Reverse Engineering für die nachträgliche Erstellung von Entity-Relationship-Diagramm basierend auf relationalen Modellen

Amani Assani: 220200533

Bachelorarbeit

Fakultät für Informatik und Elektrotechnik
Institut für Informatik
Universität Rostock

Betreuer: Dr. Hannes Grunert
Zweitgutachter: Dr.-Ing. Holger Meyer

Rostock, den 14.03.2025

Kurzfassung

Diese Bachelorarbeit befasst sich mit der Entwicklung und Implementierung eines Verfahrens zur automatisierten Erzeugung von Entity-Relationship-Diagrammen aus bestehenden relationalen Datenbanken. Im Zentrum steht die methodische Extraktion der relevanten Entitäten, Beziehungen, Attribute und Kardinalitäten aus oft unzureichend dokumentierten Datenbanken. Ziel ist es, ein interaktiv visualisierbares ERM zu erstellen, das die Komplexität moderner Datenstrukturen abbildet. Zunächst werden die theoretischen Grundlagen des Entity-Relationship-Modells und des relationalen Datenmodells erläutert, inklusive der Unterschiede und der Prinzipien der Abbildung von relationalen Strukturen auf ERMs. Anschließend wird das Reverse Engineering als Methode vorgestellt, um aus bestehenden Datenbanken die ursprünglichen Designentscheidungen und Strukturen wiederzugewinnen. Im weiteren Verlauf der Arbeit werden verschiedene Ansätze und Tools zur Visualisierung von ER-Diagrammen analysiert und verglichen. Dabei wird herausgestellt, dass benutzerfreundliche, flexible Werkzeuge wie Draw.io besonders geeignet sind, um komplexe Datenmodelle anschaulich darzustellen und gleichzeitig die Anpassbarkeit an spezifische Anforderungen zu gewährleisten. Abschließend zeigt die prototypische Implementierung, wie das entwickelte Verfahren in der Praxis eingesetzt werden kann. Neben der technischen Umsetzung (unter anderem mittels XML-basierter Codierung) wird auch die Evaluierung des Verfahrens hinsichtlich Effizienz und Praxistauglichkeit diskutiert. Die Ergebnisse unterstreichen den Mehrwert einer automatisierten und interaktiven Visualisierung bei der Dokumentation und Optimierung von Datenbankstrukturen.

Abstract

This bachelor thesis deals with the development and implementation of a method for the automated generation of entity-relationship diagrams from existing relational databases. The focus is on the methodical extraction of the relevant entities, relationships, attributes and cardinalities from often inadequately documented databases. The aim is to create an interactively visualizable ERM that maps the complexity of modern data structures. First, the theoretical foundations of the entity-relationship model and the relational data model are explained, including the differences and the principles of mapping relational structures to ERMs. Reverse engineering is then presented as a method for recovering the original design decisions and structures from existing databases. In the further course of the work, various approaches and tools for visualizing ER diagrams are analyzed and compared. It is highlighted that user-friendly, flexible tools such as Draw.io are particularly suitable for clearly displaying complex data models while ensuring adaptability to specific requirements. Finally, the prototype implementation shows how the developed method can be used in practice. In addition to the technical implementation (including using XML-based coding), the evaluation of the method in terms of efficiency and practical suitability is also discussed. The results underline the added value of automated and interactive visualization in the documentation and optimization of database structures.

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit beschäftigt sich mit der Entwicklung eines Verfahrens zur automatisierten Erstellung von Entity-Relationship-Modellen (ERM) aus relationalen Datenbanken und deren interaktiver Visualisierung. Dabei werden die spezifischen Problemstellungen, die dieser Ansatz adressiert, sowie die Zielsetzung beschreiben. Zudem wird der Aufbau der Arbeit skizziert, um einen klaren Überblick über die Struktur und den Ablauf der Untersuchung zu geben.

1.1 Problemstellung

In modernen Organisationen, sei es in der Forschung, der Wirtschaft oder der öffentlichen Verwaltung, haben sich im Laufe der Zeit umfangreiche Datensammlungen entwickelt. Diese Datensammlungen, häufig als „organisch gewachsen“ beschrieben, sind das Ergebnis jahrelanger operativer Tätigkeiten, bei denen Daten oft unsystematisch und ohne klare Strukturierung erfasst wurden. Diese Prozesse führen dazu, dass Datenbanken und Datenspeicher oft unzureichend dokumentiert sind und über eine komplexe und unübersichtliche Struktur verfügen. Ein zentrales Problem bei der Arbeit mit solchen Datensammlungen besteht darin, dass sie sowohl abteilungsintern als auch -übergreifend häufig nur schlecht dokumentiert sind. Dies führt dazu, dass der Informationsgehalt der Daten nur unzureichend verstanden und genutzt werden kann. Die mangelnde Transparenz und Struktur erschwert es, die Daten für statistische Auswertungen oder die Anwendung von Künstlicher Intelligenz (KI) effektiv aufzubereiten. In der Praxis zeigt sich oft, dass bestehende Datenbanken erst nachträglich, also ohne vorherige Modellierung, dokumentiert und strukturiert werden.

In solchen Fällen muss das ERM nachträglich aus den bestehenden Datenbanken erzeugt werden. Dies ist jedoch mit erheblichen Herausforderungen verbunden, da die relevanten Entitäts- und Beziehungstypen sowie Attribute und Kardinalitäten aus den vorhandenen Datenstrukturen extrahiert werden müssen. Der Prozess der nachträglichen Modellierung ist zeitaufwändig, fehleranfällig und erfordert ein tiefes Verständnis der zugrunde liegenden Datenstrukturen. Ein methodisch durchdachter Datenbankentwurf würde in der Regel mit einem Entity-Relationship-Modell beginnen. Das ERM dient dazu, die grundlegende Struktur der Datenbank zu definieren, indem es Entitäten, deren Beziehungen zueinander und deren Attribute beschreibt. Ohne ein solches Modell fehlt es an einer klaren Übersicht über die Datenstruktur, was die Fehleranfälligkeit erhöht und die Effizienz bei der Datenverarbeitung beeinträchtigt.

1.2 Zielsetzung und Forschungsfragen

Ziel dieser Bachelorarbeit ist es, ein Verfahren zu entwickeln und zu implementieren, das es ermöglicht, aus bestehenden Datenbanken nachträglich ein Entity-Relationship-Modell (ERM) zu erstellen und dieses interaktiv zu visualisieren. Dabei soll der Fokus auf der algorithmischen Extraktion der relevanten Entitäts- (Schwache Entitätstypen) und Beziehungstypen, Attribute (optionale Attribute) sowie der Kardinalitäten und Festlegung der Schlüssel liegen.

Die zentrale Fragestellung lautet: Wie kann ein Verfahren zur nachträglichen algorithmischen Extraktion und interaktiven Visualisierung von Entity-Relationship-Modellen aus bestehenden Datenbanken entwickelt und implementiert werden, das die Erkennung von Entitäts- und Beziehungstypen, Attributen sowie Kardinalitäten und Schlüsseln ermöglicht?

Zusätzlich wird eine Prototyp-Implementierung für die Visualisierung des ERMs schaffen, die es ermöglicht, die erzeugte Datenstruktur übersichtlich darzustellen. Schließlich soll der entwickelte Ansatz evaluiert werden, um seine Praxistauglichkeit und Effizienz zu beurteilen.

1.3 Aufbau der Arbeit

Diese Arbeit ist in mehrere Kapitel unterteilt. In Kapitel 2 vermittelt die theoretischen Grundlagen, die für das Verständnis und die Umsetzung der Arbeit wesentlich sind. Es beginnt mit einer Einführung in das Entity-Relationship-Modell, einer der zentralen Techniken im Datenbankdesign. Hier werden grundlegende Konzepte des ERM erläutert sowie seine Rolle im Datenbankentwurf.

Im Kapitel 3 wird der aktuelle Stand der Technik vorgestellt. Es werden verschiedene existierende Methoden und Werkzeuge vorgestellt, die sich mit der nachträglichen Erstellung von ER-Modellen aus relationalen Datenbanken befassen.

Kapitel 4 beschreibt die Konzeption des entwickelten Verfahrens. Hier werden die Anforderungen definiert und der methodische Ansatz vorgestellt. Zudem werden die Schritte erläutert, die für die Abbildung relationaler Modelle auf ERMs erforderlich sind.

Kapitel 5 behandelt die Implementierung des Konzepts. Es gibt einen Überblick über die verwendeten Technologien und beschreibt die konkrete Umsetzung der entwickelten Methode, einschließlich der Codierung und Verarbeitung der ERMs.

Kapitel 6 präsentiert die Ergebnisse und diskutiert deren Bedeutung.

Abschließend bietet Kapitel 7 einen Ausblick auf mögliche Weiterentwicklungen.

2 Theoretische Grundlagen

Kapitel 2 vermittelt die theoretischen Grundlagen, die für das Verständnis und die Umsetzung der Arbeit wesentlich sind. Es beginnt mit einer Einführung in das Entity-Relationship-Modell (ERM), einer der zentralen Techniken im Datenbankdesign. Hier werden grundlegende Konzepte des ERM erläutert sowie seine Rolle im Datenbankentwurf, insbesondere für die Dokumentation, Optimierung und Fehlervermeidung.

In Abschnitt 2.1 wird zunächst eine Definition des ERM gegeben. Danach wird auf seine Bedeutung im Datenbankdesign eingegangen . Abschließend wird die Visualisierung des ERM beschrieben, um das Verständnis der Modellstrukturen zu erleichtern.

Anschließend widmet sich Abschnitt 2.2 dem Reverse Engineering, das sich auf die Rückgewinnung von Datenmodellen aus bestehenden Datenbanken konzentriert. Neben einer Definition werden die Ziele und Herausforderungen dieses Ansatzes beleuchtet, insbesondere die Schwierigkeiten bei der Erkennung von Entitäten, Beziehungen und Attributen aus bestehenden, oft komplexen Datenstrukturen.

In Abschnitt 2.3 werden die Grundlagen relationaler Datenmodelle behandelt, darunter die Konzepte von Tabellen, Schlüsseln und Normalisierung. Zudem wird der Unterschied zwischen ERM und relationalen Modellen sowie das Prinzip der Abbildung relationaler Strukturen auf ERMs dargestellt.

2.1 Entity-Relationship-Modell (ERM)

Das Entity-Relationship-Modell, auch bekannt als ERM, ER-Diagramm oder ERD: ist das älteste und allgemeinste Modell zur strukturierten Datenmodellierung. Es wurde zwischen 1970 und 1976 von Chen entwickelt und seither vielfach weiter verfeinert [SC09].

darunter insbesondere Verfeinerung [Che02]:

- Computer-Aided Software Engineering (CASE): Das ER-Modell wurde Ende der 1980er Jahre als Kernkomponente von CASE-Tools zur Unterstützung der Softwareentwicklung und des Informationsmanagements verwendet.
- Repository Systeme und ANSI IRDS Standard (1987): Das ER-Modell hat sich zum Standarddatenmodell für Repository-Systeme entwickelt (z. B. IBM Repository Manager und CDD+ von DEC).
- Daten-Mining: Das ER-Modell dient als theoretische Grundlage zum Aufdecken verborgener Beziehungen zwischen Datenentitäten und ist ein wichtiger Ausgangspunkt für moderne Data-Mining-Methoden.
- Resource Description Framework (RDF): RDF wird als Teil der ER-Modellfamilie angesehen und zeigt, dass ER-Modellkonzepte einen direkten Einfluss auf moderne Webtechnologien haben.
- XLink: Das Konzept eines erweiterten Links (XLink) entspricht dem Konzept der n-ären Beziehungen im ER-Modell. Es handelt sich hierbei um eine logische Verbindung zwischen mehreren Entitäten.

In Abbildung 1 sehen wir den Zusammenhang zwischen dem relationalen Modell, dem ERM und der Implementierung. Hierbei zeigt die Abbildung, wie textuelle Anforderun-

gen zunächst in ein ERM überführt werden, bevor sie in ein relationales Modell (RM) und schließlich in die physische Implementierung.

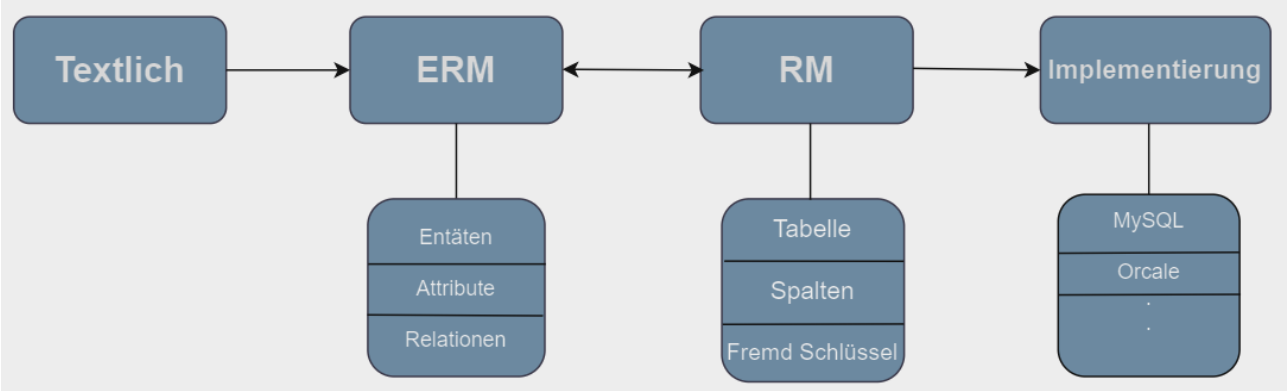


Abbildung 1: Prozess der Datenmodellierung

2.1.1 Definition des ERD

Das ER-Diagramm: ist ein konzeptioneller Ansatz, der die Datenstruktur eines bestimmten Problembereichs in Form von Entitäten und deren Beziehungen abbildet. Das Resultat dieser Modellierung wird visuell als Diagramm dargestellt. Ein ERD veranschaulicht somit die konzeptionelle Struktur des zu modellierenden Bereichs. ERDs finden häufig Anwendung beim Entwurf von Datenbanken und der Systemanalyse, um die Anforderungen eines Systems oder eines Problembereichs zu erfassen. Besonders in der Datenmodellierung hilft das ERD dem Datenbankentwickler, sowohl die Daten als auch die Regeln zu identifizieren, die in der Datenbank erfasst und verwendet werden sollen. Zudem lassen sich ERDs einfach in relationale Datenbankschemata überführen [SC09].

Sie verwenden einen definierten Satz von Symbolen wie Rechtecke, Rauten, Ovale und Verbindungslinien, um die Vernetzung von Entitätstypen, Beziehungstypen und ihren Attributen darzustellen. Sie spiegeln die grammatische Struktur wider, wobei Entitätstypen Substantive und Beziehungstypen Normalerweise Verben sind [Bad04]. Das Ziel der ERD besteht darin, ein einfaches, aber formal wirksames Werkzeug für das Datenbankdesign bereitzustellen, das die grundlegende Semantik vieler Situationen erfasst [Bad04].

Hier sind die grundlegenden Konzepte des ERM:

- **Entitätstyp**: Eine Entitätstyp ist ein Objekt aus der realen oder fiktiven Welt, über das Informationen gespeichert werden sollen, wie zum Beispiel ein Wein, ein Weingut, ein Produkt, ein Kunde oder auch eine Person wie ein Student oder Professor. Auch Vorgänge wie Bestellungen oder Prüfungen können als Objekte im Sinne des ERD betrachtet werden. außerdem, In einer Datenbank sind Entitäten jedoch nicht direkt abbildbar, sondern nur durch ihre Merkmale erkennbar. Das bedeutet, dass ein Wein nicht als solcher in der Datenbank gespeichert wird, sondern lediglich Informationen über seine Eigenschaften, wie Name, Farbe, Jahrgang usw. [SSH18].
- **Relationship/Beziehungstyp**: Eine Beziehung im ERD beschreibt, wie Entitäten aufeinander einwirken oder miteinander verbunden sind [SSH18]. Sie stellt eine Ver-

knüpfung zwischen zwei oder mehreren Entitätstypen dar, die wechselseitig (in beide Richtungen) aufeinander bezogen sind. Eine Beziehung ist daher eine Verbindung zwischen mindestens zwei Entitäten und hängt existenziell von diesen ab, das heißt, sie kann ohne die beteiligten Entitäten nicht eigenständig existieren. Beziehungen sind ein mindestens zweifach existenzabhängiges Phänomen der realen Welt und können nur indirekt, also über die Entitäten, die sie verbinden, identifiziert werden [MD97].

In konkreten Beispielen des ERD könnte eine Beziehung beschreiben, wie ein Weingut einen Wein produziert, ein Kunde ein Produkt bestellt oder ein Dozent eine Vorlesung hält. Diese Beispiele verdeutlichen, dass die Beziehung die Interaktion und Abhängigkeit der Entitäten in einem spezifischen Kontext festhält und eine strukturierte Modellierung von Prozessen und Interaktionen ermöglicht. Durch ihre mehrfache existenzielle Abhängigkeit und ihre indirekte Identifizierbarkeit unterscheidet sich eine Beziehung von einer Entität, die direkt identifizierbar ist [SSH18].

- **Attribut**: ist eine Eigenschaft oder ein Merkmal einer Entität im Entity-Relationship-Modell (ERM), das spezifische Informationen über die Entität beschreibt. Jedes Attribut hat einen Namen, der die Art der Eigenschaft angibt (z. B. „Name“ oder „Alter“), und einen Datentyp, der bestimmt, welche Art von Daten gespeichert werden kann (z. B. varchar für Zeichenfolgen oder int für Ganzzahlen) [SSH18].
  - **Schlüsselattribut**: Ein Schlüsselattribut ist eine kleinste Menge von Attributen, die ausreicht, um eine Entität eindeutig zu identifizieren. Diese Menge kann aus einem einzelnen Attribut oder mehreren Attributen bestehen. Ein Schlüssel, der nur ein Attribut umfasst, wird als „einfacher“ Schlüssel bezeichnet, während ein Schlüssel aus mehreren Attributen als „zusammengesetzter“ Schlüssel bekannt ist. Ein minimaler Schlüssel lässt sich häufig durch eine fortlaufende Nummerierung der einzelnen Entitäten erzeugen. Da solche fortlaufenden Nummern keine beschreibende Funktion haben, werden sie auch als „künstliche“ Schlüssel bezeichnet. Häufige Beispiele hierfür sind „Kunden-Nr.“, „Artikel-Nr.“ oder „Personal-Nr.“. Um Schlüsselattribute zu kennzeichnen, werden sie in der Regel unterstrichen [Gad19].

In Abb. 2 wird ein einfaches Beispiel gezeigt

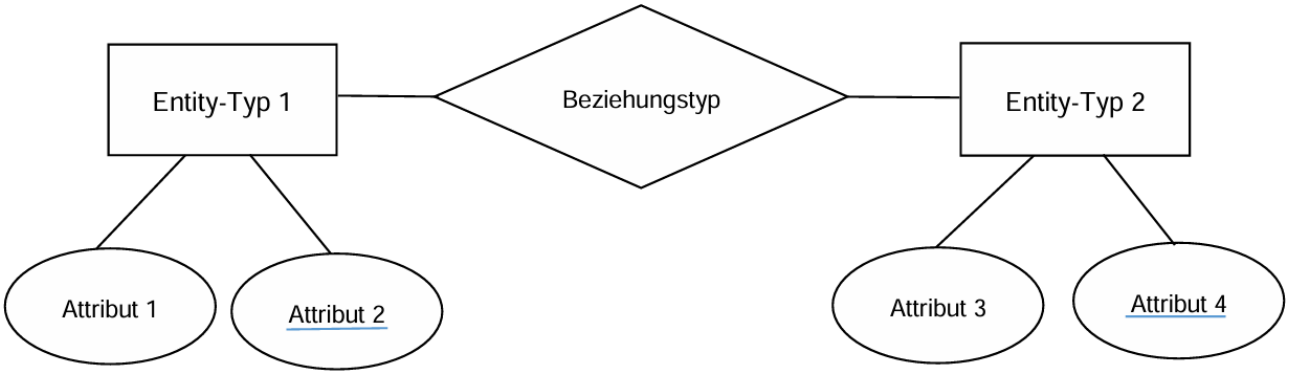


Abbildung 2: viel einfaches Modellierungsbeispiel mit der Chen-Notation

- **Schwache Entitätstypen**: Im ERD nach Chen werden schwache Entitätstypen als solche Entitätstypen definiert, die nicht eigenständig existieren können und von einem starken Entitätstyp abhängig sind. Sie sind nicht direkt an einem Geschäftsprozess beteiligt und erfordern zur Identifikation eine Verbindung zu einem starken Entitätstyp, da sie keinen eigenen, eindeutigen Schlüssel besitzen. Ein Beispiel dafür ist ein Hotelzimmer. Oftmals verfügt ein einzelnes Zimmer nicht über eine eindeutige Identifikationsnummer, da es in verschiedenen Hotels immer gleiche Zimmernummern geben kann. Ein Zimmer mit der Nummer 101 lässt sich zwar in vielen Hotels finden, eindeutig wird es allerdings erst durch die Kombination mit der Hotel-ID. Daher muss die Entität „Zimmer“ mit einem starken Entitätstyp, z.B. „Hotel“, verknüpft werden, um eindeutig identifiziert werden zu können. Schwache Entitätstypen werden im ERD durch doppelte Linien dargestellt, um ihre Abhängigkeit und fehlende Eigenständigkeit visuell hervorzuheben [Her18].

- **Optionale Attribute**: sind Attribute, die nicht zwingend für alle Entitäten einer Entitätengruppe einen definierten Wert haben müssen. Sie stehen im Gegensatz zu totalen (nicht-optionalen) Attributen, die immer einen Wert für jede Entität aufweisen müssen. Die Optionalität eines Attributs wird in manchen grafischen Notationen, wie in der von Hohenstein, durch einen kleinen Kreis an der Verbindungslinie zwischen Attributsymbol und Entitätentyp markiert. Optionalität bedeutet semantisch, dass die Funktion des Attributs eine partielle Funktion ist und daher nicht für jede Entität einen definierten Wert haben muss [SSH18].

  Stellen wir uns eine Entität 'Kunden' in einer Datenbank vor. Diese Entität hat die folgenden Attribute:
  - **Name** – Ein totales (erforderliches) Attribut, da jeder Kunde einen Namen haben muss.
  - **Adresse** – Ebenfalls ein totales Attribut, da jeder Kunde eine Adresse angeben muss.
  - **Telefonnummer** – Ein optionales Attribut, da nicht jeder Kunde eine Telefonnummer hinterlegen muss.

  Hier bedeutet die Optionalität des Attributs *Telefonnummer*, dass Kunden, die keine Telefonnummer angeben, trotzdem in der Datenbank existieren können.

- **Ternärer Beziehungstyp / N-ärer Beziehungstyp**: Ein ternärer Beziehungstyp ist eine Art von Beziehung im ERD, die drei verschiedene Entitäten miteinander verbindet. Während n-ärer Beziehungstyp beschreibt allgemein die Verknüpfung von n verschiedenen Entitäten, wobei $n \geq 2$ ist. Der ternärer Beziehungstyp beschreibt ein eine Beziehung, die gleichzeitig drei Entitäten einschließt. Beide Beziehungstypen werden verwendet, wenn die Verknüpfung zwischen den Entitäten nicht vollständig durch separate binäre Beziehungen abgebildet werden kann. [Gad19]

- **Kardinalität**: Beschreibt die Anzahl der Verknüpfungen (siehe Abb 3), die zwischen zwei Entitäten oder Gruppen von Entitäten bestehen können. Die drei Hauptarten der Kardinalität sind: Eins-zu-eins, Eins-zu-viele und Viele-zu-viele [Gad19].

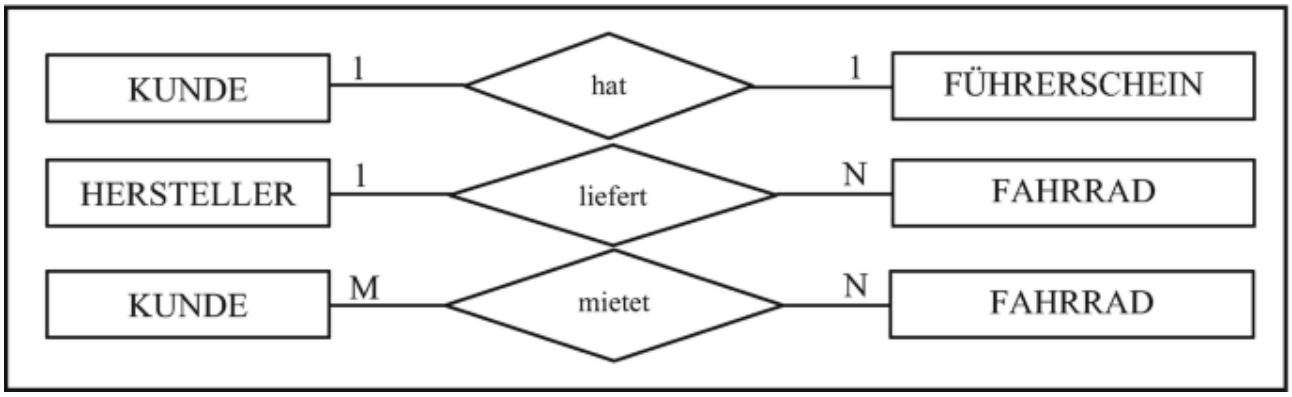


Abbildung 3: Beziehungen zwischen Entitätstypen
[Her18]

1. **1:1 Beziehungstyp**

   Eine Eins-zu-eins-Beziehung wird häufig genutzt, um eine Entität in zwei separate Bereiche zu unterteilen, damit Informationen klarer und leichter verständlich dargestellt werden können. Das bedeutet, dass ein Student nur einen Studentenausweis haben kann, und umgekehrt kann ein Studentenausweis nur einem einzigen Studenten zugeordnet sein. Es ist jedoch nicht immer erforderlich, dass jede Entität in der Menge der Studenten zwingend mit einer Entität in der Menge der Studentenausweise verknüpft ist. Ein solcher Fall könnte zum Beispiel auftreten, wenn der Immatrikulationsantrag noch nicht vollständig bearbeitet wurde [Gad19].

2. **1:N-Beziehungstyp**

   1:N-Beziehungstyp: Dieser Beziehungstyp beschreibt das Verhältnis zwischen den Entitätstypen “Student” und “Hochschule”. Hierbei gilt, dass ein Student nur an einer Hochschule eingeschrieben sein kann, während eine Hochschule gleichzeitig viele Studierende betreuen kann [Gad19].

   - **1-Seite:** Auf der Seite der Entität mit der 1-Beteiligung (z. B. „Hochschule“) kann eine Entität mit mehreren Entitäten des anderen Typs in Beziehung stehen.
   - **N-Seite:** Auf der Seite der Entität mit der N-Beteiligung (z. B. „Student“) kann jede Entität nur mit einer Entität des anderen Typs in Beziehung stehen.

   Ein Beispiel dafür wäre, dass ein Student nur an einer Hochschule eingeschrieben sein darf, aber eine Hochschule viele Studierende haben kann.

3. **N:M Beziehungstyp**

   N:M-Beziehung wird beschrieben, indem zwei Entitätstypen dargestellt werden, die in einer Beziehung stehen, bei der jede Entität der einen Seite mit mehreren Entitäten der anderen Seite verknüpft sein kann. Ein Beispiel hierfür ist die Beziehung zwischen Datenbanksystemen und Händlern: Ein Datenbanksystem kann von mehreren Händlern angeboten werden, und ein Händler kann mehrere Datenbanksysteme anbieten [Sta05].
   Ein weiteres Beispiel wäre ein Studierender, der mehrere Vorlesungen besucht, während jede Vorlesung von mehreren Studierenden belegt wird.

4. **Min-Max-Notation**:

   Im Wesentlichen wird zusätzlich zu den geänderten Symbolen für die Kardinalitäten (0,1) für „1“ oder (0,*) für „N“) eine alternative Leserichtung angewandt (wie Abb. 4)[Gad19].

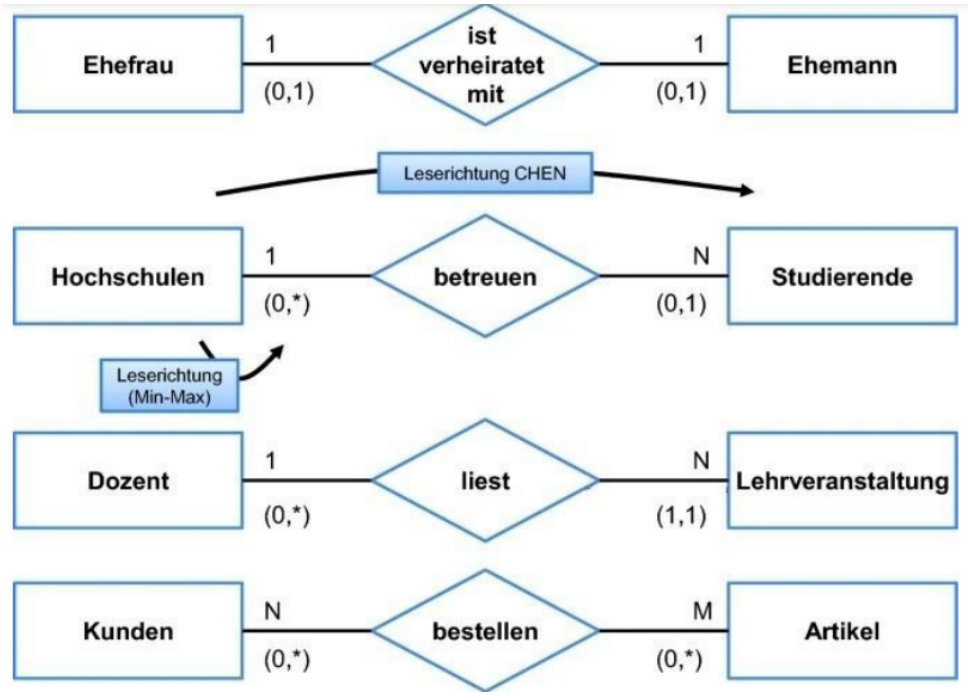


Abbildung 4: [min,max] Notation
[Gad19]; wurde erweitert.

- **Ehefrau - Ehemann**: Die Notation (0,1) bedeutet, dass die Beziehung optional ist, da eine Person entweder verheiratet oder unverheiratet sein kann.

- **Hochschulen - Studierende**: Die Kardinalität (0,*) für Hochschulen zeigt, dass eine Hochschule keine oder mehrere Studierende betreuen kann, während (0,1) bei Studierenden zeigt, dass ein Studierender von keiner oder genau einer Hochschule betreut werden kann.

- **Dozent - Lehrveranstaltung**: Die Notation (0,*) für Dozenten zeigt an, dass ein Dozent keine oder mehrere Lehrveranstaltungen lesen kann, und (1,1) für Lehrveranstaltungen bedeutet, dass jede Lehrveranstaltung genau von einem Dozenten gelesen wird.

- **Kunden - Artikel**: Die Kardinalität (0,*) zeigt, dass es keine Einschränkungen bezüglich der Anzahl der Bestellungen für Kunden und Artikel gibt.

2.1.2 Rolle des ERD im Datenbankdesign

Das Entity-Relationship-Diagramm spielt eine zentrale Rolle im Datenbankdesign, insbesondere in einer Zeit, in der die Anforderungen agiler Unternehmen sich rasch ändern. Um diesen dynamischen Herausforderungen gerecht zu werden, ist es für Datenbankdesigner unerlässlich, bestehende Datenmodelle zügig zu verstehen und anzupassen. In dieser Hinsicht bieten Entity-Relationship-Diagramme eine wertvolle Unterstützung, da sie Entwicklern ermöglichen, einen umfassenden Überblick über die Datenanforderungen und die Struktur des Informationssystems zu erhalten, noch bevor die Implementierungsphase beginnt [Cag+13].

Mit der zunehmenden Integration verschiedener Anwendungen und der Verwaltung unterschiedlichster Datentypen hat sich der Bedarf an einem effektiven Modellierungsansatz, wie dem ERD, verstärkt. Es ermöglicht, die vielschichtigen Beziehungen zwischen realen Objekten und deren Eigenschaften klar und verständlich darzustellen. Durch die graphische Abbildung von Entitäten, Beziehungen und Attributen hilft das ERD, die Komplexität dieser Strukturen zu bewältigen und ermöglicht es den Datenbankdesignern, die relevanten Elemente und deren Wechselwirkungen zu erfassen [Eng+92].

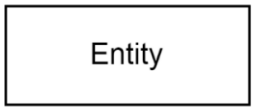
Ein weiterer wichtiger Aspekt der ERDs ist ihre Rolle bei der Fehlervermeidung. Sie bieten Entwicklern die Möglichkeit, potenzielle Fehler oder Unstimmigkeiten in der Datenbankstruktur frühzeitig zu erkennen. Durch die sorgfältige Überprüfung und Analyse von ERDs sind Softwareingenieure in der Lage, Designfehler aufzudecken, die entscheidend für die Konsistenz des Systems sind. Diese frühzeitige Fehlererkennung kann die Entwicklungskosten erheblich senken, indem sie spätere Probleme und den damit verbundenen Aufwande vermeidet. Insgesamt trägt das ERM somit wesentlich dazu bei, die Effizienz und Qualität im Datenbankdesign zu steigern [Cag+13].

2.1.3 ERM-Visualisierung

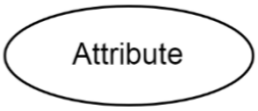
Die Visualisierung von Datenmodellen ist ein entscheidender Schritt im Datenbankdesign, da sie es ermöglicht, die Struktur und Beziehungen innerhalb der Daten zu verstehen. Die Verwendung von Diagrammen hilft, komplexe Informationen klar und prägnant darzustellen, was die Kommunikation zwischen Entwicklern, Stakeholdern und anderen Beteiligten erleichtert. Ein besonders effektives Werkzeug zur Visualisierung von Datenmodellen ist das Entity-Relationship-Modell. Dieses Modell ist eine weit verbreitete Methode zur Datenmodellierung, die die logische Struktur einer Datenbank beschreibt. Es identifiziert Entitäten, Attribute und Beziehungen zwischen diesen Entitäten. Um die Inhalte eines ERMs klar darzustellen, erfolgt die Visualisierung typischerweise durch ER-Diagramme. Diese Diagramme verwenden standardisierte Symbole, um die verschiedenen Komponenten darzustellen [HRT16]

Elemente der ERM-Visualisierung [HRT16, Man08]:

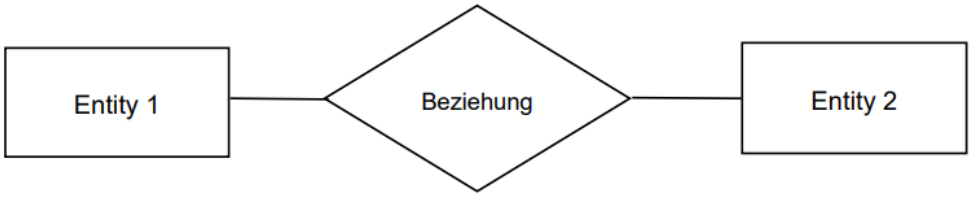
- Entitäten: Objekte oder Dinge in der realen Welt, die für die Datenbank relevant sind (z.B. Kunde, Produkt). Oft wurde als Rechtecke dargestellt.



- Attribute: Eigenschaften oder Merkmale von Entitäten (z.B. Name, Preis). Normalerweise als Ovale dargestellt, die mit ihren entsprechenden Entitäten verbunden sind.

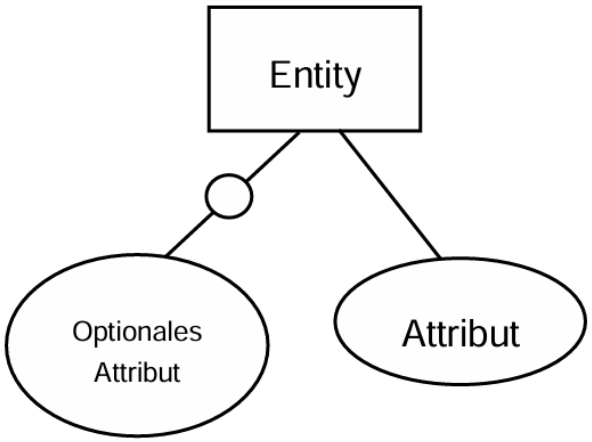


- Beziehungen: zeigen, wie verschiedene Dinge in einem System miteinander stehen. Eine Beziehung zwischen einem Kunden und einem Produkt besteht beispielsweise, wenn ein Kunde etwas kauft. In einem ERD werden diese Verbindungen als Linien zwischen Entitäten dargestellt, um die Beziehungen zu veranschaulichen..

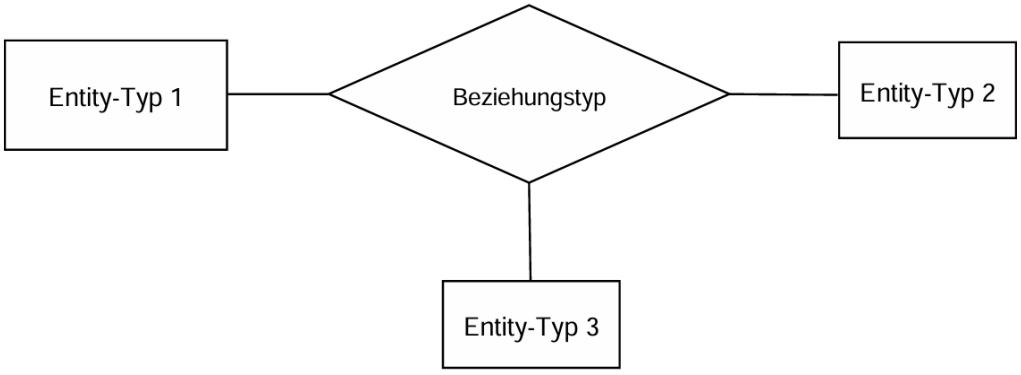


- Schwache Entitätstypen: sie werden durch eine doppelte rechteckige Box dargestellt. Ein schwacher Entitätstyp ist besonders dann sinnvoll, wenn er nur durch die Beziehung zu einer anderen Entität existiert

- Optionale Attribute: sind Attribute, die in bestimmten Fällen leer bleiben können. In ER-Diagrammen wird ein optionales Attribut häufig durch einen gestrichelten Ovalrahmen dargestellt oder durch einen Kreis auf der Attributzeile gekennzeichnet.

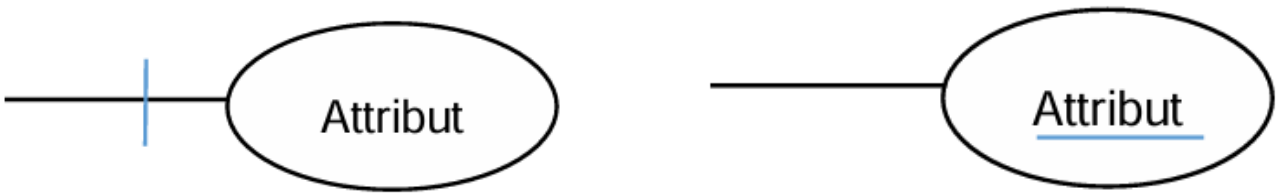


- Ternäre bzw. n-äre Beziehung: wenn drei oder mehr Entitäten gleichzeitig miteinander in einer Beziehung stehen, wobei jede der beteiligten Entitäten eine spezifische Rolle in dieser Beziehung einnimmt, die durch die Beziehung definiert wird.

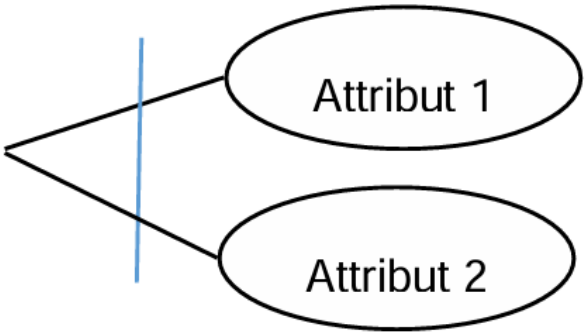


- Schlüsselattribut:

  In ER-Diagrammen wird das Schlüsselattribut durch Unterstreichen dargestellt. Darüber hinaus gibt es mehrere Möglichkeiten, Primärschlüssel zu definieren, beispielsweise die Verwendung einfacher Schlüssel oder zusammengesetzter Schlüssel, je nach Anwendungsfall.



  Zusammengesetzter Schlüssel: Kein einzelnes Attribut des Schlüssels reicht alleine aus, um eine Entität zu unterscheiden, aber die Kombination der Attribute ist eindeutig.



2.2 Reverse Engineering

Reverse Engineering (siehe Abb. 5) von Datenbanken ist seit mehreren Jahrzehnten als spezielles Problem bekannt, wird jedoch erst seit den 1980er Jahren formell untersucht. Die ersten Ansätze basierten auf einfachen Regeln, die gut mit sauber und diszipliniert entwickelten Datenbanken funktionieren. In den 1990er Jahren wurden zahlreiche Beiträge veröffentlicht, die sich praktisch mit allen veralteten Technologien befassten und Informationsquellen wie Anwendungsquellcode, Datenbankinhalte oder Anwendungsbenutzeroberflächen nutzten [Hai+09].

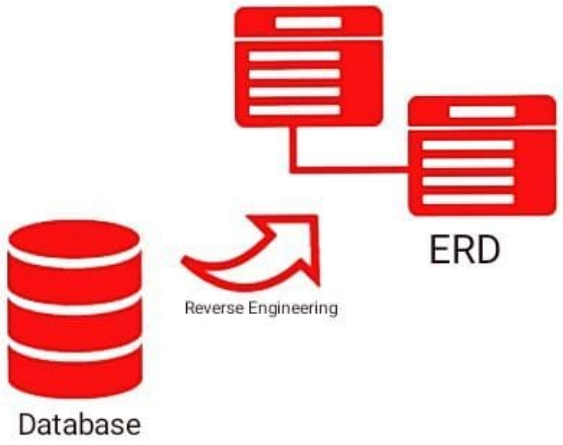


Abbildung 5: Reverse Engineering eines ERD aus einer Datenbank
[Par24]; wurde erweitert

2.2.1 Definition und Ziele

Reverse Engineering von Datenbanken (Database Reverse Engineering, DRE) ist ein bedeutender Prozess, insbesondere in Organisationen, die eine Vielzahl von Datenbanken pflegen und weiterentwickeln müssen. Da viele der ursprünglichen Designentscheidungen im Laufe der Zeit verloren gehen, ermöglicht DRE, die Struktur und Funktionsweise einer bestehenden Datenbank nachträglich zu analysieren und zu dokumentieren. Dadurch wird ein besseres Verständnis der Datenbank geschaffen, was zukünftige Entwicklungen oder Anpassungen effizienter gestaltet [Pet+94].

Das Hauptziel von DRE ist es, existierende operative Systeme wie Datenbankschemata oder Anwendungscode zu nutzen, um ein präziseres Verständnis der Datenbank zu schaffen. Dies verbessert die Qualität und Effizienz von Wartungs- und Weiterentwicklungsprozessen erheblich. DRE ermöglicht es insbesondere, schlecht dokumentierte Systeme zu verstehen und systematisch neu zu modellieren, was für die Planung und Umsetzung zukünftiger Entwicklungen von entscheidender Bedeutung ist [Pet+94].

2.2.2 Herausforderungen und typische Probleme

Beim Reverse Engineering von Datenbanken treten zahlreiche Probleme und Herausforderungen auf, die durch die Komplexität und die unterschiedlichen Arten von Datenbankstrukturen verursacht werden. Hier sind die wichtigsten Probleme und Herausforderungen zusammengefasst [Eng02]:

1. Implizite Strukturen: Versteckte oder nicht übersetzbare Datenstrukturen, die nicht direkt dokumentiert sind oder von frühen DBMS nicht unterstützt werden.

2. Optimierte Strukturen: Datenbankoptimierungen, wie Redundanz oder Denormalisierung, die für Performance-Zwecke verwendet werden und nicht dem konzeptionellen Design entsprechen.

3. Ungeschicktes Design: Schlechte oder inkorrekte Datenbankstrukturen, die von unerfahrenen Entwicklern erstellt wurden.

4. Veraltete Konstrukte: Teile der Datenbank, die nicht mehr genutzt werden, aber noch vorhanden sind.

5. Modellübergreifender Einfluss: Datenbanken, die aus älteren Systemen wie IMS-Datenbanken oder COBOL-Dateien übersetzt wurden und dadurch nicht standardisierte Strukturen aufweisen.

In dieser Arbeit konzentriere ich mich auf die Herausforderungen, die sich aus optimierten Strukturen (Punkt 2) und ungeschicktem Design (Punkt 3) ergeben. Der Fokus liegt hierbei auf der Analyse und Rückführung solcher Strukturen auf das relationale Modell (RM), während Themen wie die eigentliche Datenbankimplementierung oder die Migration älterer Systeme bewusst ausgeklammert werden.

2.2.3 Anwendungsgebiete im Datenbankbereich

Reverse Engineering im Datenbankbereich bietet zahlreiche Anwendungsgebiete, die durch die Erstellung eines Modells basierend auf einer bestehenden Datenbank unterstützt werden. Ein detailliertes Datenmodell kann die Struktur einer bestehenden Datenbank visuell darstellen und dokumentieren. Dies ist besonders hilfreich, wenn keine oder nur unzureichende Dokumentation vorhanden ist, oder bestehende Dokumentationen aktualisiert werden müssen [Sup24].

Database Reverse Engineering wird genutzt, um die semantische Struktur von Datenbanken zu verstehen und zu dokumentieren. Es unterstützt zentrale Aufgaben wie Systemwartung, Migration und Integration, indem es ein klares Bild der Datenbankstruktur liefert. DBRE erleichtert die Übertragung von Daten zwischen Systemen und die Integration neuer Datenquellen, konvertiert Altdaten und ermöglicht die Wiederverwendung von Komponenten. Dadurch wird die Weiterentwicklung und Qualitätssicherung bestehender Systeme wesentlich verbessert [Hai02].

Die Visualisierung der Datenbankstruktur hilft dabei, komplexe Datenbankbeziehungen und Datenflüsse zu analysieren und kann auch bei der Identifikation von Problemen. Ent-

wickler und Administratoren können so Abhängigkeiten und Strukturen schneller erkennen. Ein solches Modell beschleunigt die Fehleranalyse und ermöglicht gezielte Anpassungen. Schließlich dient es auch als Schulungswerkzeug, indem es neuen Mitarbeitern ein besseres Verständnis der Datenbankstruktur vermittelt und den Einarbeitungsprozess erleichtert.

2.3 Relationale Modelle

Das von Codd im Jahr 1970 vorgestellte relationale Modell ist inzwischen das am weitesten verbreitete Datenbankmodell (Abb 6). Durch seine Einfachheit und Präzision wird es seit Langem in der Forschung anerkannt und hat zahlreiche weitere Arbeiten im Bereich der Datenbanken beeinflusst. Gleichzeitig ist es in der Praxis fest etabliert, da die kommerziell erfolgreichsten Datenbanksysteme auf diesem Modell basieren [SSH18].

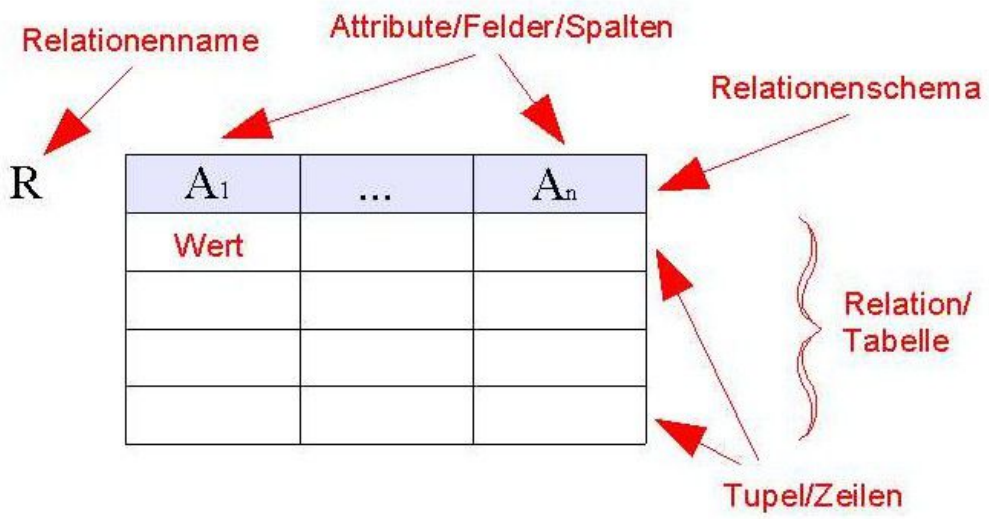


Abbildung 6: Relationale Datenbanken
[Man08]

2.3.1 Grundlagen relationaler Datenbankenmodell

Relationale Datenbanken organisieren Daten in Tabellen (Relationen), eine weit verbreitete Methode zur strukturierten Datenhaltung, insbesondere für kommerzielle Anwendungen. Jede Tabelle stellt eine Relation dar, die aus Zeilen (Tupeln) und Spalten (Attributen) besteht. Jede Zeile speichert einen Datensatz, und jede Spalte repräsentiert eine Eigenschaft (Attribut) dieses Datensatzes [SSH18].

Die wichtigsten Bestandteile des relationalen Modells sind [SSH18]:

- Relation: Eine Relation ist eine Tabelle, die Daten in Zeilen (Tupel) und Spalten (Attribute) speichert. Jede Zeile repräsentiert einen Datensatz, und jede Spalte steht für eine bestimmte Datenkategorie.
- Attribute (Spalten): Jede Relation besteht aus Attributen, die bestimmte Datentypen enthalten (z. B. INT, VARCHAR).
- Tupel (Zeilen): Jede Zeile in einer Relation stellt einen Datensatz dar. Jedes Tupel besteht aus Attributwerten. Attributwert: Ein spezifischer Wert in einer Zelle eines Tupels.
- Schlüssel: Ein Attribut oder eine Kombination von Attributen, die ein Tupel eindeutig identifizieren.
  Primärschlüssel (engl. Primary Key): Ein oder mehrere Attribute, die eine Zeile in einer Tabelle eindeutig identifizieren.
  Fremdschlüssel (engl. Foreign Key): Ein Attribut, das auf den Primärschlüssel einer anderen Tabelle verweist und damit Beziehungen zwischen Tabellen herstellt.
- Normalisierung: Der Prozess der Organisation von Daten, um Redundanz zu vermeiden und Datenabhängigkeiten zu minimieren. Dies geschieht in verschiedenen Normalformen: (1NF, 2NF, 3NF, usw.).
  - 1NF (Erste Normalform): Eine Tabelle erfüllt die Erste Normalform, wenn jeder Wert in den Attributen (Spalten) unteilbar ist. Das bedeutet, dass jede Zelle in der Tabelle nur einen einzelnen Wert enthält und keine Listen oder Gruppen von Werten [And98].
  - 2NF (Zweite Normalform): Eine Tabelle ist in der Zweiten Normalform, wenn sie bereits die Erste Normalform erfüllt und alle Attribute, die nicht Teil eines Schlüssels sind, von dem gesamten Schlüsselkandidaten vollständig abhängen. Das heißt, alle Nicht-Schlüsselattribute müssen direkt vom gesamten Schlüssel abhängen und nicht nur von einem Teil des Schlüssels [And98].
  - 3NF (Dritte Normalform): Eine Tabelle befindet sich in der Dritten Normalform, wenn sie die Zweite Normalform erfüllt und keine Nicht-Schlüsselattribute von anderen Nicht-Schlüsselattributen abhängen. Das bedeutet, dass alle Nicht-Schlüsselattribute nur vom Schlüsselkandidaten und nicht von anderen Nicht-Schlüsselattributen abhängen [And98].

2.3.2 Unterschiede zwischen ERM und relationalen Modellen

Hier ist eine Tabelle, die die Unterschiede zwischen dem Entity-Relationship-Modell und dem relationalen Modell darstellt, basierend auf den Informationen aus verschiedene Quellen [Che76, BHB88]:

Aspekt	**Entity-Relationship-Modell**	**Relationalen Modell**
Kopfzeile	Entity-Typ	Relationstyp
Spaltenüberschrift	Attribut	Attribut
Datenstruktur	Grafische Darstellung von Entitäten und Beziehungen	Tabellarische Struktur (Relationen)
Beziehungen	Beziehungen sind explizit und zeigen ihre Kardinalität an	Beziehungen werden implizit durch Primär- und Fremdschlüssel dargestellt
Semantische Klarheit	Hohe semantische Klarheit, da reale Beziehungen abgebildet werden	Semantische Ambiguitäten möglich, da Beziehungen durch Schlüssel definiert sind
Flexibilität	Flexibel bei der Modellierung komplexer Beziehungen	Weniger flexibel, kann bei Änderungen an Beziehungen komplex werden
Einfache Nutzung	Direkte Darstellung von Beziehungen erleichtert das Verständnis	Komplexere Regeln zur Darstellung von Beziehungen können verwirrend sein
Datenunabhängigkeit	Hohe Datenunabhängigkeit durch klare Trennung von Daten und Logik	Hohe Datenunabhängigkeit, kann jedoch in der Implementierung Einschränkungen aufweisen
Modellierungsebene	Konzeptionelles Modell, das die reale Welt abbildet	Logisches Modell, das sich auf die physische Datenorganisation fokussiert

Tabelle 1: Unterschiede zwischen dem Entity-Relationship-Modell und dem relationalen Modell.

2.3.3 Prinzipien der Abbildung relationaler Modelle auf ERMs

Nach einem Überblick über mehrere Quellen werde ich den Prinzipien der Abbildung relationaler Modelle auf ERMs in einem Beispiel zusammenfassen, damit wir ihn leicht verstehen können [EN16, Moh15].

Um ein relationales Modell in ein Entity-Relationship-Modell zu transformieren, werden die Tabellen (Relationen) und deren Strukturen in Entitäten und Beziehungen überführt. Hier ist ein einfaches Beispiel:

Relationales Modell (RM): Angenommen, wir haben zwei Tabellen: Kunde und Bestellung.

- Tabelle Kunde:
  Kunde({KundenID, Name, Adresse}, {{KundenID}})

KundenID	Name	Adresse
1	Anna	Hauptstr. 1
2	Ben	Nebenstr. 10

Tabelle 2: Kundendaten

- Tabelle Bestellung:
  Bestellung(KundenID)-> Kunde (KundenID)

BestellID	Datum	KundenID
101	2023-09-01	1
102	2023-09-02	1
103	2023-09-09	2

Tabelle 3: Bestelldaten

Schrittweise Umwandlung in das Entity-Relationship-Modell :

1. Relationen zu Entitäten umwandeln: Jede Tabelle im relationalen Modell wird im ERM zu einer Entität:

- Die Tabelle „Kunde“ wird zur Entität „Kunde“ mit den Attributen KundenID, Name und Adresse.
- Die Tabelle „Bestellung“ wird zur Entität „Bestellung“ mit den Attributen BestellID, Datum und KundenID (dieses Attribut wird im nächsten Schritt zur Beziehung).

2. Fremdschlüssel zu Beziehungen umwandeln:

- Der Fremdschlüssel KundenID in der Tabelle „Bestellung“ verweist auf die Tabelle „Kunde“. Dies zeigt, dass jede Bestellung einem Kunden zugeordnet ist. Diese Beziehung wird im ERM als 1:N-Beziehung dargestellt. Das bedeutet, dass ein Kunde mehrere Bestellungen aufgeben kann aber jede Bestellung jedoch nur zu einem Kunden gehört.

3. Primärschlüssel festlegen:

- Die Primärschlüssel (KundenID und BestellID) aus den Relationen bleiben im ERM bestehen und dienen dazu, die Entitäten eindeutig zu identifizieren.

Resultierendes Entity-Relationship-Diagramm

- Entität „Kunde“ mit den Attributen: KundenID (Primärschlüssel), Name, Adresse.
- Entität „Bestellung“ mit den Attributen: BestellID (Primärschlüssel), Datum.
- Beziehung: Zwischen Kunde und Bestellung besteht eine Beziehung „aufgeben“, da Kunden Bestellungen aufgeben.

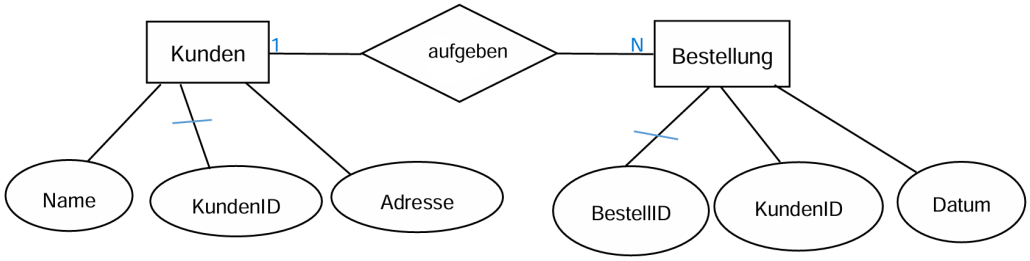


Abbildung 7: ER-Diagramm für Kunden und Bestellungen mit einer 1:N-Beziehung

Erklärung

- Entitäten: repräsentieren reale Objekte, z. B. Kunden und deren Bestellungen.
- Die Beziehung zwischen der Tabelle Kunde und der Tabelle Bestellung ist eine 1:N-Beziehung. Das bedeutet: Ein Kunde kann mehrere Bestellungen aufgeben, während jede Bestellung genau einem Kunden zugeordnet ist.
  Zum Beispiel:
  Kunde 1 (Anna) hat die Bestellungen 101 und 102 aufgegeben. Kunde 2 (Ben) hat die Bestellung 103 aufgegeben. Dies zeigt, dass die KundenID in der Tabelle Bestelldaten mehrfach vorkommen kann, um die Mehrfachbestellungen eines Kunden darzustellen.
- Fremdschlüssel: Der Fremdschlüssel KundenID verbindet die Entitäten Kunde und Bestellung im ERM.

So wird das relationale Modell in ein strukturiertes ERM überführt, welches die Daten semantisch sinnvoll und verständlich darstellt.

3 Stand der Technik

Dieses Kapitel liefert eine Übersicht über die im Rahmen dieser Arbeit recherchierten und aktuell verfügbaren Techniken und Werkzeuge. Es legt den Fokus auf die ausgewählten Methoden und Verfahren und erklärt ausführlich, weshalb sie gewählt wurden und welchen Beitrag sie zur Zielerreichung dieser Arbeit leisten.

3.1 Vorgehen bei der Suche

Um eine fundierte Auswahl der besten Werkzeuge zur Visualisierung von Entitätsbeziehungsmodellen zu ermöglichen, wurde eine umfangreiche und systematische Literaturanalyse durchgeführt. Diese Recherche konzentrierte sich auf wissenschaftliche Publikationen, Fachzeitschriftenartikel, Online-Datenbanken sowie renommierte Plattformen wie Google Scholar und ResearchGate. Ziel war es, eine gezielte Sammlung von Werkzeugen zusammenzustellen, die sich speziell für die ERM-Visualisierung eignen und in der Fachgemeinschaft weithin anerkannt sind.

Im Verlauf der Analyse wurden die identifizierten Werkzeuge anhand mehrerer Kriterien bewertet, darunter Benutzerfreundlichkeit, Funktionsumfang, Kollaborationsmöglichkeiten, Integration in gängige Entwicklungsumgebungen sowie die Fähigkeit zur automatischen Code-Generierung für Datenbanken. Diese Kriterien ermöglichten eine differenzierte Untersuchung, die es erlaubte, die Werkzeuge hinsichtlich ihrer Eignung für verschiedene Anwendungsszenarien einzuschätzen und so die jeweiligen Stärken und Schwächen gezielt hervorzuheben.

Die gewonnenen Erkenntnisse dieser Literaturrecherche wurden in einer Vergleichstabelle zusammengefasst, die die bedeutendsten ERM-Visualisierungswerkzeuge einander gegenüberstellt. Diese Tabelle soll Entwicklern und Datenmodellierern als zuverlässige Entscheidungshilfe dienen, um das für ihre spezifischen Projektanforderungen geeignete Tool auszuwählen.

Suchbegriffe: Open-source ER diagram tools, Drawing ER diagrams (with Tikz), er diagram maker, Tools to create an Entity Relationship Diagram, ER diagram creation, biger modeling tool, Draw.io, Graphviz.

3.2 Übersicht über gängige Tools zur ERM-Visualisierung

In der Datenmodellierung sind spezialisierte Tools zur ERM-Visualisierung unverzichtbar, da sie das Erstellen, Anpassen und Verstehen komplexer Datenstrukturen erleichtern. Solche Tools bieten häufig benutzerfreundliche Schnittstellen und Funktionen zur Erstellung von ER-Diagrammen, die eine klare Darstellung der Entitäten, Attribute und Beziehungen innerhalb einer Datenbank ermöglichen. Die Auswahl des richtigen Tools hängt von verschiedenen Faktoren ab, wie Benutzerfreundlichkeit, Integration mit Datenbankplattformen und unterstützten Visualisierungsstandards usw. . Ein Überblick über die gängigsten ERM-Tools hilft, das passende Werkzeug für spezifische Projektanforderungen zu identifizieren

1. **BIGER**:

   Das BIGER Tool ist ein hybrides Modellierungswerkzeug zur Erstellung von Entity-Relationship-Diagrammen in der Entwicklungsumgebung Visual Studio Code (VS Code). Es kombiniert textbasierte und grafische Modellierungsmöglichkeiten und nutzt dabei das Language Server Protocol (LSP), wodurch eine plattformübergreifende Nutzung und einfache Erweiterbarkeit möglich sind. Durch eine benutzerdefinierte Sprache und automatische SQL-Generierung unterstützt BIGER eine flexible und interaktive Modellierung von ER-Datenmodellen. BIGER ermöglicht eine simultane Nutzung von textbasierter und grafischer Modellierung. Der textbasierte Ansatz dient als führende Darstellung, während die grafische Ansicht automatisch synchronisiert wird, sodass Änderungen direkt aus der textlichen Beschreibung übernommen werden können. Innerhalb von VS Code bietet das Tool eine separate Diagrammansicht neben dem Texteditor. Die Diagrammkomponenten lassen sich über eine Toolbar anpassen und durch direkte Interaktionen mit dem Diagramm bearbeiten [GB21].

   Zur grafischen Darstellung nutzt BIGER das Sprotty-Framework sowie die Eclipse Layout Kernel (ELK) für automatische Layouts. Die Diagramme werden als skalierbare SVGs angezeigt, was eine hohe Anpassungsfähigkeit und Interaktivität gewährleistet. Eine leicht erlernbare Syntax ermöglicht es Nutzern, ER-Modelle in .erd-Dateien zu definieren, während Syntax-Highlighting, Fehlervalidierung und Hyperlinking die Arbeit im Texteditor erleichtern und die Effizienz beim Modellieren fördern. BIGER ist als Erweiterung im VS Code Marketplace verfügbar und eignet sich besonders für den Einsatz in akademischen Umgebungen, da es offen gestaltet ist und eine Integration für die Generierung von Datenbanken und Diagrammen unterstützt. BIGER stellt somit eine der ersten umfassend integrierten ER-Modellierungslösungen in VS Code dar [GB21].

   Vom relationalen Modell zum ER-Diagramm: BIGER unterstützt die automatische Generierung von Entity-Relationship-Diagrammen aus relationalen Modellen durch SQL-Code [GB21]. Die Code-Generator-Komponente des Tools analysiert SQL-Schemata und wandelt diese in textuelle und grafische ER-Modelle um. Dies ermöglicht es Nutzern, direkt mit relationalen Modellen zu beginnen und daraus ein visuelles ERD zu erstellen. Die Synchronisation erfolgt automatisch, und Änderungen in der textbasierten SQL-Definition werden sofort in der grafischen Darstellung reflektiert. BIGER nutzt dafür fortschrittliche Technologien wie das Sprotty-Framework und den Eclipse Layout Kernel (ELK), um eine intuitive und interaktive Visualisierung der ER-Diagramme sicherzustellen.

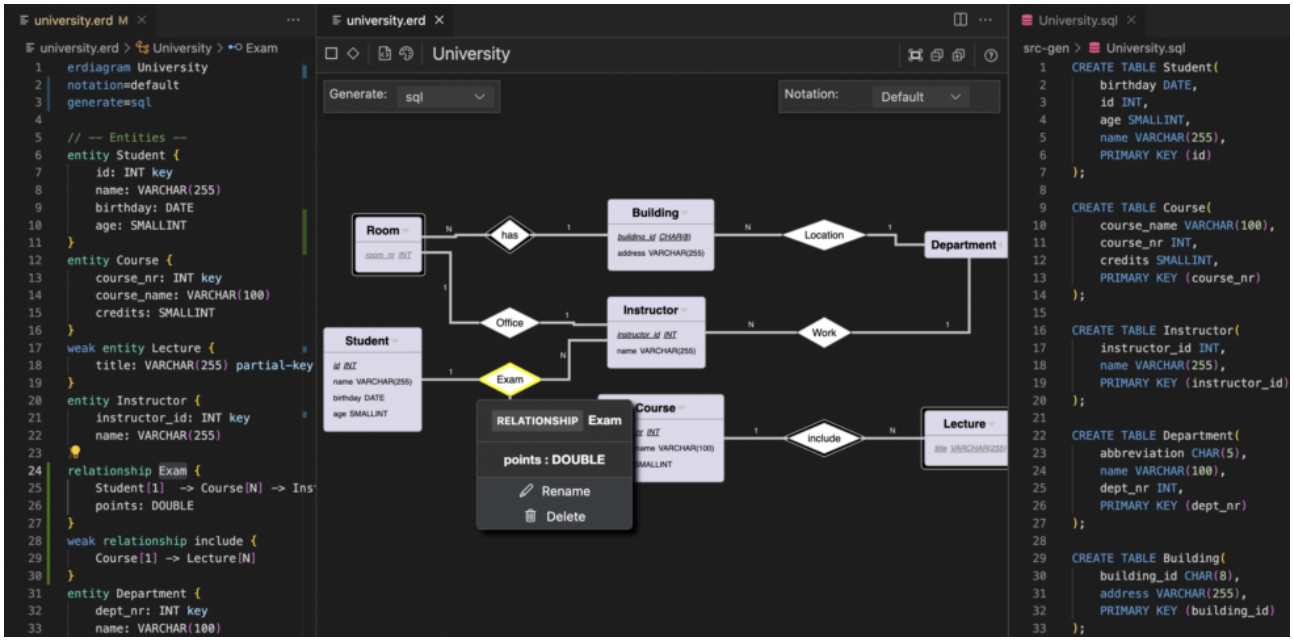


Abbildung 8: Erstellen eines ERD durch BIGER

In Abb. 8 wird im linken Bereich eine textbasierte Sprache verwendet, um das ER-Modell zu spezifizieren. Die Unterstützung durch das Language Server Protocol hilft bei der Validierung und bietet rich-text Editing, sodass die Benutzer schnell mit den Sprachkonstrukten vertraut werden.

Der mittlere Bereich zeigt das ER-Diagramm, das automatisch mit dem textbasierten Modell synchronisiert ist. Das Diagramm aktualisiert sich bei Änderungen im Text und bietet eine interaktive Werkzeugleiste für grafische Bearbeitung, Layout-Anpassungen und Unterstützung für verschiedene Notationen.

Der rechte Bereich zeigt den generierten SQL-Code, der auf dem definierten ER-Modell basiert. Dieser SQL-Code kann verwendet werden, um die Tabellen in einer echten Datenbank zu erstellen und das Modell zu integrieren.

2. **TikZ**:

   Die TikZ ist eine in LaTeX integrierte Bibliothek zur Erstellung von Diagrammen, einschließlich Entity-Relationship-Diagrammen. Sie bietet eine präzise und anpassbare Möglichkeit, Diagramme direkt aus textuellem Code zu erstellen (wie Biger). Besonders in wissenschaftlichen oder technischen Arbeiten, die mit LaTeX erstellt werden. TikZ ermöglicht die Erstellung von ER-Diagrammen mithilfe von textbasiertem Code, was eine präzise Kontrolle über das Layout und die Stiloptionen bietet. Es lässt sich nahtlos in LaTeX integrieren, sodass Diagramme direkt in wissenschaftliche Dokumente eingebettet werden können, ohne dass externe Tools erforderlich sind. Darüber hinaus bietet TikZ eine hohe Flexibilität und Anpassungsmöglichkeiten, da Elemente wie Entitäten, Attribute, Beziehungen und Kardinalitäten detailliert definiert und gestaltet werden können. Die Bibliothek ist Open Source, kostenlos verfügbar und

umfangreich dokumentiert, ergänzt durch zahlreiche praktische Beispiele, die den Einstieg erleichtern [Tan24].

```
1  % Entities
2  \node[entity] (customer) {Customer};
3  \node[entity] (order) [right=5cm of customer] {Order};
4
5  % Attribute für die Entität Customer
6  \node[attribute] (custID) [above left=1cm and 1cm of customer] {CustomerID};
7  \draw[line] (custID) -- (customer);
8
9  % Beziehung zwischen Customer und Order
10 \node[relationship] (places) [ right=2.7cm  ] {places};
11 \draw[line] (customer) -- node[above left] {1} (places);
12 \draw[line] (places) -- node[above right] {N} (order);
13
14 % Attribute für die Entität Order
15 \node[attribute] (orderID) [above right=1cm and 1cm of order] {OrderID};
16 \draw[line] (orderID) -- (order);
```

Listing 1: Erstellen eines ERD durch Tikz

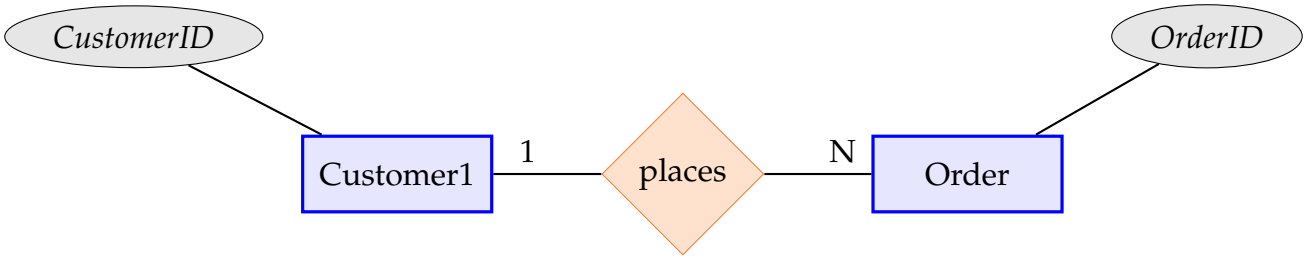


Abbildung 9: Ausgabe eines ERD durch Tikz

Der Code beschreibt ein einfaches ER-Diagramm in einer Textform.
Die Anweisungen wie Node und Draw in Zeile 2 und 7 sind:
Node definiert Elemente wie Entitäten, Attribute oder Beziehungen. Draw verbindet diese Elemente visuell mit Linien. Die Optionen wie [right=5cm of ...] oder [above left=...] bestimmen, wo Knoten platziert werden. Die node[above left] Syntax fügt Text wie Kardinalitäten (1, N) an die Verbindungslinien hinzu.

3. **Graphviz**:

   Graphviz ist ein Open-Source-Tool-Paket, das häufig zur Erstellung und Visualisierung von Graphen verwendet wird [Zha+16]. Es eignet sich hervorragend zur Erstellung von ER-Diagrammen. Solche Diagramme bestehen aus Knoten, die Entitäten, Beziehungen oder Attribute repräsentieren, und Kanten, die Verbindungen zwischen den Elementen darstellen [Fia13]. Es basiert auf der DOT-Sprache, den Quellcode des Graphen in der DOT-Sprache [1] herunterzuladen [PF10].
   Es wird oft das Tool *dot2texi* verwendet, das bereits in *TEX Live* und *MiKTEX* enthalten ist. Dieses Tool vereinfacht die Zusammenarbeit zwischen Graphviz und LaTeX, insbesondere bei der Visualisierung von ER-Diagrammen [Fia13].

   Graphviz zeichnet sich durch die Verwendung integrierter Layout-Algorithmen [2] aus, die die automatische Platzierung der Diagrammelemente übernehmen. Zu diesen Algorithmen gehören unter anderem dot, fdp, circo, neato und twopi [Zha+16]. Allerdings müssen Nutzer die Stile der Elemente selbst definieren, da Graphviz keine voreingestellten Stile bietet [Fia13]. Das Tool unterstützt die Ausgabe von Diagrammen in verschiedenen Formaten wie GIF, PDF, SVG und PNG. Dank der Möglichkeit, verschiedene Layouts zu unterstützen und automatisch anzuwenden, ermöglicht Graphviz eine schnellere und präzisere Visualisierung komplexer Datenstrukturen (siehe Abb. 11) [Zha+16].

   Die Automatisierung der grafischen Layout-Funktion mit Graphviz reduziert den manuellen Aufwand erheblich, der normalerweise bei der Erstellung von ER-Diagrammen erforderlich ist. In einem ER-Diagramm kann jede Entität als Box dargestellt werden, die Attribute enthält, während die Beziehungen zwischen Entitäten durch Pfeile mit spezifischen Markierungen (z. B. Eins-zu-Viele-Beziehungen) visualisiert werden (siehe Abb. 10) [MM15].
   Vor der Nutzung sollte sichergestellt werden, dass sowohl Graphviz als auch dot2tex auf dem System installiert sind. Die Fähigkeit von Graphviz, Diagramme mithilfe automatisierter Layout-Algorithmen zu erstellen, kombiniert mit der Unterstützung von dot2tex, macht es zu einem leistungsstarken Werkzeug für die Erstellung und visuelle Darstellung komplexer Datenbeziehungen [Fia13].

[1] DOT-Sprache ist eine visuelle Beschreibungssprache, die sowohl für Menschen als auch für Software leicht verständlich ist.

[2] Graphviz-Layout-Algorithmen ordnen Knoten und Kanten automatisch an, um Diagramme übersichtlich und ästhetisch darzustellen. Sie minimieren Überlappungen, reduzieren Kantenkreuzungen und optimieren die Struktur je nach Diagrammtyp:

- dot: Hierarchische Layouts (gerichtete Graphen, Flussdiagramme), die wir für unseren Anwendungsfall benötigen.
- neato/fdp: Kraftbasierte Layouts (ungerichtete Graphen).
- twopi: Radiale Layouts (konzentrische Kreise).
- circo: Zyklische Layouts (Ringe).

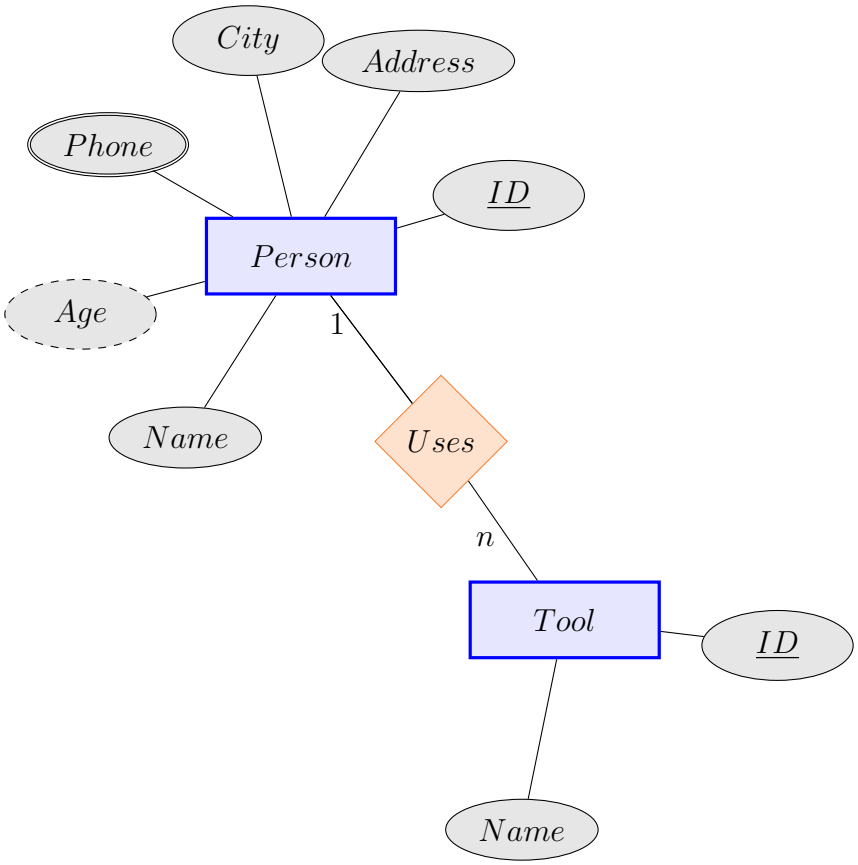


Abbildung 10: Erstellen eines ERD durch Graphviz-Dot in Kombination mit TikZ

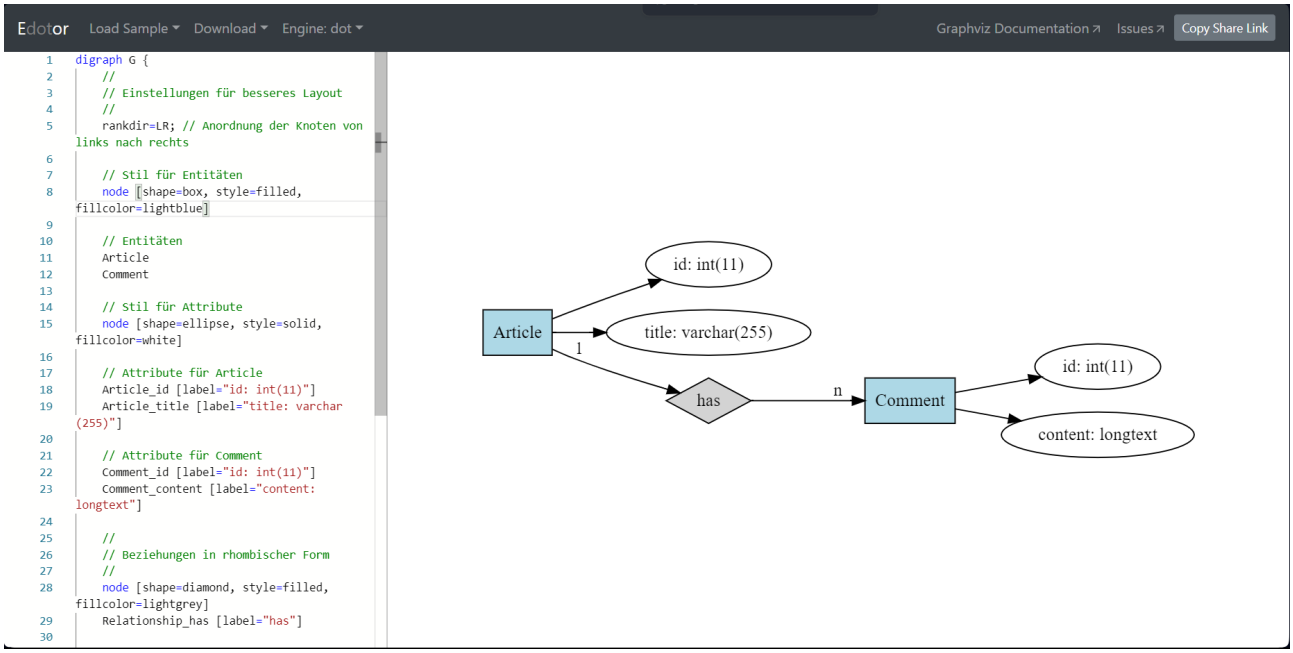


Abbildung 11: Erstellen eines ERD mit Graphviz in DOT-Sprache

In Abb. 11 die linke Seite zeigt den DOT-Code, der als Eingabe für Graphviz dient. Der Quellcode definiert:

- Entitäten: Article und Comment als Knoten.
- Attribute:

  Für Article: id und title.
  Für Comment: id und content.

- Beziehungen: Die Verbindung has zwischen Article und Comment.
- Kardinalität: 1 auf der Seite von Article und n auf der Seite von Comment.

Die rechte Seite zeigt das visuelle Ergebnis (ein ERD):
Die grafische Darstellung des Quellcodes, die mit einem Tool wie Graphviz erstellt wurde.

- Entitäten: Article und Comment sind als blaue Rechtecke dargestellt.
- Attribute: id, title und content sind mit Ellipsen verbunden.
- Beziehung: has ist als graue Raute dargestellt, die die Entitäten verbindet.
- Kardinalität: 1 auf der Seite von Article und n auf der Seite von Comment verdeutlicht die Beziehungstypen.

4. **Gliffy**:

   Gliffy ist ein vielseitiges Tool zur Erstellung von Diagrammen, das sich besonders für UML- und ER-Diagramme eignet und kostenlos für zwei Wochen getestet werden kann. Mit dieser Software können Teams auch aus der Ferne zusammenarbeiten und in Echtzeit an gemeinsamen Projekten arbeiten. Dank der intuitiven Drag-and-Drop-Oberfläche lassen sich Diagramme schnell und einfach erstellen, was die Benutzerfreundlichkeit weiter steigert. Gliffy ermöglicht zudem die Nachverfolgung von Änderungen in den Diagrammen, sodass Benutzer jederzeit frühere Versionen wiederherstellen und den Fortschritt im Hinblick auf das gewünschte Endergebnis nachvollziehen können. Neben der Möglichkeit zur Zusammenarbeit und dem Überwachen von Änderungen können Sie in Gliffy auch Bilder importieren und die erstellten Diagramme exportieren [Yud23, Gre21].
   Die Benutzerfreundlichkeit von Gliffy ist besonders im Bildungsbereich ein großer Vorteil, da Schüler und Lehrer ohne technische Hürden visuelle Inhalte gestalten können. Die Software kann in verschiedenen Fächern eingesetzt werden und unterstützt das visuelle Darstellen von Informationsverknüpfungen, Sequenzen, Vergleichen und Brainstorming. Ein weiteres Plus ist die automatische Speicherung alle 30 Sekunden, die zusätzliche Datensicherheit gewährleistet, da keine Informationen verloren gehen, selbst wenn technische Probleme auftreten. Mit seinen umfassenden Funktionen und flexiblen Einsatzmöglichkeiten ist Gliffy ein wertvolles Tool zur Erstellung und Verwaltung von Diagrammen in unterschiedlichsten Kontexten [McP07].

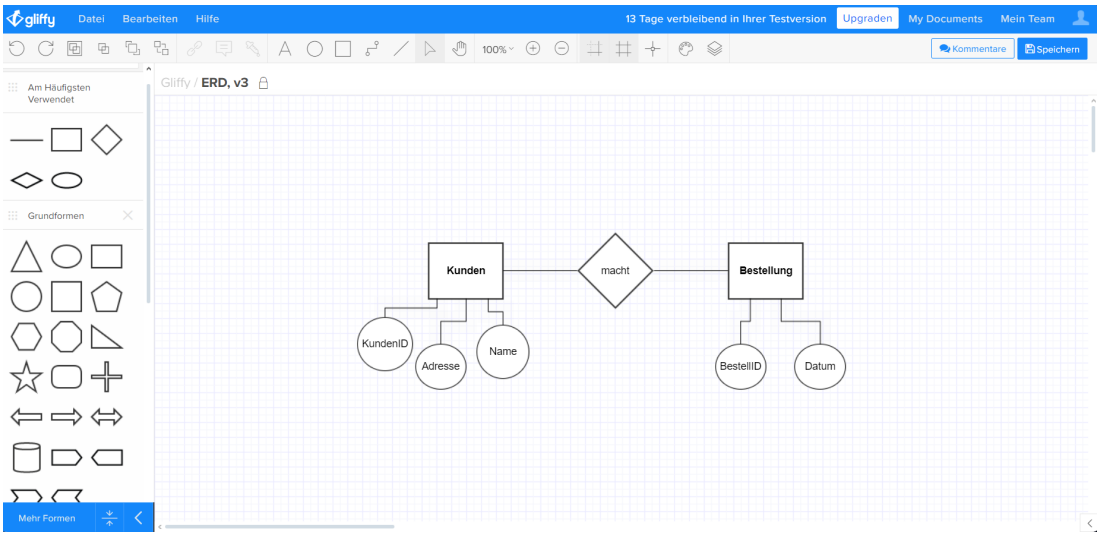


Abbildung 12: Erstellen eines ERD durch Gliffly

In dem gezeigten ER-Diagramm in Abb.12 sind die Entitäten „Kunden“ und „Bestellung“ dargestellt. Die Entität „Kunden“ besitzt die Attribute „KundenID“, „Adresse“ und „Name“, während „Bestellung“ die Attribute „BestellID“ und „Datum“ aufweist. Die beiden Entitäten sind durch die Beziehung „macht“ miteinander verbunden, was anzeigt, dass ein Kunde eine Bestellung aufgeben kann. Dieses Diagramm veranschaulicht die Beziehung zwischen Kunden und ihren Bestellungen in einer Datenbank.

5. **Visual Paradigm**

Visual Paradigm ist ein vielseitiges ER-Diagramm-Tool, das eine breite Palette von Diagrammtypen unterstützt. Dank der benutzerfreundlichen Drag-and-Drop-Oberfläche lassen sich visuelle Modelle einfach erstellen, selbst wenn man keine professionellen Designkenntnisse besitzt.
Zu den wichtigsten Funktionen gehören der Sweeper und der Magnet: Mit der Sweeper-Funktion kann der Abstand zwischen Diagrammelementen leicht vergrößert werden, während der Magnet es ermöglicht, die Elemente durch einfaches Ziehen näher zusammenzubringen. Die farbige Legende ist eine weitere nützliche Funktion, mit der sich verschiedene Prioritäten, Entwicklungsstufen und Reifegrade im ERD durch den Einsatz von Farben deutlich visualisieren lassen. Zudem spart der Formatkopierer viel Zeit, indem er es erlaubt, Stileinstellungen wie Schriftart, Linien und Füllung schnell von einem Element auf ein anderes zu übertragen [Kam24].

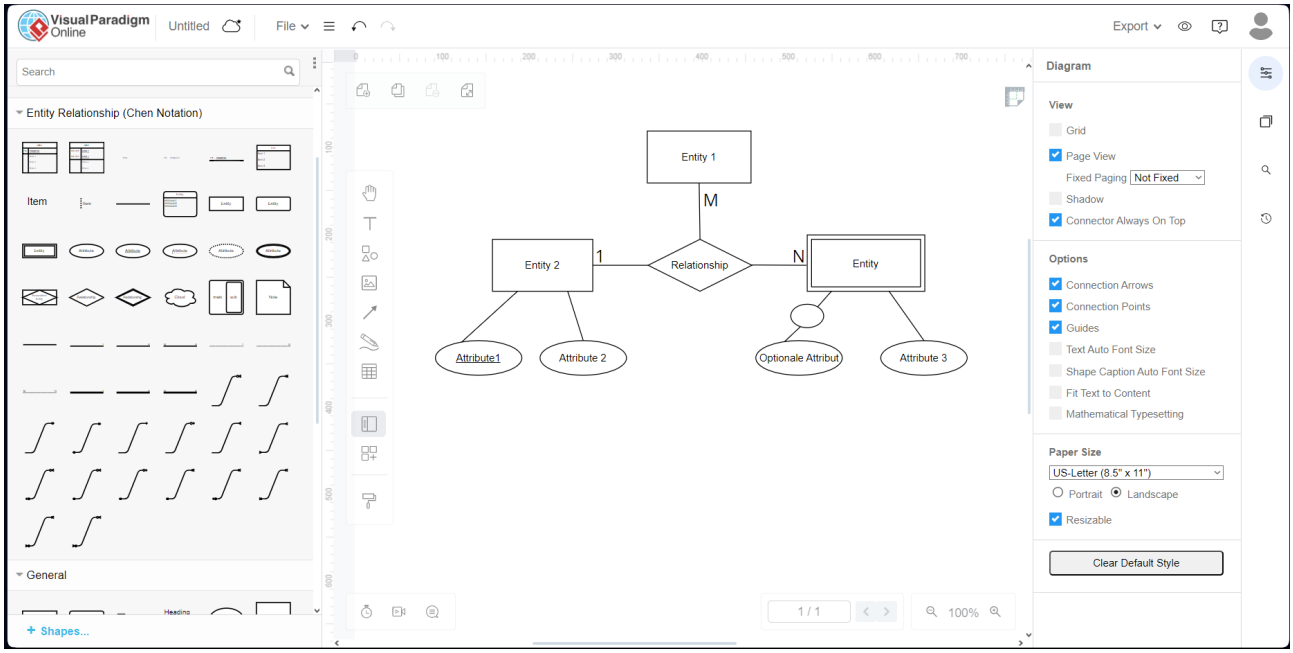


Abbildung 13: Erstellen eines ERD durch Visual Paradigm

Das Abb. 13 zeigt die Benutzeroberfläche von Visual Paradigm Online, einem Tool zur Erstellung von Entity-Relationship-Diagrammen. Auf der linken Seite befindet sich eine Werkzeugleiste mit Symbolen für die Chen-Notation, einschließlich Entitäten (Rechtecke), Attribute (Ovale, einfache und optionale) und Beziehungen (Rauten), sowie verschiedene Verbindungslinien. Auf der rechten Seite gibt es eine Einstellungsleiste, mit der Diagramme angepasst werden können, z. B. durch Aktivierung von Gitternetz, Anpassung der Papiergröße und Ausrichtung, sowie Optionen für Verbindungspunkte und Schriftgröße. In der Mitte wird ein ER-Diagramm angezeigt, welches aus mehreren Entitäten, Attributen (inklusive optionalen Attributen) usw. besteht. Das Diagramm verwendet die Chen-Notation und visualisiert die Beziehung zwischen verschiedenen Entitäten und ihren Attributen.

6. **DBDiagram.io**:

   Dbdiagram.io ist ein webbasiertes Open-Source-Tool zur Erstellung von Entity-Relationship-Diagrammen, das durch das Schreiben von Code arbeitet und sich somit besonders gut für Nutzer eignet, die lieber mit der Tastatur arbeiten. Es verwendet einen textbasierten Ansatz und implementiert eine eigene domänenspezifische Sprache (DSL), die für eine schnelle Lernkurve ausgelegt ist. Diese DSL ermöglicht eine intuitive grafische Darstellung der Diagramme, wobei die Benutzer die Elemente in Echtzeit frei organisieren können. Die Modellierung erfolgt dennoch vollständig textbasiert [Lop+21, Gre21].
   Darüber hinaus bietet Dbdiagram.io die automatische Generierung von SQL-Code, was die Erstellung und Verwaltung von Datenbankstrukturen erleichtert. Das kostenlose Tool ermöglicht es, Diagramme als Bild- oder PDF-Dateien zu exportieren, die sich mit einem Klick teilen lassen. Zudem unterstützt es die Option, das Datenbankschema bei einer Aktualisierung in Web-Frameworks wie Django hochzuladen, was die Integration in bestehende Projekte vereinfacht [Gre21, Lop+21].

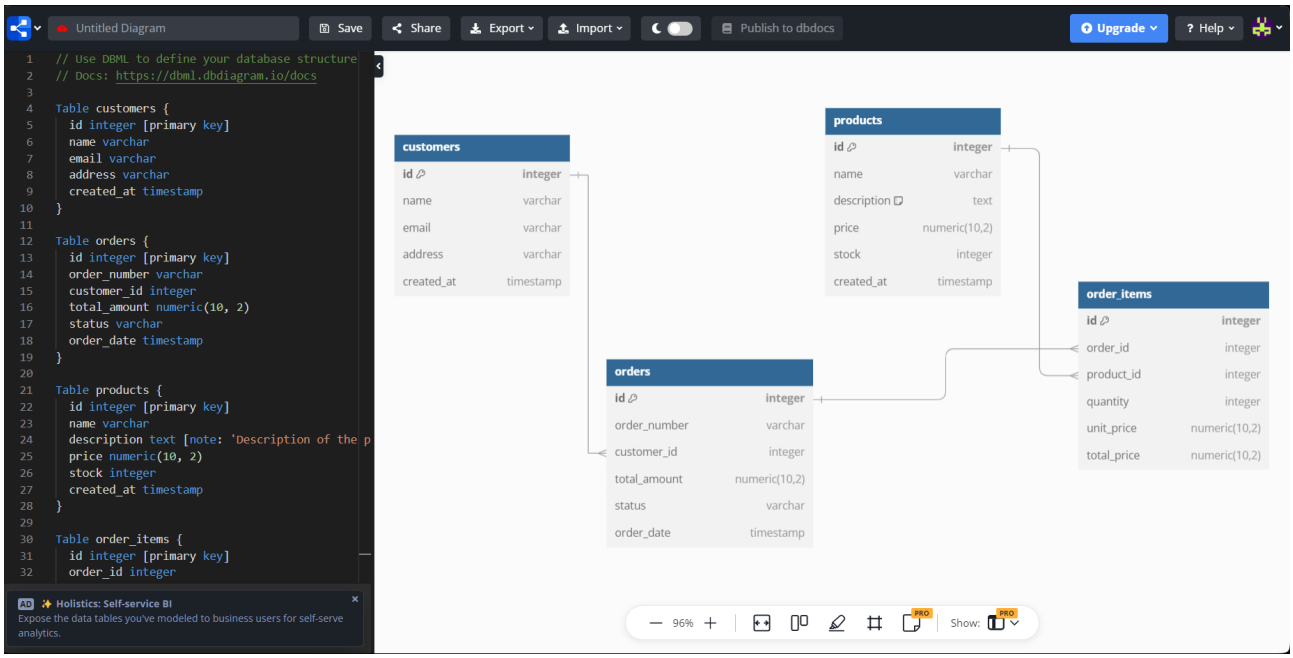


Abbildung 14: Erstellen eines ERD durch DBDiagram.io

Dieses Diagramm in Abb. 14 zeigt eine Datenbankstruktur für ein einfaches Bestellsystem und veranschaulicht die Tabellen customers, orders, products und order_items sowie deren Beziehungen. customers: Speichert Kundeninformationen, wie id, name, email, address und created_at. orders: Enthält Bestelldaten, einschließlich id, order_number, customer_id, total_amount, status und order_date. Jede Bestellung ist über customer_id mit einem Kunden verknüpft. products: Enthält Informationen zu Produkten, wie id, name, description, price, stock und created_at. order_items: Verknüpft Bestellungen und Produkte und speichert Details zu den bestellten Artikeln, einschließlich order_id, product_id, quantity, unit_price und total_price. Das Tool, das hier verwendet wird, bietet eine benutzerfreundliche Oberfläche, bei der auf der linken

Seite im Code-Editor der Datenbankcode in DBML [3] geschrieben wird, während auf der rechten Seite eine sofortige visuelle Darstellung des ER-Diagramms generiert wird. Diese duale Ansicht ermöglicht es, Datenbankstrukturen direkt im Code zu definieren und das Ergebnis gleichzeitig visuell zu überprüfen, was die Arbeit an Datenmodellen effizient und übersichtlich gestaltet.

7. **Draw.io**:

   Draw.io ist ein webbasiertes sowie als Desktop-Version verfügbares Diagrammwerkzeug, das sich durch eine Drag-and-Drop-Funktionalität und eine benutzerfreundliche Oberfläche auszeichnet. Es unterstützt die Erstellung von ER-Diagrammen mit grundlegenden Elementen wie Entitäten, Beziehungen und Attributen. Zu den Hauptfunktionen von Draw.io gehören die Diagrammspeicherung und -bearbeitung, wodurch Diagramme gespeichert und später bearbeitet werden können, was eine kontinuierliche Arbeit an Projekten ermöglicht. Anpassungsmöglichkeiten bieten die Option, Farben, Schriftgrößen, Linien und andere Elemente individuell anzupassen, um das Diagramm an spezifische Anforderungen anzupassen. Die Exportfunktion erlaubt das Exportieren der Diagramme als Bild- oder PDF-Datei, was die Nutzung der erstellten Diagramme in verschiedenen Formaten und Anwendungen erleichtert [JM20].

   Draw.io ermöglicht die Zugriffskontrolle und Organisation von Projekten im Team. Es unterstützt die Kollaborationsfunktion, die es mehreren Benutzern ermöglicht, gemeinsam an einem Diagramm zu arbeiten, was es besonders praktisch für Teams macht. Zudem bietet es die Möglichkeit zur Transformation von ER- zu relationalen Modellen, was für die Datenbankimplementierung nützlich ist. Draw.io ist zudem kostenfrei nutzbar und enthält ein Tutorial zur Einführung in die Funktionen [JM20].

[3] DBML (Database Markup Language) ist eine einfache, lesbare DSL-Sprache, die zur Definition von Datenbankstrukturen entwickelt wurde.

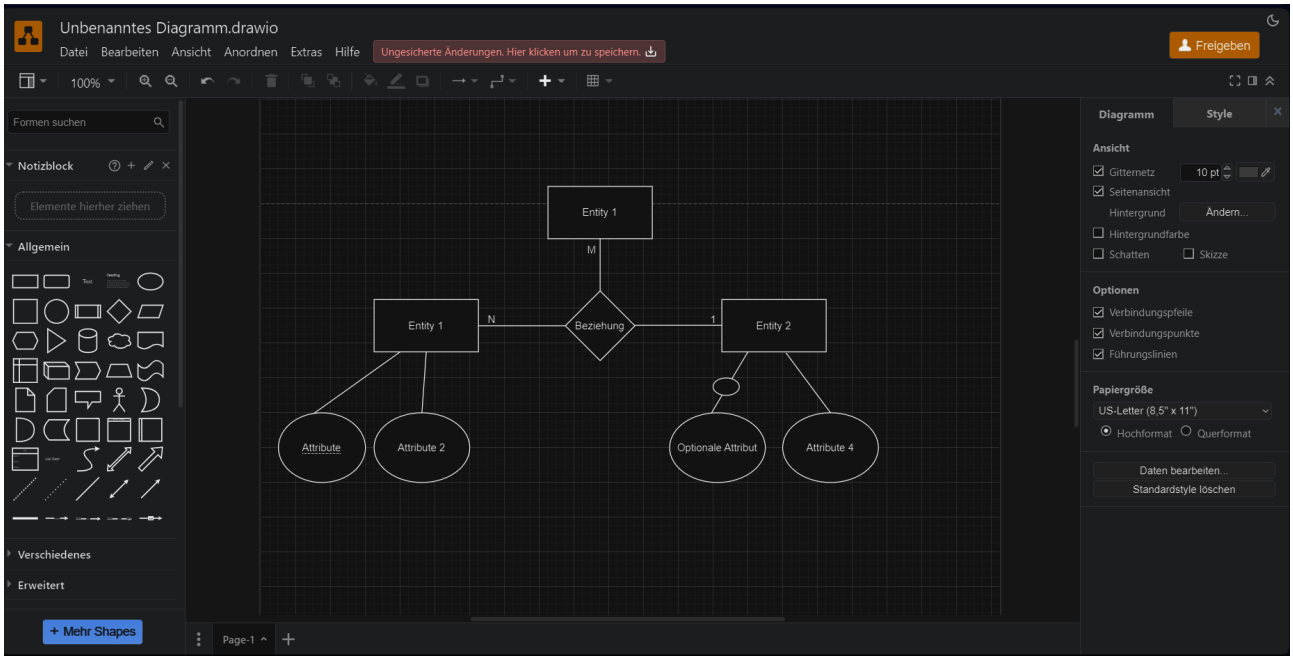


Abbildung 15: Erstellen eines ERD durch Draw.io

Das Diagramm in Abb. 15 zeigt ein einfaches ER-Diagramm. Auf der linken Seite befindet sich ein Werkzeugbereich mit verschiedenen Formen, wie Rechtecken, Kreisen und Pfeilen, die für die Erstellung des Diagramms verwendet werden können. Der mittlere Bereich zeigt das eigentliche Diagramm. Auf der rechten Seite des Bildes sind die Einstellungen des Diagramms sichtbar. Hier können Ansichtsoptionen wie Gitternetz, Hintergrundfarbe und Papierformat (z. B. US-Letter) angepasst werden, sowie Verbindungs- und Führungslinien ein- oder ausgeschaltet werden.

Tool	Entitäten	Attribute	Beziehungen	Optionale Attribute	N-äre Relationen	Schwache Entitätstypen	ERM-Stil
Biger	Ja	Ja	Ja	Ja	Ja	Ja	Chen, Crow's Foot
TikZ	Ja	Ja	Ja	Ja (manuell)	Ja (manuell)	Ja (manuell)	Chen, Crow's Foot (manuell)
Gliffy	Ja	Ja	Ja	Nein	Nein	Nein	Grundlegende Diagramme (keine spezifischen ERM-Stile)
Graphviz	Ja	Ja	Ja	Ja (indirekt)	Ja	Ja (indirekt)	Chen, Crow's Foot
Visual Paradigm	Ja	Ja	Ja	Ja	Ja	Ja	Chen, Crow's Foot
DBDiagram.io	Ja	Ja	Ja	Nein	Nein	Nein	Crow's Foot
Draw.io	Ja	Ja	Ja	Ja (manuell)	Ja (manuell)	Ja (manuell)	Chen, Crow's Foot (manuell)

Tabelle 4: Vergleich der ERM-Funktionen von Tools

3.3 Vergleich verschiedener Ansätze

Hier in Tabelle 5 werden verschiedene Tools zur Visualisierung von Entity-Relationship-Modellen hinsichtlich ihrer Vor- und Nachteile verglichen.

Tool	Vorteile	Nachteile
Graphviz	- Flexibel für komplexe Diagramme - Automatisches Layout - Exportformate: GIF, PDF, SVG, PNG	- Erfordert DOT-Sprache - Keine Drag-and-Drop-Oberfläche
TikZ	- Präzise Diagramme per LaTeX-Code - Anpassbar (Farben, Linien, Schrift)	- Erfordert LaTeX/TikZ-Kenntnisse - Manuelle Code-Erstellung zeitaufwändig - Fehleranfällig bei komplexen Diagrammen
Gliffy	- Benutzerfreundlich (Drag-and-Drop) - Echtzeit-Kollaboration - Automatische Speicherung	- Eingeschränkte Anpassung für komplexe Modelle - Einige Features kostenpflichtig
Visual Paradigm	- Anpassbar (Farben, Legenden) - Formatkopierer-Funktion - Kann für einfache Diagramme zu komplex sein	- Einige Funktionen nur kostenpflichtig
Db-diagram.io	- Textbasierte Modellierung - Automatische SQL-Generierung - Web-Framework-Integration	- Keine Drag-and-Drop-Unterstützung - Erfordert Einarbeitung in DSL
BIGER	- Kombiniert Text- und Grafikmodellierung - Plattformübergreifend	- Abhängig von VS Code - Begrenzte Anpassungsmöglichkeiten
Draw.io	- Drag-and-Drop-Oberfläche - ERD-Transformation zu relationalen Modellen - Kostenlos und webbasiert	- Eingeschränkte Funktionen für komplexe Strukturen

Tabelle 5: Vor- und Nachteile verschiedener ER-Visualisierungstools

4 Konzeption

Dieses Kapitel beschreibt den konzeptionellen Ansatz, der als Grundlage für die Entwicklung und Umsetzung des Projekts dient. Es konzentriert sich auf die Auswahl und Darstellung zentraler Ergebnisse sowie auf die methodische Überführung relationaler Modelle in Entity-Relationship-Modelle.

Zunächst werden in Abschnitt 4.1 die Hauptergebnisse der vorangegangenen Analyse zusammengefasst, die die Basis für die weitere konzeptionelle Arbeit bilden. Diese Ergebnisse liefern die Grundlage für die Strukturierung und das Verständnis der zugrunde liegenden Daten.

Anschließend wird in Abschnitt 4.2 aufgezeigt, wie relationale Modelle auf ERMs abgebildet werden können. Dieser Schritt ermöglicht es, Datenstrukturen und deren Beziehungen systematisch zu visualisieren und bietet eine solide Grundlage für eine strukturierte und effektive Implementierung

4.1 Auswahl der Hauptergebnisse

Die Auswahl des geeigneten Werkzeugs für die Rückgewinnung von Entity-Relationship-Modellen aus relationalen Datenbanken und deren Implementierung ist von zentraler Bedeutung. Nach einer sorgfältigen Analyse verschiedener verfügbarer Tools wurden die Anforderungen an das Tool definiert, die sowohl die Modellierung als auch die Implementierung unterstützen sollen. Ein essenzielles Kriterium ist die Fähigkeit zur grafischen Modellierung, da eine visuelle Darstellung von Entitäten, Attributen und Beziehungen unverzichtbar ist, um komplexe Strukturen klar zu strukturieren und die Kommunikation mit Stakeholdern zu erleichtern. Zusätzlich sollte das Tool benutzerfreundlich, plattformunabhängig und flexibel einsetzbar sein, um eine langfristige Nutzbarkeit sowie Anpassungsfähigkeit an verschiedene Projektanforderungen zu gewährleisten.

Nach einer umfassenden Bewertung mehrerer Tools, darunter Graphviz, Dbdiagram.io, Visual Paradigm, Draw.io und BIGER usw., wurde festgestellt, dass*Draw.io* diese Anforderungen am besten erfüllt. Es bietet nicht nur alle grundlegenden Funktionen, sondern auch spezifische Eigenschaften, die für die Anforderungen an die grafische Modellierung besonders vorteilhaft sind.

Draw.io bietet nicht nur die grundlegenden Funktionen zur Erstellung von Entity-Relationship-Diagrammen, sondern zeichnet sich durch eine Vielzahl zusätzlicher Features aus, die speziell auf die Anforderungen der Datenmodellierung abgestimmt sind.

Ein Vorteil ist die Flexibilität in der Gestaltung. Draw.io ermöglicht die individuelle Anpassung von Diagrammelementen wie Farben, Schriftarten, Linienstilen und Größen. Diese Anpassungsoptionen erlauben es, spezifische Anforderungen visuell hervorzuheben und komplexe Modelle übersichtlich darzustellen. Darüber hinaus können Elemente wie gestrichelte Linien für optionale Beziehungen oder doppelte Rahmen für schwache Entitäten problemlos hinzugefügt und bearbeitet werden.

Ein Pluspunkt ist die Unterstützung komplexer Notationen. Neben den Grundelementen der Chen-Notation können erweiterte Modellierungselemente integriert werden, um beispielsweise Kardinalitäten (1:1, 1:N, M:N) oder ternäre bzw. n-äre Beziehungen darzustellen. Diese Funktionalitäten sind besonders hilfreich, um detaillierte und realitätsnahe Abbildungen von Geschäftsprozessen und Datenstrukturen zu erstellen.

Ein weiterer Pluspunkt von Draw.io ist die Möglichkeit, Diagramme in verschiedenen Formaten wie PNG, PDF oder SVG zu exportieren und leicht in andere Dokumente oder Systeme zu integrieren. Die Anpassungsfähigkeit des Tools ermöglicht es, Elemente wie gestrichelte Linien, optionale Attribute und schwache Entitäten grafisch exakt umzusetzen, was besonders bei der Erstellung von ER-Modellen nach der Chen-Notation von großer Bedeutung ist. Dadurch eignet sich Draw.io hervorragend für die visuelle Modellierung und Dokumentation von relationalen Datenbankstrukturen.

Begründung für die Nicht-Auswahl anderer Tools

- Graphviz:

  - Bietet leistungsstarke Visualisierungsfunktionen, jedoch keine Drag & Drop Oberfläche. Auch wenn dies für den automatisierten Hauptprozess nicht zwingend erforderlich ist, ermöglicht eine solche Funktion mehr Flexibilität, um Modelle bei Bedarf manuell anzupassen. Insbesondere bei Feinkorrekturen wie der Positionierung von Elementen oder der Optimierung der Lesbarkeit kann eine visuelle Bearbeitungsmöglichkeit hilfreich sein, um das Ergebnis übersichtlicher zu gestalten.
  - Erfordert Kenntnisse der DOT-Sprache für Anfänger, was den Einstieg erschwert.

- Dbdiagram.io:

  - Keine vollwertige grafische Bearbeitungsmöglichkeit, was die visuelle Darstellung komplexer Strukturen einschränkt.
  - unterstützt nicht Chen-Notation

- Visual Paradigm:

  - Bietet umfangreiche Funktionen für die Modellierung, jedoch sind viele Features kostenpflichtig.
  - Hohe Komplexität und weniger geeignet für einfache, schnelle Modellierungsprozesse.

- BIGER:

  - Fehlende Einfachheit für Anwender ohne technische Vorkenntnisse und keine direkte Drag-and-Drop-Funktionalität.

4.2 Abbildung relationaler Modelle auf ERMs

Relationale Modelle und ERMs sind zwei grundlegende Konzepte im Bereich der Datenmodellierung. Während relationale Modelle auf Tabellen (Relationen) basieren, konzentriert sich das ERM auf die Darstellung von Entitäten und deren Beziehungen. Der Prozess der Abbildung relationaler Modelle auf ERMs dient dazu, ein bereits existierendes relationales Datenbankschema in ein ER-Diagramm zu transformieren, um es visuell verständlicher darzustellen oder weiterzuentwickeln.

4.2.1 Schritte der Abbildung

Die Abbildung eines relationalen Modells auf ein ERM erfolgt durch eine strukturierte Transformation der einzelnen Bestandteile des relationalen Datenbankschemas. Dabei werden Tabellen, Spalten, Fremdschlüssel und Zwischentabellen in die konzeptionellen Elemente eines ERMs überführt.

Als grundlegender Schritt werden Tabellen direkt als Entitäten im ERM interpretiert. Jede Tabelle im relationalen Modell wird in der Regel einer Entität zugeordnet, wobei die Tabelle den Entitätstyp darstellt. Beispielsweise wird eine Tabelle *Kunde* zur Entität *Kunde* im ERM. Der Primärschlüssel einer Tabelle wird das eindeutige Attribut (Schlüsselattribut) der Entität. So wird beispielsweise in der Tabelle *Kunde* die Spalte *KundenID*, die als Primärschlüssel definiert ist, im ERM als Attribut *KundenID* der Entität *Kunde* dargestellt.

Das Beispiel wird wie folgt als logisches Modell geschrieben:

Kunde {KundenID, {KundenID}}

Das Beispiel wurde in XML geschrieben:

```
<mxfile host="65bd71144e">
   <diagram  name="Page-1">
       <mxGraphModel dx="253" dy="508" grid="0" gridSize="10" guides="1" tooltips="1" connect="1"
           arrows="1" fold="1" page="1" pageScale="1" pageWidth="850" pageHeight="1100" background
           ="#ffffff" math="0" shadow="0">
           <root>
               <mxCell id="0"/>
               <mxCell id="1" parent="0"/>
               <mxCell id="2" value="Kunde" style=" whiteSpace=wrap;fillColor=#999999;strokeColor
                   =#000000;fontColor=00000" vertex="1" parent="1">
                   <mxGeometry x="254" y="243" width="100" height="40" as="geometry"/>
               </mxCell>
               <mxCell id="3" value="KundenID" style="ellipse;whiteSpace=wrap;fontStyle=4;
                   strokeColor=#000000;fontColor=#000000;fillColor=#CCCCCC;" parent="1" vertex="1">
                   <mxGeometry x="413" y="243" width="100" height="40" as="geometry"/>
               </mxCell>
               <mxCell id="4" value="" style="endArrow=none;strokeColor=#000000;" edge="1" parent
                   ="1" source="2" target="3">
                   <mxGeometry relative="1" as="geometry"/>
               </mxCell>
           </root>
       </mxGraphModel>
   </diagram>
</mxfile>
```

Listing 2: XML-Darstellung eines Diagramms für Entität mit Schlüsselattribut.

Hier ist die Erklärung der einzelnen Elemente des Codes:

(<mxCell id="2") Dies stellt die Entität Kunde durch Anweisungen dar:

- value= Kunde: Der Text der Entität, der im Diagramm angezeigt wird.
- style: whiteSpace=wrap: Der Text innerhalb der Form wird umgebrochen.
  fillColor=#999999: Der Hintergrund der Entität ist grau.
  strokeColor=#000000: Die Kantenfarbe (Umrandung) ist schwarz.
  fontColor=00000: Die Schriftfarbe des Textes ist schwarz.
  vertex= 1: Gibt an, dass es sich um einen Knoten (Entität) handelt.
- Position und Größe: x=254, y=243: Die Position der Entität auf der Zeichenfläche. width=100, height=40: Die Breite und Höhe der Entität.

(<mxCell id="3") Dies ist das Attribut KundenID, das als Primärschlüssel der Entität Kunde dient. Sie wird als Ellipse darstellt (`style="ellipse;whiteSpace=wrap;"`).

- fontStyle=4: Der fett formatierte Text (fontStyle=4) zeigt an, dass es sich um ein Schlüsselattribut handelt.

(<mxCell id="4") Dies ist die Kante, die die Entität Kunde mit dem Attribut KundenID verbindet.

- value=: Es gibt keinen Text für diese Verbindung.
- style:
  endArrow=none: Die Linie hat keine Pfeilspitze.
  strokeColor=#000000: Die Farbe der Linie ist schwarz.
- edge=1: Gibt an, dass es sich um eine Kante (Verbindung) handelt.
- source=2: Die Quelle der Verbindung ist die Entität Kunde (id="2").
- target=3: Das Ziel der Verbindung ist das Attribut KundenID (id="3").
- relative=1: Die Verbindung wird relativ zwischen den beiden verbundenen Elementen gezeichnet.

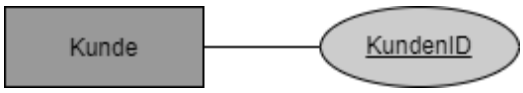


Abbildung 16: Ausgabe Entität mit Schlüsselattribut in ERD

Die Spalten der Tabellen, die keine Fremdschlüssel sind, werden als Attribute der entsprechenden Entität dargestellt. Beispielsweise wird die Spalte *Name* in der Tabelle *Kunde* zu einem Attribut *Name* der Entität *Kunde*. Auch die Datentypen der Spalten im relationalen Modell werden im ERM berücksichtigt und durch geeignete Annotationen an den Attributen abgebildet.

Das Beispiel wird wie folgt als logisches Modell geschrieben:

Kunde {Name}

Das Beispiel wurde in XML geschrieben:

```
1 <mxfile host="65bd71144e">
2     <diagram name="Page-1">
3         <mxGraphModel dx="310" dy="556" grid="0" gridSize="10" guides="1" tooltips="1" connect="1"
              arrows="1" fold="1" page="1" pageScale="1" pageWidth="850" pageHeight="1100" background
              ="#ffffff" math="0" shadow="0">
4             <root>
```

```
                <mxCell id="0"/>
                <mxCell id="1" parent="0"/>
                <mxCell id="2" value="Kunde" style="whiteSpace=wrap;" parent="1" vertex="1">
                    <mxGeometry x="374" y="279" width="100" height="40" as="geometry"/>
                </mxCell>
                <mxCell id="3" value="Name" style="ellipse;whiteSpace=wrap;fillColor=#CCCCCC;
                    fontColor=#000000;strokeColor=#000000;" parent="1" vertex="1">
                    <mxGeometry x="549" y="279" width="100" height="40" as="geometry"/>
                </mxCell>
                <mxCell id="4" value="" style="endArrow=none;" edge="1" parent="1" target="3">
                    <mxGeometry relative="1" as="geometry">
                      </mxGeometry>
                </mxCell>
            </root>
        </mxGraphModel>
    </diagram>
</mxfile>
```

Listing 3: XML-Darstellung eines Diagramms für Enität und Attribut.

Erläuterung der einzelnen Code-Elemente:

(<mxCell id="3") Dies ist das Attribut Name, das der Entität Kunde zugeordnet ist.

- fontStyle fehlt: Es gibt keinen Hinweis darauf, dass es sich um einen Schlüssel handelt.

Abbildung 17: Ausgabe Entität und Attribut in ERD

Fremdschlüssel in Tabellen werden genutzt, um Beziehungen zwischen Entitäten im ERM darzustellen. Eine Tabelle *Bestellung*, die beispielsweise einen Fremdschlüssel *KundenID* enthält, welcher auf die Tabelle *Kunde* verweist, wird im ERM durch eine Beziehung zwischen der Entität *Kunde* und der Entität *Bestellung* repräsentiert. Diese Beziehung kann je nach Struktur und Kardinalität entweder eine 1:N-Beziehung oder eine M:N-Beziehung sein. Eine 1:N-Beziehung entsteht, wenn eine Entität (z. B. *Kunde*) mit vielen anderen Entitäten (z. B. *Bestellung*) verknüpft ist.

Das Beispiel wird wie folgt als logisches Modell geschrieben:

Kunde {KundenID, {KundenID}} Bestellung {BestellID, {BestellID}}

Das Beispiel wurde in XML geschrieben:

```
<root>
                <mxCell id="0"/>
                <mxCell id="1" parent="0"/>
                <mxCell id="2" value="Kunde" style="whiteSpace=wrap;" parent="1" vertex="1">
                    <mxGeometry x="236" y="223" width="109" height="53" as="geometry"/>
                </mxCell>
                <mxCell id="3" value="Bestellung" style="whiteSpace=wrap;" parent="1" vertex="1">
                    <mxGeometry x="494" y="223" width="109" height="53" as="geometry"/>
                </mxCell>
                <mxCell id="5" value="KundenID" style="ellipse;whiteSpace=wrap;fontStyle=4;" parent
                    ="1" vertex="1">
                    <mxGeometry x="98" y="194" width="100" height="40" as="geometry"/>
                </mxCell>
                <mxCell id="6" value="BestellID" style="ellipse;whiteSpace=wrap;fontStyle=4;" parent
                    ="1" vertex="1">
```

```
14                      <mxGeometry x="513" y="137" width="100" height="40" as="geometry"/>
15                  </mxCell>
16                  <mxCell id="8" value="" style="endArrow=none;" parent="1" target="2" edge="1">
17                      <mxGeometry relative="1" as="geometry">
18                         </mxGeometry>
19                  </mxCell>
20                  <mxCell id="10" value="" style="endArrow=none;" parent="1" source="6" edge="1">
21                      <mxGeometry relative="1" as="geometry">
22                          </mxGeometry>
23                  </mxCell>
24                  <mxCell id="12" value="aufgeben" style="shape=rhombus;perimeter=rhombusPerimeter;
                        whiteSpace=wrap" parent="1" vertex="1">
25                      <mxGeometry x="386" y="225.5" width="78" height="48" as="geometry"/>
26                  </mxCell>
27                  <mxCell id="13" value="1" style="endArrow=none;" parent="1" source="2" target="12"
                        edge="1">
28                      <mxGeometry x="-0.6098" y="6" relative="1" as="geometry">
29                         </mxGeometry>
30                  </mxCell>
31                  <mxCell id="14" value="N" style="endArrow=none;" parent="1" source="12" target="3"
                        edge="1">
32                      <mxGeometry x="0.5333" y="7" relative="1" as="geometry">
33              ...
34
35  <root>
```

Listing 4: XML-Darstellung eines Diagramms für Kardinalitäten und Beziehungen.

Erläuterung der einzelnen Code-Elemente:

(<mxCell id="12") Dies stellt die Beziehung „aufgeben“ dar, die Kunde und Bestellung miteinander verbindet. Sie wurd durch Anweisung (shape=rhombus) als Raute dargestellt

(<mxCell id="13") und (<mxCell id="14") haben (value="1") (value="N") diese zeigen die Kardinalität der Beziehung an.

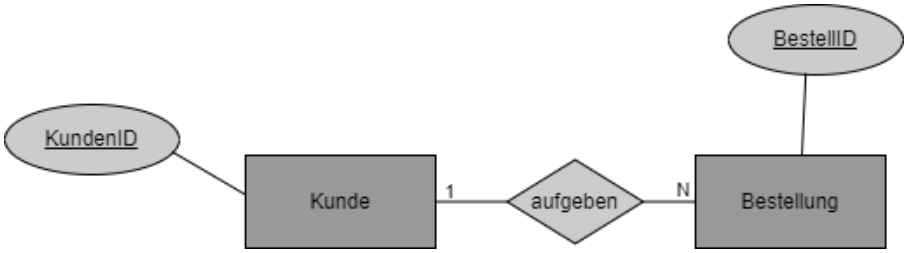


Abbildung 18: Ausgabe Kardinalitäten und Beziehungen in ERD

Eine 1:N-Beziehung entsteht, wenn eine Entität *(Kunde)* mit vielen anderen Entitäten *(Bestellung)* verknüpft ist. Dies ist erkennbar, wenn der Fremdschlüssel *KundenID* in der Tabelle *Bestellung* eindeutig auf eine einzelne Instanz der Entität *Kunde* verweist, während die Tabelle *Bestellung* mehrere Einträge mit demselben Wert in *KundenID* enthalten kann.

Eine M:N-Beziehung hingegen entsteht, wenn eine Tabelle Fremdschlüssel enthält, die auf mehrere andere Entitäten verweisen und so Verbindungen zwischen diesen herstellen. Solche Beziehungen werden in relationalen Modellen oft durch Zwischentabellen realisiert. Zum Beispiel wird eine Zwischentabelle Kunde-Produkt mit den Spalten *KundenID* und *ProduktID*, die Beziehungen zwischen den Tabellen *Kunde* und *Produkt* abbildet, im ERM als eine M:N-Beziehung zwischen den Entitäten *Kunde* und *Produkt* dargestellt. Hier ist er-

kennbar, dass jede Instanz in *Kunde-Produkt* mehrere Werte sowohl für *KundenID* als auch *ProduktID* enthalten kann, was auf die Existenz einer M:N-Beziehung hinweist.

4.2.2 Spezielle Aspekte

Spezielle Aspekte bei der Abbildung relationaler Modelle auf ERMs umfassen verschiedene besondere Fälle, die bei der Modellierung von Datenstrukturen berücksichtigt werden müssen. Dieser Aspekte sind Ist-Beziehung, optionale Attribute, n-äre Relationen und schwache Entitäten.

- **Ist-Beziehung**:

  Eine Ist-Beziehung beschreibt Generalisierungs- und Spezialisierungshierarchien zwischen Entitäten. Sie tritt auf, wenn ein Entitätstyp eine spezifischere Version einer allgemeineren Entität darstellt. Im ERM wird die allgemeinere Entität als Elternentität dargestellt, während die spezifischeren Entitäten als Kindentitäten abgebildet werden. Zum Beispiel kann die Entität „Fahrzeug“ spezialisierte Unterentitäten wie „Auto“ und „Motorrad“ haben. Diese Spezialisierung wird durch eine Ist-Beziehung dargestellt, wobei „Auto ist ein Fahrzeug“ und „Motorrad ist ein Fahrzeug“ veranschaulichen, wie solche Hierarchien modelliert werden. Im relationalen Modell werden die allgemeine Entität „Fahrzeug“ und die spezialisierten Entitäten „Auto“ und „Motorrad“ in separaten Tabellen gespeichert. Hierbei wird die Inklusion Abhängigkeit genutzt, um sicherzustellen, dass jede spezifische Entität eine gültige Referenz zur allgemeinen Entität hat. Konkret bedeutet dies:

  1. **Tabellenstruktur:**:

     - Die Tabelle Fahrzeug enthält alle gemeinsamen Attribute, wie z. B. *FahrzeugID*, *Typ*, und *Marke*.
     -Die Tabellen Auto und Motorrad enthalten jeweils spezialisierte Attribute, wie *AnzahlTüren* für Autos oder *Hubraum* für Motorräder.
     -Jede spezifische Tabelle (Auto, Motorrad) enthält zudem einen Fremdschlüssel *FahrzeugID*, der auf die Tabelle Fahrzeug verweist.

  2. **Inklusionsabhängigkeit**:

     Um sicherzustellen, dass jede Instanz in den spezifischen Tabellen (Auto oder Motorrad) auch in der allgemeinen Tabelle (Fahrzeug) existiert, wird die Inklusionsabhängigkeit geprüft. Dies wird durch die Definition des Fremdschlüssels realisiert: Der Wert in der Spalte *FahrzeugID* der spezifischen Tabellen muss in der Tabelle Fahrzeug vorhanden sein.

  3. **Überprüfung durch relationales Modell**:

     -In einer relationalen Datenbank wird die Inklusionsabhängigkeit durch referenzielle Integrität garantiert. Die Fremdschlüssel-Beziehung zwischen *FahrzeugID* in Auto bzw. Motorrad und *FahrzeugID* in Fahrzeug stellt sicher, dass keine spezifische Entität ohne eine zugehörige allgemeine Entität existieren kann.

Das Beispiel wird wie folgt als logisches Modell geschrieben:

Fahrzeug {FahrzeugID, {FahrzeugID}}
Auto {FahrzeugID, {FahrzeugID}}
Motorrad {FahrzeugID,{FahrzeugID}}

Das Beispiel wurde in XML geschrieben:

```
<root>
    <mxCell id="0"/>
    <mxCell id="1" parent="0"/>
    <mxCell id="2" value="Fahrzeug" style="rounded=0;whiteSpace=wrap;" parent="1"
        vertex="1">
        <mxGeometry x="305" y="80" width="120" height="80" as="geometry"/>
    </mxCell>
    <mxCell id="14" style="endArrow=classic" parent="1" source="3" target="2" edge
        ="1">
        <mxGeometry relative="1" as="geometry"/>
    </mxCell>
    <mxCell id="3" value="Auto" style="rounded=0;" parent="1" vertex="1">
        <mxGeometry x="420" y="240" width="100" height="60" as="geometry"/>
    </mxCell>
    <mxCell id="18" style="endArrow=classic" parent="1" source="17" target="2"
        edge="1">
        <mxGeometry relative="1" as="geometry"/>
    </mxCell>
    <mxCell id="17" value="Motorrad" style="rounded=0;whiteSpace=wrap;" parent="1"
         vertex="1">
        <mxGeometry x="190" y="240" width="100" height="60" as="geometry"/>
    </mxCell>
    <mxCell id="25" value="FahrzeugID" style="ellipse;whiteSpace=wrap;" parent="1"
         vertex="1">
        <mxGeometry x="490" y="90" width="100" height="40" as="geometry"/>
    </mxCell>
    <mxCell id="28" value="" style="endArrow=none;" parent="1" target="25" edge
        ="1">
        <mxGeometry x="0.0401" y="7" relative="1" as="geometry">
            <mxPoint as="offset"/>
            ....
    <root>
```

Listing 5: XML-Darstellung eines Diagramms für Ist-Beziehung.

Die Ist-Beziehung zwischen der allgemeinen Entität Fahrzeug und ihren Unterklassen (Auto und Motorrad) wird durch Kanten (Edges) realisiert. Diese Kanten zeigen, dass Auto und Motorrad von Fahrzeug abgeleitet sind.
(<mxCell id="14") hat die Verbindung zwischen Auto und Fahrzeug und hat den Form durch Anweisung (`style="endArrow=classic"`):

- style= (endArrow=classic;): Fügt eine klassische Pfeilspitze am Ende der Linie hinzu, die auf das Ziel (target) zeigt.

Gleichzeitig wird die Verbindung zwischen Fahrzeug und Motorrad durch (<mxCell id="18") definiert. Hier verbindet die Kante (id="18") die spezifische Entität Motorrad (id="17") mit der allgemeinen Entität Fahrzeug (id="2"). Sie hat die Form durch die Anweisung (`style="endArrow=classic"`) erhalten.

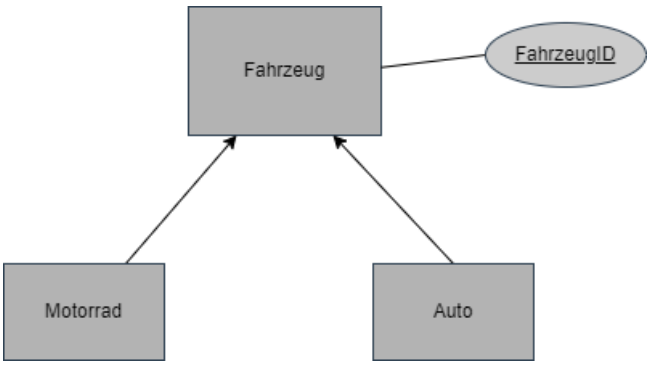


Abbildung 19: Ausgabe von Ist-Beziehung in ERD

- **Optionale Attribute**:

  Ein Attribut ist optional, wenn es nicht zwingend für jede Instanz einer Entität gefüllt sein muss. Im ER-Diagramm wird ein optionales Attribut durch ein Kreis-Symbol markiert, durch einen gestrichelten Ovalrahmen dargestellt, oder mit einer Annotation versehen, die darauf hinweist, dass es nicht obligatorisch ist. Zum Beispiel kann die Entität „Kunde“ das Attribut „Telefonnummer“ haben, das jedoch nicht für jeden Kunden ausgefüllt sein muss. Im relationalen Modell wird ein optionales Attribut durch einen NULL-Wert in der entsprechenden Spalte dargestellt. Alternativ kann das Attribut in eine separate Tabelle ausgelagert werden, um NULL-Werte zu vermeiden. Um zu überprüfen, ob ein Attribut optional ist, sollte analysiert werden, ob in der entsprechenden Attributspalte der Tabelle mindestens ein NULL-Wert vorhanden ist. Wenn dies der Fall ist, kann das Attribut als optional gekennzeichnet werden. Ist dies nicht der Fall, wird das Attribut als obligatorisch angesehen.

  Das Beispiel wird wie folgt als logisches Modell geschrieben:

  Kunde {KundenID, Telefonnummer, {KundenID}}

  Das Beispiel wurde in XML geschrieben:

```
            <root>
                <mxCell id="0"/>
                <mxCell id="1" parent="0"/>
                <mxCell id="2" value="Kunde" style="whiteSpace=wrap;" parent="1" vertex="1">
                    <mxGeometry x="300" y="200" width="120" height="60" as="geometry"/>
                </mxCell>
                <mxCell id="3" value="KundenID" style="ellipse;whiteSpace=wrap;" parent="1"
                    vertex="1">
                    <mxGeometry x="120" y="200" width="100" height="40" as="geometry"/>
                </mxCell>
                <mxCell id="4" style="endArrow=none;" parent="1" source="3" target="2" edge
                    ="1">
                    <mxGeometry relative="1" as="geometry"/>
                </mxCell>
                <mxCell id="5" value="Telefonnummer" style="ellipse;whiteSpace=wrap;" parent
                    ="1" vertex="1">
                    <mxGeometry x="521" y="205" width="100" height="40" as="geometry"/>
                </mxCell>
                <mxCell id="6" style="endArrow=none;" parent="1" source="2" target="5" edge
                    ="1">
                    <mxGeometry relative="1" as="geometry">
                    </mxGeometry>
```

```
19                </mxCell>
20                <mxCell id="7" style="ellipse;whiteSpace=wrap;" parent="1" vertex="1">
21                    <mxGeometry x="460" y="220" width="15" height="15" as="geometry"/>
22                  ......
23             <root>
```

Listing 6: XML-Darstellung eines Diagramms für Optionale Attribute.

Das optionales Attribut „Telefonnummer“ wird in (id="5") definiert. Es wird als Ellipse dargestellt (`style="ellipse;whiteSpace=wrap;"`). Die Kante (id="6") verbindet die Entität Kunde mit dem Attribut Telefonnummer. Die kleine Ellipse (id="7") in Zeile ("20") symbolisiert die Option und wird mit der Hauptellipse des Attributs verbunden.

Abbildung 20: Ausgabe von optionale Attribute in ERD

- **N-äre Relationen**:

  N-äre Relationen stellen eine Besonderheit dar, da sie Beziehungen zwischen mehr als zwei Entitäten modellieren. Sie erlauben die Abbildung komplexerer Abhängigkeiten zwischen mehreren Entitäten. Im ERM wird eine N-äre Beziehung durch ein Raute-Symbol dargestellt, das mit mehreren Verbindungen zu den beteiligten Entitäten verknüpft ist. Zum Beispiel beschreibt die Beziehung „Projektzuordnung“ zwischen den Entitäten „Mitarbeiter“, „Projekt“ und „Rolle“ eine N-äre Beziehung, da drei Entitäten beteiligt sind. Binäre Relationen hingegen, bei denen nur zwei Entitäten miteinander in Beziehung stehen, sind ein Spezialfall der N-ären Relationen mit N=2.
  Im relationalen Modell werden N-äre Beziehungen durch eine zusätzliche Beziehungstabelle realisiert. Diese Tabelle enthält Fremdschlüssel zu allen beteiligten Entitäten und kann auch zusätzliche Attribute enthalten, die spezifische Details zur Beziehung beschreiben. Für binäre Relationen entspricht diese Tabelle einer regulären Beziehungstabelle zwischen zwei Entitäten.

  Das Beispiel wird wie folgt als logisches Modell geschrieben:

  Mitarbeiter {MitarbeiterID, Name, Position, {MitarbeiterID}}
  Projekt {ProjektID, Projektname, Startdatum, Enddatum, {ProjektID}}
  Rolle {RollenID, Rollenname, Beschreibung, {RollenID}}
  Projektzuordnung {MitarbeiterID, ProjektID, RollenID, Zuordnungsdatum, Status, {MitarbeiterID, ProjektID, RollenID }}

  Das Beispiel wurde in XML geschrieben:

```
1             <root>
2                 <mxCell id="0"/>
3                 <mxCell id="1" parent="0"/>
4                 <mxCell id="2" value="Projektzuordnung" style="shape=rhombus;perimeter=
                      rhombusPerimeter;whiteSpace=wrap;" parent="1" vertex="1">
5                     <mxGeometry x="340" y="270" width="120" height="60" as="geometry"/>
6                 </mxCell>
```

```
7                 <mxCell id="3" value="Mitarbeiter" style="whiteSpace=wrap;" parent="1" vertex
                      ="1">
8                     <mxGeometry x="150" y="280" width="100" height="40" as="geometry"/>
9                 </mxCell>
10                <mxCell id="4" value="Projekt" style="whiteSpace=wrap;" parent="1" vertex="1">
11                    <mxGeometry x="350" y="150" width="100" height="40" as="geometry"/>
12                </mxCell>
13                <mxCell id="5" value="Rolle" style="whiteSpace=wrap;" parent="1" vertex="1">
14                    <mxGeometry x="600" y="280" width="100" height="40" as="geometry"/>
15                </mxCell>
16                <mxCell id="22" value="" style="endArrow=none;" parent="1" edge="1">
17                    <mxGeometry relative="1" as="geometry">
18                       </mxGeometry>
19                </mxCell>
20                <mxCell id="23" value="" style="endArrow=none;" parent="1" source="3" edge
                      ="1">
21                    <mxGeometry relative="1" as="geometry">
22                        </mxGeometry>
23                </mxCell>
24                <mxCell id="27" value="" style="endArrow=none;" parent="1" source="4" target
                      ="2" edge="1">
25                    <mxGeometry relative="1" as="geometry">
26                       </mxGeometry>
27                </mxCell>
28                <mxCell id="34" value="Status" style="ellipse;whiteSpace=wrap;" parent="1"
                      vertex="1">
29                    <mxGeometry x="414" y="380" width="66" height="40" as="geometry"/>
30                </mxCell>
31                <mxCell id="36" value="" style="endArrow=none;" parent="1" source="34" edge
                      ="1">
32                    <mxGeometry relative="1" as="geometry">
33                  ....
34              <root>
```

Listing 7: XML-Darstellung eines Diagramms für N-äre Reltionen.

Das bereitgestellte XML beschreibt eine N-äre Beziehung mit der Hauptbeziehung Projektzuordnung. Diese Beziehung verknüpft drei Entitäten: Mitarbeiter, Projekt, und Rolle.
Die Beziehung Projektzuordnung, die durch (<mxCell id="2") definiert hat, ist die zentrale Verbindung, die die Entitäten Mitarbeiter, Projekt, und Rolle miteinander verknüpft und wird als Raute dargestellt (`shape=rhombus; perimeter=rhombusPerimeter`).
Die Zeile ('16 -> 27') enthält Anweisungen, die alle Entitäten mit der Hauptbeziehung verbinden.

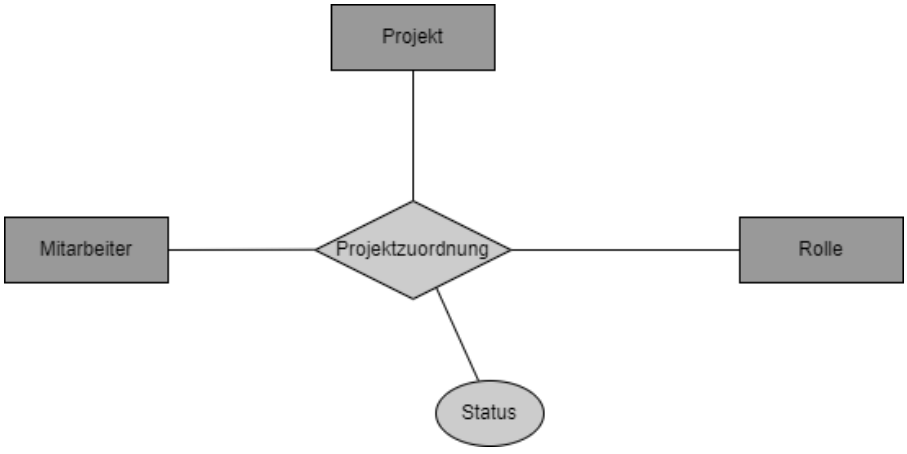


Abbildung 21: Ausgabe von N-äre Relationen in ERD

- **Schwache Entität**:

  Eine schwache Entität ist eine Entität, die ohne eine andere Entität nicht existieren kann. Im Entity-Relationship-Diagramm wird eine schwache Entität durch ein Rechteck mit einer doppelten Umrandung dargestellt. Der zugehörige Primärschlüssel wird in diesem Fall durch eine Kombination aus dem Fremdschlüssel der übergeordneten Entität und einem partiellen Schlüssel der schwachen Entität gebildet. Zusätzlich wird die Beziehung zwischen der schwachen Entität und der übergeordneten Entität durch eine doppelt umrandete Raute dargestellt, die ihre Abhängigkeit symbolisiert. Zum Beispiel kann eine schwache Entität „Adresse“ und ihre Beziehung zu einer übergeordneten Entität „Kunde“ wie folgt dargestellt werden:
  Die Entität „Kunde“ hat ein eindeutiges Attribut, wie etwa „KundenID“, das als Primärschlüssel dient. Gleichzeitig existiert die schwache Entität „Adresse“, die Attribute wie „AdresseID“ (partieller Schlüssel) und „Straße“ enthält. Da die Entität „Adresse“ ohne die Existenz von „Kunde“ nicht existieren kann, enthält sie zusätzlich die „KundenID“ als Fremdschlüssel. Die Beziehung „hat“, die „Kunde“ mit „Adresse“ verbindet, wird im ERD durch eine doppelt umrandete Raute dargestellt. Diese zeigt klar an, dass „Adresse“ vollständig von „Kunde“ abhängt.

  Das Beispiel wird wie folgt als logisches Modell geschrieben:
  Kunde {KundenID, {KundenID}}
  Adresse {AdresseID, KundenID, {AdresseID, KundenID}}

  Das Beispiel wurde in XML geschrieben:

```
1         <root>
2                 <mxCell id="0"/>
3                 <mxCell id="1" parent="0"/>
4                 <mxCell id="2" value="Kunde" style="rounded=0;whiteSpace=wrap;" parent="1"
                      vertex="1">
5                     <mxGeometry x="329" y="145" width="100" height="40" as="geometry"/>
6                 </mxCell>
7                 <mxCell id="3" value="KundenID" style="ellipse;whiteSpace=wrap;" parent="1"
                      vertex="1">
8                     <mxGeometry x="177" y="120" width="100" height="40" as="geometry"/>
9                 </mxCell>
10                <mxCell id="4" value="" style="endArrow=none;" parent="1" source="3" edge="1">
11                    <mxGeometry relative="1" as="geometry">
12                      </mxGeometry>
13                </mxCell>
14                <mxCell id="7" value="hat" style="shape=rhombus;double=1;perimeter=
                      rhombusPerimeter;whiteSpace=wrap;" parent="1" vertex="1">
15                    <mxGeometry x="478" y="143" width="104" height="44" as="geometry"/>
16                </mxCell>
17                <mxCell id="8" value="" style="endArrow=none;" parent="1" source="7" target
                      ="2" edge="1">
18                    <mxGeometry relative="1" as="geometry">
19                      </mxGeometry>
20                </mxCell>
21                <mxCell id="9" value="Adresse" style="shape=ext;margin=3;double=1;whiteSpace=
                      wrap;" parent="1" vertex="1">
22                    <mxGeometry x="644" y="142" width="102" height="46" as="geometry"/>
23                </mxCell>
24                <mxCell id="10" value="" style="endArrow=none;" parent="1" source="9" target
                      ="7" edge="1">
25                    <mxGeometry relative="1" as="geometry">
26                         </mxGeometry>
27                </mxCell>
28                <mxCell id="13" value="&lt;span style=&quot;border-bottom: 1px dotted&quot;&gt
```

```
                    ;AdresseID&lt;/span&gt;" style="ellipse;whiteSpace=wrap;" parent="1"
                    vertex="1">
29                  <mxGeometry x="669" y="224" width="100" height="40" as="geometry"/>
30              </mxCell>
31              <mxCell id="15" value="" style="endArrow=none;" parent="1" source="13" edge
                    ="1">
32                  <mxGeometry relative="1" as="geometry">
33                  ....
34                  .....
35          <root>
```

Listing 8: XML-Darstellung eines Diagramms für schwache Entitätstypen.

Das XML beschreibt eine schwache Entität, die vollständig von einer übergeordneten Entität abhängig ist. Die Entität Adresse ist eine schwache Entität, die durch (<mxCell id="9") definiert und wurde als Rechteck mit einer doppelten Umrandung dargestellt (`shape=ext;margin=3;double=1`).

- shape=ext: Definiert die Form als Rechteck (extended shape), das typischerweise für schwache Entitäten verwendet wird.
- margin=3: Fügt einen Innenabstand von 3 Pixeln zwischen dem Inhalt und der Kante der Form hinzu.
- double=1: Aktiviert eine doppelte Umrandung, die für schwache Entitäten und abhängige Beziehungen verwendet wird, um die Abhängigkeit hervorzuheben.

Die Beziehung (hat), die durch (<mxCell id="7") definiert, verbindet die starke Entität Kunde mit der schwachen Entität Adresse. Sie wird als doppelt umrandete Raute dargestellt (`shape=rhombus;double=1`), um die Abhängigkeit der schwachen Entität zu signalisieren.

- shape=rhombus: Definiert die Form als Raute (Rhombus). Diese wird in ER-Diagrammen verwendet, um Beziehungen darzustellen.

Das Attribut (AdresseID) ist partieller Schlüssel und wurde durch (<mxCell id="13") definiert. Der Text (AdresseID) wird mit einer gestrichelten Unterstreichung durch Anweisung (`style="border-bottom: 1px dotted"`) dargestellt, was oft verwendet wird, um partielle Schlüssel oder schwache Attribute darzustellen.

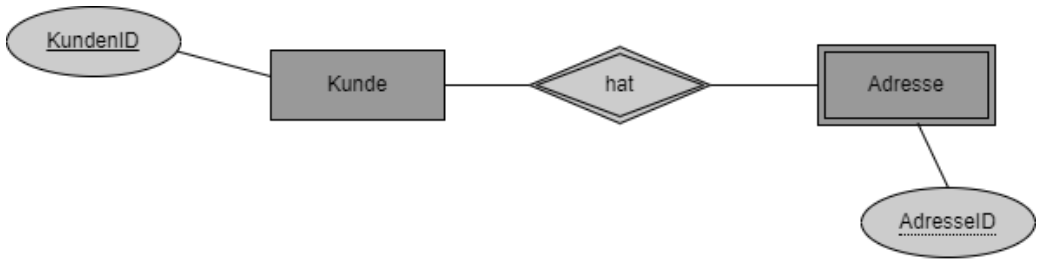


Abbildung 22: Ausgabe eines schwachen Entitätstypen in ERD

5 Implementierung des Konzepts

Dieses Kapitel widmet sich der praktischen Umsetzung des zuvor entwickelten Konzepts. Es konzentriert sich auf die Implementierung technischer und konzeptioneller Ansätze, um die geplanten Ziele zu realisieren. Ziel ist es, die in der Konzeption entwickelten Modelle und Ansätze in die Praxis zu überführen und deren Funktionalität in einem realen Anwendungskontext zu demonstrieren.

Zunächst wird in Abschnitt 5.1 ein Überblick über die verwendeten Technologien gegeben, die bei der Implementierung des Konzepts eine zentrale Rolle spielen. Diese Technologien bilden die Grundlage für die Entwicklung und stellen sicher, dass die gewählten Ansätze technisch umsetzbar und effizient sind.

Während in Abschnitt 5.2 die Codierung der ERMs beschrieben. In diesem Schritt werden die zuvor erstellten Modelle in eine ausführbare Form überführt, wobei sowohl die Datenstrukturen als auch die Beziehungen zwischen den Entitäten realisiert werden. Ziel ist es, eine funktionierende Datenbankstruktur zu schaffen, die den Anforderungen des Projekts entspricht.

5.1 Überblick über die verwendeten Technologien

Die Umsetzung des Konzeptes basiert auf Python in der Version 3.13 und wurde speziell für die automatische Erstellung von ER-Diagrammen in einer Draw.io-kompatiblen XML-Struktur entwickelt. Die Technologien und Ansätze wurden ausgewählt, um eine flexible und erweiterbare Lösung zu ermöglichen.

Verwendete Programmiersprache und Bibliotheken:

Für die Erstellung des ER-Diagramms wurde Python [4] als Programmiersprache gewählt. Python zeichnet sich insbesondere durch seine Einfachheit und Lesbarkeit aus, was eine schnelle und intuitive Umsetzung komplexer Prozesse ermöglicht. Dank der umfangreichen Standardbibliothek können Aufgaben wie die Dateiverarbeitung, die XML-Manipulation sowie die Generierung eindeutiger Identifikatoren direkt und ohne zusätzlichen Aufwand realisiert werden. Ein weiterer wesentlicher Vorteil von Python ist seine Plattformunabhängigkeit: Der gleiche Code kann auf verschiedenen Betriebssystemen wie Windows, Linux oder macOS ausgeführt werden. Zudem profitiert man von einer aktiven Community, die kontinuierlich neue Module und Erweiterungen bereitstellt. Die wichtigsten Bibliotheken im Code sind:

[4] `https://docs.python.org/3/whatsnew/3.13.html`. [Zugriff am 02.02.2025].

- os: [5]

Das `os`-Modul stellt Funktionen bereit, die den Zugriff auf Betriebssystemressourcen ermöglichen. In diesem Projekt wird es verwendet, um Datei- und Verzeichnisoperationen durchzuführen – etwa um zu prüfen, ob bestimmte Dateien existieren oder um neue Verzeichnisse anzulegen. Dadurch können notwendige Dateien und Ordner dynamisch erstellt und verwaltet werden, was die Automatisierung und die Robustheit des Programms erhöht.

- uuid: [6]

Das Modul `uuid` in der Version 4 wird eingesetzt, um eindeutige Identifikatoren (IDs) für die verschiedenen Elemente wie Entitäten, Attribute und Kanten zu generieren. Durch diese einzigartigen Bezeichner wird sichergestellt, dass jedes Element innerhalb des XML-Dokuments klar voneinander unterschieden werden kann. Dies ist besonders wichtig, wenn es darum geht, zahlreiche miteinander verknüpfte Elemente konsistent und fehlerfrei zu verwalten.

- xml.etree.ElementTree: [7]

Diese Bibliothek ermöglicht es, XML-Strukturen einfach und übersichtlich zu erstellen und zu bearbeiten. XML (Extensible Markup Language) bietet ein standardisiertes Format zur Darstellung hierarchischer Daten, das sich hervorragend für die Beschreibung von Diagrammelementen eignet. Besonders hervorzuheben ist, dass die durch ET erzeugten XML-Dokumente leicht lesbar und wartbar bleiben, selbst wenn die zugrunde liegende Struktur komplex wird. Diese Vorteile machen `xml.etree.ElementTree` zu einem idealen Werkzeug für die dynamische Generierung von Diagrammen, wie sie auch von Tools wie Draw.io verwendet werden.

5.2 Codierung der ERMs

Im folgenden Abschnitt wird zunächst der grundlegende Aufbau der XML-Struktur erläutert, bevor auf spezifische Aspekte der Verarbeitung von Eingabedateien sowie die Behandlung besonderer Fälle eingegangen wird.

5.2.1 Aufbau der XML-Grundstruktur

Der erste Schritt im Code besteht darin, die Grundstruktur des XML-Dokuments zu erstellen. Mithilfe der Bibliothek `xml.etree.ElementTree` wird ein Element `mxfile` erzeugt, das als Container für das Diagramm dient. Dazu sind folgende hierarchische Elemente enthalten:

- mxfile: Das oberste Element, das Metadaten zum Diagramm enthält (z.B. den Host, hier "draw.io").

[5] `https://docs.python.org/3/library/os.html#os.extsep` [Zugriff am 07.02.2025].
[6] `https://docs.python.org/3/library/uuid.html`. [Zugriff am 07.02.2025].

[7] `https://docs.python.org/3/library/xml.etree.elementtree.html` [Zugriff am 10.02.2025].

- diagram: Das Diagramm selbst wird als untergeordnetes Element von mxfile erstellt. Es kann mehrere Diagramme enthalten.

- mxGraphModel: Dieses Element definiert das Modell, das unter anderem Parameter wie Seitenabmessungen, Skalierung, Raster und Hintergrundfarbe enthält. Hier werden auch Attribute wie `dx, dy, grid` und `pageWidth` gesetzt.

- root: Als Container für die eigentlichen Diagrammelemente (Entitäten, Relationen, Attribute und Kanten) dient das `root`-Element.

```
1  mxfile = ET.Element("mxfile", {"host": "draw.io"})
2  diagram = ET.SubElement(mxfile, "diagram", {"name": "ERD"})
3  mxGraphModel = ET.SubElement(diagram, "mxGraphModel", {
4      "dx": "1000", "dy": "1000", "grid": "1", "gridSize": "10",
5      "guides": "1", "tooltips": "1", "connect": "1", "arrows": "1",
6      "fold": "1", "page": "1", "pageScale": "1", "pageWidth": "1200",
7      "pageHeight": "800", "background": "#ffffff", "math": "0", "shadow": "0"
8  })
9  root = ET.SubElement(mxGraphModel, "root")
10 ...
```

Listing 9: Erstellung einer XML-Datei für draw.io mit einer Diagrammstruktur

Mit dem Befehl `ET.Element("mxfile", {"host": "draw.io"})`, der in Zeile ("1") ist, wird ein XML-Element namens mxfile erstellt. Das dabei gesetzte Attribut `"host": "draw.io"` signalisiert, dass das XML-Dokument für Draw.io bestimmt ist. Anschließend wird mit `ET.SubElement(mxfile, "diagram", {"name": "ERD"})` in Zeile ("2") innerhalb des mxfile-Elements ein weiteres Element namens diagram erzeugt. Das Attribut `"name": "ERD"` gibt dabei an, dass der Name des Diagramms „ERD“ lautet. Diese beiden Zeilen bilden den Grundaufbau des XML-Dokuments, das später mit weiteren Elementen wie Entitäten, Attributen und Kanten ergänzt wird. Für das weitere Unterelement `mxGraphModel` in Zeile ("3") innerhalb des `diagram`-Elements werden die grundlegenden Eigenschaften des Diagramms definiert, wie beispielsweise Offset-Werte (dx und dy), die das zugrunde liegende Koordinatensystem bestimmen. Weitere Einstellungen wie *guides, tooltips, connect, arrows, fold und page* steuern, wie Elemente angeordnet, verbunden und interaktiv gestaltet werden, um eine korrekte und ansprechende Darstellung in Draw.io zu ermöglichen. Zudem definiert das Element die Seitenabmessungen des Diagramms über *pageWidth* und *pageHeight* und legt mit dem *background-Attribut* die *Hintergrundfarbe* fest..

Das `root`-Element wird als untergeordnetes Element des `mxGraphModel` erzeugt. Es dient als Container für alle weiteren Elemente, die später (z.B. Entitäten, Attribute, Verbindungen) dem Diagramm hinzugefügt werden. Dieses Element ist essenziell, um die Hierarchie des Diagramms im XML-Dokument zu strukturieren.

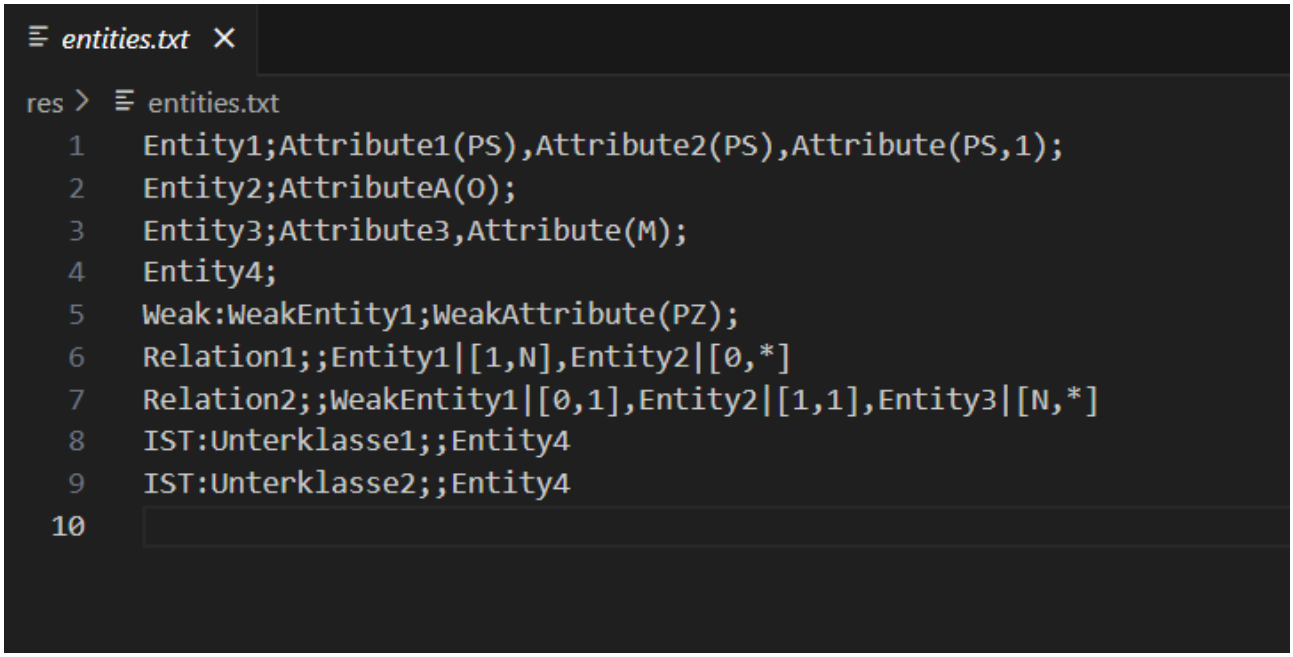


Abbildung 23: Der Eingabedatei

5.2.2 Verarbeitung der Eingabedatei

Zunächst wird im Code geprüft, ob die Eingabedatei (in diesem Fall res/entities.txt) sieh. Abbildung 23 existiert. Falls nicht, wird der Prozess beendet. Anschließend wird jede Zeile der Datei zeilenweise gelesen und in drei Teile zerlegt:

- Name der Entität: Dieser erste Teil kann auch Präfixe enthalten, die den Typ der Entität festlegen. Beispielsweise werden Einträge, die mit `Weak`: beginnen, als schwache Entität und Einträge, die mit `IST`: beginnen, als Vererbungsbeziehung (IST-Beziehung) interpretiert.

- Attribute: Die Attribute werden als kommagetrennte Liste übergeben. Dabei sind Schlüsselattribute durch den Zusatz (PS) als Primärschlüssel gekennzeichnet. Partielle Schlüssel werden mit (PZ) markiert, und optionale Attribute sind mit (O) gekennzeichnet.

- Verbindungen (Connections): Eine kommagetrennte Liste von weiteren Entitäten, zu denen diese Entität in Beziehung steht.

```
1     input_file = "res/entities.txt"
2     if not os.path.exists(input_file):
3         print(f"Datei '{input_file}' nicht gefunden.")
4         return
5  ....
6  ....
7     with open(input_file, "r", encoding="utf-8") as file:
8         for line in file:
9             line = line.strip()
10            if not line:
```

```
11                  continue  # Leere Zeilen überspringen
12              parts = line.split(";")
13              if len(parts) < 1:
14                  print(f"Ungültige Zeile: {line}")
15                  continue
16
17              # Aufteilung der Zeile in Name, Attribute, Verbindungen
18              name = parts[0].strip()
19              attributes = split_attributes(parts[1]) if len(parts) > 1 and parts[1].strip() else []
20              connections = split_connections(parts[2]) if len(parts) > 2 and parts[2].strip() else []
21  ....
```

Listing 10: Einlesen der Eingabedatei

Hier wird der Pfad zur Eingabedatei als String in der Variable `input_file` in Zeile ("1") gespeichert. Die Datei heißt in diesem Fall (entities.txt) und befindet sich im Ordner (res). Mit der Funktion `os.path.exists(input_file)`, die in Zeile ("2") befindet, wird geprüft, ob die angegebene Datei tatsächlich existiert.

Wenn die Datei nicht existiert, wird eine Fehlermeldung ausgegeben, die den Benutzer darauf hinweist, dass die Datei nicht gefunden wurde. Der Befehl `return` beendet daraufhin die Ausführung der aktuellen Funktion (in diesem Fall vermutlich die `main()`–Funktion), sodass der weitere Code nicht ausgeführt wird.

Mit `open(input_file, "r", encoding="utf-8")` (Zeile 7) wird die Datei im Lese-Modus geöffnet, wobei die Kodierung `utf-8` verwendet wird. Der with-Block sorgt dafür, dass die Datei nach Beendigung der Verarbeitung automatisch wieder geschlossen wird.

Entfernt `line.strip()`: in Zeile ("9") überflüssige Leerzeichen und Zeilenumbrüche am Anfang und Ende der Zeile. Dies sorgt dafür, dass die nachfolgenden Operationen sauber mit dem eigentlichen Inhalt arbeiten. Mit der Bedingung `if not line: continue` werden leere Zeilen erkannt und übersprungen, sodass sie nicht weiter verarbeitet werden.

Jede nicht-leere Zeile wird mit `line.split(";")` (Zeile 12) in einzelne Teile zerlegt.

Sollte eigentlich `len(parts) < 1` nie vorkommen, da split(";") immer eine Liste mit mindestens einem Element zurückgibt.

- parts[0] in Zeile ("18"): Enthält den Namen der Entität. Mit `strip()` wird sichergestellt, dass keine überflüssigen Leerzeichen vorhanden sind.
- parts[1] in Zeile ("19"): Falls vorhanden und nicht leer, wird dieser Teil (Attribute) weiter mit `split(",")` in einzelne Attribute aufgespalten. Anschließend werden auch hier die einzelnen Attribute mit `strip()` von Leerzeichen befreit.
- parts[2] in Zeile ("20"): Enthält optional die Verbindungen (Relationsangaben) zu anderen Entitäten. Auch hier wird zunächst geprüft, ob der Teil vorhanden ist und nicht leer, bevor er in einzelne Verbindungsnamen aufgesplittet wird.

5.2.3 Behandlung besonderer Fälle

In ER-Modellen gibt es neben den „normalen“ Entitäten und Relationen einige Sonderfälle, die besondere Notationen oder Platzierungslogiken benötigen.

- **Schwache Entitäten:**
  In meinem Code wird das am Präfix (Weak:) in der Eingabedatei erkannt.

```
1 is_weak = name.startswith("Weak:")
2 if is_weak:
3     name = name.replace("Weak:", "")
```

  Mit `name.startswith("Weak:")` wird überprüft, ob der String in der Variablen name mit dem Präfix ("Weak:") beginnt. Falls das zutrifft, wird ein Flag `is_weak` auf `"True"` gesetzt, was darauf hinweist, dass es sich um eine schwache Entität handeln könnte. Danach wird der Text `"Weak:"` aus dem Namen entfernt, sodass der Name ohne dieses Präfix weiterverwendet werden kann.

- **IST-Beziehung:**
  Es wird durch das Präfix (IST:) verwendet, um eine Vererbung zu kennzeichnen.

```
1 is_ist_beziehung = name.startswith("IST:")
2 if is_ist_beziehung:
3       name = name.replace("IST:", "")
```

  Mit `name.startswith("IST:")` wird überprüft, ob der String in der Variablen name mit dem Präfix ÏST:"beginnt. Falls das zutrifft, wird ein Flag `is_ist_beziehung` auf `"True"` gesetzt, was darauf hinweist, dass es sich um eine Ist-Beziehung handeln könnte. Danach wird der Text `"IST:"` aus dem Namen entfernt, sodass der Name ohne dieses Präfix weiterverwendet werden kann.

- **Optionale Attribute:**
  In mein Skript erkennt (O) im Attributnamen:

```
1 offset_index_opt = offset_counter + len(normal_attrs)
2             for k, opt_attr in enumerate(optional_attrs):
3                 disp_name = opt_attr.replace("(O)", "").strip()
4                 opt_style = "ellipse;whiteSpace=wrap;fillColor=#FFF2CC;strokeColor=#000000;"
5                 attr_id, ax, ay = create_attribute_cell(disp_name, opt_style, offset_index_opt
                      + k)
6       .....
```

  Die Anweisung `offset_index_opt` in erste Zeile bestimmt, wo die optionalen Attribute in der Reihenfolge der Darstellung beginnen. Dazu wird `offset_counter`, der wahrscheinlich die Anzahl der bereits erstellten Attribute oder eine Anfangsposition enthält, mit der Länge der normalen Attribute `(normal_attrs)` addiert.
  Jedes Attribut wird überprüft und das Markierungszeichen (O), das optionale Attribute kennzeichnet, entfernt. Der bereinigte Name wird in `disp_name`, die in Zeile ("3") ist, gespeichert.
  `create_attribute_cell(...)` ist eine Funktion, die ein grafisches Attribut-Element in XML erstellt.

- **Erzeugung der Schlüssel:**
  In unserem Fall erkennt der Code verschiedene Arten von Schlüsseln anhand spezifischer Markierungen in der Eingabedatei (entities.txt) wie zum Beispiel (PS, PZ, usw.). Wenn ein Attribut die Markierung (PS) oder (PS,1) hat, wird es als Primärschlüssel in `pk_attrs` gespeichert. Steht (PZ) dabei, gilt es als partieller Schlüssel und wird unter `pz_attrs` geführt. Ohne eine spezielle Markierung zählt es einfach zu den normalen Attributen `normal_attrs`. Es wurde durch der folgende Code-Abschnitt erzeugt:

```
            pk_attrs = []
            pz_attrs = []
            normal_attrs = []
            for attr in attributes:
                if "(PS" in attr:  # Erfasst sowohl (PS) als auch (PS,   )
                    pk_attrs.append(attr)
                elif "(PZ)" in attr:
                    pz_attrs.append(attr)

                else:
                    normal_attrs.append(attr)
```

Der folgende Code-Abschnitt bezieht sich auf eine Unterscheidung, ob es sich um numerische oder nicht-numerische Primärschlüssel handelt. Falls eine Entität mehrere Primärschlüssel besitzt, wird eine Mittellinie erzeugt, um die (PS)-Attribute miteinander zu verbinden (dies gilt für nicht-numerische Primärschlüssel). Bei numerische Primärschlüssel wird für jedes numerische (PS) eine direkte Verbindung (Edge) von der Entitätszelle zum Attribut erzeugt:

```
            numeric_pk_positions = []      # Liste für numerische PKs
            non_numeric_pk_positions = []  # Liste für andere PKs
            offset_counter = 0
            for pk_attribute in pk_attrs:
                # Prüfe, ob das Attribut den speziellen Marker (PS,   ) enthält
                if "(PS," in pk_attribute:
                    start_index = pk_attribute.find("(PS,")
                    end_index = pk_attribute.find(")", start_index)
                    # Der Inhalt innerhalb der Klammern (z. B . "1") dient als Indikator
                    number_part = pk_attribute[start_index+4:end_index]
                    # Entferne den Marker inklusive Inhalt aus dem Attributnamen
                    disp_name = (pk_attribute[:start_index] + pk_attribute[end_index+1:]).
                        strip()
                    is_numeric = True
                else:
                    if "(PS)" in pk_attribute:
                        disp_name = pk_attribute.replace("(PS)", "").strip()
                    else:
                        disp_name = pk_attribute.strip()
                    try:
                        float(disp_name)
                        is_numeric = True
                    except ValueError:
                        is_numeric = False
        .......
```

Dazu steht die Abkurzung (PZ) für Partielle Schlüssel, die durch eine gestrichelte Unterstreichung (HTML formatiert) erscheinen. Dabei wird eine Liste von Attributen durchlaufen, wobei (PZ) entfernt und Leerzeichen mit `.strip()` in der Zeile 2 bereinigt werden. Die separate Entität ist in der HTML-Div-Struktur

enthalten. Zu den wichtigsten Stilregeln gehört die Eigenschaft in der Zeile 5 `text-decoration: underline;`, die den Text unterstreicht. Außerdem wird `text-decoration-style: dashed;` verwendet, wodurch die Unterstreichung als gestrichelte Linie erscheint.

```
1     for pz_attribute in pz_attrs:
2                 disp_name = pz_attribute.replace("(PZ)", "").strip()
3
4                 pz_value_html = (
5                     f"<div style='text-decoration: underline; text-decoration-style: dashed
                          ;'>"
6                     f"{disp_name}</div>"
7                 )
```

6 Ergebnisse und Diskussion

Das in Kapitel 5 entwickelte Verfahren zur automatischen Erstellung von Entity-Relationship-Modellen aus relationalen Datenbanken hat sich in mehreren Tests bewährt. Die Ergebnisse zeigen, dass es zuverlässig Entitäten, Attribute, Primär- und Fremdschlüssel sowie Beziehungen zwischen Entitäten erkennt und korrekt in ein ER-Diagramm überträgt. Besonders die interaktive Visualisierung erweist sich dabei als großer Vorteil, da sie die Datenbankstruktur übersichtlich darstellt und sowohl die Analyse als auch die Optimierung deutlich erleichtert. Bei der Umsetzung Umsetzung traten mehrere Herausforderungen auf, die wertvolle Einblicke für die Weiterentwicklung des Systems lieferten. Ein spezifisches Problem war die Visualisierung optionaler Attribute in der Chen-Notation, da diese einen zusätzlichen Mittelpunkt benötigen, um eindeutig dargestellt zu werden. Gleiches gilt für Gruppen von Primärschlüsseln, die eine zentrale Anbindung erfordern. Aufgrund dieser Ergebnisse wurde das System entsprechend angepasst, sodass eine normgerechte und exakte Visualisierung dieser Positionen möglich ist. Darüber hinaus wurde das entwickelte Verfahren anhand eines konkreten Anwendungsbeispiels (Hotel-Datenbank) getestet. Dabei zeigte sich erneut, dass alle relevanten Entitäten, Attribute und Beziehungen zuverlässig erkannt und dargestellt wurden. Das ER-Diagramm wurde in wenigen Sekunden erstellt und bestätigte damit die praktische Verwendbarkeit des Systems. Allerdings zeigte sich, dass insbesondere bei komplexeren Datenstrukturen die automatisierte Positionierung der Diagrammelemente noch nicht ideal ist. Um eine bessere Lesbarkeit und Übersichtlichkeit zu erreichen, mussten die Elemente manuell neu angeordnet werden. Durch diese Anpassungen konnte das Verfahren nicht nur die korrekte Visualisierung komplexer Datenstrukturen sicherstellen, sondern auch seine Flexibilität und Anpassungsfähigkeit verbessern. Diese positiven Entwicklungen zeigen, dass das System nicht nur zuverlässig funktioniert, sondern auch erweiterbar ist und sich an spezifische Modellierungsanforderungen angepasst werden kann.

7 Ausblick

Die in dieser Bachelorarbeit behandelte zentrale Forschungsfrage lautet: Wie kann ein Verfahren zur nachträglichen algorithmischen Extraktion und interaktiven Visualisierung von Entity-Relationship-Modellen aus bestehenden relationalen Datenbanken entwickelt und implementiert werden, das die Erkennung von Entitäts- und Beziehungstypen, Attributen sowie Kardinalitäten und Schlüsseln ermöglicht?

Zur Beantwortung dieser Fragestellung wurde ein Verfahren entwickelt und prototypisch implementiert, welches eine interaktive Visualisierung des generierten ER-Modells ermöglicht. Die methodische Vorgehensweise, insbesondere mittels Reverse Engineering und XML-basierter Kodierung, ermöglichte es, Datenbankstrukturen erfolgreich zu analysieren und klar visuell darzustellen. Die Ergebnisse bestätigten die Zuverlässigkeit des entwickelten Verfahrens bei der Identifikation und Darstellung von ERD-Elementen wie Entitäten, Attributen und Beziehungen anhand eines konkreten Anwendungsbeispiels (Hotel-Datenbank). Allerdings besteht Verbesserungsbedarf insbesondere bei der automatischen Erkennung von schwachen Entitäten und optionalen Attributen, ohne dass diese vorher in der Eingabedatei definiert werden müssen. Dadurch werden Analyse und Modellierung vereinfacht und das System kann auch bei nicht vollständig dokumentierten Datenbankstrukturen zuverlässig arbeiten. Ein weiterer Verbesserungspunkt liegt in der automatischen Bestimmung der Kardinalitäten. Aktuell müssen diese manuell in der Eingabedatei angegeben werden. Könnte das System stattdessen Kardinalitäten direkt aus den vorhandenen Datenbeziehungen ableiten, würde dies den Prozess nicht nur effizienter, sondern auch deutlich benutzerfreundlicher machen. Um diese Einschränkungen in Zukunft zu überwinden, könnten weitere Forschung durchgeführt werden, um automatisierte Verfahren zur Erkennung von Entitäten, Attributen und Kardinalitäten zu verbessern. Dabei könnten insbesondere neue algorithmische Ansätze und Heuristiken entwickelt und eingesetzt werden, um die Genauigkeit und Benutzerfreundlichkeit weiter zu erhöhen. Eine solche Weiterentwicklung wäre insbesonders für komplexe und weniger gut dokumentierte Datenbanken sinnvoll. Die vorgeschlagenen Weiterentwicklungen bieten somit einen vielversprechenden Ansatz, um die Erstellung von ER-Modellen zukünftig weiter zu automatisieren und damit einen deutlichen praktischen Mehrwert für Datenbankdokumentation und Verbesserung zu schaffen.

Literatur

[And98] Kelz Andreas. *Relationale Datenbanken*. Zugriff 27-11-2024. 1998. URL: https://www.hdm-stuttgart.de/~riekert/lehre/db-kelz/chap4.htm#Chap4.4.

[Bad04] Antonio Badia. „Entity-Relationship modeling revisited“. In: *ACM SIGMOD Record* 33.1 (März 2004), S. 77–82. DOI: 10.1145/974121.974135.

[BHB88] Dinesh Batra, Jeffrey A. Hofrer und Robert P. Bostrom. „A Comparison of User Performance Between the Relational and the Extended Entity Relationship Models in the Discovery Phase of Database Design“. In: Hrsg. von International Conference on Information Systems. 1988. URL: https://aisel.aisnet.org/icis1988/43/.

[Cag+13] Nergiz Ercil Cagiltay, Gul Tokdemir, Ozkan Kilic und Damla Topalli. „Performing and analyzing non-formal inspections of entity relationship diagram (ERD)“. In: *Journal of Systems and Software* 86.8 (Apr. 2013), S. 2184–2195. DOI: 10.1016/j.jss.2013.03.106.

[Che02] Peter Chen. *Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned*. Jan. 2002, S. 296–310. DOI: 10.1007/978-3-642-59412-0.

[Che76] Peter Pin-Shan Chen. „The entity-relationship model—toward a unified view of data“. In: *ACM Transactions on Database Systems* 1.1 (März 1976), S. 9–36. DOI: 10.1145/320434.320440.

[EN16] Ramez Elmasri und Shamkant B. Navathe. *FUNDAMENTALS OF database systems*. Techn. Ber. 2016. URL: https://www.auhd.edu.ye/upfiles/elibrary/Azal2020-01-22-12-28-11-76901.pdf.

[Eng+92] Gregor Engels, Martin Gogolla, Uwe Hohenstein, Klaus Hülsmann, Perdita Löhr-Richter, Gunter Saake und Hans-Dieter Ehrich. „Conceptual modelling of database applications using an extended ER model“. In: *Data Knowledge Engineering* 9.2 (Dez. 1992), S. 157–204. DOI: 10.1016/0169-023x(92)90008-y.

[Eng02] J.-L. Hainaut; J.-M. Hick; J. Henrard; D. Roland; V. Englebert. „Knowledge transfer in database reverse engineering: a supporting case study“. In: *IEEE* (Aug. 2002), S. 194–203. DOI: 10.1109/wcre.1997.624590.

[Fia13] Claudio Fiandrino. *Drawing ER diagrams with TikZ*. Techn. Ber. Apr. 2013. URL: https://www.guitex.org/home/images/ArsTeXnica/AT015/drawing-ER-diagrams-with-TikZ.pdf.

[Gad19] Andreas Gadatsch. *Erstellung von Fachkonzepten mit dem Entity-Relationship-Modell (ERM)*. Jan. 2019, S. 9–34. DOI: 10.1007/978-3-658-25730-9_2.

[GB21] Philipp-Lorenz Glaser und Dominik Bork. „The bigER Tool - Hybrid Textual and Graphical Modeling of Entity Relationships in VS Code“. In: *2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW)*. 2021, S. 337–340. DOI: 10.1109/EDOCW52865.2021.00066.

[Gre21] Nicko Green. *8 besten kostenlosen ER Diagramm Tools in 2021*. Zugriff 06-12-2024. März 2021. URL: https://gitmind.com/de/er-diagramm-tool.html.

[Hai+09] Jean-Luc Hainaut, Jean Henrard, Vincent Englebert, Didier Roland und Jean-Marc Hick. *Database Reverse engineering*. Jan. 2009, S. 723–728. DOI: 10.1007/978-0-387-39940-9_646.

[Hai02] Jean-Luc Hainaut. *Introduction to Database Reverse Engineering*. Institut d'Informatique - LIBD, Namur, Mai 2002. URL: https://pure.unamur.be/ws/portalfiles/portal/270906/DBRE-2002.pdf.

[Her18] Frank Herrmann. *Konzeptioneller Datenbankentwurf*. Jan. 2018, S. 11–27. DOI: 10.1007/978-3-658-21331-2_3.

[HRT16] Jeffrey A Hoffer, Venkataraman Ramesh und Heikki Topi. *Modern database management*. Pearson, 2016. URL: https://industri.fatek.unpatti.ac.id/wp-content/uploads/2019/03/167-Modern-Database-Management-Jeffrey-A.-Hoffer-V.-Ramesh-Heikki-Topi-Edisi-12-2016.pdf.

[JM20] Carlos Jaimez-González und Jazmín Martínez-Samora. *DiagrammER: A Web Application to Support the Teaching-Learning Process of Database Courses Through the Creation of E-R Diagrams*. Okt. 2020. URL: https://www.learntechlib.org/p/217931/.

[Kam24] Titus Kamunya. *die 8 besten ER-Diagramm-Ersteller zur Visualisierung und Gestaltung von Datenbanken*. Zugriff 06-12-2024. Mai 2024. URL: https://geekflare.com/de/best-er-diagram-makers/.

[Lop+21] Jonnathan Lopes, Maicon Bernardino, Fábio Basso und Elder Rodrigues. „Textual Approach for Designing Database Conceptual Models: A Focus Group". In: *scitepress* (Jan. 2021), S. 171–178. DOI: 10.5220/0010232801710178.

[Man08] Klaus Manhart. *Grundlagenserie Business Intelligence: Business Intelligence (Teil 3): Datenmodellierung – relationale und multidimensionale Modelle*. Zugriff 12-11-2024. Feb. 2008. URL: https://www.computerwoche.de/article/2858311/business-intelligence-teil-3-datenmodellierung-relationale-und-multidimensionale-modelle.html.

[McP07] Keith McPherson. *new online technologies for new literacy instruction - ProQuest*. 2007. URL: https://www.proquest.com/openview/c9cced0d84fda660a6aa2f998e27c73e/1?cbl=38018&pq-origsite=gscholar&parentSessionId=LsiuTo%2FquN%2BDGegMs4M%2FIWL5JDLivdKXlS8HUxuumRk%3D.

[MD97] Alfred Moos und Gerhard Daues. *Datenbank-Engineering*. Jan. 1997. DOI: 10.1007/978-3-322-91986-1.

[MM15] Leonid Mamchenkov und Leonid Mamchenkov. *Using Graphviz dot for ERDs, network diagrams and more - Leonid Mamchenkov*. Zugriff 04-12-2024. Aug. 2015. URL: https://mamchenkov.net/wordpress/2015/08/20/graphviz-dot-erds-network-diagrams/.

[Moh15] Danial Abdulkareem Muhammed Mohammed Anwar Mohammed Jaza Abdullah. „Practical Approaches of Transforming ER Diagram into Tables“. In: *International Journal of Multidisciplinary and Scientific Emerging Research* (2015). URL: https://www.researchgate.net/publication/315380202_Practical_Approaches_of_Transforming_ER_Diagram_into_Tables.

[Par24] Visual Paradigm. *ERD / Database Generation Tool*. Zugriff: 04-03-2025. 2024. URL: https://www.visual-paradigm.com/features/database-engineering-tools/.

[Pet+94] J. -M. Petit, J. Kouloumdjian, J. -F. Boulicaut und F. Toumani. *Using queries to improve database reverse engineering*. Jan. 1994, S. 369–386. DOI: 10.1007/3-540-58786-1_91.

[PF10] Tobias G. Pfeiffer und SplineTalks Freie Universität Berlin. *Graphen visualisieren mit Graphviz*. Techn. Ber. Zugriff 05-12-2024. Apr. 2010. URL: https://spline.de/talks/graphviz.pdf.

[SC09] Il-Yeol Song und Peter P. Chen. *Entity relationship model*. Jan. 2009, S. 1003–1009. DOI: 10.1007/978-0-387-39940-9_148.

[SSH18] Gunter Saake, Kai-Uwe Sattler und Andreas Heuer. *Datenbanken - Konzepte und Sprachen*. 6. Aufl. mitp Verlags GmbH, Apr. 2018. ISBN: 978-3-95845-654-2. URL: https://www.mitp.de/IT-WEB/Datenbanken/Datenbanken-Konzepte-und-Sprachen-oxid.html.

[Sta05] Josef L. Staud. „Entity Relationship-Modellierung“. In: *Datenmodellierung und Datenbankentwurf: Ein Vergleich aktueller Methoden* (2005), S. 129–191.

[Sup24] Microsoft Support. *Erstellen eines Modells anhand einer vorhandenen Datenbank mittels Reverse Engineering*. Letzter Zugriff am 19-11-2024. 2024. URL: https://support.microsoft.com/de-de/topic/erstellen-eines-modells-anhand-einer-vorhandenen-datenbank-mittels-reverse-engineering-fb034862-acfc-45bc-88b2-f33d1e1f8614.

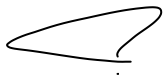
[Tan24] Till Tantau. *Entity-Relationship Diagram Drawing Library*. Zugriff 04.12.2024. 2024. URL: https://tikz.dev/library-er.

[Yud23] Keshav Dev Yudhishthir Singh. *View of Use of Various Information Communication Tools (ICT) in Online Teaching-Learning in Higher Education*. 2023. URL: https://journal.ijarps.org/index.php/IJARPS/article/view/39/34.

[Zha+16] Yanhong Zhao, Hongqi Li, Liping Zhu und Jiandong Wang. „Automatic Building of ER and Data Flow Graph: A Business Process-based Approach“. In: *Revista Técnica de Ingeniería de la Universidad del Zulia* (2016), S. 37–47. DOI: 10.21311/001.39.6.05.

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben. Ich erkläre weiterhin, dass die vorliegende Arbeit in gleicher oder ähnlicher Form noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.
Zur Unterstützung bei der Korrektur und zur Verbesserung der sprachlichen sowie grammatikalischen Qualität dieser Arbeit wurde das KI-gestützte Tool ChatGPT-4o von OpenAI verwendet. Diese Nutzung beschränkte sich ausschließlich auf die Überprüfung der Sprache und Grammatik. Alle inhaltlichen Beiträge und die wissenschaftliche Auseinandersetzung mit dem Thema erfolgten eigenständig und ohne direkte Einflussnahme durch ChatGPT.



Rostock, den 14. März 2025 Amani Assani