Duale Hochschule Baden-Württemberg

Fallstudie

Datenbanken und ETL-Prozesse

Studiengang Wirtschaftsinformatik

Studienrichtung Data-Science

Verfasser(in):	Rico Joel Siegelin, Karsten Kuczera, Olena Lavrikova, Thi Vu, Nils-Jannik Klink, Marvin Spurk
Matrikelnummer:	6577951, 1199837, 5436924, 1039624, 2538158, 1573694
Kurs:	WWI-20-DSB
Studiengang:	Wirtschaftsinformatik Data Science
Bearbeitungszeitraum:	16.11 2021 - 28.01.2022

Ehrenwörtliche Erklärung

Wir versichern hiermit, dass wir unsere Fallstudie selbstständig verfasst und keine anderen als die angegeben Quellen und Hilfsmittel benutzt haben.

Ort, Datum, Unterschrift

Oberzent, 27.02.2022, R. Siegelin
, K. Kuczera
, O. Lavrikova
, T. Vu
, N. Wirth
, M. Spurk

Inhaltverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1 Problemstellung

Datenbanksysteme sind heutzutage bei den Unmengen an Daten nicht mehr wegzudenken, jedoch gibt es das Problem der verteilten Datenhaltung. Daten sind hierbei in unterschiedlichen Datenbanken und Programmen abgelegt ohne eine großartige Struktur. Auch besteht das Problem das die Daten in unterschiedlichen Formaten vorliegen (Excel, JSON, Word, DB). Des Weiteren fehlt ein anständiges und funktionsfähiges ETL-System, um die Daten aus der verteilten Datenhaltung in ein Data Warehouse kommen.

1.2 Vorgehensweise

Zuerst haben wir die Datengeneration gestartet, um Daten für den ETL-Prozess nutzen zu können. Danach haben wir die ein neues ER-Modell entwickelt und erstellt, um eine Relation zwischen den Tabellen in der Datenbank herzustellen. Der ETL-Prozess wir als nächstes entwickelt, um die Datenbank bzw. das Data Warehouse zu befüllen.

2. Theorieteil

2.1 Datenbanken

2.1.1 Definition und Aufgaben

Eine Datenbank ist nach Definition „eine selbstständige und auf Dauer ausgelegte Datenorganisation, welche einen Datenbestand sicher und flexibel verwalten kann.“ (Steiner René, S.5, 2021)

Eine Datenbank hat folgende Aufgaben

- Die Datenbank muss dem Nutzer einen Zugriff auf gespeicherte Daten ermöglichen, ohne dass dieser die Struktur hinter dem Daten und die Organisation im System kennen muss
- Sie muss Zugriffberechtigungen verwalten, sodass ein Nutzer ohne Zugriffsberechtigung die Daten auch nicht manipulieren kann bzw. diese auch nicht Lesen oder Löschen kann. Des Weiteren muss garantiert werden, dass durch eine Fehlmanipulation des Benutzers die Daten bzw. der gesamte Datenbestand unbrauchbar gemacht wird.

- Es muss ermöglicht sein, die interne Struktur der Datenbank ändern zu können ohne, dass der Benutzer seine Applikationen anpassen muss.

2.2.2 Grundsätze

In Datenbanken gibt es gewisse Grundsätze, welche eingehalten werden müssen, um einen marklosen Betrieb zu ermöglichen.

Demnach besitzt eine ideale Datenbank folgende charakteristischen Eigenschaften:

- Gespeicherte Daten müssen eine überschaubare Struktur aufweisen, um nicht mehrfach bzw. redundant gespeichert zu werden.
- Die jeweiligen Applikationen der Nutzer müssen datenunabhängig funktionieren, damit Reorganisationen innerhalb des Datenbanksystems die Anwenderprogramme nicht beeinflussen
- Der Datenbank muss es auch möglich sein, aus bestehenden Daten eine neue Anwendung zu entwickeln. Die Datenbank muss also flexibel sein.
- Die Datenbank muss Datenintegrität gewährleisten d.h. fehlerbehaftete Eingaben von Benutzern müssen gesichert werden und gespeicherte Daten gesichert werden, damit bei technischen bzw. manuellen Fehlern keinerlei Datenverluste auftreten (vgl. (Steiner René, S.6, 2021)

2.2.3 Datenbankmodelle

Grundsätzlich lassen sich Datenbanken in 3 Hauptkategorien unterteilen:

- Hierarchische Datenbanken
- Relationale und objektorientierte Datenbanken
- Objektorientierte Datenbanken

Innerhalb unserer Fallstudie haben wir die relationalen Datenbanken verwendet. Deswegen werden auch in der Dokumentation nur die relationalen Datenbanken erläutert. Bei den relationalen Datenbanken gibt es anders als bei den Objektorientierten Datenbanken eine feste Anzahl an Standarttypen (var, char, int, string). Hierbei werden die Daten nicht hierarchisch, sondern geordnet nach Entitäten (Themenkreisen) in Form von Tabellen abgelegt. Diese Entität besteht dann je nachdem wie viele Attribute diese Entität besitzt aus mehr oder weniger vielen

Attributen. Die Entitäten werden dann mithilfe von Relationen miteinander zu einem Schema verknüpft (ER-Modell).

2.2 Data Warehouse

Data Warehouses dienen schon seit längerem der Entscheidungsfindung bzw. der Unterstützung bei Entscheidungen im Unternehmen. „Sie integrieren Informationen aus verschiedenen meist heterogenen, operativen Informationssystemen, bereiten diese gemäß der Zielsetzung des Data-Warehouses auf und stellen sie Front-End-Systemen zur Darstellung für den Endnutzer zur Verfügung“ (Sascha Qualitz, S.1, o.D.). Die extrahierten Daten werden hierbei zuerst transformiert, um die benötigte Datenqualität zu erhalten. Zu den wichtigen Anforderungen an die Daten für das Data Warehouse gehören also die Detailtiefe, die Datenqualität und deren Struktur. Die Anforderungen an die Datenqualität müssen innerhalb des Extraktion-, Transformation- und Ladeprozesses kurz ELT-Prozess vorliegen (vgl. Sascha Qualitz, S.1, o.D).

2.3 ETL-Prozess

„Ein ETL-Prozess ist ein Vorgang, der aus mehreren unterschiedlichen heterogenen Informationssystemen Daten extrahiert, integriert und in ein Zielsystem lädt“ (Sascha Qualitz, S.1, o.D). Dazu zählt der ETL-Prozess zu dem wichtigsten Teil im Betrieb eines Data Warehouses. Außerdem dient der Prozess dazu Daten aus mehreren Quellen zu integrieren und aufzubereiten. Diese Daten werden dann in den Arbeitsbereich des Data Warehouses transformiert und auf der Basisdatenbank des Data Warehouses gespeichert. Die Datenverarbeitung erfolgt in mehreren Schritten. Zuerst werden Unterschiede in der Darstellung der Daten bereinigt beispielsweise wird das Datenformat oder die Kodierung bereinigt. Danach werden extrahierten Daten validiert, gefiltert und um zusätzliche Informationen/Attribute ergänzt werden. Dabei werden sowohl semantische als auch strukturelle Konflikte, sowie Datenkonflikte aufgelöst.

Dies alles dient dazu ein einheitliches Schema aus unterschiedlichen Quellen mithilfe des ETL-Prozesses zu generieren.

2.4 Scrum

2.4.1 Rollen in Scrum

Scrum Master:

- Managt die Grenzen des Scrum Teams
- Sorgt für die Einhaltung des Scrum-Prozesses
- Implementiert Scrum
- Arbeitet mit dem Dev-Team Produktivitätsverbesserungen aus
- „Führungskraft“ ohne disziplinarische Verantwortung

Product Owner:

- Erstellt die Product Vision (das ideale Produkt)
- Erstellt das Product Backlog
- Stellt die Profitabilität sicher (achtet streng auf Return on Investment)
- Führt die Produktentwicklung und ist für fachliche Fragen des Dev-Teams da

Development Team (Dev-Team):

- Verantwortlich für die Lieferung des Produktes
- Verantwortlich für die Funktionalität und Qualität des Produktes
- Einschätzungsfunktion der Umsetzungsdauer und Möglichkeit der Umsetzung

2.4.2 Bestandteile von Scrum

Scrum ist heutzutage in der agilen Softwareentwicklung Standard. Meistens wird Scrum auf Team- oder Projektebene eingesetzt als agile Managementmethode. In Abbildung 1 zu sehen ist der typische Ablauf von Scrum.

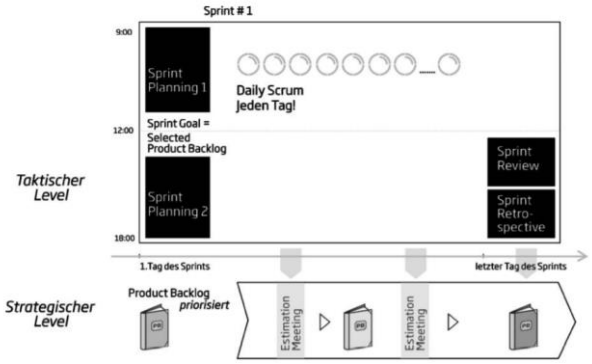

Abbildung 1: Ablauf Scrum

Demnach wird nach Scrum tägliche ein „Daily Scrum“ also ein tägliches Meeting durchgeführt, in dem der aktuelle Stand und die heutigen Tätigkeiten von jedem kurz besprochen werden.

Des Weiteren gibt es ein „Sprint Planing“ am Anfang jedes Sprints, erstellt das Entwicklungsteam eine Analyse der zu liefernden Funktionalitäten/Ziele (vgl. Boris Gloger, S.4, 2010) Ein Sprint ist demnach ein Arbeitsabschnitt ca. 1-4 Wochen, je nach Anforderungen.

Am Ende jedes Sprints gibt es ein „Sprint Review“, indem es um das testen und besprechen der aktuellen Funktionalität geht. Die dabei entstehenden Ideen und Einwürfe werden dann vom Product Owner mit in das Product Backlog mit aufgenommen (vgl. Boris Gloger, S.5, 2010).

In der „Sprint Retroperspektive“ wird mit dem Scrum Master diskutiert und versucht Lösungen zu finden, die zu einer höheren Produktivität führen.

Das „Product Backlog“ wird vom Product Owner basierend auf den Anforderungen, Wünschen des Kunden und aktuellen Markterfordernissen erstellt. Es wird dabei priorisiert erstellt. D.h. die wichtigsten Anforderungen stehen weiter oben und die unwichtigeren weiter unten.

3. Praxisteil

3.1 Anfangszustand (verteilte Datenhaltung)

Wir haben eine extrem verteilte Datenhaltung, keine richtige Struktur und keinen angemessenen ETL System. Entwickeln Sie eine Unternehmensspezifische sinnvolle Datenhaltungslösung mit einem Data-Warehouse.

3.2 Rollenverteilung und User-Story

Rollenverteilung

Die Projektumsetzung wurde das Team mit Scrum durchgeführt, dies wurde am 16.11.2021 festgelegt.

Es wurde sich für folgende Rollenverteilung entschieden:

Scrum Master:	Nils-Jannik
Product Owner:	Rico Siegelin
Dev-Team:	Olena Lavrikova
	Thi Vu
	Karsten Kuczera
	Marvin Spurk

User Story

Als erstes hat sich unser Product Owner mit dem Partner getroffen. Die Firma Faber Bau GmbH ist ein langjähriger Kunde von NO Data, welche Bauprojekte für Dritte plant. Die Projektdaten sollen zukünftig automatisiert verwaltet werden. Dabei haben unser Product Owner Rico Siegelin mit einem Mitglied des Dev-Teams Thi Vu den Partner besucht und gemeinsam eine Datenanalyse durchgeführt.

Hierbei wurden folgende Anpassungen mit dem Kunden vereinbart. Die Mitarbeitertabelle soll folgende Attribute beinhalten: Name, Anschrift, Telefonnummer,

Geschlecht und das Einstellungsdatum. Jeder Mitarbeiter bekommt eine eindeutige vierstellige ID.

Des Weiteren ist die Faber Bau GmbH in Abteilungen gegliedert. Jede Abteilung hat eine Bezeichnung und eine eindeutige Nummer. Als Abteilungsleiter ist jeweils ein Mitarbeiter zugeordnet, der über die Verwendung des Abteilungsetats entscheidet. Jeder Mitarbeiter gehört zu einer Abteilung.

Da im Unternehmen an verschiedenen Projekten gearbeitet wird, enthalten alle Projekte eine eindeutige Nummer und eine eindeutige Bezeichnung, welche sich besser merken und finden lässt. Darüber hinaus werden für jedes Projekt Auftragswert, gezahlter Betrag, Datum von Projektbeginn und -ende, die Feststellung, ob das Projekt storniert wurde, sowie der Projektleiter erfasst.

Jeder Mitarbeiter kann einem Projekt zugeordnet werden, an dem er arbeitet. Da es auch kleinere Projekte gibt, kann ein Mitarbeiter auch in mehreren Projekten tätig sein. Es kommt auch vor, dass ein Mitarbeiter mehrere Projekte leitet. Jedem Projekt ist mindestens ein Mitarbeiter zugeordnet, dies kann der Projektleiter sein. Um den Aufwand für ein Projekt besser bemessen zu können, wird für jeden Mitarbeiter der Arbeitszeitanteil für dieses Projekt festgelegt. Der Arbeitszeitanteil beträgt min. 1- bis max. 40-Arbeitsstunden pro Woche.

Mithilfe dieser Informationen soll ein Datenbankmodell seitens NO Data erstellt werden.

Folgende Informationen sollen ersichtlich werden:

- Welcher Mitarbeiter arbeitet in welcher Abteilung?
- Welcher Mitarbeiter leitet eine Abteilung?
- Wer arbeitet an welchen Projekten mit wie vielen Stunden?

3.3 Umsetzung nach Scrum

Danach hat der Product Owner die Anforderungen vom Kunden für das Scrum Team als User Stories auf einem Product-Backlog erfasst.

Der Scrum Master hat die Scrum-Meetings organisiert. Im Team wurde alle zwei Wochen ein Sprint Planning veranschlagt; Daily Scrum fand zweimal die Woche statt und Review und Retro folgten am Ende des Sprints.

Während des Projektes haben wir ein Projektverwaltungstool namens Kanban angewendet. Damit wurden die einzelnen Prozessabläufe geplant und festgehalten. Jede Story wurde auf dem Backlog des Projektes von oben nach unten unter der Berücksichtigung der Priorität im Kanban-Board hingelegt. Dieses beinhaltet detaillierte Informationen der zu erledigenden Aufgaben sowie den Termin der Abgabe.

Unser Projekt hat mit einem Sprint-Planning gestartet, an dem das gesamte Dev-Team, der Product Owner und Scrum Master teilgenommen haben. Dabei wurde die Aufgaben für das Projekt vorgestellt. Mithilfe von Product Backlogs hat unser Product Owner Rico Siegelin die Aufgaben für den kommenden Sprint vorgestellt. Dabei ging er auf die Priorität der Aufgabe sowie die genaue Erläuterung dieser ein. Danach wurden die Aufgaben in einer User Story festgehalten. Anschließend wurde geschätzt, welchen Aufwand das Team zur Fertigstellung der vorgestellten Aufgaben erwartet.

Die Abbildung stellt die Übersicht des Product-Backlogs mit allen Aufgaben dar, welche wir als User Stories in Kanban definiert haben.

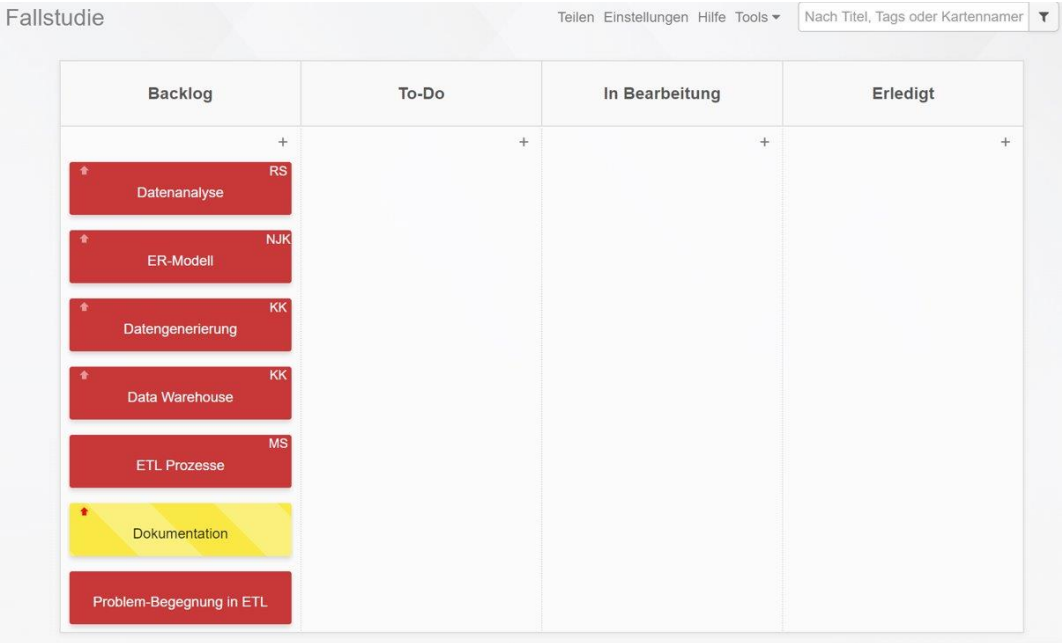

Abbildung 2: Product-Backlog

In der Abbildung ist die erste Story ‚Datenanalyse' aus dem Product Backlog dargestellt. Diese beinhaltet eine detaillierte Beschreibung sowie Akzeptanzkriterien. Darüber hinaus wurden Kartentyp, Priorität, Termin und die Zuordnung zu einem Mitglied bestimmt.

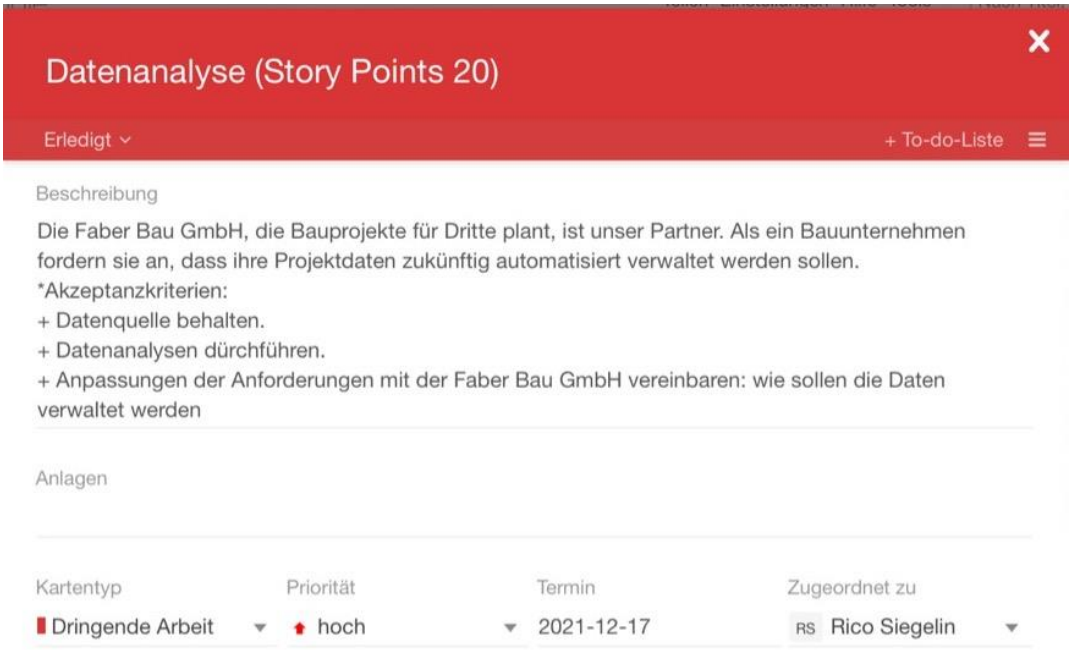

Abbildung 3: User Story (Kanban)

Sobald die Anforderungen klar definiert wurden, schätzen die Mitglieder des Dev-Teams die User Story mithilfe des Planning Poker (siehe die dazugehörige Abbildung). Unser Team hat den Aufwand dieser Aufgabe auf 20 Story Points geschätzt. Die Vorgehensweise für die nächsten User Storys werden nach demselben Prinzip durchgeführt.

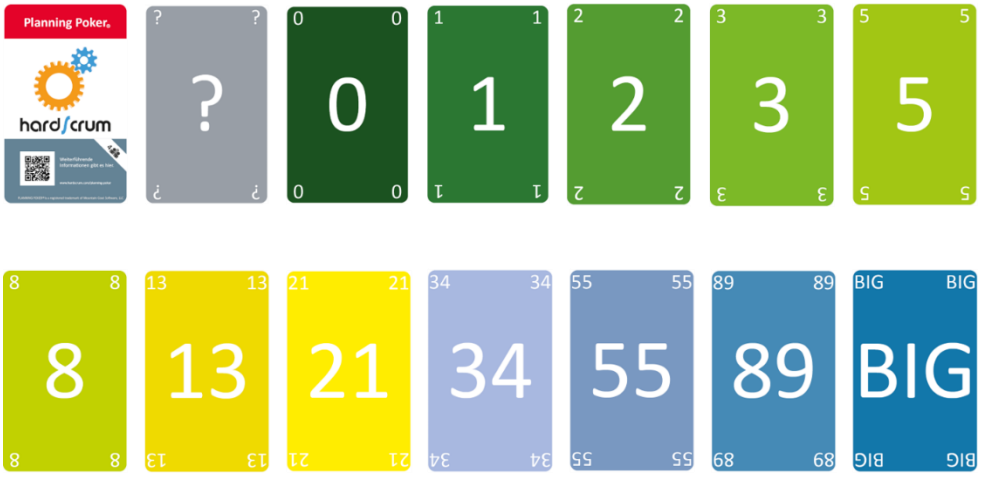

Abbildung 4: Planning Poker

3.4 ER-Modell

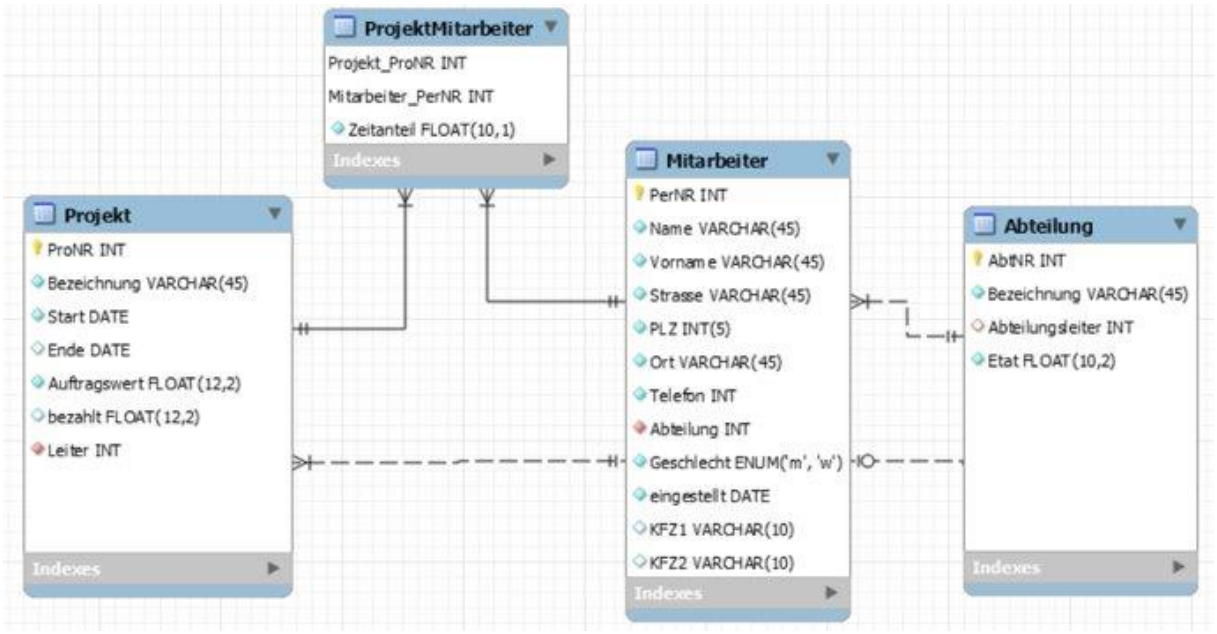

Abbildung 5: ER-Modell

Das Entity-Relationship-Modell wird anhand der User Story erstellt. Hierfür wurde ein Gespräch mit der Faber Bau GmbH geführt, um die Firmenstruktur und somit die Anforderungen an das Entity-Relationship-Modell herauszuarbeiten. Dabei konnte festgestellt werden das folgende Tabellen benötigt werde:

- Projekt

- Mitarbeiter
- Abteilung

Den Tabellen oder auch Entitys genannt, können verschiedene Eigenschaften oder auch Attribute genannt, zugeordnet werden. Folgend werden die verschiedenen Attribute zu den entsprechenden Entitys aufgelistet und beschrieben.

Projekt:

ProNr – Jedes Projekt hat eine eigene individuelle Projektnummer, welches nur aus einer Zahlenfolge besteht und einzigartig ist (INT & Primary key)

Bezeichnung – Jedes Projekt hat eine Bezeichnung, welches dieses mit maximal 45 Buchstaben & Zahlen beschreibt (Varchar (45))

Start – Projektstart als Datumsformat (DATE)

Ende – Projektende als Datumsformat (DATE)

Auftragswert – Projektwert mit maximal 12 Zahlen und 2 Nachkommastellen (Float (12,2))

Bezahlt – Wie viel vom Projektwert schon bezahlt wurde mit maximal 12 Zahlen und 2 Nachkommastellen (Float (12,2))

Leiter – Jedes Projekt hat einen Projektleiter, welcher mit seiner Mitarbeiter-ID in der Tabelle Projekt gespeichert wird (INT)

Abteilung:

AbtNr - Jede Abteilung hat eine eigene individuelle Abteilungsnummer, welche nur aus einer Zahlenfolge besteht und einzigartig ist (INT & Primary key)

Bezeichnung – Jede Abteilung hat eine Bezeichnung, welche diese mit maximal 45 Buchstaben & Zahlen beschreibt (Varchar (45))

Abteilungsleiter – Jede Abteilung hat einen Abteilungsleiter, welcher mit seiner Mitarbeiter-ID in der Tabelle Abteilung gespeichert wird (INT)

Etat – Jede Abteilung hat einen Etat welcher aus maximal 10 Zahlen und 2 Nachkommastellen (Float (12,2))

Mitarbeiter:

PerNr - Jeder Mitarbeiter hat eine eigene individuelle Personalnummer, welche nur aus einer Zahlenfolge besteht und einzigartig ist (INT & Primary key)

Name – Nachname des Mitarbeiters mit maximal 45 Buchstaben & Zahlen beschreibt (Varchar (45))

Vorname – Vorname des Mitarbeiters mit maximal 45 Buchstaben & Zahlen beschreibt (Varchar (45))

Straße – Straße des Mitarbeiters mit maximal 45 Buchstaben & Zahlen beschreibt (Varchar (45))

PLZ – Postleitzahl des Mitarbeiters mit 5 Ziffern (INT (5))

Ort – Wohnort des Mitarbeiters mit maximal 45 Buchstaben & Zahlen beschreibt (Varchar (45))

Telefon – Telefonnummer des Mitarbeiters (INT)

Abteilung – Die Abteilungs-ID des Mitarbeiters (INT)

Geschlecht – Das Geschlecht des Mitarbeiters. Vorauswahl mit m oder w für männlich oder weiblich (ENUM (`m`, `w`)

Eingestellt – Das Einstellungsdatum des Mitarbeiters (DATE)

KFZ1 – Beschreibung des ersten Firmenautos des Mitarbeiters, wenn vorhanden, mit maximal 45 Buchstaben & Zahlen beschreibt (Varchar (45))

KFZ2 – Beschreibung des zweiten Firmenautos des Mitarbeiters, wenn vorhanden, mit maximal 45 Buchstaben & Zahlen beschreibt (Varchar (45))

Aktuell wurden nun die Entitys mit ihren Attributen erstellt, jedoch existieren noch keine Relationen zwischen den Entitys. Um die Relationen zwischen den Entitys zu erstellen müssen sich folgende Logikfragen gestellt werden.

Ein Projekt kann mehrere Mitarbeiter haben und ein Mitarbeiter kann mehrere Projekte haben. Hieraus ergibt sich eine N zu M Beziehung. Daher muss eine zwischen Tabelle

ProjektMitarbeiter erstellt werden, welche die Fremdschlüssel Projekt_ProNR (INT) und Mitarbeiter_PerNR (INT) sowie das Attribut Zeitanteil (Float (10,1)) enthält. Die Fremdschlüssel dienen zur Verknüpfung der Tabellen Projekt und Mitarbeiter. Das Attribut Zeitanteil beschreibt mit maximal 10 Ziffern und einer Nachkommastelle den Stundenaufwand, welcher ein Mitarbeiter in einem Projekt hatte.

Ein Projekt hat immer einen Mitarbeiter als Projektleiter und ein Mitarbeiter kann in mehreren Projekt Projektleiter sein. Hieraus ergibt sich eine 1 zu N Beziehung.

Eine Abteilung kann mehrere Mitarbeiter haben und ein Mitarbeiter kann nur einer Abteilung angehören. Hieraus ergibt sich eine N zu 1 Beziehung.

Eine Abteilung hat immer einen Mitarbeiter als Abteilungsleiter und ein Mitarbeiter kann in einer Abteilung Abteilungsleiter sein. Hieraus ergibt sich eine 1 zu 1 Beziehung.

Jetzt ist unser Entity-Relationship-Modell fertig und kann nun auf den Datenbankserver importiert werden.

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;

SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION';

DROP SCHEMA IF EXISTS `Firma`;

CREATE SCHEMA IF NOT EXISTS `Firma` DEFAULT CHARACTER SET utf8;

USE `Firma`;


DROP TABLE IF EXISTS `Firma`. `Abteilung`;

CREATE TABLE IF NOT EXISTS `Firma`.`Abteilung` ( `AbtNR` INT ZEROFILL UNSIGNED NOT NULL AUTO_INCREMENT, `Bezeichnung` VARCHAR(45) NOT NULL, `Abteilungsleiter` INT ZEROFILL UNSIGNED NULL, `Etat` FLOAT(10,2) NOT
```

NULL, PRIMARY KEY (`AbtNR`), INDEX `fk_Abteilung_Mitarbeiter1_idx` (`Abteilungsleiter` ASC) VISIBLE, UNIQUE INDEX `Bezeichnung_UNIQUE` (`Bezeichnung` ASC) VISIBLE, UNIQUE INDEX `Abteilungsleiter_UNIQUE` (`Abteilungsleiter` ASC) VISIBLE, CONSTRAINT `fk_Abteilung_Mitarbeiter1` FOREIGN KEY (`Abteilungsleiter`) REFERENCES `Firma`.`Mitarbeiter` (`PerNR`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE = InnoDB;

DROP TABLE IF EXISTS `Firma`.`Mitarbeiter` ;

CREATE TABLE IF NOT EXISTS `Firma`.`Mitarbeiter` (`PerNR` INT UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT, `Name` VARCHAR(45) NOT NULL, `Vorname` VARCHAR(45) NOT NULL, `Strasse` VARCHAR(45) NOT NULL, `PLZ` INT(5) NOT NULL, `Ort` VARCHAR(45) NOT NULL, `Telefon` INT NOT NULL, `Abteilung` INT NOT NULL, `Geschlecht` ENUM('m', 'w') NOT NULL, `eingestellt` DATE NOT NULL, `KFZ1` VARCHAR(10) NULL, `KFZ2` VARCHAR(10) NULL, PRIMARY KEY (`PerNR`), INDEX `fk_Mitarbeiter_Abteilung1_idx` (`Abteilung` ASC) VISIBLE, CONSTRAINT `fk_Mitarbeiter_Abteilung1` FOREIGN KEY (`Abteilung`) REFERENCES `Firma`.`Abteilung` (`AbtNR`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE = InnoDB;

DROP TABLE IF EXISTS `Firma`.`Projekt` ;

CREATE TABLE IF NOT EXISTS `Firma`.`Projekt` (`ProNR` INT NOT NULL, `Bezeichnung` VARCHAR(45) NOT NULL, `Start` DATE NOT NULL, `Ende` DATE NULL, `Auftragswert` FLOAT(12,2) NOT NULL, `bezahlt` FLOAT(12,2) NULL, `Leiter` INT NOT NULL, PRIMARY KEY (`ProNR`), INDEX `fk_Projekt_Mitarbeiter1_idx` (`Leiter` ASC) VISIBLE, CONSTRAINT `fk_Projekt_Mitarbeiter1` FOREIGN KEY (`Leiter`) REFERENCES `Firma`.`Mitarbeiter` (`PerNR`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE = InnoDB;

DROP TABLE IF EXISTS `Firma`.`ProjektMitarbeiter` ;

CREATE TABLE IF NOT EXISTS `Firma`.`ProjektMitarbeiter` (`Projekt_ProNR` INT NOT NULL, `Mitarbeiter_PerNR` INT NOT NULL, `Zeitanteil` FLOAT(10,1) NOT NULL, INDEX `fk_ProjektMitarbeiter_Projekt1_idx` (`Projekt_ProNR` ASC) VISIBLE, INDEX `fk_ProjektMitarbeiter_Mitarbeiter1_idx` (`Mitarbeiter_PerNR` ASC) VISIBLE, PRIMARY KEY (`Projekt_ProNR`, `Mitarbeiter_PerNR`), CONSTRAINT `fk_ProjektMitarbeiter_Projekt1` FOREIGN KEY (`Projekt_ProNR`) REFERENCES

```
`Firma`.`Projekt` (`ProNR`) ON DELETE NO ACTION  ON UPDATE NO ACTION, CONSTRAINT `fk_ProjektMitarbeiter_Mitarbeiter1` FOREIGN KEY (`Mitarbeiter_PerNR`) REFERENCES `Firma`.`Mitarbeiter` (`PerNR`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

3.5 Daten Generierung

Da es bei einem ETL Prozess darum geht, dass man bestimmte Daten extrahieren müssen wir zunächst irgendwelche Testdaten besorgen oder selbst welche erstellen. Im Rahmen dieses Projektes haben wir beschlossen einen eigenen Datensatz zu erstellen. Dies bieten den Vorteil, dass wir so die Daten individueller an unserem Modell und unsere Story anpassen können. Zusätzlich muss man sich so keine Sorgen darum machen, dass wir herausfinden müssen, ob wir den jeweiligen Datensatz benutzen dürfen.

Um die Daten zu generieren haben wir das Python Paket „Faker“ benutzt. Dieses Paket bietet die Möglichkeit zu verschiedenen Themen (z.B. Name, Adresse) zufällige Werte zu generieren. Hierbei kann man zusätzlich angeben in welcher Sprache die verteilten werte sein sollen. Das ist sehr wichtig, da zum Beispiel die Art und Weise wie man im Amerikanischen eine Adresse angibt deutlich anders ist als im Deutschen. Dies funktioniert so, dass jedes Mal, wenn eine bestimmte Funktion aufgerufen wird, diese einen anderen Wert ausgibt. So kann man mit Hilfe einer Schleife sehr schnell große Datensätze erzeugen. Zusätzlich hat man die Möglichkeit eine Funktion als „unique“ aufzurufen. Dies führt dazu, dass jeder Wert dieser Funktion nur einmal im Datensatz vorkommt. Die Letzte „Faker“-Funktion, die wir im Rahmen unseres Projektes benötigen ist die Möglichkeit eigene „Provider“ zu erstellen. Als Provider

werden einfach die verschiedenen Funktionen bezeichnet. Hiermit kann man nun eine Liste aus eigenen Begriffen angeben, die dann beim späteren Aufrufen zufällig verteilt werden.

Nun zum Code. Als erstes werden die benötigten Imports getätigt. Dies verläuft genauso wie man es von Python kennt. Als nächstes wird das Faker Objekt mit der

Generator for Dummy Data

```
: from faker import Faker
from faker.providers import DynamicProvider
from faker_vehicle import VehicleProvider
from collections import defaultdict
import pandas as pd
import random
from dateutil import parser
```

```
: rows = 10000

fake = Faker("de_DE")
fake.add_provider(VehicleProvider)

department_provider = DynamicProvider(
     provider_name="department_names",
     elements= ["IT", "Controlling", "Sales", "Consulting", "Marketing"],
)
fake.add_provider(department_provider)
```

Abbildung 6: Import und Tabellenerstellung

richtigen Sprache instanziiert und gleichzeitig wird ein Seed gesetzt. Der Seed sorgt dafür das die zufälligen Elemente bei jeder Ausführung dieselben Werte ausspucken. Dies vereinfacht es im Nachhinein noch Anpassungen zu machen.

Als nächsten Schritt wird ein sogenannter „Community-Provider“ geladen und zusätzlich noch ein eigener Provider erstellt. Das „Faker-Paket“ gibt die Möglichkeit, dass die Community eigene Provider bereitstellen kann. Der restliche Code ist nun nur noch verschiede Schleifen in denen die jeweiligen Tabellen nach einem vorher in der Gruppe bestimmten Schema erstellt werden und danach in eine Excel Tabelle gepackt

Entry for each Employee

```
main_data = defaultdict(list)
gender=["m", "w"]

for i in range(rows):
    main_data["PerNR"].append(i)
    main_data["Name"].append(fake.name())

    temp_address = fake.address().split("\n")
    main_data["Strasse"].append(temp_address[0])
    main_data["Ort"].append(temp_address[1])

    main_data["Telefon"].append(fake.phone_number())
    main_data["Abteilung"].append(fake.department_names())
    main_data["Geschlecht"].append(gender[random.randint(0,1)])
    main_data["eingestellt"].append(fake.date())
    main_data["KFZ1"].append(fake.vehicle_make_model())
    main_data["KFZ2"].append(fake.vehicle_make_model())

main_df = pd.DataFrame(main_data)
main_df.to_excel("data/mitarbeiter.xlsx", index=False)
main_df.head()
```

	PerNR	Name	Strasse	Ort	Telefon	Abteilung	Geschlecht	eingestellt	KFZ1	KFZ2
0	0	Janette Hahn	Andrej-Ortmann-Straße 4	29508 Lübeck	07227 043345	Consulting	w	1973-02-03	Ford Crown Victoria	Volvo XC60
1	1	Univ.Prof. Hans-Josef Liebelt	Henry-Striebitz-Platz 5/0	53389 Staffelstein	+49(0)0854409714	Sales	m	1976-03-22	Volkswagen Passat	BMW X3
2	2	Katalin Rosemann	Sieringring 8/9	65494 Berlin	09638029691	Sales	m	2000-08-30	Audi S4	INFINITI J
3	3	Dipl.-Ing. Torsten Tschentscher	Hans-Walter-Thies-Platz 5	24423 Hofgeismar	+49(0)9044293896	IT	w	1985-08-11	Suzuki Swift	Chevrolet Silverado 1500 Crew Cab
4	4	Solveig Weinhage-Heser	Denis-Lachmann-Ring 9/4	25232 Hainichen	(03120) 57644	IT	w	2021-11-15	Volkswagen Touareg	Toyota Solara

Abbildung 7: Definition der Attribute und Header

werden.

Um zu gewährleisten, dass man Relationen bilden kann wurde noch ein Provider erstellt, welcher die bisher verwendeten Namen in den nächsten Tabellen wiederverwendet.

Saving existing Names to create Relations

```
uniques = main_df["Name"].unique()

existing_names_provider = DynamicProvider(
     provider_name="existing_names",
     elements= list(uniques),
)
fake.add_provider(existing_names_provider)
```

Abbildung 8: Relationsgenerierung

3.6 Problemlösung (ETL-Prozess und Data Warehouse)

X.1 Überblick:

Der ETL-Prozess wurde in unserem Projekt in Python erstellt. Hierzu ist zuerst die Extraktion der Daten aus den verschiedenen Abteilungen erfolgt. Im Anschluss wurden diese gewonnenen Daten in Python importiert und dann transformiert, um sie für den Import in das Datawarehouse vorzubereiten.

X.2 Detaillierte Übersicht über den Prozess:

X.2.1 Extraktion:

Die Daten wurden in Python von uns selbst erstellt, weswegen sich der Extraktion Prozess in unserem Projekt erübrigt.

X.2.2 Transformation:

Bei der Python Datei handelt es sich um eine „.ipynb“, also eine Jupyter Notebook Datei. Für die Transformation importieren zunächst das Modul „pandas“ als Kurzform „pd“.

```
import pandas as pd
```

Abbildung 9: Import Pandas

Mitarbeiter Transformation:

Im Anschluss an den Import des erforderlichen Moduls „pandas“ haben wir die Erste Funktion mitarbeiter_func() erstellt um die Mitarbeiter Tabelle zu transformieren.

```
def mitarbeiter_func():
    #Import Mitarbeiter Tabelle
    mitarbeiter = pd.read_excel('mitarbeiter.xlsx', delimiter=' ')

    #Spaltung Ort und PLZ
    mitarbeiter.insert(3,"PLZ", mitarbeiter.Ort.str.split(" ").str[0].tolist())
    mitarbeiter["Ort"] = mitarbeiter.Ort.str.split(" ").str[1].tolist()

    #Abteilung in AbteilungsNR tauschen
    mitarbeiter.loc[mitarbeiter["Abteilung"] == "IT","Abteilung"] = 2
    mitarbeiter.loc[mitarbeiter["Abteilung"] == "Consulting","Abteilung"] = 4
    mitarbeiter.loc[mitarbeiter["Abteilung"] == "Sales","Abteilung"] = 0
    mitarbeiter.loc[mitarbeiter["Abteilung"] == "Marketing","Abteilung"] = 3
    mitarbeiter.loc[mitarbeiter["Abteilung"] == "Controlling","Abteilung"] = 1

    #Spaltung in Vorname und Name
    mitarbeiter.insert(1,"Vorname", mitarbeiter.Name.str.split(" ").str[0].tolist())
    mitarbeiter["Name"] = mitarbeiter.Name.str.split(" ").str[1].tolist()

    #Typanpassung
    mitarbeiter.infer_objects()

    #Export der Datei
    mitarbeiter_csv_data = mitarbeiter.to_csv('ETL Daten/mitarbeiter_etl.csv', header=False, index=False)
```

Abbildung 10: Transformation Mitarbeitertabelle

In der Funktion wird zunächst die „mitarbeiter.xlsx“ importiert mit der Pandas Funktion read_excel(), da dieses Programm auf dem Betriebssystem MacOS verfasst wurde, benötigt die read_excel() Funktion noch den Befehl delimiter um die Datei zu separieren.

Das entstehende Dataframe wird nun zur Normalisierung der Daten verwendet. Hierbei beginnen wir mit der Spaltung der Adressen in die zwei Spalten „PLZ“ und „Ort“. Dazu verwenden wir die Funktion str.split() mit dem Seperator „Leerzeichen“ um die Spalte zu zuteilen. Der erste Teil der Adresse wird in der Spalte „PLZ“ gespeichert und mit dem Befehl insert() an der dritten Stelle des Dataframes eingefügt. Der zweite Teil wird als Ort der Adresse gespeichert, welche sich an der vierten Stelle des Dataframes befindet.

Um den Fremdschlüssel der Mitarbeiter Tabelle zur Abteilungstabelle zu erstellen, verwenden wir die pandas „loc“ Funktion. Dabei wird auf den Namen der Abteilung überprüft und durch den Primärschlüssel der Abteilungstabelle ersetzt.

Der Name wird in die zwei Spalten „Vorname“ und „Name“ gespalten. Hierzu benutzen wir wieder die str.split() Funktion und die Insert Funktion, um den Vornamen an der zweiten Stelle des Dataframes einzufügen hinter der „PerNR“ und vor dem „Namen“.

Das Dataframe wird mit dem Befehl infer.objects() noch an die jeweiligen Typen angepasst. Danach erstellen wir eine mitarbeiter_etl.csv aus dem bestehenden Dataframe um diese im Laden-Prozess in unser Datawarehouse zu importieren.

Projekt Transformation:

Erstellen der projekt_func() Funktion um die Projekt Tabelle zu transformieren.

```
def projekt_func():
    #Import Projekt Tabelle
    projekt = pd.read_excel('projekt.xlsx', delimiter=' ')
    #Import Mitarbeiter Tabelle
    mitarbeiter = pd.read_excel('mitarbeiter.xlsx', delimiter=' ')

    #Projektleiter Fremdschlüssel hinzufügen
    for i in range(len(projekt)):
        for j in range(len(mitarbeiter)):
            if projekt["Leiter"][i] == (mitarbeiter["Name"][j]):
                projekt["Leiter"][i] = mitarbeiter["PerNR"][j]
                break

    #Typanpassung
    projekt.infer_objects()

    #Export der Datei
    projekt_csv_data = projekt.to_csv('ETL Daten/projekt_etl.csv', header=False, index=False)
```

Abbildung 11: Transformation Projekttabelle

Zu Anfang der Funktion werden zuerst die Projekt sowie auch die Mitarbeiter Tabelle importiert. Hierbei wird wieder als Delimiter ein Leerzeichen verwendet. In der for-Schleife im Anschluss wird der Projektleiter der Projekttabelle, durch den Primärschlüssel, also die Personalnummer der Mitarbeitertabelle ersetzt, um die Referenz später zu importieren.

Konkret werden zwei in einander verschachtelte for-Schleifen verwendet, hierbei wird überprüft, ob der Projektleitername mit dem Namen des Mitarbeiters an der jeweiligen Position in der Mitarbeitertabelle übereinstimmt, falls dies der Fall ist wird das Attribut durch die Personalnummer des entsprechenden Mitarbeiters ersetzt. Mit dem Befehl break wird die innere Schleife abgebrochen sobald das Ergebnis feststeht. Mit dem Befehl infer_objects() werden bei den Daten wieder der Typ angepasst. Zum Schluss

der Funktion wird wieder eine .csv Datei mit den Werten erstellt, um sie später zu importieren.

Abteilung Transformation:

Erstellen der abteilung_func() Funktion um die Projekt Tabelle zu transformieren. Der Code der abteilungs_func() entspricht dem Code der Projektfunktion. Hier wird der Abteilungsleiter, jedoch durch die Personalnummer ersetzt, um die Referenz zu erstellen.

```
def abteilung_func():
    #Import Abteilung Tabelle
    abteilung = pd.read_excel('abteilung.xlsx', delimiter=' ')
    #Import Mitarbeiter Tabelle
    mitarbeiter = pd.read_excel('mitarbeiter.xlsx', delimiter=' ')

    #Abteilungsleiter Fremdschlüssel hinzufügen
    for i in range(len(abteilung)):
        for j in range(len(mitarbeiter)):
            if abteilung["Abteilungsleiter"][i] == (mitarbeiter["Name"][j]):
                abteilung["Abteilungsleiter"][i] = mitarbeiter["PerNR"][j]

    #Typanpassung
    abteilung.infer_objects()

    #Export der Datei
    abteilung_csv_data = abteilung.to_csv('ETL Daten/abteilung_etl.csv', header=False, index=False)
```

Abbildung 12: Transformation Abteilungstabelle

Projektzeiten Transformation:

Erstellen der projektzeiten_func() Funktion um die Projektzeitentabelle zu transformieren. Der Code der projektzeiten_func() entspricht dem Code der Projektfunktion. Hier wird der Mitarbeiter, jedoch durch die Personalnummer ersetzt, um die Referenz zu erstellen.

```python
def projektzeiten_func():
    #Import Projektzeiten Tabelle
    projektzeiten = pd.read_excel('projektzeiten.xlsx', delimiter=' ')
    #Import Mitarbeiter Tabelle
    mitarbeiter = pd.read_excel('mitarbeiter.xlsx', delimiter=' ')

    #Abteilungsleiter Fremdschlüssel hinzufügen
    for i in range(len(projektzeiten)):
        for j in range(len(mitarbeiter)):
            if projektzeiten["Mitarbeiter"][i] == (mitarbeiter["Name"][j]):
                projektzeiten["Mitarbeiter"][i] = mitarbeiter["PerNR"][j]
                break

    #Typanpassung
    projektzeiten.infer_objects()

    #Export der Datei
    projektzeiten_csv_data = projektzeiten.to_csv('ETL Daten/projektzeiten_etl.csv', header=False, index=False)
```

Abbildung 13: Transformation Projektzeitentabelle

X.2.3 Laden:

Für den Import der ETL-Daten verwenden wir das Modul des mysql.connector, um die .csv Dateien in die Tabellen zu laden.

```python
import mysql
import mysql.connector
```

Abbildung 14: Import mySQL Connector

Bei diesem Prozess beginnen wir mit der Herstellung der Verbindung unseres MySQL Servers bzw. der Datenbank mit dem unserem Python Skript. Dabei müssen wir den host, sowie unseren User auf dem der MySQL Server läuft angeben.

```python
connection = mysql.connector.connect(host = "127.0.0.1", user = "root", passwd = "", db = "")
dd
```

Abbildung 15: Verbindungsaufbau zu MySQL

```
cursor = connection.cursor()
cursor.execute("SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0")

cursor.execute("SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0")
cursor.execute("SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,

cursor.execute("DROP SCHEMA IF EXISTS `Firma` ")
cursor.execute("CREATE SCHEMA IF NOT EXISTS `Firma` DEFAULT CHARACTER SET utf8 ")
cursor.execute("USE `Firma` ")
```

Abbildung 16: Erstellung der Datenbank

Der Coding-Bereich zeigt das Erstellen der Datenbank in Python und speichert, dies in dem Schema Firma.

```
cursor.execute("DROP TABLE IF EXISTS `Firma`.`Abteilung` ")
cursor.execute("CREATE TABLE IF NOT EXISTS `Firma`.`Abteilung` ( `AbtNR` INT ZEROFILL UNSIGNED

cursor.execute("DROP TABLE IF EXISTS `Firma`.`Mitarbeiter` ")
cursor.execute("CREATE TABLE IF NOT EXISTS `Firma`.`Mitarbeiter` ( `PerNR` INT UNSIGNED ZEROFIL

cursor.execute("DROP TABLE IF EXISTS `Firma`.`Projekt` ")
cursor.execute("CREATE TABLE IF NOT EXISTS `Firma`.`Projekt` ( `ProNR` INT NOT NULL, `Bezeichnu

cursor.execute("DROP TABLE IF EXISTS `Firma`.`ProjektMitarbeiter` ")
cursor.execute("CREATE TABLE IF NOT EXISTS `Firma`.`ProjektMitarbeiter` (`Projekt_ProNR` INT NO

cursor.execute("SET SQL_MODE=@OLD_SQL_MODE")
cursor.execute("SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS")
cursor.execute("SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;")

cursor.close()
```

Abbildung 17: Tabellen erstellen

Der zweite Teil zeigt das Erstellen der Tabellen in der Datenbank mit den Attributen und den verschiedenen Datentypen, ebenfalls werden noch die Primär- und Fremdschlüsselbeziehungen hinzugefügt.

```python
cursor = connection.cursor()
cursor.execute("Use Firma")
cursor.execute("LOAD DATA INFILE 'C:/Users/Admin/Desktop/ETL_Project/mitarbeiter_etl.csv' INTO TABLE mitarbeiter")
cursor.close()
```

Python

```python
cursor = connection.cursor()
cursor.execute("Use Firma")
cursor.execute("LOAD DATA INFILE 'C:/Users/Admin/Desktop/ETL_Project/projekt_etl.csv' INTO TABLE projekt")
cursor.close()
```

Python

```python
cursor = connection.cursor()
cursor.execute("Use Firma")
cursor.execute("LOAD DATA INFILE 'C:/Users/Admin/Desktop/ETL_Project/abteilung_etl.csv' INTO TABLE abteilung")
cursor.close()
```

Python

```python
cursor = connection.cursor()
cursor.execute("Use Firma")
cursor.execute("LOAD DATA INFILE 'C:/Users/Admin/Desktop/ETL_Project/projektzeiten_etl.csv' INTO TABLE projektzeiten")
cursor.close()
```

Abbildung 18: Import in die Tabellen

Der letzte Teil des Programms zeigt wie wir die .csv Dateien aus den oben erstellten Funktionen in die Tabellen importieren. Hierzu benutzen wir den Befehl „Load Data Infile“ aus MySQL.

3.7 Frage: Inwiefern ist es sinnvoll Data Marts und OLAP Cubes für interne Abteilung als Kunden aufzusetzen?

Mit OLAP ist es möglich kurzfristig große Datenbestände auszuwerten und schnell Prognosen sowie Trends abzuleiten. Seine Multidimensionalität macht es möglich beliebig viele Dimensionen zusammen zu betrachten und auszuwerten, wie z.B. Projekt, Budget, Dauer des Projektes oder Anteil der beschäftigten Mitarbeiter in einem Projekt.

Der Analyseprozess befindet sich auf einem Client-Server-Architecture, wo die Abfragen angefordert werden. Die Daten werden an das OLAP zur Verarbeitung weitergegeben. Die Datenhaltung selbst befindet sich auf einem anderen Server. Das führt dazu, dass die Ressourcen somit effizient eingesetzt werden.

Für ein Unternehmen mit diversen Abteilungen sowie vielen Daten und Informationen ist es sinnvoll einen Data Marts aufzusetzen. Während in den Data Warehouses die ganzen Informationen des Unternehmens erfasst werden, werden bei den Data Marts nur die Anforderungen bestimmter Abteilungen oder Geschäftsfunktionen erfüllt. Sie beinhalten hauptsächlich einen kleineren Teil des gesamten Datensatz. Ein Data Mart kann in einem bestehenden Data Warehouse oder aus anderen internen oder externen Daten Quellen erstellt werden.

Da bei den Abfragen nicht der komplette Stamm aufgerufen werden sollte und es bereits genügt, ein paar Datenquellen zu sammeln, werden im Data Warehouse die Data Marts, eingerichtet. Jeder Data Mart hat einen eigenen Speicherort, in denen Daten bis zu ihrer Verwendung gelagert werden.

4. Fazit / kritische Reflexion

Abschließend lässt sich nun sagen, dass das Projekt voller Erfolg war. Das Zeitmanagement war sehr gut geplant und der aktuelle Stand war immer gleichauf mit dem im Voraus festgelegten Plan. Des Weiteren gab es durch die häufigen Meetings und die Kommunikation untereinander wenig Probleme. Die Aufgaben waren klar verteilt und konnten dank eingebundenem GitHub auch von jedem verfolgt werden. Auch bei Problemen in der Entwicklung, gab es schneller Unterstützung innerhalb des Dev-Teams. Wenn es also ein Problem gab hat sich das gesamte Dev-Team getroffen und hat gemeinsam an einer Lösung für das Problem gearbeitet.

Vereinzelnde Probleme ganz am Anfang, dass dort die Aufgabenverteilung noch nicht klar geregelt wurde. Das Problem wurde jedoch schnell behoben und es hatte jeder seine Aufgabe zugeteilt bekommen.

Für das nächsten Projekt bzw. wenn das Projekt nochmal gemacht werden würde, dann würde sich unser Team für eine vorgefertigte ETL-Lösung entscheiden, da das den Großteil der Zeit in Anspruch genommen hat.

Literaturverzeichnis

Vergleich von Open-Source und kommerziellen Programmen zur Durchführung eines ETL-Prozesses. Online verfügbar unter http://zope.informatik.hu-berlin.de/de/forschung/gebiete/wbi/teaching/studiendiplomarbeiten/finished/2013/qualitz_expose_130420.pdf.

Gloger, Boris (2010): Scrum. In: *Informatik Spektrum* 33 (2), S. 195–200. DOI: 10.1007/s00287-010-0426-6.

Google Books (2022): Kanban für die Softwareentwicklung. Online verfügbar unter https://books.google.de/books?hl=de&lr=&id=uMIoBAAAQBAJ&oi=fnd&pg=PR4&dq=kanban&ots=-bNi7FAOhF&sig=Gt_Cv3M0nNe_etiQL-u888z9gdM&redir_esc=y#v=onepage&q=kanban&f=false, zuletzt aktualisiert am 23.02.2022, zuletzt geprüft am 23.02.2022.

Sascha Qualitz: qualitz_expose_130420. Online verfügbar unter https://www.informatik.hu-berlin.de/de/forschung/gebiete/wbi/teaching/studienDiplomArbeiten/finished/2013/qualitz_expose_130420.pdf, zuletzt geprüft am 06.01.2022.

Steiner, René (2021): Grundkurs relationale Datenbanken. Einführung in die Praxis der Datenbankentwicklung für Ausbildung, Studium und IT-Beruf. 10., aktualisierte Auflage. Wiesbaden: Springer Vieweg. Online verfügbar unter https://link.springer.com/content/pdf/10.1007%2F978-3-658-32834-4.pdf, zuletzt geprüft am 17.01.2022.