis wird auf 1 gesetzt.
  + Der Aritkelname des ersten Artikels einer Rechnung wird auf „aaaaaaaaa“ gesetzt
  + Der IsHidden Zustand eines Artikels wird auf seinen Negierten Wert gesetzt.

#### Delete Spezifisch

Für alle Beziehungsvarianten wird eine Liste Z von Größe 500 außerhalb der Messung erstellt und in die Datenbank eingefügt. Die Daten werden anschließend nicht neu geladen, sondern die gleiche Liste wird wiederverwendet zum Löschen der Daten. Zu Beginn des Löschens muss sich der Cache des Frameworks in einem leeren Zustand befinden. Spätestens nachdem ein Datensatz gelöscht wurde, soll der Zustand in der Datenbank gespeichert werden. Ausnahme bilden hierbei die Viele-zu-Viele Beziehungen, hierbei werden nur die Rechnungen gelöscht. Zu Beginn der Messung dürfen sich keine Objekte im ORM Framework Cache befinden.

### Testaufbau im Detail

Nähere Informationen zu dem Aufbau der Tests sind im GIT Repository zu finden. Eine Benennung der Tests und Klassen erfolgt nachfolgendem Muster:

<FrameworkName><CRUD-Operation>Performance<Beziehungsart>.cs

Ein Beispiel für das Framework F0 bei der Operation Insert liefert folgende Klassen:

* F0InsertPerformanceNoRelation.cs
* F0InsertPerformance1to1.cs
* F0InsertPerformance1ton.cs
* F0InsertPerformanceNtoM.cs

### Datenbankschema

Die erstellten Entitäten repräsentieren folgende Entity Relationship Modelle

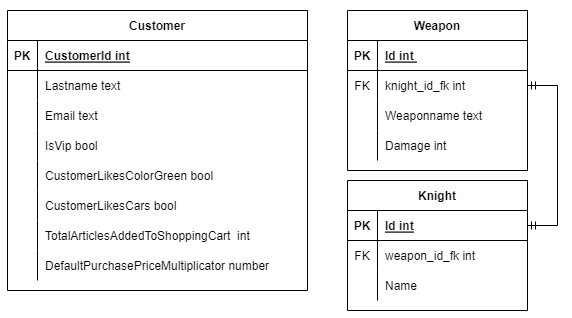


Abbildung 5 ER-Diagramm NoRelation (links) und Eins-zu-Eins (rechts)

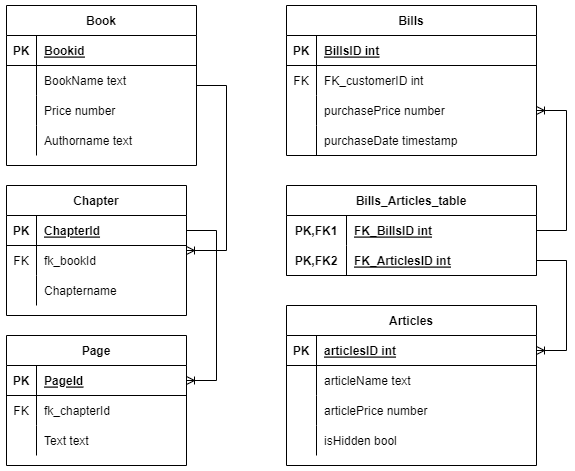


Abbildung 6 ER-Diagramm Eins-zu-Viele (links) und Viele-zu-Viele(rechts)

## SQL DDL Create Tables

Es gibt für jedes Framework ein eigenes SQL-Drop und Create Skript, das den entsprechenden Zustand der Datenbank erstellt. Die Daten sind zu finden im Projekt Data. Jedes Framework enthält ein Script mit nachfolgendem Schema: <ORMKürzel>CreateAndDrop.sql

## Source Code Manipulation

Im Zuge der Arbeit wurden Veränderungen am Sourcecode der neu entwickelten ORM Frameworks durchgeführt.

### Änderungen Console.Writeline;

F3 hatte im Sourcecode mehrere Console.Writeline Statements zum Debuggen und verfolgen des Programmflusses. Dies hat bewirkt, dass die Testlogs durch Debug Statements des Frameworks gefüllt wurden. Die Console.Writeline Statements wurden entfernt, da es sich dabei um Debugstatements haltet, die in Produktion Code entfernt werden würden.

### Debug

In F1, Orm/Orm.Core/DbContext.cs wurde in Zeile 199-201 in Absprache wurde ein Debug Statement entfernt das bewirkt hat das, sobald ein Objekt mit ID 5 auftritt, ein Fehler entsteht. Dies hat bewirkt, dass kein erfolgreiches Ausführen des ORM Frameworks möglich wäre.

Im angeführten Auszug entspricht Zeile 1, Zeile 196 der Klasse DbContext.cs im Namespace Orm.Core. Der Bug wird ausgelöst durch Zeile 4/5

CustomExpression? expression = new CustomExpression(table.PrimaryKey.ColumnName, CustomOperations.Equals,

id);

//if (Convert.ToInt32(id) == 5)

//expression = null;

DataTable result = \_db.Select(table.Name, null, null, null, expression);

\_logger.Information($"Getting Entity {typeof(T).FullName} successfully");

### Cache

Um die entsprechenden Frameworks sinnvoll zu testen, muss bereitgestellt sein, dass ein Framework Test immer unter den gleichen Bedingungen stattfinden kann und keine Unterschiede durch mehrmaliges durchlaufen des gleichen Tests geschehen sollen. Die Frameworks F1-F3 besitzen einen internen Objektcache. Dieser ist standardgemäß als eine private Variable innerhalb des Projektes definiert. Für die Tests wurde der entsprechende Cache zu einer Public Variable umgewandelt. Es ist nun möglich, den Cache für jede Ausführung durch einen neuen leeren Cache zu ersetzen. Für das Framework F0 wird ein neuer Context bei jedem Run angelegt.

# Ergebnisse Unterschiede in der Anwendung der ORM-Frameworks anhand der Erstellten Tests

In diesem Kapitel werden Besonderheiten der Tests bei der Umsetzung beschrieben.

### F0 Entity Framework Core

Bei Viele-zu-Viele Beziehungen im Bereich Insert werden zunächst für alle Artikel die Referenzen zu den Rechnungen auf eine neue Liste gesetzt, um das automatische Hinzufügen aller Referenzen zu ermöglichen

foreach (Bill item in listInsertBills)

{

foreach (Article item2 in item.FkArticles)

{

item2.FkBills = new List<Bill>();

}

}

for (var i = 0; i < listInsertBills.Count; i++)

{

context.Bills.Add(listInsertBills[i]);

context.SaveChanges();

}

Für die Select-Tests kann ein Eager-Loading verhalten mittels Include erzwungen werden

for (int i = 0; i < listInsertKnight.Count; i++)

{

listGetKnight.Add(context.Knights.Include(c => c.FkWeapon).First(c => c.Id == listInsertKnight[i].Id));

}

## Framework F1

Bei einem Insert von Daten müssen alle Referenzen auf neue Listen gesetzt werden, um ein hinzufügen der Objekte zu ermöglichen.

for (int i = 0; i < listInsertBooks.Count; i++)

{

List<Chapters> tempPointerI = listInsertBooks[i].Chapter;

listInsertBooks[i].Chapter = new List<Chapters>();

\_dbContext.Add(listInsertBooks[i]);

for (int k = 0; k < tempPointerI.Count; k++)

{

List<Pages> tempPointerK = tempPointerI[k].Pages;

tempPointerI[k].Pages = new List<Pages>();

\_dbContext.Add(tempPointerI[k]);

for (int j = 0; j < tempPointerK.Count; j++)

{

\_dbContext.Add(tempPointerK[j]);

}

}

}

Alle Referenzen müssen manuell bearbeitet werden.

for (int i = 0; i < listInsertKnight.Count; i++)

{

\_dbContext.Delete<Knight>(listInsertKnight[i].Id);

\_dbContext.Delete<Weapon>(listInsertKnight[i].Weapon.Id);

}

Bei einem Insert von Viele-zu-Viele Beziehungen muss mittels eines Umweges erfolgen, sollten noch keine Artikel mit der entsprechenden Id enthalten sein.

for (int i = 0; i < listInsertBills.Count; i++)

{

List<Articles> tempPointerI = listInsertBills[i].FkArticles;

for (int k = 0; k < tempPointerI.Count; k++)

{

if (FluentApi.Get<Articles>().EqualTo("id", tempPointerI[k].Id).Execute(\_dbContext).Count == 0)

{

tempPointerI[k].FkBills = new List<Bills>();

tempPointerI[k] = \_dbContext.Add(tempPointerI[k]);

}

}

\_dbContext.Add(listInsertBills[i]);

}

## Framework F2

Alle Referenzen müssen manuell bearbeitet werden für alle Operationen außer bei der Read Operation im Gegensatz zu Framework F1 ist ein zurücksetzen der Listen nicht erforderlich.

for (int i = 0; i < listInsertBooks.Count; i++)

{

\_dbContext.Save(listInsertBooks[i]);

for (int k = 0; k< listInsertBooks[i].chapters.Count; k++)

{

\_dbContext.Save(listInsertBooks[i].chapters[k]);

for (int j = 0; j < listInsertBooks[i].chapters[k].Pages.Count; j++)

{

\_dbContext.Save(listInsertBooks[i].chapters[k].Pages[j]);

}

}

}

Bei der Select Operation wird ein Eager-Loading verhalten folgendermaßen erzwungen.

for (int i = 0; i < listInsertBooks.Count; i++)

{

listGetBooks.Add(\_dbContext.GetById<books>(listInsertBooks[i].bookid));

for (int j = 0; j < listGetBooks[i].chapters.Count; j++)

{

for (int k = 0; k < listGetBooks[i].chapters[j].Pages.Count;k++)

{

\_ = listGetBooks[i].chapters[j].Pages[k].pagesid;

}

}

## Framework F3

Für alle Operationen muss für alle Referenzen die Operation manuell ausgeführt werden.

for (int i = 0; i < listInsertBooks.Count; i++)

{

listInsertBooks[i].Save();

for (int k = 0; k < listInsertBooks[i].Chapters.Count; k++)

{

listInsertBooks[i].Chapters[k].Save();

for (int j = 0; j < listInsertBooks[i].Chapters[k].Pages.Count; j++)

{

listInsertBooks[i].Chapters[k].Pages[j].Save();

}

}

}

Ein Aufbau von Tests für die Beziehungsart Viele-zu-Viele bei allen CRUD-Operationen war nicht möglich. Diese Beziehungsart ist von dem Framework nicht unterstützt. Ebenso konnten keine Tests für die Select Operation im Bereich Eins-zu-Eins, Eins-zu-Viele und Viele-zu-Viele angelegt werden. Hier kommt es zu einer Reflection-Exception bei der Erstellung der Objekte. Vermutlich wurde bei dem Entwickeln von Lazy-Loading nicht auf eine weiterhin funktionale Version von Eager-Loading geachtet.

## Unterschiede Entitys

In diesem Kapitel werden weitere Unterschiede der Anwendung beschrieben.

### F0 Entity Framework Core

Die Entitys für das Framework Entity Framework Core wurden mit einem bereitgestellten CLI-Tool automatisch generiert. Die Konfiguration des Modells erfolgt über einen DBContext der die Beziehungen abbildet.

Dotnet ef dbcontext scaffold <connection\_string> Npgsql.EnitiyFrameworkCore.PostgreSQL -d

Ein DBContext dient als EntryPoint für das ORM Framework.

### F1 Framework

Die Entitys mussten manuell erstellt werden. Es wurden Primarykeys definiert mittels entsprechenden Attributs. Zusätzlich wurde der Parameter autoIncrement auf false gesetzt, da bei den Tests von mir generierte Ids vergeben werden. Foreignkeys für Eins-zu-Viele und Viele-zu-Viele Beziehungen müssen manuell gesetzt werden mittels vorgegebener Attribute. Alle Primarykeys müssen trotz Kennzeichnung mittels Primarykeyattribute Id genannt sein. Hier wäre nur eine Kennzeichnung durch ein autoincrement Attribut sinnvoller.

Es gibt eine einen DBContext der Entrypoint für das ORM Framework abbildet.

### F2 Framework

Die Entitys mussten manuell erstellt werden. Für dieses Framework ist nur die Vergabe eines Keyattributes zur Kennzeichnung des Primarykeys notwendig. Entitys leiten Eins-zu-Viele und Viele-zu-Viele Beziehungen automatisch ab. Es wurde ähnlich zu Entity Framework Core ein DBContext angelegt. Dieser inkludiert DBSets die wiederum der Einstiegspunkt für die Entitys in den ORMapper widerspiegeln.

Es gibt einen DBContext der Entrypoint für das ORM Framework abbildet.

### F3 Framework

Die Entitys mussten manuell erstellt werden. Alle Entitys müssen von einem ORM Entity namens Entitys ableiten. Dieses fügt die Methoden Save und Delete hinzu, diese rufen intern das ORM Framework auf.

Gleich wie in F2 müssen für die Verwendung dieses Frameworks nur Primarykey Attribute gesetzt werden. Komplexe Beziehungen (Eins-zu-Viele,Viele-zu-Viele) werden automatisch abgeleitet. Dies führt zu dem Problem, das Foreignkeys und die Viele-zu-Viele zwischen Tabelle einen fest vorgegebenen Namen im Datenbank Schema haben muss. Zusätzlich müssen alle Primarykeys den Namen Id haben, da ein GetByID nach einem Parameter Id des Objektes sucht.

Es gibt eine statische Referenz auf das ORM Framework die entsprechenden Methoden Aufrufe zur Verfügung stellt.

## Ergebnisse Performance

In diesem Kapitel werden die Ergebnisse der Vergleichsmerkmale Tabellarisch dargestellt.

### Insert Eins-zu-Eins

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 332.8 ms | 3.5 ms | 3.89 ms |
| F1 | 500 | 2137 ms | 17.6 ms | 20.2 ms |
| F2 | 500 | 931.2 ms | 7.63 ms | 8.79 ms |
| F3 | 500 | 975.3 ms | 6.11 ms | 7.03 ms |

Tabelle 2: Resultate Insert Eins-zu-Eins

### Insert NoRelation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 748.2 ms | 4.27 ms | 4.92 ms |
| F1 | 500 | 975.0 ms | 3.50 ms | 4.03 ms |
| F2 | 500 | 481.4 ms | 4.16 ms | 4.79 ms |
| F3 | 500 | 483.5 ms | 2.62 ms | 2.91 ms |

Tabelle 3: Resultate Insert NoRelation

### Insert Eins-zu-Viele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 4023 ms | 159.6 ms | 183.8 ms |
| F1 | 500 | 1299 ms | 10.7 ms | 11.9 ms |
| F2 | 500 | 3557 ms | 20.0 ms | 23.0 ms |
| F3 | 500 | 3585 ms | 15.5m s | 17.8 ms |

Tabelle 4: Resultate Insert Eins-zu-Viele

### Insert Viele-zu-Viele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 1064 ms | 5.3 ms | 6.2 ms |
| F1 | 500 | 12330 ms | 611 ms | 704 ms |
| F2 | 500 | 2041 ms | 16.6 ms | 19.1 ms |
| F3 | - | - | - | - |

Tabelle 5: Resultate Insert Viele-zu-Viele

### Select Eins-zu-Eins

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 988.8 ms | 6.47 ms | 7.45 ms |
| F1 | 500 | 407.8 ms | 7.40 ms | 8.52 ms |
| F2 | 500 | 631.7 ms | 25.12 ms | 28.93 ms |
| F3 | - | - | - | - |

Tabelle 6: Resultate Select Eins-zu-Eins

### Select NoRelation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 318.7 ms | 3.41 ms | 3.79 ms |
| F1 | 500 | 215.4 ms | 3.69 ms | 4.10 ms |
| F2 | 500 | 245.3 ms | 6.64 ms | 7.38 ms |
| F3 | 500 | 181.8 ms | 2.38 ms | 2.74 ms |

Tabelle 7: Resultate Select NoRelation

### Select Eins-zu-Viele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 1110 ms | 111.1 ms | 109.2 ms |
| F1 | 500 | 1342 ms | 22.0 ms | 25.3 ms |
| F2 | 500 | 1368 ms | 59.5 ms | 68.6 ms |
| F3 | - | - | - | - |

Tabelle 8: Resultate Select 1 zu N

### Select Viele-zu-Viele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 391.6 ms | 4.23 ms | 4.87 ms |
| F1 | 500 | 795.2 ms | 11.62 ms | 13.39 ms |
| F2 | - | - | - | - |
| F3 | - | - | - | - |

Tabelle 9: Resultate Select Viele-zu-Viele

### Update Eins-zu-Eins

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 1021 s | 3.4 ms | 3.9 ms |
| F1 | 500 | 1678 s | 15.5 ms | 17.8 ms |
| F2 | 500 | 932.4 ms | 6.09 ms | 7.01 ms |
| F3 | 500 | 961.0 ms | 5.43 ms | 6.04 ms |

Tabelle 10: Resultate Update Eins-zu-Eins

### Update NoRelation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 892.7 ms | 6.19 ms | 6.88 ms |
| F1 | 500 | 743.0 ms | 4.65 ms | 5.35 ms |
| F2 | 500 | 471.4 ms | 4.99 ms | 5.75 ms |
| F2 | 500 | 479.8 ms | 5.31 ms | 6.11 ms |

Tabelle 11: Resultate Update NoRelation

### Update Eins-zu-Viele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 2816 ms | 6.8 ms | 7.3 ms |
| F1 | 500 | 8347 ms | 102.4 ms | 117.9 ms |
| F2 | 500 | 1446 ms | 20.7 ms | 23.1 ms |
| F3 | 500 | 1406 ms | 11.6 ms | 13.3 ms |

Tabelle 12: Resultate Update Eins-zu-Viele

### Update Viele-zu-Viele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 1089 ms | 6.1 ms | 6.8 ms |
| F1 | 500 | 9489 ms | 28.5 ms | 31.7 ms |
| F2 | 500 | 2322 ms | 17.1 ms | 19.0 ms |
| F3 | - | - | - | - |

Tabelle 13: Resultate Update n zu m

### Delete Eins-zu-Eins

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 888.3 ms | 2.63ms | 3.03ms |
| F1 | 500 | 1010 ms | 7.4 ms | 8.5 ms |
| F2 | 500 | 927.1 ms | 9.04 ms | 10.41 ms |
| F3 | 500 | 984.7 ms | 11.69 ms | 13.46 ms |

Tabelle 14: Resultate Delete Eins-zu-Eins

### Delete NoRelation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 733.4 ms | 3.24 ms | 3.6 ms |
| F1 | 500 | 485.0 ms | 11.57 ms | 13.32 ms |
| F2 | 500 | 455.1 ms | 7.21 ms | 8.01 ms |
| F3 | 500 | 459.6 ms | 3.38 ms | 3.90 ms |

Tabelle 15: Resultate Delete NoRelation

### Delete Eins-zu-Viele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| F0 | 500 | 2264 ms | 87.8 ms | 101.2 ms |
| F1 | 500 | 3617 ms | 35.5 ms | 34.9 ms |
| F2 | 500 | 4529 ms | 446.3 ms | 513.9 ms |
| F3 | 500 | 9320 ms | 794.5 ms | 914.9 ms |

Tabelle 16: Resultate Delete Eins-zu-Viele

### Delete Viele-zu-Viele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | numberOfStatements | Mean | Error | StdDev |
| f0 | 500 | 1085 ms | 4.6 ms | 5.2 ms |
| f1 | 500 | 1099m s | 161.0 ms | 172.2 ms |
| f2 | 500 | 451.4 ms | 3.99 ms | 4.60 ms |
| f3 | - | - | - | - |

Tabelle 17: Resultate Delete Viele-zu-Viele

## Weitere Statistiken

Weitere Statistiken und Daten können im Repository im Ordner Datengesammelt gefunden werden.

## Fehlende Werte

Für alle Werte, bei denen ein – als Wert eingetragen wurden, konnten keine Tests erstellt werden oder erfolgreich ausgeführt werden.

# Analyse

Im Durchschnittsfall bieten F1, F2 und F3 gleich gute Performance oder leicht bessere Performance unter den gegebenen Bedingungen im Vergleich zu Entity Framework Core.

Hierbei ist es wichtig zu beachten, dass die neu entwickelten Frameworks weniger Features als Entity Framework Core haben. Unter anderem verwenden diese Frameworks, keine Transactions. Zusätzlich werden von Entity Framework verschiedene Modellierungstypen unterstützt. Ebenso unterstützen die neu entwickelten Frameworks nur ein Datenbankmanagementsystem.

Im Detail wird ersichtlich, dass das Framework F1 in Teilgebieten um ein Vielfaches schlechter abschneiden als Entity Framework Core. Als Beispiel dient hier Insert Viele-zu-Viele, F1 ist hier etwa 12-fach langsamer als Entity Framework Core und etwa 6-mal langsamer als Framework F2. Die schlechtere Performance für komplexe Beziehungstypen (Eins-zu-Viele, Viele-zu-Viele) für das Framework F1 kann erklärt werden, da bei jeder CRUD Operation alle Referenzen des Objektes mittels eines Read neu geladen und aktualisiert werden. Vor allem im Beziehungsbereich Viele-zu-Viele ist dies in den Daten ersichtlich.

Im Bereich unterstütze Beziehungen bieten F0, F1 und F2 eine Unterstützung für alle Beziehungstypen. Ausnahme bildet hier F2 im Bereich Select Viele-zu-Viele Beziehungen. Eine theoretische Unterstützung der Beziehung ist vorhanden. Mit fortlaufender Iterationanzahl erhöht sich die Laufzeit massiv. Zu Beginn benötigt eine Iteration etwa 5 Sekunden, am Ende mehr als 5 Minuten pro Iteration. Ein Einsatz des Frameworks in diesem Bereich nicht realistisch und zu empfehlen.

Das Framework F3 schneidet im Bereich unterstütze Beziehungen am schlechtesten ab. Im Voraus wurde von dem\*der Entwickler\*in angegeben, dass Viele-zu-Viele Beziehungen für jede CRUD Operation nicht unterstützt werden. Dies hat sich auch als solches bewahrheitet. Zusätzlich zu den nicht funktionalen N zu M Beziehungen sind im Bereich Select sämtliche Beziehungen nicht verwendbar. Es kommt hier zu einer Exception. Der Fehler entsteht dadurch, dass in den Tests kein Lazy-Loading verwendet wurde. Beim Select wird für alle Referenztypen versucht, den „Underlying-Type“ zu bekommen. Ein Beispiel hierfür ist Lazy<T>, hierbei ist T der Underlying-Type. In den Tests wurde Eager-Loading verwendet, wodurch es nur ein T gibt und kein Lazy<T>. Da es sich bei diesem Fehler um einen Logikfehler handelt, an dem an mehreren Stellen Fehler behoben werden müssten, erfolgte hier keine Korrektur des Source-Codes.

# Conclusio

Das Framework F0, Entity Framework Core unterstützt alle Beziehungen und bietet die beste oder annähend gleiche Performance in den meisten Teilbereichen. Es ist daher das beste Framework unter den gegebenen Bedingungen.

Unter der Bedingung, dass alle Beziehungstypen relevant sind, ist das Framework F1 am zweit besten und somit das beste Framework der neu Entwickelten. Wenn auf eine Verwendung von n zu m Beziehungen verzichtet werden kann, ist das Framework F2 dem Framework F1 vorzuziehen. Die Performance von dem Framework F2 ist Bereich Insert und Update besser als F1.

Das Framework F3 ist am schlechtesten zu bewerten, da am wenigsten Beziehungstypen unterstützt werden, in den Bereichen Insert, Delete und Update ist die Performance ähnlich zu F1 und F2, die fehlende Unterstützung für Viele-zu-Viele Beziehungen und auch der nicht funktionale Bereich Select führt dazu, dass das Framework nur mit vielen Einschränkungen benutzbar ist.

Im Rahmen dieses Tests ist das Framework F0, also Entity Core, das beste Framework im Bereich Anwendung. Referenzbeziehung werden automatisch bearbeitet und es werden keine zwischen Referenzen benötigt. Am nächstbesten ist das Framework F2 und F3, es müssen Referenzen einzeln bearbeitet werden, aber es wird ein komplexeres Arbeiten mit Pointern und zwischen Referenzen vermieden. Das Framework F1 ist im Bereich Anwendung das Schlechteste. Es muss mit zwischen Referenzen und Pointern gearbeitet werden. In einem Fall muss bei einem Insert ein GET erfolgen, um ein anderes Verhalten einzuleiten, sollten noch keine Objekte des Referenztyps in der Datenbank sein.

## Ausblick

Ein Thema für Folgearbeiten ergibt sich in der Fehlerhaftigkeit der Frameworks F2 und F3. Hier könnten die Programmfehler analysiert und Ausbesserung Vorschläge etabliert werden. Für das Framework F1 könnte eine bessere Referenz Update und nachlade Algorithmus programmiert werden.

Literaturverzeichnis

1. B. L. Gorman, „Data Access (Create, Read, Update, Delete)“, in *Practical Entity Framework Core 6*, Berkeley, CA: Apress, 2022, S. 269–297. doi: 10.1007/978-1-4842-7301-2\_6.
2. „ECTS-Punkte“, *ECTS-Punkte*, 1. Juni 2021. https://www.oesterreich.gv.at/themen/bildung\_und\_neue\_medien/universitaet/Seite.160120.html (zugegriffen 24. April 2022).
3. D. Zmaranda, L.-L. Pop-Fele, C. Gyorödi, R. Gyorödi, und G. Pecherle, „Performance Comparison of CRUD Methods using NET Object Relational Mappers: A Case Study“, *IJACSA*, Bd. 11, Nr. 1, 2020, doi: 10.14569/IJACSA.2020.0110107.
4. L. Schweinberger, „Github Repository ORM Test“, *Github Repository*. https://github.com/rothexD/ORMTestBachelorArbeit/tree/master (zugegriffen 26. Mai 2022).
5. S. Cvetković und D. Janković, „A Comparative Study of the Features and Performance of ORM Tools in a .NET Environment“, in *Objects and Databases*, Bd. 6348, A. Dearle und R. V. Zicari, Hrsg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 147–158. doi: 10.1007/978-3-642-16092-9\_14.
6. . „net ORM performance test analysis“, . *net ORM performance test analysis*, 25. Dezember 2019. https://developpaper.com/net-orm-performance-test-analysis/ (zugegriffen 24. April 2022).
7. „Overview | Benchmark .Net“, *Overview*. https://benchmarkdotnet.org/articles/overview.html (zugegriffen 24. April 2022).
8. „Brauchbare OR Mapper“, *Brauchbare OR Mapper*, 8. April 2022. http://greiterweb.de/spw/ORMapper.html (zugegriffen 24. Februar 2022).
9. „Object-Relational Mapping“, *Object-Relational Mapping*. https://de-academic.com/dic.nsf/dewiki/1040411 (zugegriffen 24. April 2022).
10. D. Dhruw, „ORM and ODM — A Brief Introduction“, *ORM and ODM — A Brief Introduction*, 11. November 2022. https://medium.com/spidernitt/orm-and-odm-a-brief-introduction-369046ec57eb (zugegriffen 24. April 2022).
11. Krybot, „ORM overview, advantages and disadvantages, and introduction to common frameworks“, *ORM overview, advantages and disadvantages, and introduction to common frameworks*. https://blog.krybot.com/a?ID=00950-0cb801a7-7ee8-48be-9f75-a2b0af80465d (zugegriffen 24. April 2022).
12. M. Rehman, „Advantages and disadvantages of using an ORM“, *Advantages and disadvantages of using an ORM*, 22. April 2018. https://www.linkedin.com/pulse/advantages-disadvantages-using-orm-maaz-rehman/ (zugegriffen 24. April 2022).
13. S. Rogers, „The pros and cons of Object Relational Mapping (ORM)“, *The pros and cons of Object Relational Mapping (ORM)*, 22. Januar 2019. https://centralblue.co.uk/blog/2019/01/the-pros-and-cons-of-object-relational-mapping-orm (zugegriffen 24. April 2022).
14. „An introduction to NuGet“, *What is Nuget and what does it do?* https://docs.microsoft.com/en-us/nuget/what-is-nuget (zugegriffen 24. April 2022).
15. „Microsoft.EntityFrameworkCore“, *Nuget Gallary*. https://www.nuget.org/packages/Microsoft.EntityFrameworkCore (zugegriffen 24. April 2022).
16. „Entity Framework Core“, *Entity Framework Core*, 25. Mai 2021. https://docs.microsoft.com/en-us/ef/core/ (zugegriffen 24. April 2022).
17. „NHibernate“. https://www.nuget.org/packages/NHibernate/ (zugegriffen 24. April 2022).
18. H. Schwichtenberg, . „NET Core 1.0“, .*NET Core 1.0*, November 2016. https://entwickler.de/dotnet/net-core-10-001
19. „Announcing Entity Framework Core 1.0“, *Announcing Entity Framework Core 1.0*, 06 2016. https://devblogs.microsoft.com/dotnet/entity-framework-core-1-0-0-available/ (zugegriffen 24. April 2022).
20. M. Chand, „Difference Between .NET and .NET Core“, *Difference Between .NET and .NET Core*, 5. Juli 2020. https://www.c-sharpcorner.com/article/difference-between-net-framework-and-net-core/ (zugegriffen 24. April 2022).
21. „Reverse Engineering | EF Core“, *Reverse Engineering | EF Core*, 3. September 2022. https://docs.microsoft.com/en-us/ef/core/managing-schemas/scaffolding?tabs=dotnet-core-cli (zugegriffen 25. April 2022).
22. P. P.-S. Chen, „The entity-relationship model—toward a unified view of data“, *ACM Trans. Database Syst.*, Bd. 1, Nr. 1, S. 9–36, März 1976, doi: 10.1145/320434.320440.
23. Il-Yeol Song und K. Froehlich, „Entity-relationship modeling“, *IEEE Potentials*, Bd. 13, Nr. 5, S. 29–34, 1995, doi: 10.1109/45.464652.
24. roj, gewarren, smitpatel, und bricelam, „Loading Related Data“, *Loading Related Data*. https://docs.microsoft.com/en-us/ef/core/querying/related-data (zugegriffen 24. April 2022).
25. „Was ist Caching und wie Funktioniert es?“, *Was ist Caching und wie Funktioniert es?* https://aws.amazon.com/de/caching/ (zugegriffen 24. April 2022).
26. „UPDATE ON CHRISTMAS ISSUES“, *UPDATE ON CHRISTMAS ISSUES*, 30. Dezember 2015. https://store.steampowered.com/oldnews/19852 (zugegriffen 24. April 2022).
27. L. Richardson, S. Ruby, und L. Richardson, *Web Services mit REST: frischer Wind für Web Services durch REST*, 1. Aufl., [Dt. Ausg.]. Beijing Köln: O’Reilly, 2007.
28. „Was ist eine REST-API und was ist REST (Representational State Transfer)?“, *Was ist eine REST-API und was ist REST (Representational State Transfer)?*, 8. Mai 2020. https://www.redhat.com/de/topics/api/what-is-a-rest-api (zugegriffen 24. April 2022).
29. N. Dragoni *u. a.*, „Microservices: Yesterday, Today, and Tomorrow“, in *Present and Ulterior Software Engineering*, M. Mazzara und B. Meyer, Hrsg. Cham: Springer International Publishing, 2017, S. 195–216. doi: 10.1007/978-3-319-67425-4\_12.
30. G. J. Stigler, „The Economies of Scale“, *The Journal of Law and Economics*, Bd. 1, S. 54–71, Okt. 1958, doi: 10.1086/466541.
31. S. Anupom, „Optimizing data warehouse storage“, *Optimizing data warehouse storage*, 21. Dezember 2020. https://netflixtechblog.com/optimizing-data-warehouse-storage-7b94a48fdcbe (zugegriffen 24. April 2022).
32. V. Cortellessa, A. Di Marco, und P. Inverardi, „What Is Software Performance?“, in *Model-Based Software Performance Analysis*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 1–7. doi: 10.1007/978-3-642-13621-4\_1.
33. P. Mandl, *Grundkurs Betriebssysteme Architekturen, Betriebsmittelverwaltung, Synchronisation, Prozesskommunikation*. 2010. Zugegriffen: 24. April 2022. [Online]. Verfügbar unter: https://doi.org/10.1007/978-3-8348-9384-0

Abbildungsverzeichnis

[Abbildung 1: Eins zu Eins Beziehung [23] 15](#_Toc102908809)

[Abbildung 2: Eins zu Viele Beziehung [23] 15](#_Toc102908810)

[Abbildung 3: Viele zu Viele Beziehung [23] 16](#_Toc102908811)

[Abbildung 4: Screenshot Hardware und Software 24](#_Toc102908812)

[Abbildung 5 ER-Diagramm NoRelation (links) und Eins-zu-Eins (rechts) 26](#_Toc102908813)

[Abbildung 6 ER-Diagramm Eins-zu-Viele (links) und Viele-zu-Viele(rechts) 27](#_Toc102908814)

Tabellenverzeichnis

[Tabelle 1: Download Statistiken ORM Frameworks nach NuGet 13](#_Toc102908815)

[Tabelle 2: Resultate Insert Eins-zu-Eins 32](#_Toc102908816)

[Tabelle 3: Resultate Insert NoRelation 33](#_Toc102908817)

[Tabelle 4: Resultate Insert Eins-zu-Viele 33](#_Toc102908818)

[Tabelle 5: Resultate Insert Viele-zu-Viele 33](#_Toc102908819)

[Tabelle 6: Resultate Select Eins-zu-Eins 33](#_Toc102908820)

[Tabelle 7: Resultate Select NoRelation 33](#_Toc102908821)

[Tabelle 8: Resultate Select 1 zu N 34](#_Toc102908822)

[Tabelle 9: Resultate Select Viele-zu-Viele 34](#_Toc102908823)

[Tabelle 10: Resultate Update Eins-zu-Eins 34](#_Toc102908824)

[Tabelle 11: Resultate Update NoRelation 34](#_Toc102908825)

[Tabelle 12: Resultate Update Eins-zu-Viele 34](#_Toc102908826)

[Tabelle 13: Resultate Update n zu m 35](#_Toc102908827)

[Tabelle 14: Resultate Delete Eins-zu-Eins 35](#_Toc102908828)

[Tabelle 15: Resultate Delete NoRelation 35](#_Toc102908829)

[Tabelle 16: Resultate Delete Eins-zu-Viele 35](#_Toc102908830)

[Tabelle 17: Resultate Delete Viele-zu-Viele 35](#_Toc102908831)

Abkürzungsverzeichnis

|  |  |
| --- | --- |
| API | Application Programming Interface |
| CRUD | Creat Read Update Deleate |
| DBMS | Datenbankmanagementsystem |
| DoS | Denial of Service |
| ER | Entity Relationship |
| Rest | Representational State Transfer |