Inhaltsverzeichnis

1. Einleitung

In der Produktion fallen viele sogenannte Qualitätsdaten an, diese sind logisch stark mit Produktionsdaten in ERP-Systemen verknüpft. Sie sind logistisch, kostenrechnerisch oder buchhalterisch relevant. Neben der hohen Datenmenge, erzeugen die vielen Sensoren auch einen sehr heterogenen Datenbestand. Durch Veränderungsprozesse im Unternehmen kommt noch eine gewisse dynamik hinzu, diese entsteht durch Änderungen im Messinstrumenten Bestand. Alle diese Daten müssen persistiert werden, auch wenn nur ein Teil davon zunächst relevant erscheint wäre es fatal, wenn Informationen die später nützlich sein könnten einfach verloren gehen würden. Somit benötigt es einen flexiblen, verlässlichen und performanten Datenspeicher.

Da es viele Anbieter auf dem Datenbankmarkt gibt war es Ziel dieses Projektes vier verschiedene Lösungsansätze an einem Praxisbeispiel zu evaluieren. Dabei sollte eine proprietäre Datenstruktur, eine universell relational Struktur, eine Schlüssel-Werte-Datenbank und eine dokumentenorientierte Datenbank zum Einsatz kommen. In allen Lösungen sollten Daten konsistent gespeichert werden. Weiterhin sollten verschiedene Analysen möglich sein und auch auch das Darstellen der Analyseergebnisse.

2. Problemstellung

2.1. Einführung in das Praxisbeispiel

In dem konkreten zu implementierenden Beispiel ging es um fünf Maschinen beteiligt an einem Carbonteil Fertigungs Prozess. Dabei werden teure Carbonfasern mit einem günstigen Trägerstoff vernäht. Ziel dabei ist es eine bessere Handhabbarkeit bei geringeren Herstellungskosten bereitzustellen. Die Maschine erzeugt einen Eingangsdatensatz mit 23 Werten und einen Ausgangsdatensatz mit 16 Werten. Dabei werden neben der Seriennummer, die spezifisch für ein Werkstück (Teil) ist, der Fertigungsauftrag und die Ladungsträgernummer gespeichert. Zu einem Fertigungsauftrag gehören mehrere Teile. Weiterhin wird auch die Nummer des Ladungsträgers abgespeichert, dieser ist Abhängig von der Teilart und kann für mehrere Wekstücke verwendet werden. Die restliche Werte geben aufschluss über die Teil Qualität und den Fertigungsablauf.

Der Einlesealgorhytmus stellt etwas vereinfacht dar wie die Datensatzerstellung mit der Fertigung auf der Maschine zusmmenhängt. Sobald das Teil korrekt in die Maschine eingelegt wurde wird der Eingangsdatensatz erstellt. Nach der eigentlichen Fertigungsarbeit auf der Maschine wird das Teil überprüft. Dabei entsteht der Ausgangsdatensatz. Falls die Prüfung nicht erfolgreich war wird das Produkt erneut überprüft. Danach verlässt das Teil die Maschine, falls nun Probleme mit dem Werkstück gefunden wurden wird das Werkstück erneut in die Maschine gegeben, falls nicht endet der von uns betrachtete Prozess.

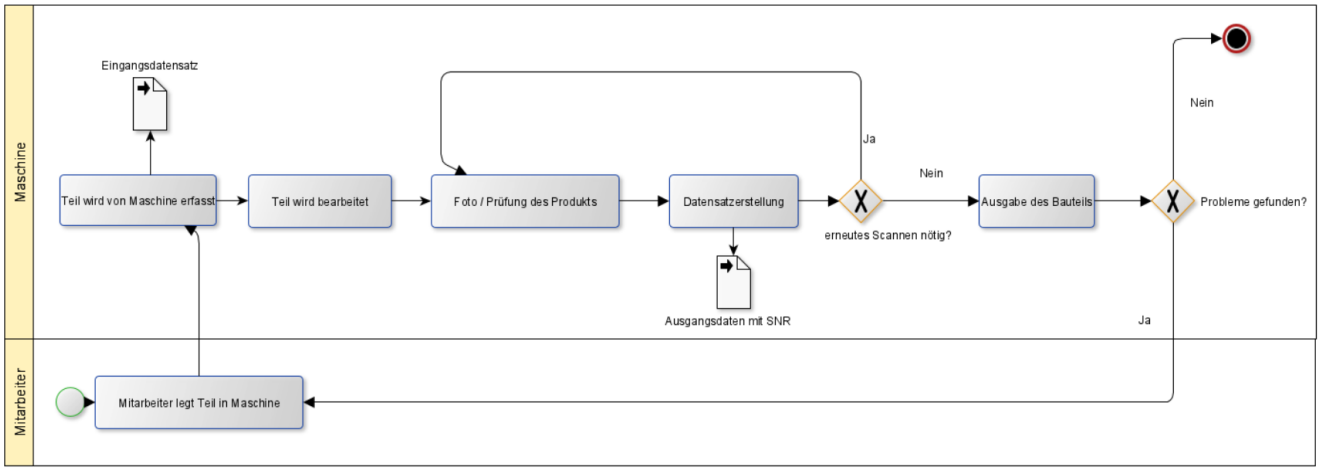


Abbildung 1. Einlesealgorhytmus

2.2. Allgemeine Informationen zur Umsetzung

Da die Eingangs- und Ausgangsdatensätze in einer pro Datensatz neu angelegten CSV gespeichert werden, war es eine Anforderung an das System auf diese Events zu reagieren. Dafür wurde ein Watchdog implementiert, da dies mit Python erfolgte und die sich alle Teammitglieder auf diese Programmiersprache geeinigt hatten, konnte dieser wiederverwendet werden. Dabei wurde die Bibliothek Wachtdog genutzt, diese erlaubt es mit wenigen Zeilen Veränderungen im Dateisystem festzustellen. Man übergibt dazu einfach das zu überwachende Verzeichnis und bei Änderungen wird das jeweilige Event geworfen. Darunter sieht man wie die Behandlung des genutzen Created Event aussieht. Um herauszufinden welche Datensatzart erstellt wurde wird durch eine IF-Anweisung überprüft ob es sich um einen Input Datensatz handelt und wenn ja, dann wird der der

"InputLoader" aufgerufen, falls das nicht der Fall ist der "OutputLoader". Dabei wird der richtig aufbereitete Datei Pfad mit übergeben.

```
def on_any_event(event):
        if event.event_type == 'created':
            if "input" in event.src_path.replace:
                InputLoader.save(event.src_path.replace('\\','/'))
            else:
                OutputLoader.save(event.src_path.replace('\\','/'))
```

3. State of the Art Lösungen

Um die Lösungen des Projektes zu evaluieren war zunächst ein Blick auf die in der Indsustrie eingesetzten Technologien notwendig. Dafür werden im folgenden Lösungsansätze aus dem Praxiseinsatz disktutiert.

3.1. Iconics Hyper Historian

Iconics bietet Automatisierungssoftware Lösungen welche Echtzeitinformationen für jede Anwendung, visualisieren, historisieren, analysieren und mobilisieren. Dabei spielt der Hyper Historian eine Schlüsselrolle in der Archivierung und Analyse historischer Daten. Dazu wurde von Iconics eine Datenbanklösung selbst entwickelt, diese basiert auf Zeitreihen und hat deshalb den Vorteil der Datenkompression. Nach außen wird ein SQL Interface bereitgestellt,

3.2. Amazon

Amazon hat mit der Amazon DynamoDB ebenfalls eine eigene NoSQL-Datenbank im Portfolio. Dabei werden Ansätze der Schlüssel-Wert- und der Dokumentendatenbank kombiniert. Daneben existiert auch noch Amazon Timestream, dies ist ein Zeitreihen-Datenbankservice. Hier lässt sich schon erkennen, dass in der Datenbankwelt keine passende Lösung für alle Probleme gibt, sondern unterschiedliche Anforderungen zu unterschiedlichen Systemen führen.

3.3. Kontron AIS GmbH

Die Kontron AIS hat je nach Projekt verschiedene Datenbanklösungen im Einsatz, dabei werden oft verschiedene Ansätze kombiniert. So ist meist auf der Auswertungsseite ein relationaler Datenbank Server wie Beispielsweise von Oracle oder Microsoft. Da der Aufbau von den Relationen, aber beim Speichern Zeit benötigt wird mit einem sogenannten Fast Layer gearbeitet. Um das diesen zu implementieren werden verschiedene Lösungen genutzt. Zum einem das Big Data System Hadoop. Es ermöglicht eine hoch performante Speicherung in einem redundanten und parallelisiertem Dateisystem. Ebenfalls werden als Zwischenspeicher die auch im folgenden diskutierten Ansätze wie Schlüssel-Wert-Datenbank und Dokumentenorientierte Datenbank genutzt, dabei werden die nicht nur Datensammlung, sondern auch als Zwischenspeicher für Auswertungen genutzt. Auffallend dabei ist das die Lösungen von Projekt zu Projekt variieren und das oft auch nur eine Kombination eine adequate Problemlösung schafft.

3.4. Rückschlüsse auf das Projekt

Da die Datenlast mit vier Maschinen und damit 8 Datensätzen pro 5 Minuten recht gering ist, wäre ein Big Data System wie Hadoop eine zu übertriebene Lösung, da diese Systeme für mehrere tausend Datensätze pro Sekunde ausgelegt sind. Ein Zeitreihendatenbankservice wäre keine Alternative die Infrage kommt, da diese für das beständige Aufzeichnen von Datenpunkten optimiert sind. Mit dieser Voreingrenzung ist es wahrscheinlich, dass die im Projekt diskutierten Ansätze auch für ein reales Projekt infrage kommen.

4. Analysen

4.1. Analyse 1

Die erste Analyse beschäftigt sich mit der Taktung pro Artikel. Dabei wurde die Differenz zwischen Eingangs- und Ausgangsdatensatz gemessen. Danach wurde pro Fertigungsauftrag gruppiert und das Minimum, das Maximum und der Durchschnitt ermittelt. Die Ergebnisse wurden auf eine Stunde beschnitten um nicht zu sehr von Ausreißern beeinflusst zu werden. Die Alternative wäre gewesen nur die Zeiten zu nutzen, die in der Fertigungszeit des Unternehmens liegen, da aber die Auswertungen nur als Grundlage zum Vergleich der Datenbanklösungen dienen sollte wurde das nicht implementiert.

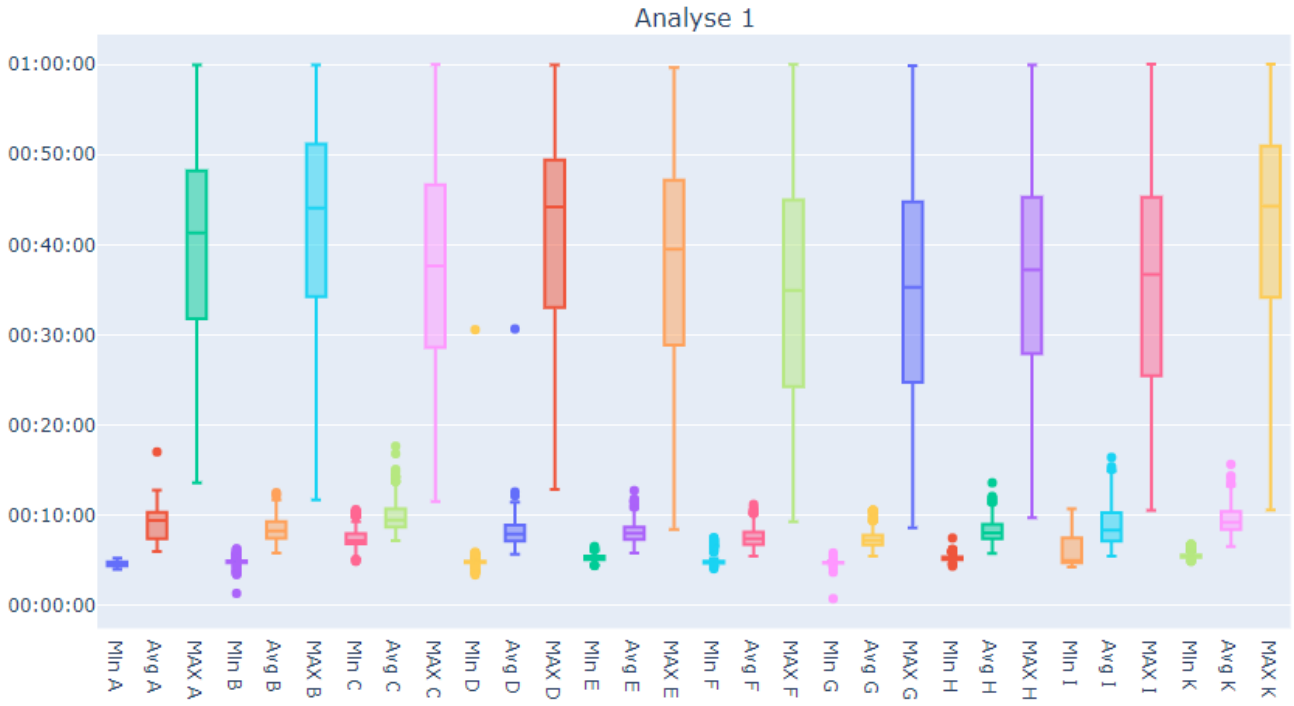


Abbildung 2. Analyse 1

4.2. Analyse 2

Diese Analyse beschäftigt sich nur mit Ausschuss Artikeln, also Werkstücke die mehrmals in die Maschine eingelegt worden sind, also deren SNR n In-Datensätze besitzt. Dabei wurde zunächst die Differenz zwischen Ende des fehlerhaften Vorgangs und Beginn des neuen Vorgangs gemessen. Auch hier wurde wieder Minimun, Maximun und der Durchschnitt berechnet. Aber über die Gruppierung Teil. Weiterhin wurde der Anteil an Fehlerhaften Bearbeitungen wurde ermittelt.

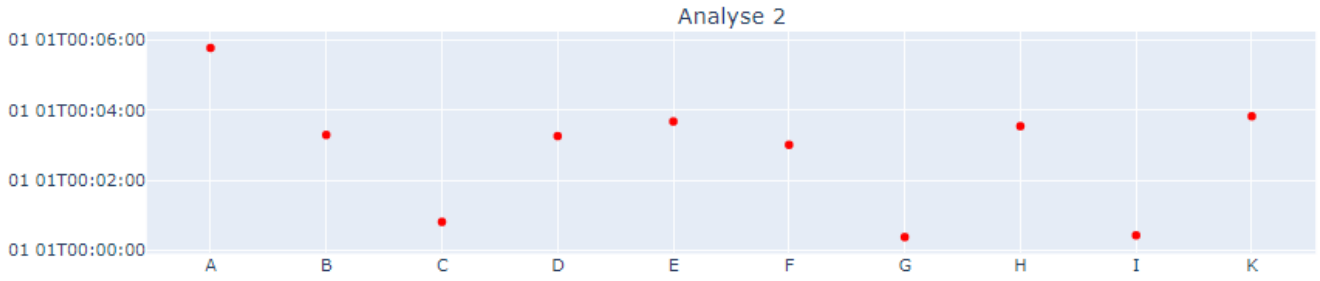


Abbildung 3. Minimun

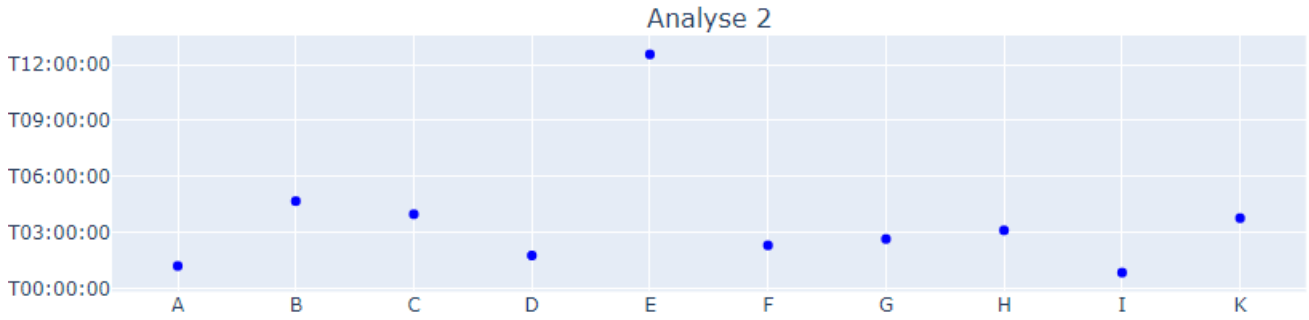


Abbildung 4. Durschschnitt

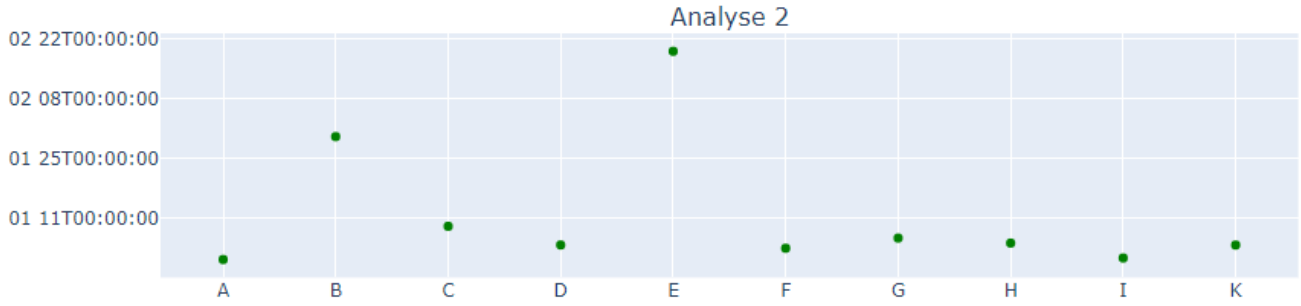


Abbildung 5. Maximum

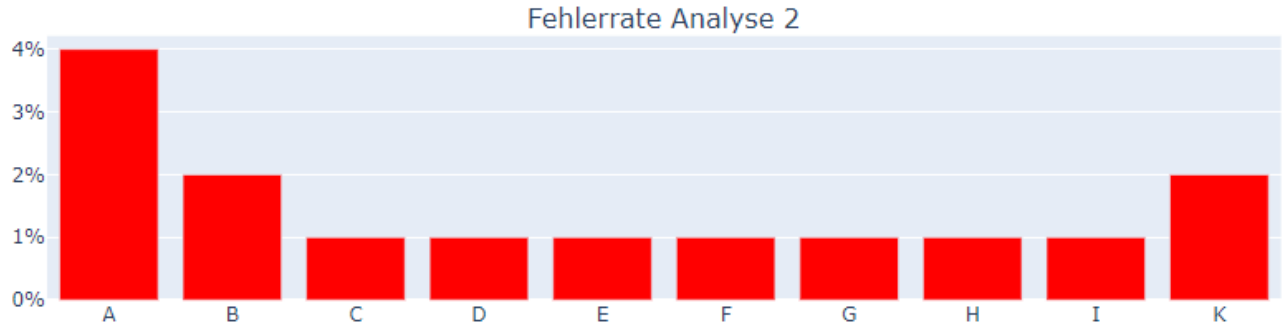


Abbildung 6. Fehlerrate

4.3. Analyse 3

Die dritte Analyse nutzt die gleichen Daten wie Analyse 1. Nur mit veränderter Darstellungsform. Ziel war es hierbei Ausreißer zu finde, deshalb wird nun nicht mit Boxplots gearbeitet, sondern mit einem Streudiagramm. Damit ist es besser ersichtlich, wo entfernte Ausreißer zu finden sind.

Abbildung 7. Minimun

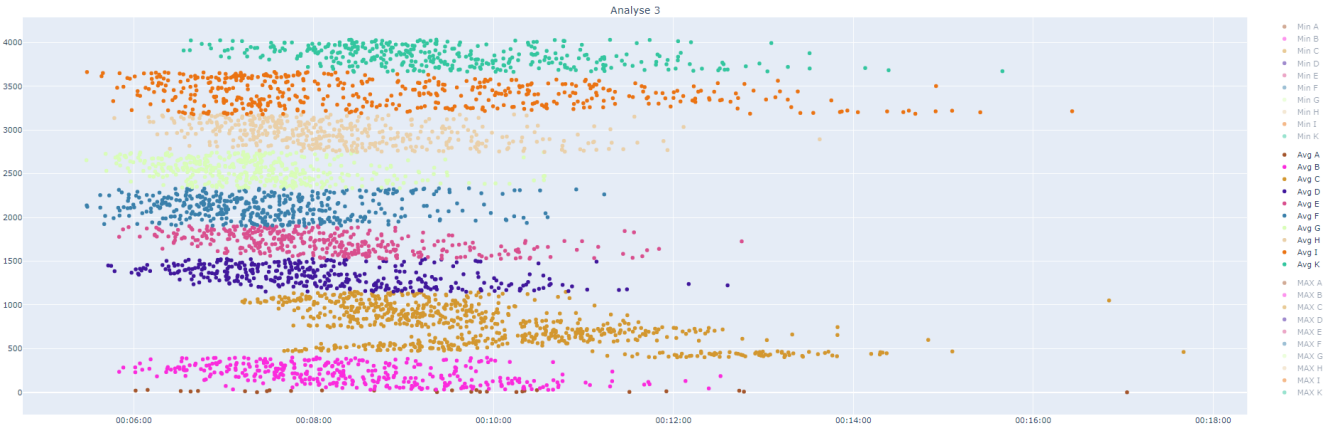


Abbildung 8. Durschschnitt

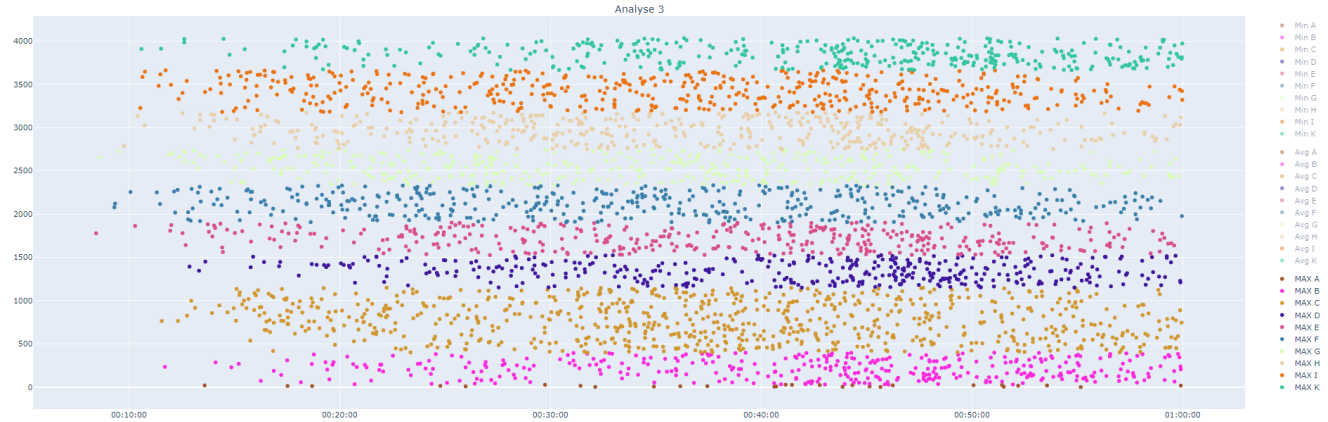


Abbildung 9. Maximum

4.4. Analyse 4

Die vierte Analyse zeigt die Nutzungszeit der Ladungsträger. Dabei ist die Nummer an der Y-Achse irrelevant und stellt jeglich die Datensatznummer dar, denn in den Ladungsträgernummern gab es eine zu große Streuung um diese adequat darzustellen.

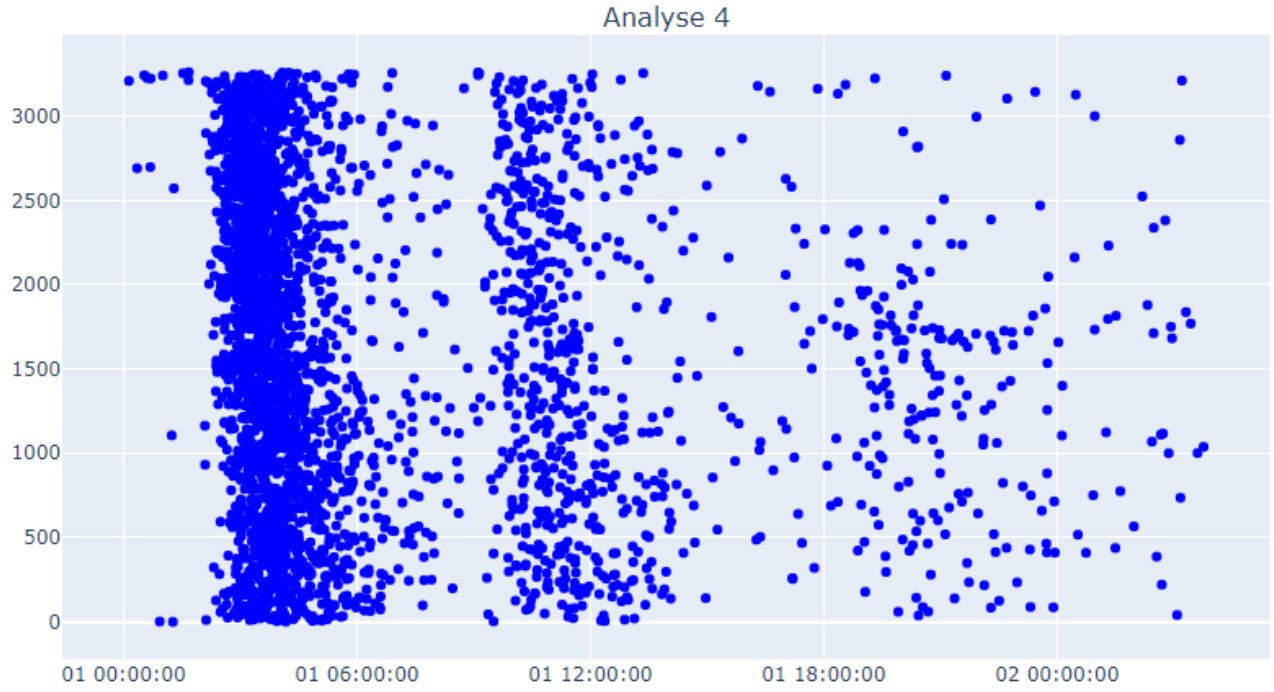


Abbildung 10. Analyse 4

4.5. Analyse 5

Die fünfte Analyse ist analog zur ersten Analyse, nur das dieses mal die Zwischenaggretion nicht pro Fertigungsauftrag sondern pro Ladungsträger erfolgte. Danach wurde die Daten wieder nach Teil sortiert und jeweils ein Boxplot für Minimum, Maximum und den Durchschnitt gezeichnet.

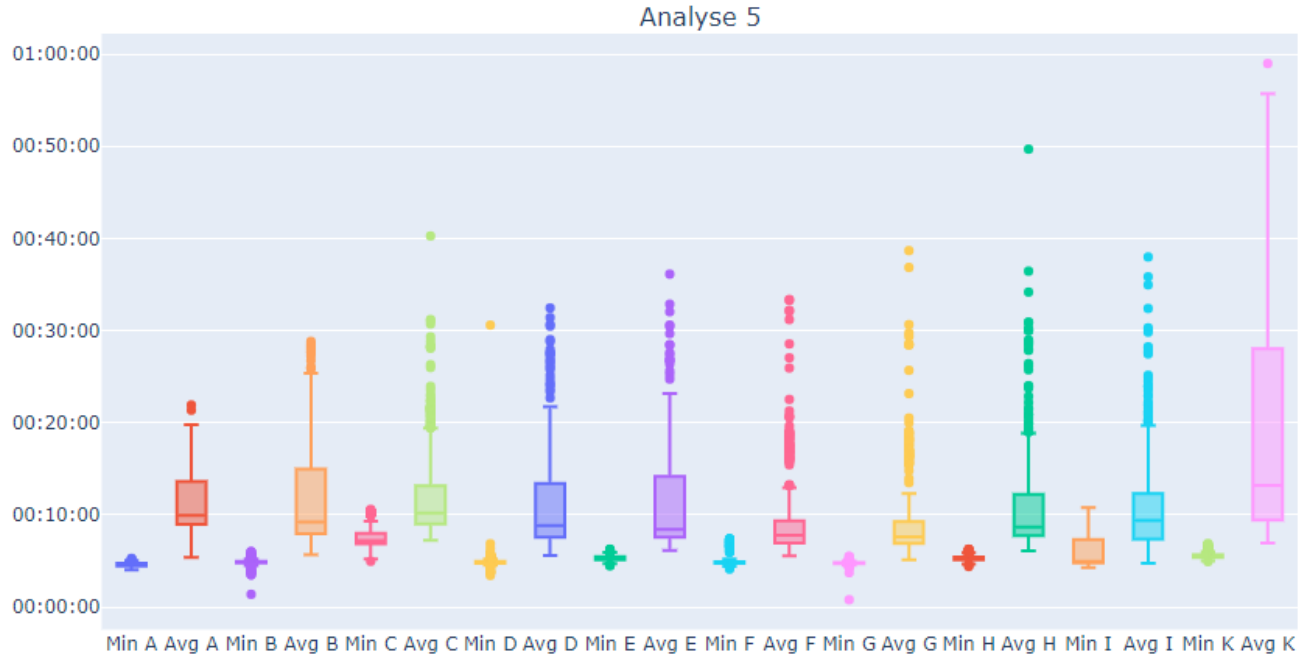


Abbildung 11. Minimun und Durschnitt

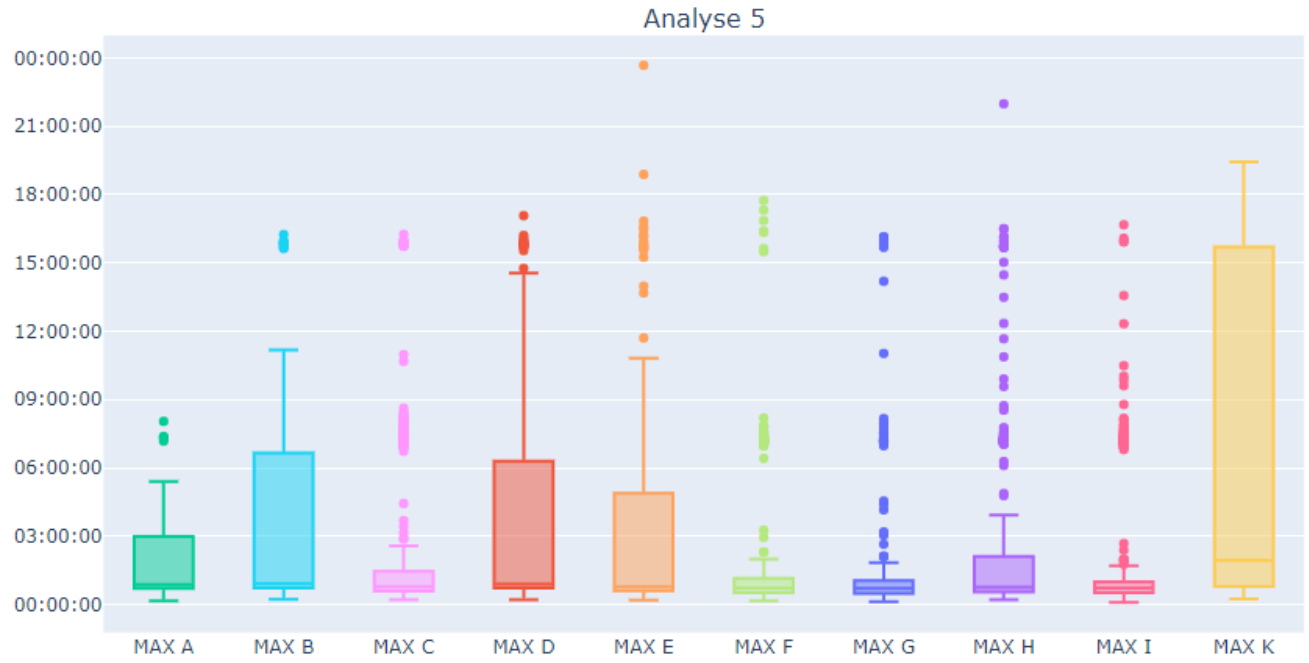
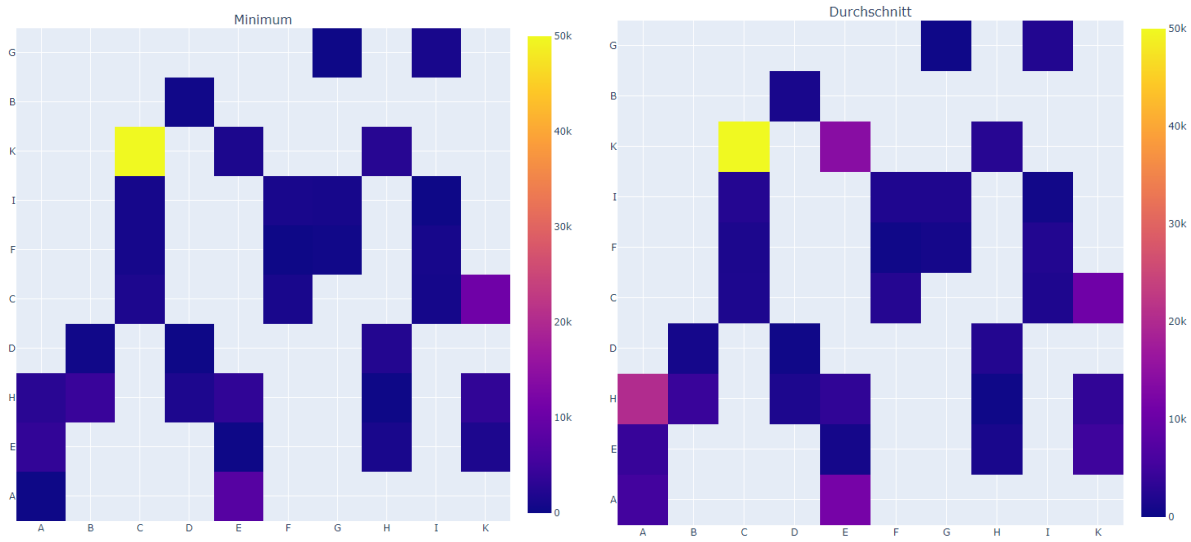


Abbildung 12. Maximum

4.6. Analyse 6

Die letzte Analyse zeigt die Umrüstzeiten zwischen den Teilen. An der Y-Achse stehen die Teile von denen auf die Teile die an der X-Achse umgerüstet wurde. Je heller die Farbe ist, desto höher ist der Wert. Hier wurde Minimun, Maximum und der Durchschnitt pro Umrüstvorgang bestimmt.



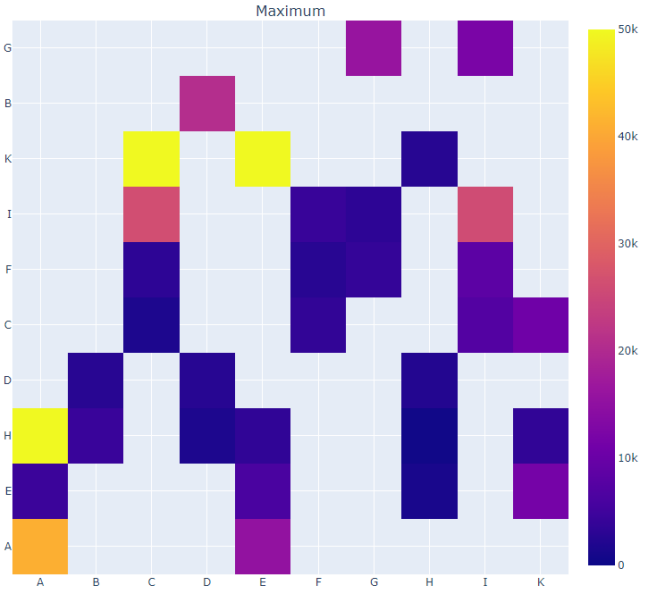


Abbildung 13. Maximum

5. Vorstellung der Lösungen

5.1. Proprietäre Datenstruktur

5.1.1. Einführung in Datenbankstruktur

Datenbankvorstellung

5.1.2. Entwurf

Erstes Datenmodell

5.1.3. Implementierung

Implementierung der Struktur

Implementierung der Datenloader

Implementierung der Analysen

```
@app.route('/')
def home():
    return render_template("inline.html")
```

Evolution des Datenmodells

5.2. Universelle relationale Struktur

5.2.1. Einführung in Datenbankstruktur

Datenbankvorstellung

Grundlage dieser Datenstruktur ist, wie bei der proprietären Struktur, ein relationales Datenmodell. Dieses Datenmodell soll die speziellen Daten beliebiger Struktur aufnehmen.
Jedoch ist im Vergleich zur proprietären Struktur, ein weiteres Ziel, dass die Struktur beliebig, weitere spezielle Daten aufnehmen kann, ohne dass diese erweitert werden muss. Genauer betrachtet, bedeutet dies, dass zur Aufnahme neuer Daten lediglich Data Manipulation Language (DML), anstatt Data Definition Language (DDL), verwendet werden soll.

Listing 1. Code 1 - Beispiel für Data Manipulation Language

```
INSERT INTO Table1 VALUES('Test', 123);
```

Listing 2. Code 2 - Beispiel für Data Definition Language

```
ALTER TABLE Table1 ADD COLUMN zipCode CHAR(5);
```

Um bereits jetzt einen Grund für die Umsetzung und gleichzeitig einen Vorteil dieser Struktur zu nennen, ist es wichtig zu verstehen, dass mit Codebeispiel 1 ausschließlich Tupel in eine Relation geschrieben werden, währenddessen im Codebeispiel 2 die Datenstruktur verändert wird, was viele Systemtest und damit einen Aufwand mit sich bringt.

5.2.2. Entwurf

Konzeptioneller Entwurf

Der konzeptionelle Entwurf basiert auf der Anforderungsanalyse und beschreibt den abzubildenden Weltausschnitt (T. Kudraß, 2015, S.46). "Die Beschreibung erfolgt unabhängig von der Realisierung in einem konkreten Datenbankmanagementsystem (DBMS) und unabhängig von konkreten Anwendungen, um eine stabile Basis für die weiteren Entwurfsphasen zu besitzen." (T. Kudraß, 2015, S.46)
Eine abstrakte Modellierung wurde mittels eines Entity-Relationship-Models (ERM) umgesetzt. (siehe Abbildung 1)

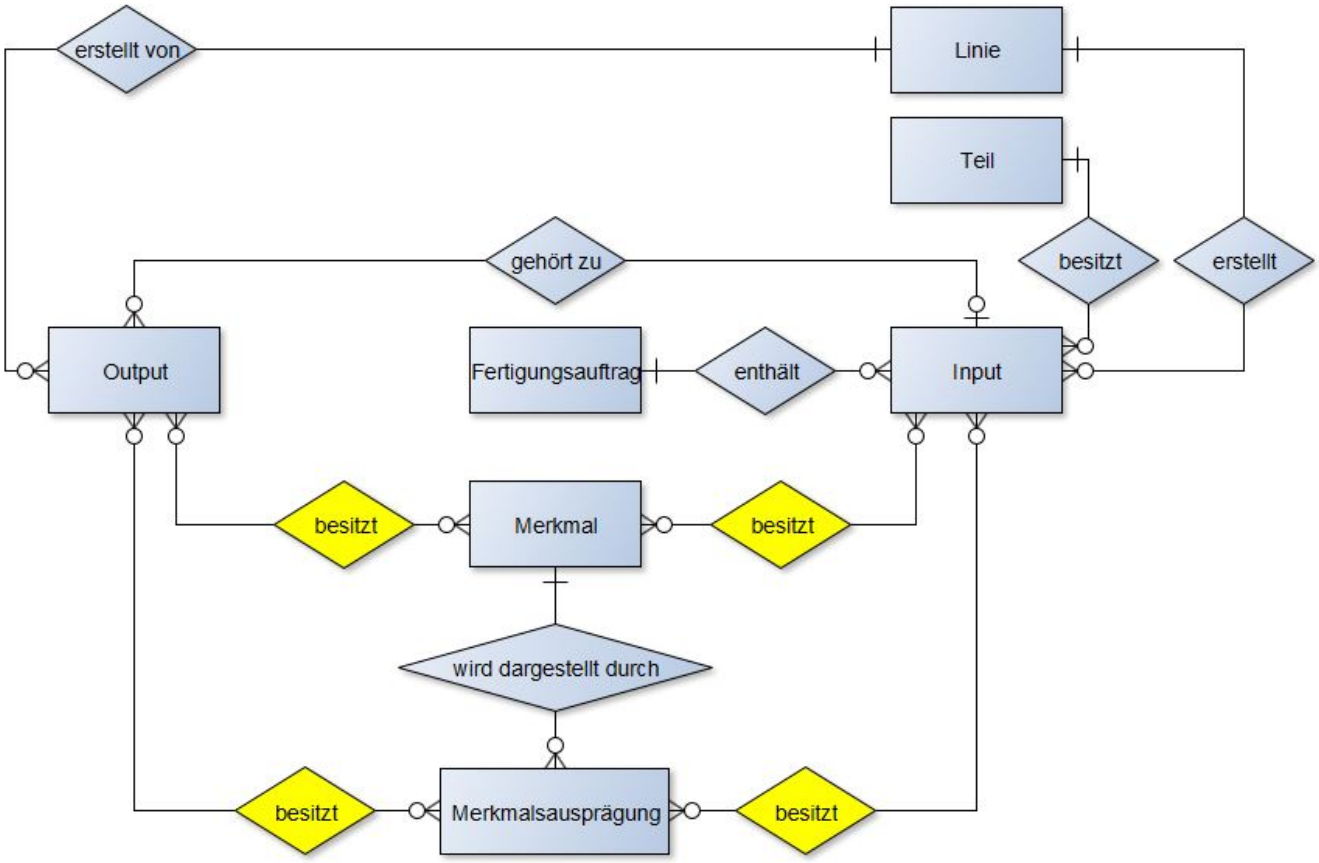


Abbildung 14. Entity-Relationship-Model

In Abbildung 1 kann man drei grundlegende Modellelemente eines ERM erkennen.
Als blaue Rechtecke sind Entitäten (auch Entity) dargestellt, welche im Allgemeinen ein abgrenzbares Objekt der Realität darstellen. Im Fallbeispiel wären dies Input, Output, Fertigungsauftrag, Linie und Teil. Diese Entitäten besitzen normalerweise Attribute bzw. spezielle

Eigenschaften, welche die Entitäten charakterisieren, identifizieren und klassifizieren. Um jedoch eine universelle relationale Struktur zu entwickeln, ist es notwendig die Eigenschaften, welche in keinem Verhältnis zu einer anderen Entität stehen, als eigene Entität zu betrachten. Da Attribute im Gegensatz zu Entitäten Wertausprägungen haben, ist es erforderlich, die Ausprägungen der Entität Merkmal in der Entität Merkmalsausprägung darzustellen.
"Zwischen Entities werden Beziehungen oder Relationships definiert. Eine Beziehung ist die logische Verknüpfung von zwei oder mehreren Entities." (T. Kudraß, 2015, S.51) Diese Beziehung sind in Abbildung 1 als Rauten dargestellt. Die schwarzen Verbindungen zwischen Entitäten und Beziehungen beschreiben die Komplexitätsgrade.
Im Fallbeispiel besitzen nur die Entitäten Input und Output spezielle Eigenschaften. Deshalb haben nur diese Entitäten eine Beziehung (gelb markiert) zu Merkmal und Merkmalsausprägung, aber es kann auch jede andere Entität diese Beziehungen besitzen.

Logischer Entwurf

Ziel des logischen Entwurfs ist es, das konzeptionelle Datenmodell in ein relationales Datenmodell zu überführen.

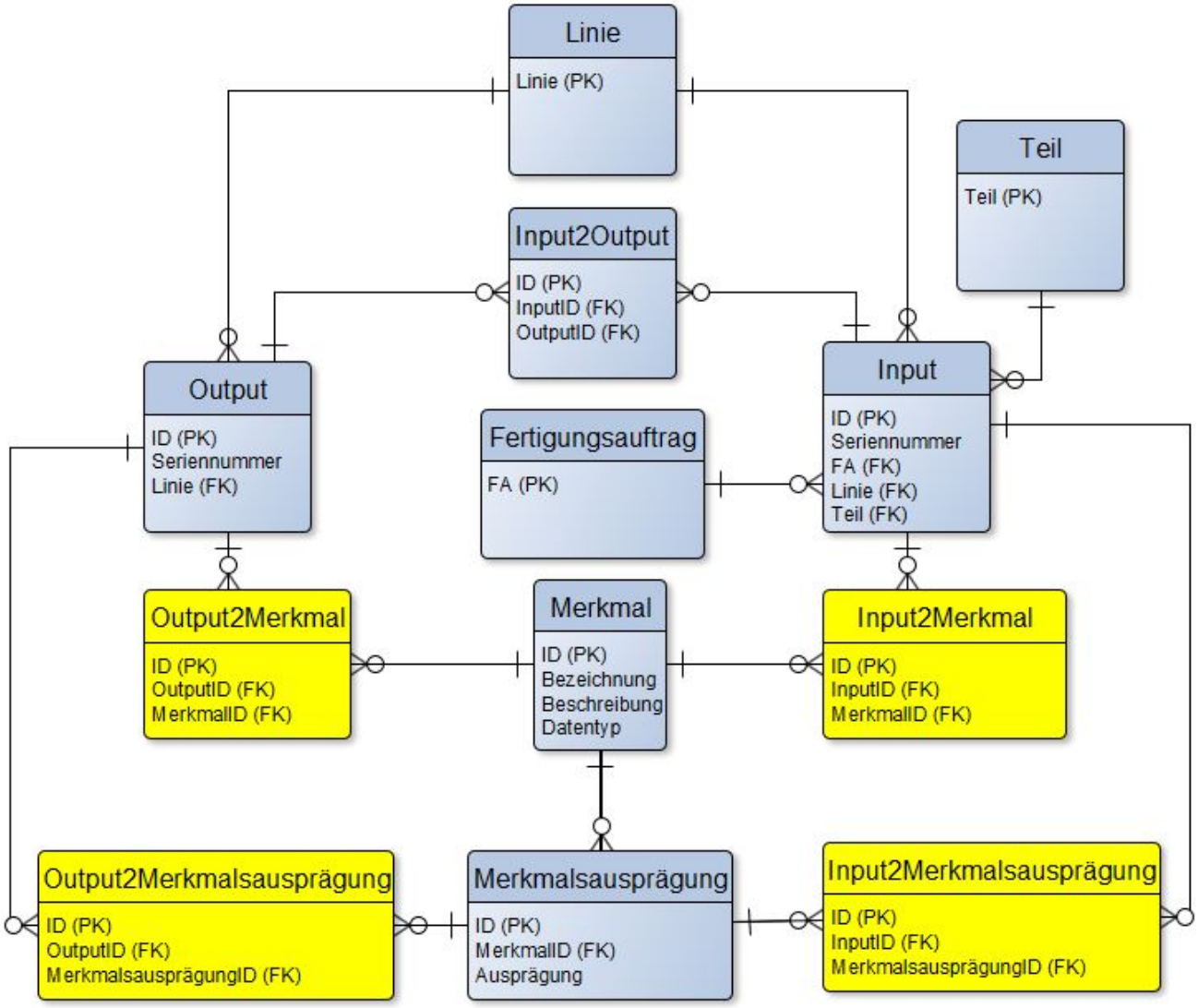


Abbildung 15. Relationenmodell

Die Abbildung 2 zeigt, dass abgeleitete Relationenmodell. Die Beziehungen, welche im konzeptionellen Entwurf noch bestanden, wurden gemäß den Ableitungsregeln aufgelöst. Wichtig

für die universelle relationale Struktur sind insbesondere die gelb markierten Verbindungsrelationen, die auf Grund einer mc:mc Verbindung im konzeptionellen Entwurf entstanden. Diese Verbindungsrelationen definieren die Merkmale eines Tupels und dessen Ausprägungen und bestehen aus einer eindeutigen ID und zwei Fremdschlüsseln. Da jedoch mit jeder Relation, die spezielle Merkmale hat, zwei neue Verbindungsrelationen entstehen, steigt die Komplexität schnell an. Um die Komplexität zu kontrollieren, kann die Ähnlichkeit der Verbindungsrelationen zur Relation Merkmal und die Ähnlichkeit der Verbindungsrelationen zur Relation Merkmalsausprägung genutzt werden, um diese zu vereinen.
In Abbildung 2 wären das Output2Merkmal und Input2Merkmal bezüglich der Relation Merkmal und Output2Merkmalsausprägung und Input2Merkmalsauprägung bezüglich der Relation Merkmalsausprägung, welche zusammengefasst werden können. Beim Zusammenfassen entsteht jedoch das Problem, dass die neu entstandene Relation sich in einem Attribut auf zwei Relationen, in unserem Fall beispielhaft auf Input und Output, bezieht, wodurch ein Tupel beiden Relationen zugeordnet werden kann.
Um dieses Problem zu lösen und dem Ansatz der geringeren Komplexität zu folgen, ist es notwendig die referentielle Integrität aufzulösen.
Daraufhin ergibt sich das Problem, dass ein Tupel der Verbindungsrelation nun nicht mehr eindeutig einer Relation, auf die es sich bezieht, zugeordnet werden kann. Zur eindeutigen Zuordnung wird deshalb ein Diskriminator verwendet.
Ein Diskriminator ist ein Attribut einer Relation, das festlegt, welcher Relation ein Tupel der Verbindungsrelation angehört. Dieses Attribut ist ein Fremdschlüssel, welcher sich in der Diskriminatortabelle definiert. Diese Tabelle bzw. Relation enthält alle definierten Objekttypen der Domäne.
Das angepasste Relationenmodell für die universelle relationale Relationenmodell ist in Abbildung 3 dargestellt.

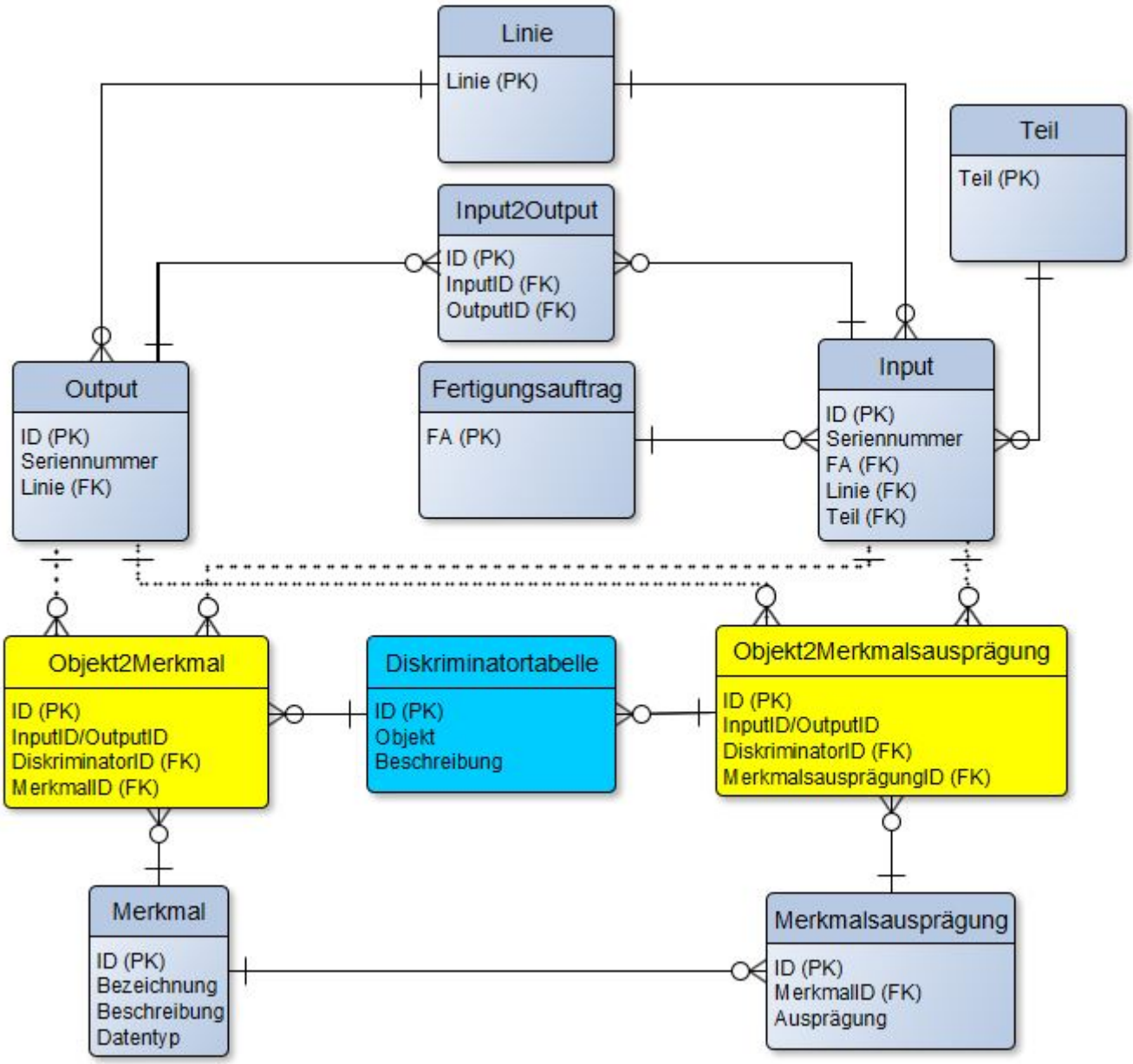


Abbildung 16. universelle relationale Struktur im Relationenmodell

Ein wichtiger Punkt im Relationenmodell ist, dass die Relation Merkmal das Attribut Datentyp besitzt. Grund für dieses Attribut ist, dass in der Relation Merkmalsausprägung alle Ausprägungen als *varchar* gespeichert werden, da diese alle in einem Attribut stehen. Um später den Datentyp einer Ausprägung einfach abfragen zu können, wird er mit dem Merkmal gespeichert.

5.2.3. Implementierung

Systemvoraussetzungen

Alle Implementierungen wurden unter den folgenden Voraussetzungen vorgenommen:

- Windows 10 Pro 64-bit
- Intel Core i5-8265U
- 16 GB RAM

Datenbank

Allgemein

Die universelle relationale Struktur kann einfach über ein relationales DBMS (RDBMS) realisiert werden.
Es wurden der MySQL Server 8.0, als Open-Source Variante, und der kostenlose SQL Server Express 2017 (MSSQL-Server) von Microsoft genutzt.
Beide Systeme können einfach durch die im Internet bereitgestellten Installer der Hersteller installiert und mit Hilfe eines Assistenten konfiguriert und danach genutzt werden.
Für das Anlegen der Datenstruktur ist es wichtig zu wissen, dass die Hersteller unterschiedliche Dialekte der Structured Query Language (SQL) nutzen.
Die folgenden beispielhaften Quelltexte beziehen sich, sofern nicht anders angegeben, auf den MySQL-Dialekt.

Listing 3. Code 3 - Anlegen der Tabelle Input mittels Data Definition Language

```
CREATE TABLE Input (
  ID    int(10) NOT NULL AUTO_INCREMENT,
  SNR   char(18) NULL,
  FA    char(20) NULL,
  TEIL  char(1) NULL,
  LINIE char(1) NULL,
  PRIMARY KEY (ID));
```

Listing 4. Code 4 - Anlegen eines Fremdschlüssels zwischen den Tabellen Input und Teil

```
ALTER TABLE Input ADD CONSTRAINT FKInput_TEIL FOREIGN KEY (TEIL) REFERENCES TEIL (
TEIL);
```

Da bei den Analysen, deren Implementierung später noch beschrieben wird, große Datenmengen der Tabellen abgefragt und teilweise über JOINS miteinander verbunden werden, spielen Indizes eine wichtige Rolle, um die Datenabfragen performant zu gestalten. Durch Indizes wird, vereinfacht beschrieben, das vollständige Durchsuchen einer Menge von Tupeln (FullTableScan) vermieden, da bestimmte Attributwerte in einem B-Baum gespeichert werden, was die Laufzeit verringert.

Listing 5. Code 5 - Anlegen von Indizes für die Tabelle Input

```
CREATE INDEX INDEX_TBL_Input ON Input (SNR);
CREATE INDEX INDEX_TBL_Input2 ON Input (TEIL, SNR);
CREATE INDEX INDEX_TBL_Input3 ON Input (FA, SNR);
CREATE INDEX INDEX_TBL_Input4 ON Input (LINIE);
```

Die Auswahl der Attribute, welche in einen Index aufgenommen werden, steht in starkem Zusammenhang, mit den abgefragten Spalten in den Analysen. Beispielsweise bezieht sich Analyse 1 auf die Teilart, weshalb *INDEX_TBL_Input2* angelegt wurde.
An dieser Stelle ist es wichtig zu erwähnen, dass Microsoft mit seinem SQL-Dialekt dem Nutzer

mehr Möglichkeiten gibt einen Index zielgenauer zu definieren.

Listing 6. Code 6 - Anlegen von Indizes mittels Microsoft SQL-Dialekt

```
CREATE INDEX INDEX_TBL_Input2 ON Input (TEIL) INCLUDE (FA);
CREATE INDEX INDEX_TBL_MA ON Merkmalsausprägung (MerkmalID) INCLUDE (Ausprägung) WHERE
MerkmalID = 21;
```

Über die INCLUDE-Klausel ist es möglich Werte zu speichern und schnell abzufragen, welche nicht im Indexschlüssel stehen.
Über die WHERE- Klausel ist es möglich einen Index nur für gefilterte Werte zu erstellen.

Die meisten Tupel der Struktur werden dynamisch über die Datenloader erstellt, sofern ein neues Tupel eines Objekts angelegt wird. Doch bevor dieses Laden geschehen kann, müssen die Merkmale der speziellen Daten in der Tabelle Merkmal und die Objekttypen definiert sein. Dabei müssen der Tabelle Merkmal Werte für Bezeichnung, Beschreibung und Datentyp übergeben werden (Code 7). Das Attribut ID muss im MySQL-Dialekt nicht mittels *NULL* übergeben werden.

Listing 7. Code 7 - Anlegen der Merkmale mittels DML

```
INSERT INTO Merkmal VALUES ('DateIn', 'Zeitstempel der Prüfdaten', 'timestamp');
INSERT INTO Merkmal VALUES ('NR', 'Eingangszähler', 'int');
INSERT INTO Merkmal VALUES ('E', 'GreiferID', 'string');
```

Listing 8. Code 8 - Anlegen der Objettypen mittels DML

```
INSERT INTO ObjektTyp VALUES ('Input', 'Input Datensätze');
INSERT INTO ObjektTyp VALUES ('Output', 'Output Datensätze');
```

Messung der Datenbankausführungszeiten der Analysen

MySQL

Der MySQL-Server stellt standardmäßig die Status der 100 zuletzt ausgeführten Queries in der Systemtabelle *INFORMATION_SCHEMA.PROFILING* mit bestimmten Merkmalen bereit, sofern das Profiling aktiviert wurde (siehe Code 9).

Listing 9. Code 9 - Aktivieren des Profilings im MySQL-Server

```
SET @@profiling = 1;
```

Normalerweise verfügt diese Variante über die Möglichkeit, die Größe der Historie (auch *profiling_history_size*) zu bestimmen (siehe Code 10). Jedoch funktionierte dies im Projekt unzuverlässig, weshalb immer der Standardwert von 100 genutzt wurde, um die Zeiten zuverlässig zu messen und die ausgeführten Queries zu zählen (siehe Code 11).

Listing 10. Code 10 - Setzen der Query-Historie auf 500

```
SET @@profiling_history_size = 500;
```

Listing 11. Code 11 - Messen der Ausführungszeiten und Zählen der ausgeführten Queries

```
SELECT SUM(DURATION) FROM INFORMATION_SCHEMA.PROFILING;
SELECT COUNT(Query_ID) FROM INFORMATION_SCHEMA.PROFILING WHERE STATE = 'end';
```

Das Zurücksetzen der Historie kann einfach über die folgende Befehlsfolge im Codeabschnitt 12 erfolgen.

Listing 12. Code 12 - Initialisieren der Systemtabelle

```
SET @@profiling = 0;
SET @@profiling_history_size = 0;
SET @@profiling_history_size = 100;
SET @@profiling = 1;
```

MSSQL

Für die Nutzung des SQL Server Express 2017 wurde das Microsoft SQL Server Management Studio 17 genutzt. Diese Software ermöglicht eine einfache Administration des Datenbankservers.
Über den integrierten *XEventProfiler* können, ab Aufruf des Profilers, alle Events und Queries des Datenbankservers bzw. einer Datenbank, welche in diesem Zeitraum stattfinden, getrackt werden.
Da Systemevents während der Ausführung auftreten, muss nach dem Stoppen des Datenfeed die Ergebnismenge nach dem *client_app_name* gruppiert werden, um nur die gewünschten Ereignisse auszuwerten. Nach der Gruppierung ist noch eine Aggregation zur Summe des Feldes *duration* möglich, um die Ausführungszeit direkt abzulesen.

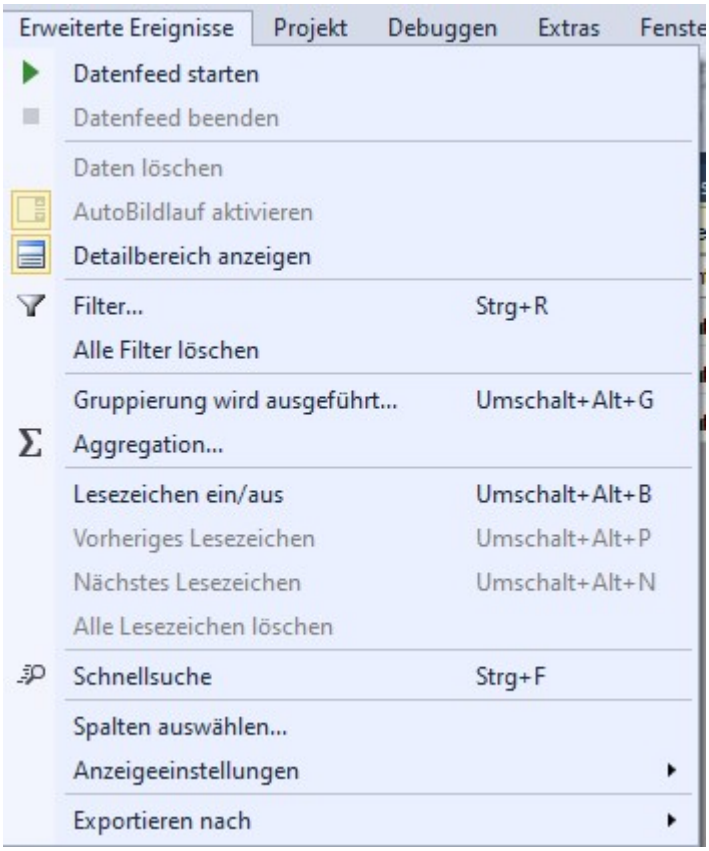


Abbildung 17. Menü zur Gruppierung und Aggregation der getrackten Queries im Microsoft SQL Server Management Studio 17

Lessons Learned

Nachdem in beiden Systemen dieselbe Struktur mit gleichen Indizes (auf Basis MySQL) erstellt wurde und erste Analysen gefahren wurden, zeigte sich, dass drei der fünf Analysen auf dem MSSQL-Server langsamer liefen. Nach der Fehlersuche stellte sich heraus, dass der Buffer des MSSQL-Servers, mit 1.4 GB, sehr schnell aufgebraucht ist. Da es sich um eine kostenlose Variante von Microsoft handelt, besteht keine Möglichkeit diesen Buffer zu erhöhen.
Da beim MySQL-Server der Buffer auch noch nicht betrachtet wurde, wurde hier die Größe überprüft (800 MB voreingestellt) und auf 6 GB erhöht. Die Erhöhung ist möglich, indem in der Datei */ProgramData/MySQL/MySQLServer8.0/my.ini* die Variable *innodb_buffer_pool_size* auf *6G* gesetzt wurde. Wichtig ist dabei, dass die Datei mit Rechten des Administrators geändert werden muss.
Durch diese Veränderung ließ sich eine starke Senkung in den Ausführungszeiten der Analysen erreichen (siehe Tabelle 1 und Bild).

Tabelle 1. Ausführungszeiten MySQL-DB in Abhängigkeit von der Puffergröße

Analyse	Ausführungszeit 800MB Puffer	Ausführungszeit 6GB Puffer	Senkung
001	19 min : 19 sek	06 min : 12 sek	67.9 %
002	00 min : 56 sek	00 min : 07 sek	86.6 %
004	23 min : 20 sek	06 min : 24 sek	72.6 %
005	31 min : 49 sek	07 min : 28 sek	76.5 %
007	07 min : 31 sek	00 min : 43 sek	90.4 %

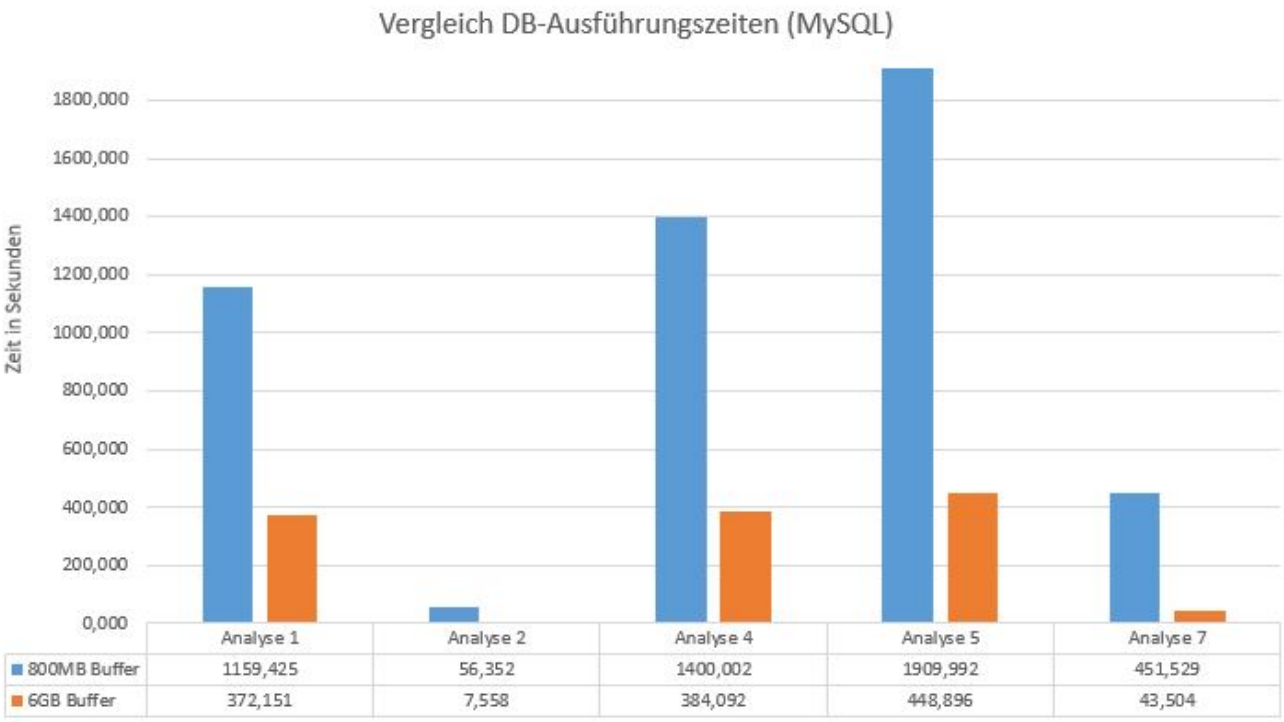


Abbildung 18. Ausführungszeiten MySQL-DB in Abhängigkeit von der Puffergröße

Anwendungen

Programmiersprache

Zur Implementierung der Anwendungen wurde die Programmiersprache Python verwendet.
Im Projekt wurde Visual Studio Code als Entwicklungsumgebung (IDE) genutzt, welche es ermöglicht, einfach die Python-Extension herunterzuladen und zu nutzen.
Für die Implementierung wurde die Python-Version 3.7.3 genutzt.

Verwendete Bibliotheken zur Kommunikation zwischen Anwendung und Datenbank

MySQL

Zur Verbindung zwischen Anwendung und MySQL-Datenbankserver wurde die Python-Bibliothek *mysql-connector-python* genutzt. Diese kann in Visual Studio Code über die Konsole durch den im Beispiel 13 dargestellten Code installiert werden.

Listing 13. Code 13 - Installieren der MySQL-Bibliothek für Python

```
pip install mysql-connector-python
```

Damit die Anwendung eine Verbindung zur Datenbank herstellt, muss die Bibliothek eingebunden und die Parameter *user*, *password*, *host* und *database* übergeben werden. Um Operationen ausführen zu können, muss ein Cursor genutzt werden. (siehe Code 14)

Listing 14. Code 14 - Herstellen der Verbindung und Erstellen eines Cursors

```
import mysql.connector

connection = mysql.connector.connect(user = "root", password = "demo", host =
"127.0.0.1",  database = "project")
cursor = connection.cursor()
```

Für SELECT-Abfragen muss nun lediglich ein Statement der Cursor-Funktion *execute* übergeben werden, damit die Abfrage ausgeführt wird. Der Cursor bietet drei Methoden, um zu definieren, welche Menge der Ergebnismenge bereitgestellt wird:

- *fetchall()* für die komplette Ergebnismenge
- *fetchone()* für die erste Zeile der Ergebnismenge
- *fetchmany(size = x)* für die ersten x Zeilen der Ergebnismenge

Listing 15. Code 15 - Ausführen einer Abfrage und Fetch der kompletten Ergebnismenge

```
statement = "SELECT Input.FA FROM Input WHERE TEIL = 'A' GROUP BY Input.FA ORDER BY
Input.FA;"
  cursor.execute(statement)
  FA_List = cursor.fetchall()
```

Sofern ein Insert, Update oder Delete durchgeführt wurde, muss nach der Ausführung mittels *execute()* ein Commit erfolgen, um die Änderungen zu übernehmen. (siehe Code 16)

Listing 16. Code 16 - Verbindungscommit nach Insert-Anweisung

```
statement = "INSERT INTO LINIE VALUES (1);"
  cursor.execute(statement)
  connection.commit()
```

Am Ende der Anwendung können der Cursor und die Verbindung einfach über die Funktion *close()* geschlossen werden. (siehe Code 17)

Listing 17. Code 17 - Schließen des Cursors und Abbau der Verbindung

```
cursor.close()
connection.close()
```

Genauere Ausführungen und weitere Informationen sind in der MySQL-Dokumentation verfügbar.

MSSQL

Zur Verbindung zwischen Anwendung und MSSQL-Datenbankserver wurde die Python-Bibliothek *pyodbc* genutzt. Diese kann in Visual Studio Code über die Konsole durch den im Beispiel 18 dargestellten Code installiert werden. Außerdem muss der "Microsoft ODBC Driver for SQL Server", welcher in der Microsoft Dokumentation zu finden ist (ODBC Driver), installiert werden.

Listing 18. Code 18 - Installieren der pyodbc-Bibliothek für Python

```
pip install pyodbc
```

Im Unterschied zu MySQL muss zum Verbindungsaufbau noch der weitere Parameter *DRIVER* übergeben werden. Um Operationen ausführen zu können, muss auch hier ein Cursor genutzt werden. (siehe Code 19)

Listing 19. Code 19 - Herstellen der Verbindung und Erstellen eines Cursors

```
import pyodbc

connection = pyodbc.connect(driver = '{ODBC Driver 17 for SQL Server}', server =
'Desktop\\SQLEXPRESS' , database = 'project', UID = 'root', PWD = 'demo')
cursor = connection.cursor()
```

Alle weiteren im MySQL-Teil ausgeführten Befehle gelten unter pyodbc ebenfalls in der gleichen Form.

Messung der Skriptausführungszeiten

Zur Messung der Skriptausführungszeiten wurde von der Python-Bibliothek *time* die Methode *process_time_ns()* geladen, mit der die Summe der System- und Benutzer-CPU-Zeit des aktuellen Prozesses in Nanosekunden berechnet werden kann. Diese Methode schließt die während des Ruhezustands verstrichene Zeit nicht ein.

Listing 20. Code 20 - Messen der Skriptausführungszeit

```
from time import process_time_ns()

start = process_time_ns()
# Code der auszuführen ist
stop = process_time_ns()

duration = stop - start
```

Datenloader

Die Datenloader, über die Datensätze in die Struktur geladen werden, unterscheiden sich auf Grund der unterschiedlichen SQL-Dialekte. Jedoch ist das allgemeine Vorgehen, welches hier erläutert wird, gleich. Ein kleiner Unterschied liegt nur in der Verknüpfung des Outputs mit dem Input, was später erläutert wird.
Voraussetzung, bevor Datensätze eingelesen werden können, ist wie bereits erwähnt, dass Merkmale und Objekttypen bereits in der Struktur definiert wurden.

Aus dem bereits erläuterten Watchdog, erhält die Anwendung den Pfad des Textdokuments, welches ausgelesen werden muss. Im Codebeispiel 21 ist dargestellt, wie eine Datei mit Leserechten geöffnet wird, der Inhalt mittels *read()* ausgelesen und als String gespeichert wird und dieser String aufgearbeitet wird, dass alle Elemente, die durch ein Semikolon getrennt sind, ein Element in einer

Liste werden.

Listing 21. Code 21 - Auslesen der vorhandenen Datei

```
def insert (file):
  datei = open(file,'r')
  values = datei.read()
  data = values.split(';')
```

Die Verfahren zum Einlesen der Input- und Output-Datensätze sind sehr ähnlich. Deshalb wurden die Verfahren zusammen in den Bildern 6 und 7 dargestellt.

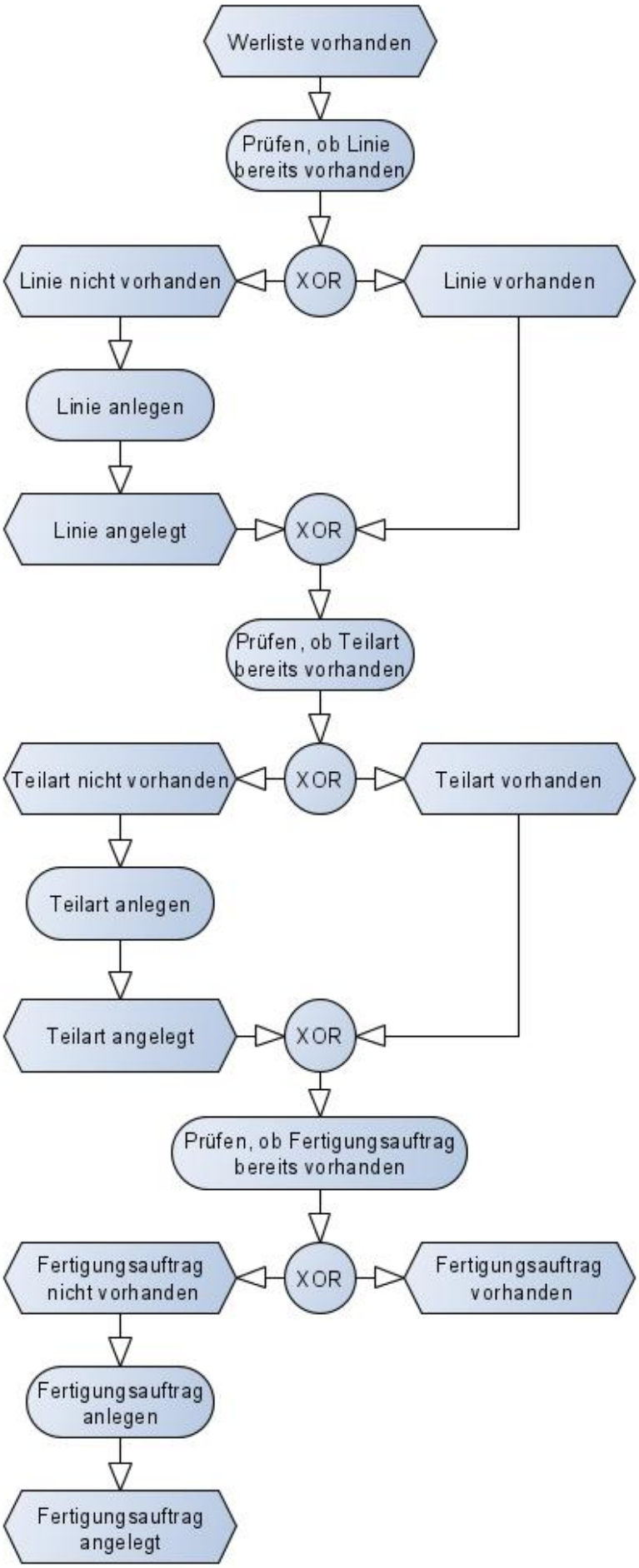


Abbildung 19. ereignisgesteuerte Prozesskette zur Darstellung des Einlesens von Werten anderer

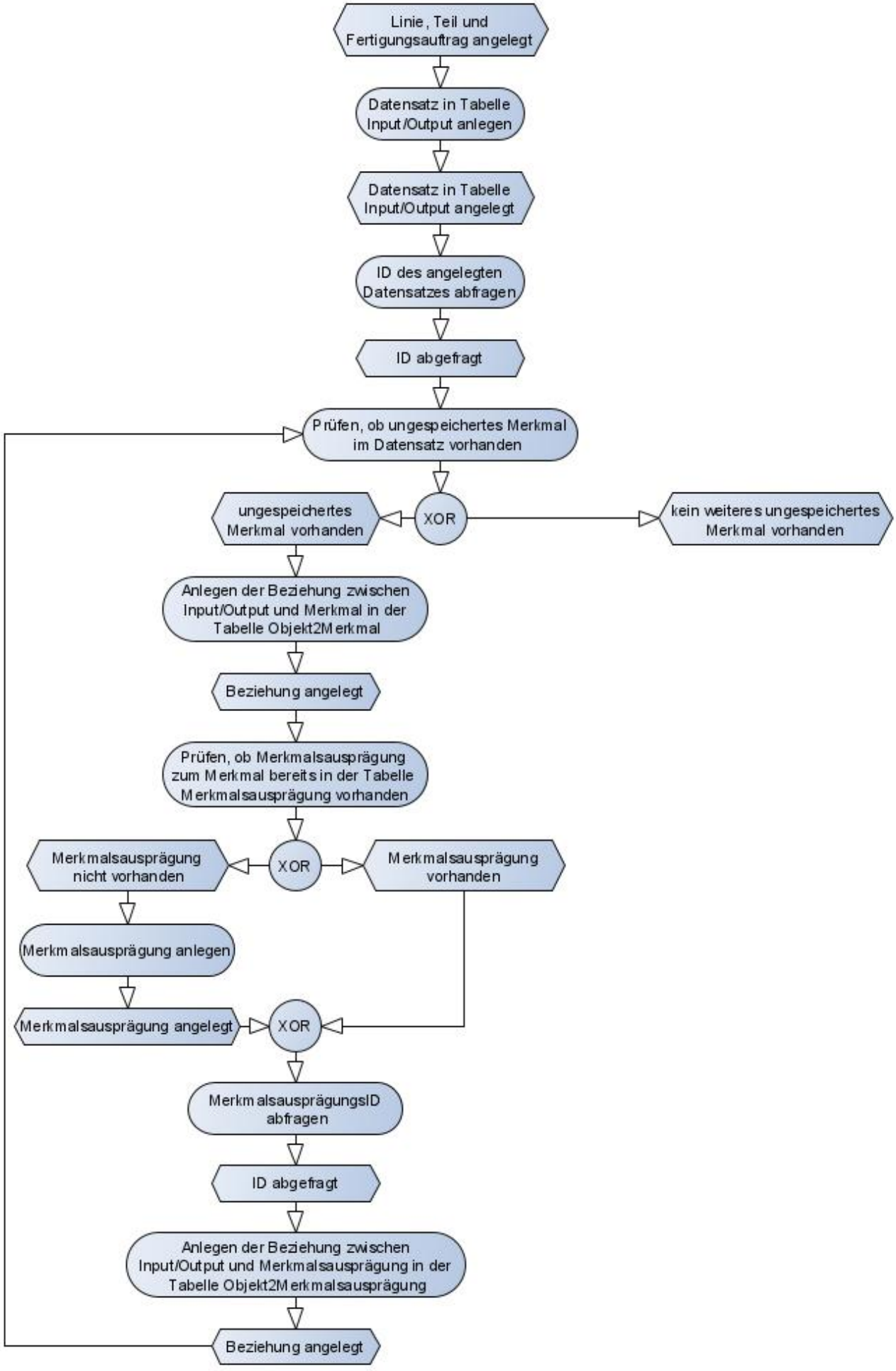

Linie, Teil und Fertigungsauftrag angelegt
Datensatz in Tabelle Input/Output anlegen
Datensatz in Tabelle Input/Output angelegt
ID des angelegten Datensatzes abfragen
ID abgefragt
Prüfen, ob ungespeichertes Merkmal im Datensatz vorhanden
ungespeichertes Merkmal vorhanden
XOR
kein weiteres ungespeichertes Merkmal vorhanden
Anlegen der Beziehung zwischen Input/Output und Merkmal in der Tabelle Objekt2Merkmal
Beziehung angelegt
Prüfen, ob Merkmalsausprägung zum Merkmal bereits in der Tabelle Merkmalsausprägung vorhanden
Merkmalsausprägung nicht vorhanden
XOR
Merkmalsausprägung vorhanden
Merkmalsausprägung anlegen
Merkmalsausprägung angelegt
XOR
MerkmalsausprägungsID abfragen
ID abgefragt
Anlegen der Beziehung zwischen Input/Output und Merkmalsausprägung in der Tabelle Objekt2Merkmalsausprägung
Beziehung angelegt

Abbildung 20. ereignisgesteuerte Prozesskette zur Darstellung des Einlesens spezieller Merkmale sind

Nachdem ein Output-Datensatz in der Tabelle Output angelegt worden ist (Bild 7, Ereignis 2), muss dieser noch, sofern möglich, mit einem Input-Datensatz verknüpft werden. Dies passiert über die Zeitstempel der Datensätze. Sofern es für die Seriennummer nur einen Input-Datensatz gibt, erfolgt eine direkte Verknüpfung, außer die Zeitdifferenz zwischen Output und Input ist negativ. Sollten jedoch mehrere Input-Datensätze zu einer Seriennummer vorhanden sein, muss die Zeitdifferenz zwischen Output und jedem passenden Input berechnet werden. Dabei wird der Output mit dem Input verknüpft zu dem die kleinste nicht negative Differenz besteht.

Analysen

Allgemein

Bevor mit der Implementierung der vorgegebenen Analysen begonnen wurde, wurde über Möglichkeiten der Realisierung in Python nachgedacht. Grundsätzlich lassen sich drei Varianten realisieren, welche mit ihren Vor- und Nachteilen, die sich auch auf LessonsLearned des Projekts zurückführen lassen, in der folgenden Tabelle dargestellt sind.

Tabelle 2. Realisierungsmöglichkeiten der Analysen

	(1) kleine Abfragen mit genauen WHERE-Klauseln (bspw. je SNR)	**(2) mittlere Abfragen mit Mengen in WHERE-Klauseln (bspw. je FA)**	**(3) große Abfragen**
Vorteile	• gesamtes Vorgehen einfach nachvollziehbar • geringer Aufwand in Programmiersprache • verständlichere SQL-Abfragen	• geringere Netzwerklast als bei kleinen häufigen Abfragen • Verteilung der Komplexität in Abfragen und Programmiersprache	• einmalige Netzwerklast
Nachteile	• Netzwerklast dauerhaft • in Summe höhere Abfragezeiten auf der Datenbank	• dauerhafte Netzwerklast größer als bei großen Abfragen	• Vorgehen schwerer nachvollziehbar • höherer Aufwand in Programmiersprache • Gruppierungen, die bereits einfach mit SQL gelöst werden können, müssen in der Programmiersprache erledigt werden

Zum Test wurde versucht über jede Variante eine vordefinierte Datenmenge aus der Datenbank abzufragen. Da sich die Zeitergebnisse für diese Datenmenge nur gering unterschieden, wurde entschieden, um die unbekannten Analysen vorerst in kleinen logischen Schritten zu lösen, dass die Variante 1 umgesetzt wird.

Nach Fertigstellung der Variante 1 für jede Analyse wurde zum Vergleich Variante 2 für die Analysen 1, 4 und 5 umgesetzt, da dort relativ lange Zeiten auftraten.

In den folgenden Absätzen werden kurz selbstdefinierte Funktionen gezeigt und das Vorgehen in den Analysen für die verschiedenen Varianten als Pseudocode, zur einfachen Verständlichkeit erläutert.

Für die Realisierung der Variante 2 wurde die Python-Bibliothek *pandas* genutzt, welche einfache und flexible Möglichkeiten der Datenanalyse und -manipulation bietet.

Listing 22. Code 22 - Installieren der pandas-Bibliothek für Python

```
pip install pandas
```

Eigene Funktionen

Zur Umsetzung der Implementierungen wurden zwei selbstdefinierte Funktionen genutzt.
Zum einen eine Funktion, um Datumswerte, welche in der Struktur als *VARCHAR* gespeichert sind, in Sekunden für die Zeitdifferenzberechnung umzuwandeln. (siehe Code 23)

Listing 23. Code 23 - Umwandeln eines Datumsstrings in Sekunden

```
import datetime, time

def convert_from_datestring( TimeString ):
  Date = datetime.datetime.strptime(TimeString, "%Y-%m-%dT%H:%M:%S.%f")
  Second = time.mktime(Date.timetuple())
  return Second
```

Zum anderen wurde eine Funktion zur Umwandlung der Zeitdifferenzen in Sekunden verwendet, um diesen Wert in einen einfach menschlichen String bestehend aus Tagen, Stunden, Minuten und Sekunden umzurechnen. (siehe Code 24)

Listing 24. Code 24 - Umwandeln eines Sekundenwerts in einen einfach lesabren String

```
def convert_from_s( seconds ):
  minutes, seconds = divmod(seconds, 60)
  hours, minutes = divmod(minutes, 60)
  days, hours = divmod(hours, 24)
  string = str(int(days))+"T:"+str(int(hours))+"h:"+str(int(minutes))+"m:"+str(int
(seconds))+ "s"
  return string
```

Analyse 1 - Taktung pro Artikel

Variante 1 - kleine Abfragen

Listing 25. Code 25 - Pseudocode Analyse 1.1

```
Abfrage aller Teilarten;

FOR EACH Teilart der Teilarten {
  Anzahl gefertigter Teile pro Teiltyp ermitteln;
  Fertigungsaufträge des Teiltyps abfragen;

  FOR EACH Fertigungsauftrag in Fertigungsaufträgen {
    Anzahl gefertigter Teile pro Fertigungsauftrag ermitteln;
    Alle Seriennummern abfragen, die mehr als einen Input in diesem Fertigungsauftrag
haben (Auschuss);

    FOR EACH Seriennummer im Ausschuss {
      Anzahl Inputs für Seriennummer finden;
    }

    Minimum, Maximum, Durchschnitt des Ausschusses bestimmen;
    InputID's abfragen, die einen Output haben, zum Fertigungsauftrag gehören und eine
Seriennummer haben;

    FOR EACH InputID in InputID's {
      Input-Zeit abfragen und konvertieren;
      Alle OutputID's für Input ID abfragen;

      FOR EACH OutputID in OutputID's {
        Output-Zeit abfragen, konvertieren und Differenz zu Input-Zeit berechnen;
      }

      Maximum der Differenzen bestimmen;
    }

    Minimum, Maximum, Durchschnitt aller Differenzen pro Fertigungsauftrag bestimmen;
    Ausgabe pro Fertigungsauftrag;
  }
}
```

Variante 2 - mittlere Abfragen

Listing 26. Code 26 - Pseudocode Analyse 1.2

```
Abfrage aller Teilarten;

FOR EACH Teilart der Teilarten {
  Anzahl gefertigter Teile pro Teiltyp ermitteln;
  Fertigungsaufträge des Teiltyps abfragen;

  FOR EACH Fertigungsauftrag in Fertigungsaufträgen {
    Anzahl gefertigter Teile pro Fertigungsauftrag ermitteln;
    Alle InputID's mit ihren Input-Zeitstempeln abfragen;
    Für alle InputID's den maximalen Output-Zeitstempel (über erstellte Verknüpfung)
ermitteln;

    FOR EACH InputID in InputID's {
      Suche des passenden Outputs in Outputs;
      Zeitstempel konvertieren und Differenz berechnen;
    }

    Minimum, Maximum, Durchschnitt aller Differenzen pro Fertigungsauftrag bestimmen;
    Alle Seriennummern abfragen, die mehr als einen Input in diesem Fertigungsauftrag
haben (Auschuss);
    Anzahl des Ausschusses pro Seriennummer für alle Seriennummern abfragen;
    Minimum, Maximum, Durchschnitt des Ausschusses bestimmen;
    Ausgabe pro Fertigungsauftrag;
  }
}
```

Analyse 2 - Auftrennung

Listing 27. Code 27 - Pseudocode Analyse 2

```
Abfrage aller Teilarten;

FOR EACH Teilart der Teilarten {
  Anzahl gefertigter Teile pro Teiltyp ermitteln;
  Fertigungsaufträge des Teiltyps abfragen;

  FOR EACH Fertigungsauftrag in Fertigungsaufträgen {
    Anzahl gefertigter Teile pro Fertigungsauftrag ermitteln;
    Alle Seriennummern abfragen, die mehr als einen Input in diesem Fertigungsauftrag
haben (Auschuss);

    FOR EACH Seriennummer im Ausschuss {
      InputID's und die Zeitstempel ermitteln;

      FOR EACH InputID in InputID's {
        Output-Zeitstempel der InputID abfragen;

        IF kein Output-Zeitstempel vorhanden {
          nächste InputID;
        }

        IF aktuelle InputID nicht die Letzte {
          Output-Zeitstempel der InputID konvertieren;
          Input-Zeitstempel der nächsten InputID konvertieren;
          Differenz berechnen;
        }
      }
    }
  }
  Minimum, Maximum, Durchschnitt aller Differenzen pro Teilart bestimmen;
  Ausgabe pro Teilart;
}
```

Analyse 4

Variante 1 - kleine Abfragen

Listing 28. Code 28 - Pseudocode Analyse 4.1

```
Abfrage aller LadungsträgerIn;

FOR EACH Ladungsträger der LadungsträgerIn {
  Anzahl gefertigter Teile pro Ladungsträger ermitteln;
  InputID's des aktuelle Ladungsträgers abfragen;

  FOR EACH InputID in InputID's {
    Input-Zeitstempel abfragen und konvertieren;
    OutputID's zur aktuellen InputID ermittlen;

    FOR EACH OutputID in OutputID's {
      Output-Zeitstempel abfragen und konvertieren;
    }
  }

  Minimum Input-Zeitstempel bestimmen;
  Maximum Output-Zeitstempel bestimmen;
  Differenz berechnen;
  Ausgabe pro Ladungsträger;
}
```

Variante 2 - mittlere Abfragen

Listing 29. Code 29 - Pseudocode Analyse 4.2

```
Abfrage aller LadungsträgerIn;

FOR EACH Ladungsträger der LadungsträgerIn {
  Anzahl gefertigter Teile pro Ladungsträger ermitteln;
  InputID's des aktuelle Ladungsträgers abfragen;
  minimalen Input-Zeitstempel der InputID's abfragen;
  maximalen Output-Zeitstempel der mit den InputID's verknüpften Outputs ermitteln;
  Differenz berechnen;
  Ausgabe pro Ladungsträger;
}
```

Analyse 5

Variante 1 - kleine Abfragen

Listing 30. Code 30 - Pseudocode Analyse 5.1

```
Abfrage aller Teilarten;

FOR EACH Teilart der Teilarten {
  genutzte LadungsträgerIn für den Teiltyp abfragen;

  FOR EACH Ladungsträger in LadungsträgerIn {
    Anzahl gefertigter Stücke pro Ladungsträger des Teiltyps ermitteln;
    InputID's des Teiltypen abrufen, die auf dem aktuellen Ladungsträger gefertigt
wurden;

    FOR EACH InputID in InputID's {
      Input-Zeitstempel abfragen und konvertieren;
      OutputID's zur InputID abfragen;

      FOR EACH OutputID in OutputID's {
        Output-Zeitstempel abfragen und konvertieren;
        Differenz zwischen Output und Input berechnen;
      }
      Maximum der Differenzen bestimmen;
    }
    Minimum, Maximum, Durchschnitt aller Differenzen pro Ladungsträger bestimmen;
    Ausgabe pro Ladungsträger;
  }
}
```

Variante 2 - mittlere Abfragen

Listing 31. Code 31 - Pseudocode Analyse 5.2

```
Abfrage aller Teilarten;

FOR EACH Teilart der Teilarten {
  genutzte LadungsträgerIn für den Teiltyp abfragen;

  FOR EACH Ladungsträger in LadungsträgerIn {
    Anzahl gefertigter Teile pro Ladungsträger des Teiltyps ermitteln;
    InputID's des Teiltypen abrufen, die auf dem aktuellen Ladungsträger gefertigt
wurden;
    alle InputID's mit ihren Input-Zeitstempeln abfragen;
    maximale Output-Zeitstempel der mit den InputID's verknüpften Outputs ermitteln;
    Differenzen berechnen zwischen zusammengehörigen Outputs und Inputs;
    Minimum, Maximum, Durchschnitt ermitteln;
    Ausgabe pro Ladungsträger;
  }
}
```

Analyse 6

Listing 32. Code 32 - Pseudocode Analyse 6

```
Abfrage aller Linien;

FOR EACH Linie in Linien {
  Fertigungsaufträge der Linie abfragen;

  FOR EACH Fertigungsauftrag in Fertigungsaufträge {
    alle Input-Zeitstempel und Teilart des Fertigungsauftrags abfragen;
    minimalen und maximalen Input-Zeitstempel mit Teilart als ein Element in einer
Liste speichern;
  }
  Liste nach minimaler Input-Zeit sortieren;

  FOR EACH Element der Liste {
    maximalen Input-Zeitstempel des aktuellen Elements konvertieren;
    minimalen Input-Zeitstempel des nächsten Elements konvertieren;
    Differenz zwischen maximalen Input-Zeitstempel des aktuellen Elements und
minimalen Input-Zeitstempel des nächsten Elements bilden;
    IF Differenz positiv {
      Wechsel der Teilart mit Differenzzeit notieren;
    }
  }

  Minimum, Maximum, Durchschnitt der Wechselzeiten pro Linie berechnen;
}
```

Auswertung

Nach Messung aller Ausführungszeiten ergab sich eine deutliche Senkung der Datenbankausführungszeiten durch die Umstellung der Analysevorgehen. (siehe Bild 8)

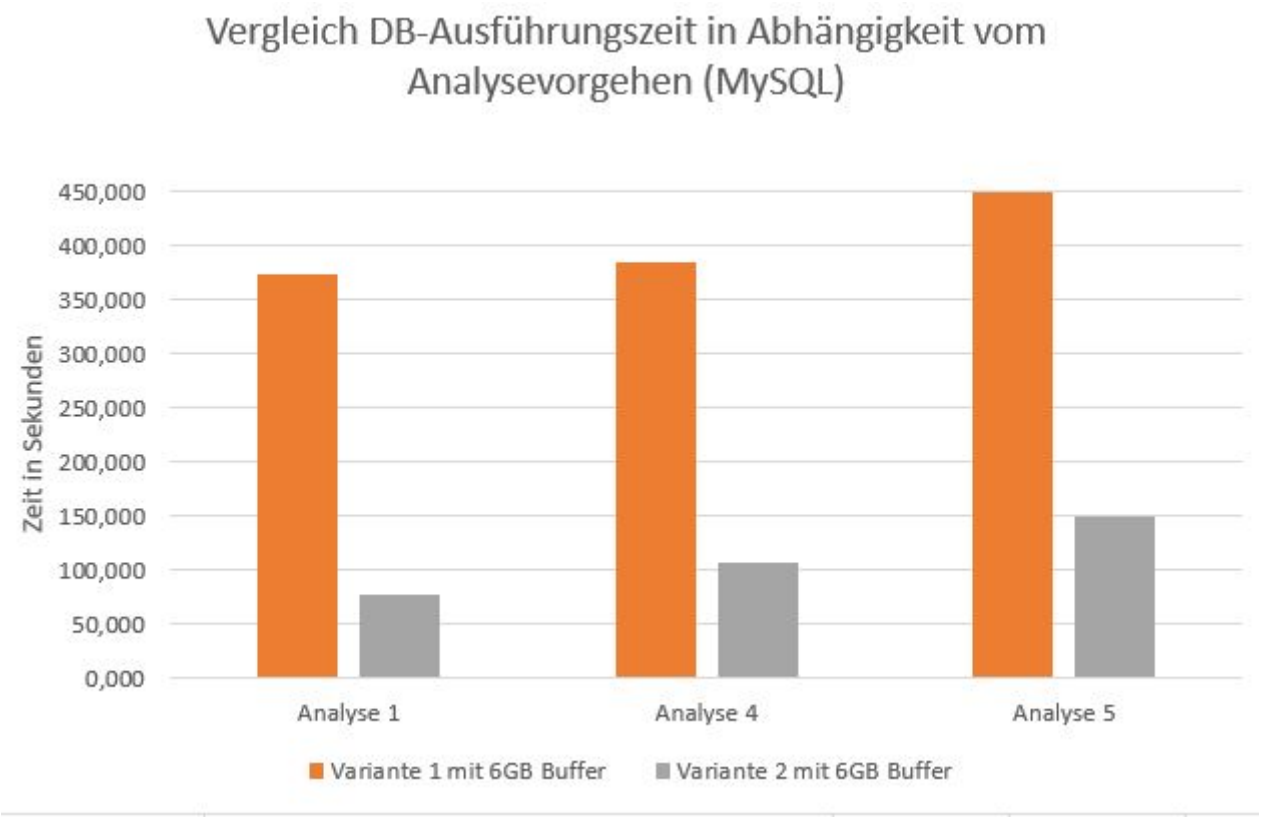


Abbildung 21. Vergleich der Analysevorgehen bezüglich der Datenbankausführungszeit

Jedoch zeigte sich auch in der Ausführungszeit der Skripte ein deutliche Zeitverbesserung. (siehe Bild 9)

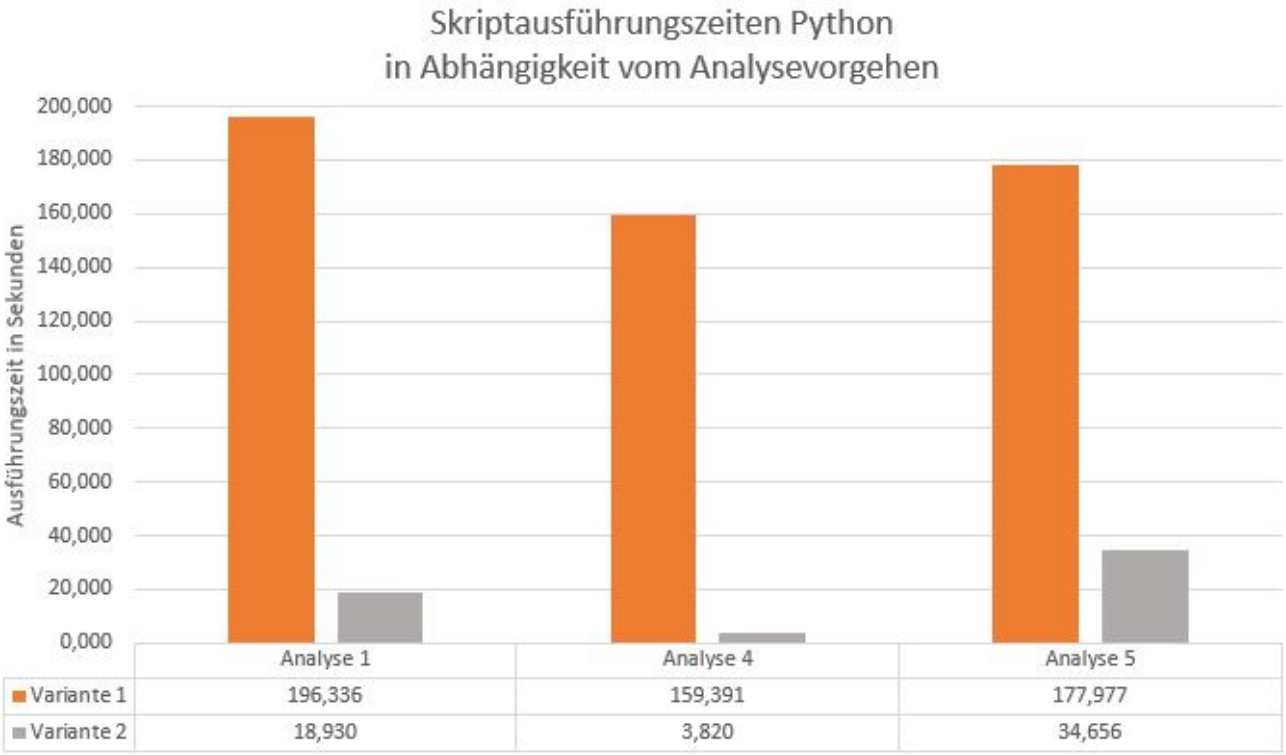


Abbildung 22. Vergleich der Analysevorgehen bezüglich der Skriptausführungszeit

Lessons Learned

Durch eine starke Verschachtelung in FOR-Schleifen der Variante 1 aller Analysen ist es möglich

sehr genaue SQL-Statements zu entwerfen und so nur einen kleinen Teil der gebrauchten Daten zu manipulieren, was den Manipulationsaufwand in Python verringert. Jedoch entstehen dadurch sehr großen Analysen mit sehr vielen Abfragen, welche der Datenbank gestellt werden müssen.

Mit Variante 2 sind weniger Abfragen nötig, jedoch müssen die Daten aufwendiger mittels Python manipuliert werden. Demgegenüber zeigte sich aber, dass dieser Mehraufwand sich deutlich in den Datenbank- und Skriptausführungszeiten widerspiegelt.

Interessant wäre noch ein Vergleich mit Variante 3 gewesen, wofür aber leider die Zeit fehlte.

5.3. Dokumentenorientierte Datenbank

5.3.1. Einführung in Datenbankstruktur

Datenbankvorstellung

5.3.2. Entwurf

Erstes Datenmodell

5.3.3. Implementierung

Implementierung der Struktur

Implementierung der Datenloader

Implementierung der Analysen

```
@app.route('/')
def home():
    return render_template("inline.html")
```

Evolution des Datenmodells

5.4. Schlüssel-Werte-Datenbank

5.4.1. Einführung in Datenbankstruktur

Datenbankvorstellung

5.4.2. Entwurf

Erstes Datenmodell

5.4.3. Implementierung

Implementierung der Struktur

Implementierung der Datenloader

Implementierung der Analysen

```
@app.route('/')
def home():
    return render_template("inline.html")
```

Evolution des Datenmodells

6. Lösungsvergleich

6.1. Übersicht

Um die Lösungen zu evaluieren wurden verschiedene Kritierien genutzt, diese sind in der Tabelle zu finden. Dabei wurden die Abstufungen sehr positiv (++) bis sehr negativ (--) genutzt. Es wurde keine absolute Skale verwendet, sondern jeglich die Lösungen untereinander zu vergleichen. Natürlich ist dieser Vergleich subjektiv, allein schon von der Vergleichskriterien Auswahl.

Das erste Kriterium vergleicht den programmier Aufwand der einzelnen Lösungen. Dabei wird die Komplexität und Größe der Abfragen als wie der Verwaltungsaufwand der Datenbank berücksichtigt. Das nächste Kriterium handelt von dem Aufwand das Datenmodell zu erstellen und der Verständlichkeit davon. Ein weiteres Kriterium für den Vergleich war der Entwicklungsaufwand für das hinzufügen eines neuen Attributes, also wenn eine neue Werte Art auch gespeichert werden muss. Die Schreib Performance befasst mit der benötigten Zeit um neue Datensätze in die Datenbank zu bringen, die dazugehörigen Grafiken sind unter Einleseperformance zu finden. Die Lese Performance beschäftigt sich im besonderen mit der Dauer der Analysen, die Diagramme sind unter der Analyseperformance zu finden. Für das nächste Kriterium wurde der von der Datenbank genutzte Speicher gemessen, damit auch der benötigte Speicherbedarf für Indices. Im letzten kriterium wurde das vorhandensein von Entwicklerdokumentation im Zusammenhang mit einer aktiven Nutzergemeinschaft verglichen.

Tabelle 3. Lösungsvergleich

Kriterium/Datenbank	Proprietäre Datenstruktur	Universelle relationale Struktur	Schlüssel-Werte-Datenbank	Dokumentenorientierte Datenbanklösung
geringer Entwicklungs-aufwand	++	- -	- -	+
geringe Komplexität des Datenmodells	++	-	-	++
Hinzufügen eines Attribut	- -	++	++	++
Schreib Performance	-	- -	+	++
Lese Performance	++	- -	+	+
geringer Speicher-aufwand	++	- -	+	++
Entwickler Dokumentation	++	++	-	+
kumuliertes Ergebnis	7	-3	1	11

6.2. Performance

6.2.1. Einleseperformance

Die Diagramme zeigen die Speicherdauer für einen Datensatz in Millisekunden. Dabei war die Einlesezeit für die Input Datensätze der Dokumentendatenbank nicht messbar, da diese kleiner als eine Millisekunde waren.

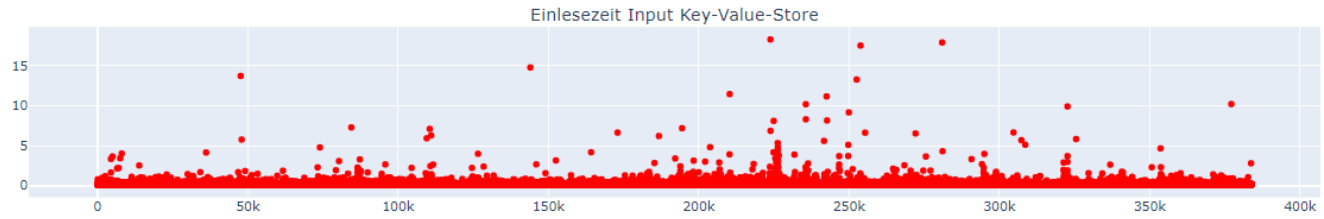


Abbildung 23. Einlesezeit Input

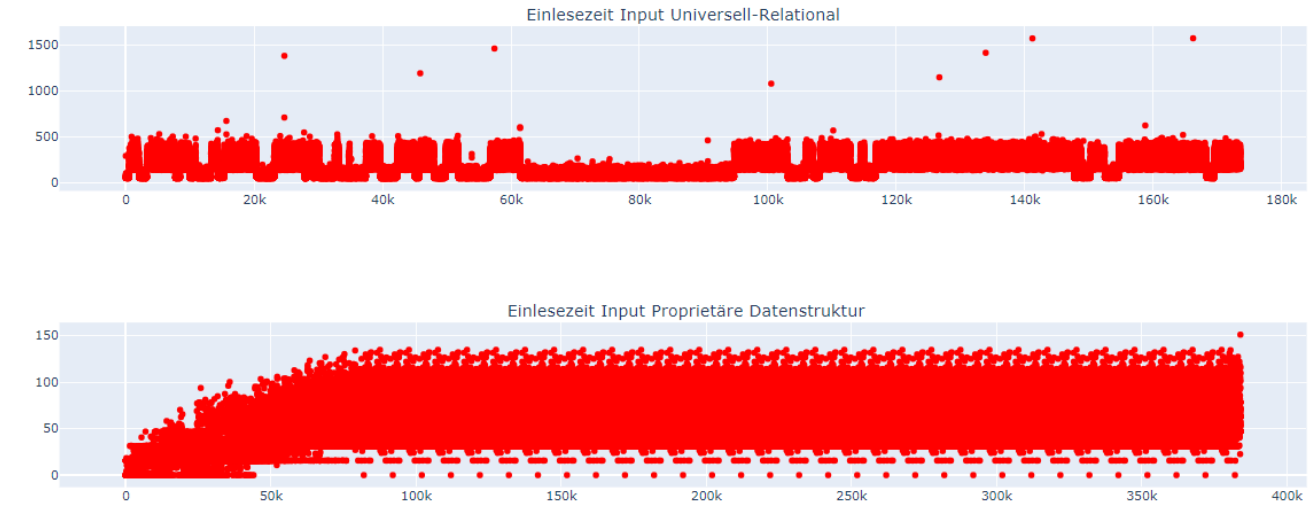


Abbildung 24. Einlesezeit Input

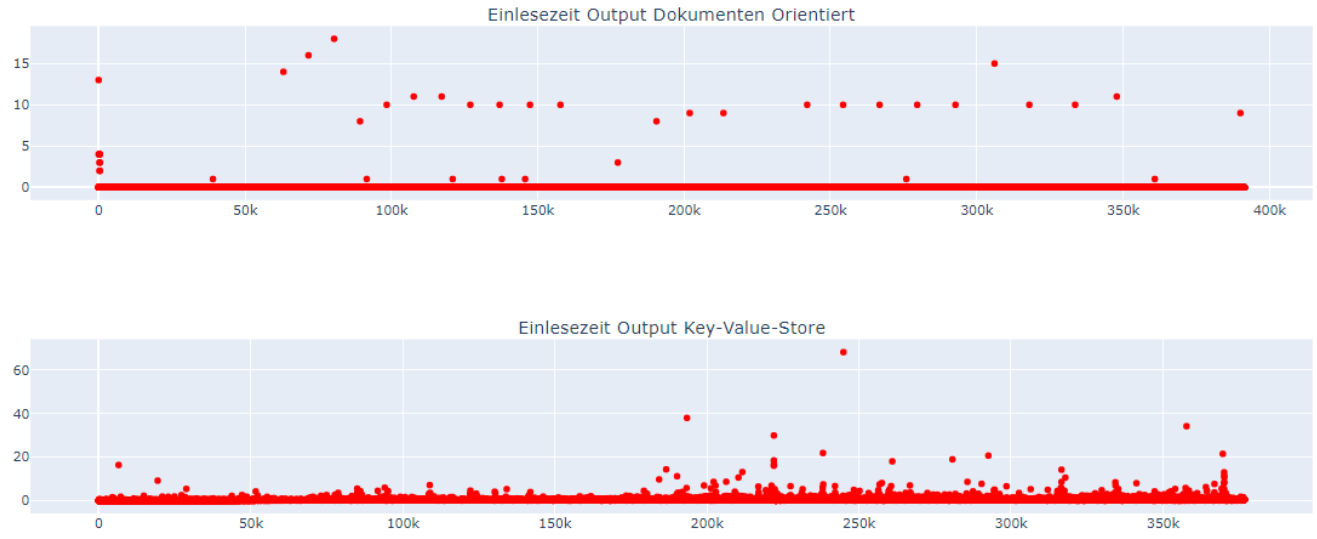


Abbildung 25. Einlesezeit Output

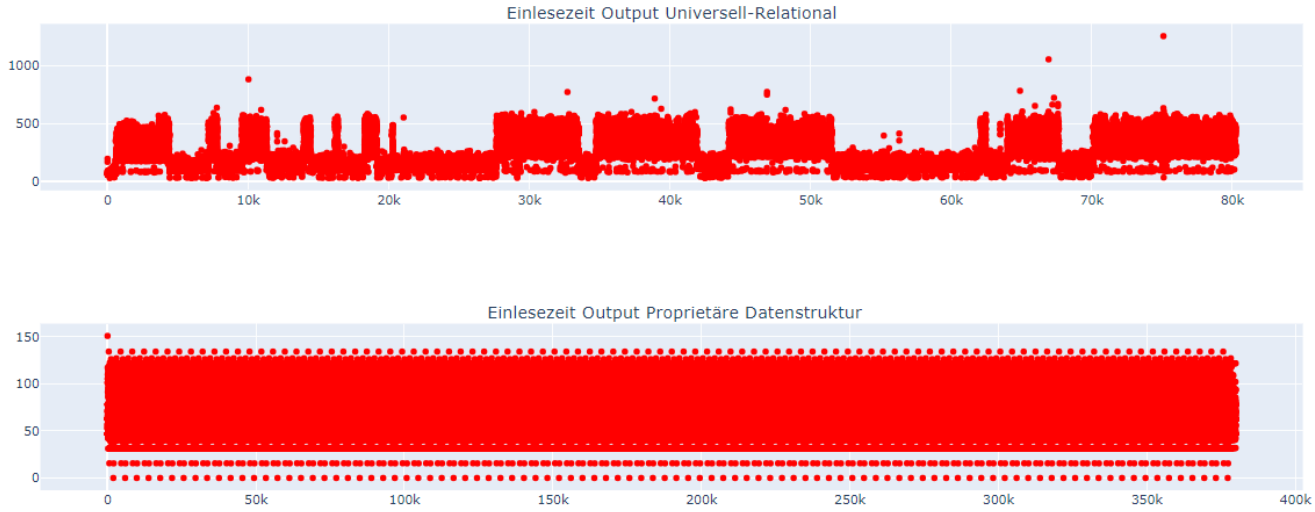


Abbildung 26. Einlesezeit Output

6.2.2. Analysenperformance

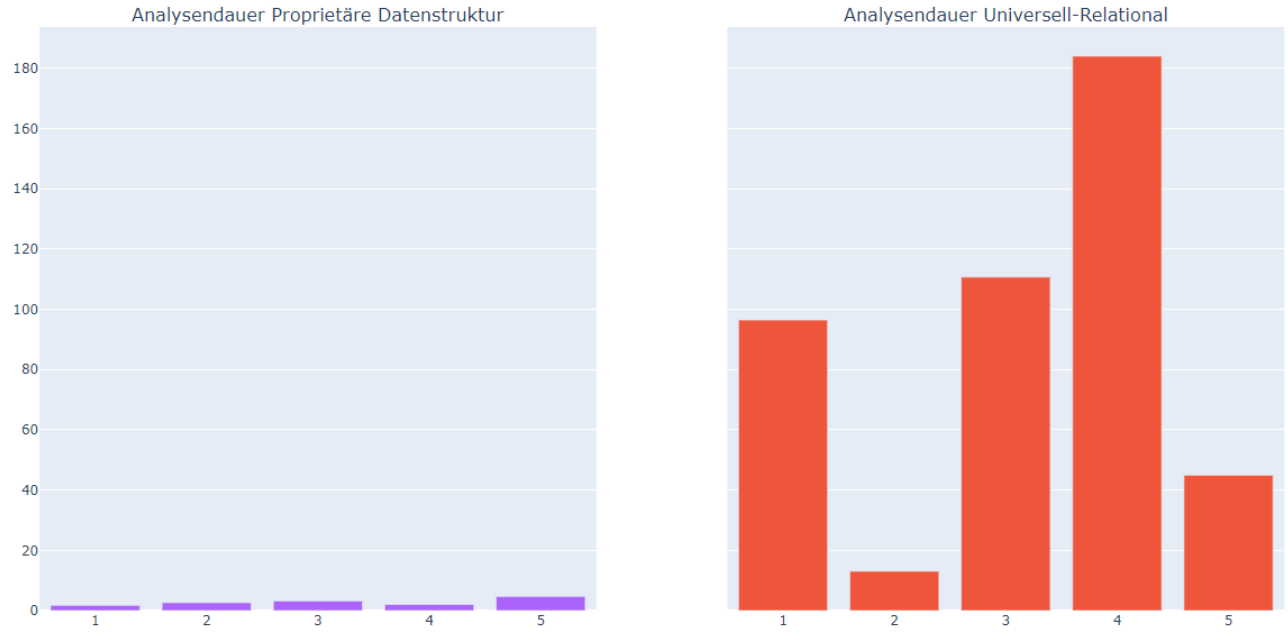


Abbildung 27. Analysedauer in den SQL basierten Lösungen

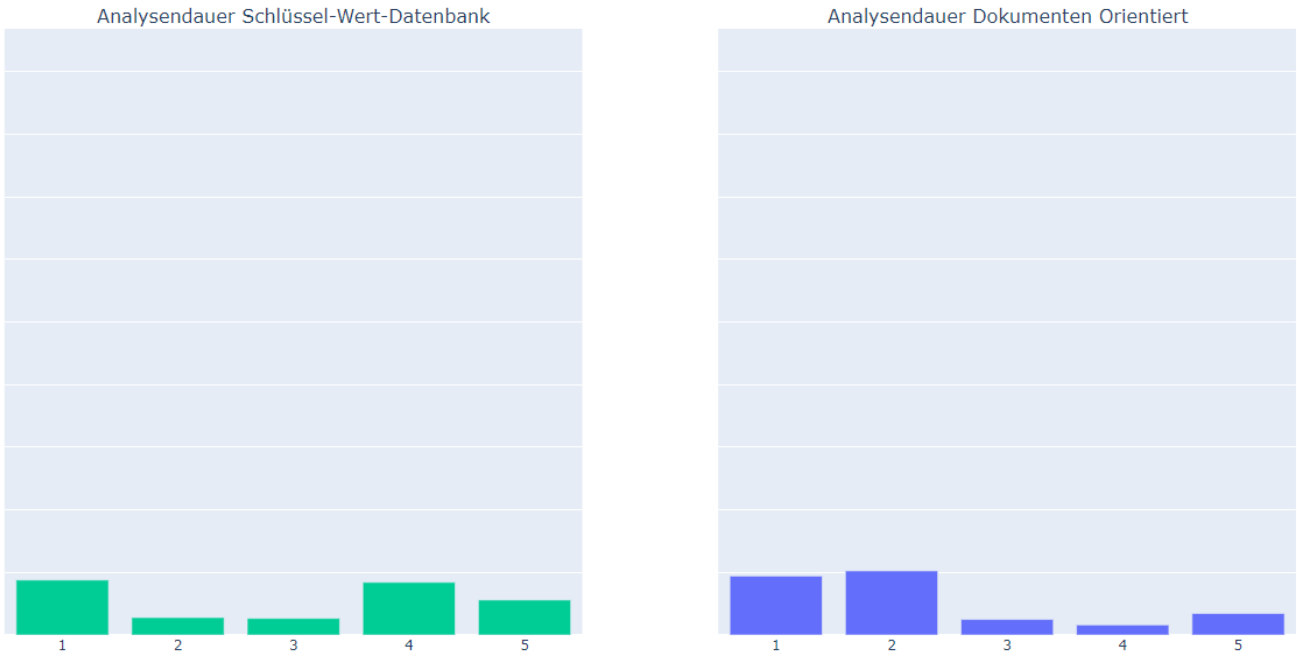


Abbildung 28. Analysedauer in den NOSQL Datenbanken

6.3. Vor- und Nachteile der einzelnen Lösungen

6.3.1. Proprietäre Datenstruktur

Der wohl klassischste Lösungansatz besticht im besonderen mit einer hohen Abfrageperformance, diese können auch mit einfach bedienbaren Werkzeugen wie Excel erstellt werden. Die Speicherperformance könnte im besonderen bei komplexeren mit Datenmodellen mit verschiedenen Foreign-Key-Constraints problematisch werden, da sich diese Lösung hier nur Platz drei belegt. Auch kann der Entwicklungsaufwand verbunden mit Änderungen im Sensorbestand ein Auschlusskriterium für manche Anwendungsfälle sein.

6.3.2. Universelle relationale Struktur

Das größter Problem mit der vorangegangen Lösung wird mit dem universellen Ansatz behoben, denn mit Änderungen im Messtechnik Bestand hat diese Modell keine Problem. Dabei bleibt aber der Vorteil bestehen, dass SQL basierte Lösungen auf ein potentes Umfeld aus Datenbankservern und Dokumentationen zurückgreifen können. Die flexibilität hat dabei aber leider den Preis eines sehr komplexen Datenmodells, was auch zu der mit Abstand schlechtesten Lese- und Schreibperformance führt.

6.3.3. Schlüssel-Werte-Datenbank

Diese Performance Probleme hat der dritte Lösungsansatz nicht, besonders bei Einzelabfragen hat diese Lösung ihre Stärke. Um aber Datenzusammenhänge darzustellen ist einiges an Aufwand nötig, der für Entwickler aus dem SQL Umfeld zunächst irritierend wirken kann. Weiterhin sind Operationen wie Joins und Gruppierungen nicht ohne weitere implementierung möglich, was Abfragen komplizierter macht.

6.3.4. Dokumentenorientierte Datenbanklösung

Dieser Abfrage Komplexität wird bei bei der Dokumentenorientierten Datenbanklösung, durch ein SQL ähnliches Abfrageraster entgegengewirkt. Da dieses Raster erst bei der Abfrage über die Datenbank gelegt wird, hat die Datenbank auch keine Performance Probleme und eignet sich somit auch für den Einsatz in einem Fast Layer. Damit hat diese Lösung eine sehr ausgeglichene Leistung über all unsere Kritieren, deshalb hat sie auch das größte Lösungspotential der besprochenen Lösungen.

7. Zusammenfassung und Ausblick

7.1. Lessons Learned

Das Projekt hat gezeigt, dass es bei der Datenbank und Datenmodell Auswahl wie bei den meisten Architekturentscheidungen keine allgemeingültige Antwort auf die Frage "Was ist die beste Lösung?" gibt. Dafür ist die Variabilität der Anforderungen zwischen den verschiedenen Anwendungsgebieten zu hoch. Man muss somit von Projekt zu Projekt unterscheiden, was die am besten Umsetzbare Lösung ist. In realen Projekten kommen natürlich auch neben technischen Anforderungen, einschränkende Faktoren wie Kosten, bestehende Datenbankserver und Fähigkeiten des Entwicklerteams kommen hinzu. Damit kann die Arbeit hoffentlich dem Leser vermitteln, welche Lösung der vier in welcher Situation die meisten Vorteile hat.

Die wichtigste Entscheidung ist welche Daten Zusammenhänge wie gespeichert werden. Wenn diese Zusammenhänge durch die Datenbankstruktur vorgegeben werden sind Abfragen einfach und performant. Wenn diese Beziehungen aber nicht gespeichert werden müssen diese Informationen durch die Abfragen wiederhergestellt werden. So sind Abfragen in der proprietären Struktur mit einfachen SQL-Abfragen möglich, während bei der universellen Struktur Abfrage Ergebnisse zum Teil zwischengespeichert werden müssen um die richtige Analyse zu ermöglichen. Das sollte man berücksichtigen, im besonderen, da Analysen besonders wertvoll sind, wenn diese durch Mitarbeiter der Fachdomäne erstellt werden können.

7.2. Fazit

8. Quellen

Laurenz Wuttke (29 Apr. 2020) Hadoop Einfach Erklärt: Was Ist Hadoop? Was Kann Hadoop?, URL: https://datasolut.com/apache-hadoop-einfuehrung/#Unterschied-zwischen-Hadoop-relationalen-Datenbank [Letzter Besuch: 5 Feb. 2021].
Übersicht Über AWS IoT Analytics Amazon Web Services, URL: https://aws.amazon.com/de/iot-analytics/?c=i&sec=srv [Letzter Besuch: 8 Feb. 2021].