KONVERTIERUNG EINER ORACLE DATENBANKTABELLE IN DAS HADOOP DISTRIBUTED FILE SYSTEM

Dokumentation zur betrieblichen Projektarbeit

IHK Abschlussprüfung Winter 2022/23

Fachinformatiker für Anwendungsentwicklung

Prüfungsbewerber:

Viktor Kövesdi
Hofangerstr. 158
81735 München
Azubi-Identnr.: 3454141

Ausbildungsbetrieb:

UniCredit S.p.A.
Branch Germany
Abteilung ARAMIS Factory
Am Tucherpark 1
80538 München

Projektbetreuer:

Herr Stefan Schmiedberger

Tel.: 089 - 378 48 114

Inhaltsverzeichnis

1. Einleitung

In dieser Dokumentation wird der Ablauf des Abschlussprojekts, das im Rahmen der Abschlussprüfung zum Fachinformatiker mit Fachrichtung Anwendungsentwicklung durch den Autor Viktor Kövesdi realisiert wurde, beschrieben.

1.1. Projektumfeld

Die UniCredit S.p.A. ist eine paneuropäische Bankengruppe mit Wurzeln in Italien. Der Firmenhauptsitz befindet sich im wirtschaftlichen Zentrum Italiens in Mailand. Zu der Gruppe gehören viele Tochtergesellschaften, hauptsächlich in Mittel- und Osteuropa. Seit der internen Fusion im Oktober 2022 dient die ehemals als UniCredit Services S.C.p.A. bekannte Tochter nun unter dem Gewand der Muttergesellschaft als hauseigener IT-Dienstleister. Die Abteilung entwickelt und betreut in über einem Dutzend Ländern der ganzen Welt Hard- und Software-lösungen für sämtliche Unternehmen des Großkonzerns.

Dieses Projekt wird in der Abteilung US41455 (ARAMIS Factory) umgesetzt. Sie betreut und entwickelt die reguläre Meldeplattform ARAMIS für die UniCredit Group weiter. Beispielsweise betrifft das unter anderem die Operationen am Rechenkern sowie die Abgabe von Unternehmenskennzahlen an die europäische Zentralbank (EZB).

1.2. Ist-Analyse

ARAMIS ist die zentrale Plattform für die Errechnung und Bereitstellung wichtiger Risikoparameter und Rechnungslegungskennziffern der UniCredit Gruppe. Anhand dieser Kennzahlen soll sichergestellt werden, dass das betroffene Finanzinstitut für Stakeholder ausreichend transparent ist.

Für Länder der EU und somit auch Deutschland gilt unter anderem der International Financial Reporting Standard 9 (IFRS 9) des International Accounting Standards Board (IASB).

Für die deutsche Tochterbank, die HypoVereinsBank AG, berechnet der ARAMIS-Rechenkern anhand der IFRS 9-Kennzahlen den Inhalt der sogenannten FINSAP-Tabelle (Financial Data for SAP). Dabei werden die IFRS 9 Kennziffern in dem Fast-Forward-Modul auf die nächsten 50

Jahre („lifetime“) hochgerechnet und in einer Oracle Datenbank festgehalten. Die errechneten Daten werden zu Analysezwecken zunächst an den Chief Risk Officer und anschließend an die Accounting-Abteilung der Bank weitergeleitet.

In der vom Fachbereich zur Verfügung gestellten Abbildung 1 ist der Entstehungsprozess von der Zentralen ARAMIS Datenbank (ARAMIS CDB), bis hin zur Generierung der für das Projekt relevanten „T_EG_RK_FINSAP“-Tabelle unter der Nummer 7 zu sehen. Wie man sieht, betreffen allein diesen Prozess, auch nach Erstellung der FINSAP-Tabelle viele weitere Schritte, bei denen im Kontext von Big Data ein optimierter Ablauf unerlässlich ist.

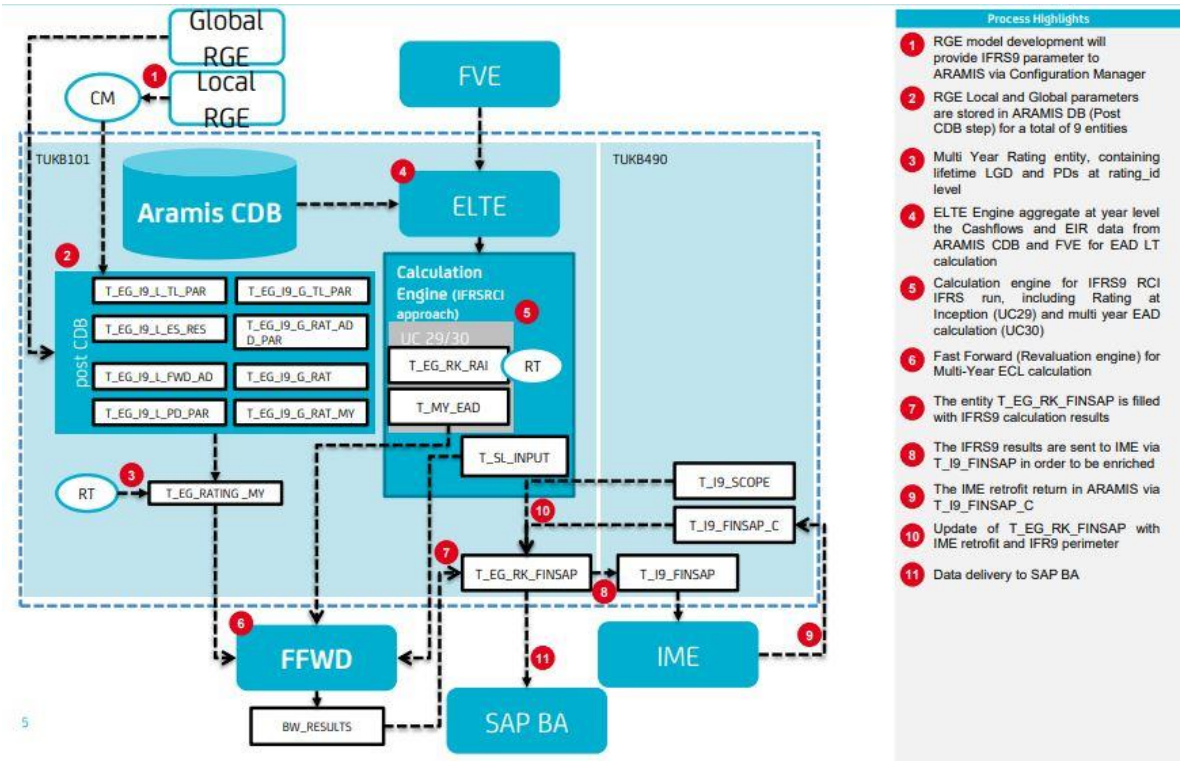


Abbildung 1: Übersicht der Entstehung der FINSAP-Tabelle

Die Bearbeitung der Daten im Oracle Datenbanksystem dauert aufgrund der Datengröße jedoch sehr lange. Da die meisten Arbeitsschritte an das ins Firmennetz integrierte Computer Cluster ausgelagert werden, blockieren diese großen Zugriffe neben der Datenbank selbst auch weitere wichtige (Hardware-)Ressourcen, die andere Workflows dementsprechend ausbremst, da diese erst gestartet werden können, wenn die zugeteilten Ressourcen frei werden.

2. Analysephase

2.1. Projektziel

Ziel des Projektes ist die Erstellung eines Workflows innerhalb des Apache Spark Frameworks, welches die vorhandene FINSAP-Tabelle von einer Oracle-Datenbanktabelle in ein Hadoop Distributed File System (HDFS) kompatibles Format konvertiert.

Daraus ergeben sich folgende Vorteile:

- Zeitersparnis durch geringere Belastung der Oracle-Datenbank
- höhere Verfügbarkeit der Daten
- Kosteneinsparungen bei den Speichermedien

Zunächst wird in der Programmiersprache Java und mithilfe zahlreicher Frameworks die Applikation entworfen und geschrieben. Anschließend wird das lokal getestete Programm auf die Testumgebung des Clusters übertragen und gemäß den üblichen Standards, wie Namenskonventionen und Datenstrukturen, eingebettet. Dort finden dann weitere Tests statt, um zusätzlich zur Funktionalität die Abrufbarkeit als Workflow zu kontrollieren.

Hat das Programm dort die Testphase bestanden, wird es nach erfolgreicher Abnahme vom Fachbereich als Workflow in der Workflowgruppe DB_Finsap_TRANSFER_CLOUDERA in der Event Engine der Produktionsumgebung integriert. Dieser letzte Schritt ist somit eine Fremdleistung und wird nicht mehr vom Autor selbst durchgeführt.

2.2. Bestimmung der technischen Voraussetzungen

Der Workflow muss folgenden Anforderungen genügen:

- Einhaltung der in der Abteilung üblichen Codenormen
- Parametrisierbarkeit des Partitionsschlüssels
- Automatisches Überschreiben veralteter Daten
- Unbetroffene Partitionen

Wie bereits die meisten Workflows wird auch dieser in der Cloudumgebung der Unicredit Gruppe CLOUDERA via Cluster Computing durchgeführt. Das bedeutet, dass die Applikation anhand eines passenden Frameworks über die Cloud abrufbar sein muss. Hierfür wird das bereits erwähnte Apache Spark verwendet. Es ist eine open-source Cluster Computing Engine bestehend aus mehreren Hauptkomponenten zur Verwaltung, Bearbeitung und Streaming großer Datenmengen (Stichwort: Big Data). Eines dieser Komponenten ist das Spark SQL-Modul. Dieses dient als Schnittstille für den Zugriff auf die Oracle-Datenbank und findet hier Anwendung.

Die Zielplattform ist Apache Hadoop, genauer gesagt das titelgebende HDFS. Es ist das Framework, welches für die Speicherung großer Datenmengen in den verfügbaren Clustern des Unternehmens verantwortlich ist. Für die Verwaltung integrierter Hadoop-Datenbanken ist die Apache Hive Software zuständig. Mittels auf SQL-basierender Syntax kann so auf Tabellen zugegriffen werden und dient somit als Format, in welches die Oracle-Tabelle konvertiert werden soll.

Als Programmiersprache hat sich der Autor für Java entschieden, da der Großteil der Prozesse der UniCredit in Java geschrieben werden. Somit ist eine Integration am einfachsten umzusetzen, da keinerlei Kompatibilitätsprobleme zu erwarten sind.

2.3. Abgrenzung der betroffenen Schnittstellen

Um die geforderten Daten abrufen und speichern zu können, muss der Workflow über die Java Database Connectivity (JDBC) API auf die entsprechende Oracle-Datenbank zugreifen und diese auslesen. Die Tabelle muss jedoch nicht von diesem Programm erstellt werden, da diese in einem anderen Schritt, der nicht Teil dieses Projektes ist, dem Schema der Abbildung 1 auf Seite 3 folgend bereits angelegt wurde.

Die Datenbank befindet sich im internen Firmennetz und soll über das Spark Framework angesprochen werden. Spark greift dann wiederum über den Cloudanbieter des Unternehmens auf die Datenbank zu und speichert den Inhalt dann via dem Hive System ab.

Dies geschieht durch die Integration in die Event Engine mittels eines Triggers automatisch, was jedoch wie bereits erwähnt Aufgabe vom Fachbereich ist und nicht vom Autor selbst durchgeführt wird.

3. PLANUNGSPHASE

3.1. ZEITPLANUNG DER PROJEKTSCHRITTE

Für die Umsetzung dieses Projektes standen dem Autor insgesamt 80 Stunden zur Verfügung. In Absprache mit dem Projektbetreuer wurden diese Projektschritte herausgearbeitet und eine grobe Zeitplanung erstellt, die in der nachfolgenden Tabelle 1 zu sehen ist.

Projektphase	Geplante Zeit
Analysephase	7 h
Planungsphase	13 h
Implementierungsphase	39 h
Testphase	4 h
Wirtschaftlichkeitsbetrachtung	1 h
Übergabe an den Fachbereich	2 h
Erstellen der Dokumentation	14 h
Gesamt	**80 h**

Tabelle 1: Grobe Zeitplanung

Eine detailliertere Übersicht über die aufgelisteten Punkte findet sich im Anhang A3: Detaillierte Zeitplanung.

3.2. EINRICHTUNG DER BENÖTIGTEN SOFTWAREKOMPONENTEN UND ZUGRIFFSRECHTE

Da dies die erste Spark-Umsetzung des Autors innerhalb des Unternehmens mit dem Ziel ist, für die Produktionsumgebung eingesetzt zu werden, bedarf es zunächst der Installation und Einrichtung benötigter Software und der Anforderung der nötigen Zugriffsrechte.

Denn sowohl die Datenbanken selbst als auch die Cloudumgebung des Unternehmens unterstehen unterschiedlicher Rollenverteilungen, die die Zugriffsmöglichkeiten regeln. Dies gilt nicht nur für die Produktionsumgebung, sondern auch für die Test- und die Qualitätssicherungsumgebung (QSU), auf denen natürlich die ersten Testläufe stattfinden sollen.

Bevor jedoch auf der Clusterumgebung getestet werden kann, sollen zuerst lokale Tests stattfinden, um die korrekte Kompilierung des Codes zu erreichen, ohne unnötig Ressourcen zu besetzen. Um Spark lokal nutzen zu können, mussten somit die entsprechenden Anwendungen auf dem Computer installiert werden, namentlich Apache Hadoop und Apache Spark.

Zur Projektverwaltung wurde das Management-Tool Apache Maven benutzt. Als Grundlage dient die Konfigurationsdatei (POM.xml). Sie kann einfach Unternehmensstandards übernehmen und mit sog. „dependencies“ schnell versionsspezifische Bibliotheken herunterladen, von denen das Projekt sozusagen abhängt. So können Projekte innerhalb des Unternehmens schnell und unkompliziert zugänglich gemacht werden. Sehen Sie dafür beispielsweise die Zeilen 18-25 der folgenden Abbildung 2, auf denen die benutzten Versionen der entsprechenden Software als „properties“ angegeben werden.

```
16          <maven.compiler.target>1.8</maven.compiler.target>
17
18          <!-- Apache Spark -->
19          <version.spark>2.4.5</version.spark>
20
21          <!-- ORACLE -->
22          <version.ojdbc>11.2.0.3</version.ojdbc>
23
24          <!-- Testing -->
25          <version.junit>4.13.1</version.junit>
26      </properties>
27
28      <dependencies>
29          <!-- Apache Spark -->
30          <dependency>
31              <groupId>org.apache.spark</groupId>
32              <artifactId>spark-sql_2.12</artifactId>
33              <version>${version.spark}</version>
34              <scope>provided</scope>
35          </dependency>
36          <dependency>
37              <groupId>org.apache.spark</groupId>
38              <artifactId>spark-hive_2.12</artifactId>
39              <version>${version.spark}</version>
40              <scope>provided</scope>
41          </dependency>
42
43          <!-- ORACLE -->
44          <dependency>
45              <groupId>com.oracle</groupId>
46              <artifactId>ojdbc6</artifactId>
47              <version>${version.ojdbc}</version>
48              <scope>provided</scope>
49          </dependency>
50
```

Abbildung 2: Auszug der POM.xml des Projekts

3.3. Entwurf der benötigten Tests

Die Anforderungen an das Programm sind sehr eindeutig definiert. Da es für einen bestimmten Workflow entwickelt wird, existieren nicht sehr viele Szenarien, die einen Testfall vermuten lassen.

Die Möglichkeit einer bereits existierenden Tabellenpartition stellt so einen Fall dar. Falls von der Event Engine eine Partition abgerufen wird, die bereits existiert, muss das Programm in der Lage sein, die bestehende Partition zuerst zu verwerfen, um sie anschließend neu zu befüllen. Ebenso dürfen Partitionen mit anderen Schlüsseln nicht durch den Vorgang verändert oder gar gelöscht werden.

Welches zum nächsten Punkt führt, nämlich die Parametrisierbarkeit des Partitionsschlüssels. Für das Programm muss es möglich sein, die richtige Partition zu erfassen und diese dann wie eben beschrieben abzuspeichern.

Durch das Erstellen von Testtabellen können diese Testfälle in der Applikation anhand einfacher Testläufe in der Clusterumgebung die verschiedenen Szenarien abgebildet und überprüft werden. Die Logs der Testumgebung geben aufschlussreiches Feedback, wie lange einzelne Jobs gelaufen sind oder etwa welche Fehler einem erfolgreichen Ablauf noch im Wege stehen.

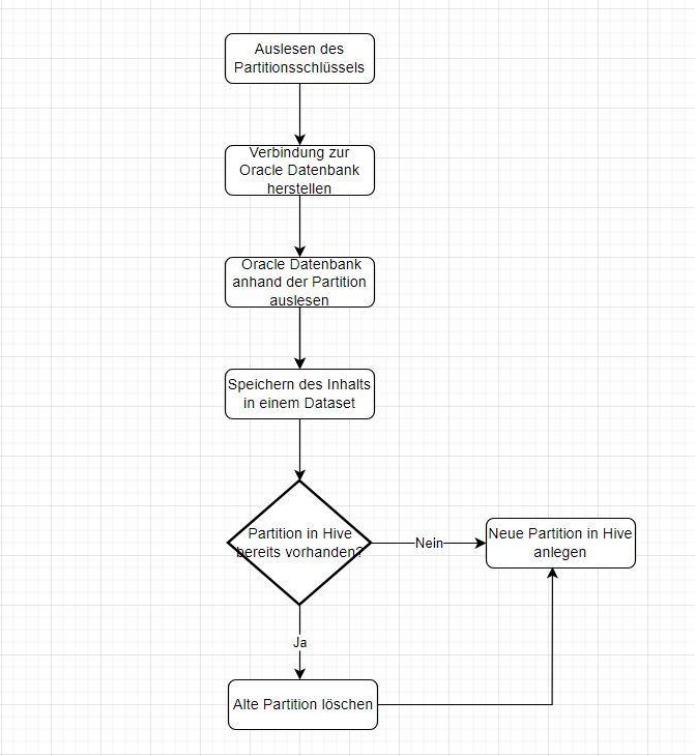


Abbildung 3: UML-Ablaufplan des Programms

3.4. Einarbeitung in die Testumgebung

Zunächst musste eine testbare Tabelle in der Hive-Testdatenbank angelegt werden. Sie musste der Struktur der originalen Oracle-Tabelle entsprechen, inkl. dem angegeben

Partitionsschlüssel. Dieser ist in diesem Fall eine Konkatenation aus den Spalten „STICHTAG“ (der angeforderte Bilanzstichtag), „MANDANT“ (die betroffene Entität der UniCredit Gruppe) UND „LAUF“ (die Bezeichnung des auszuführenden Laufs, der kennzeichnet mit welchem Rechnungslegungsstandard diese Daten berechnet wurden).

Dieser wird später mittels Parsing aus den vorhergehenden Workflows ausgelesen und dann angegeben. Für die bevorstehenden Tests wurde der Einfachheit halber zunächst jedoch ein fixer Partitionsschlüssel angegeben, um sich zunächst auf die grundlegende Funktionalität zu konzentrieren und möglichst wenig Abhängigkeiten mit anderen Workflows zu erzeugen.

Mittels eines angefertigten SQL-Skripts, welches an die leicht veränderte Hive Syntax HiveQL angepasst wurde, konnte dann eine unbeschriebene Tabelle erstellt werden, die als Zielobjekt verwendet wird.

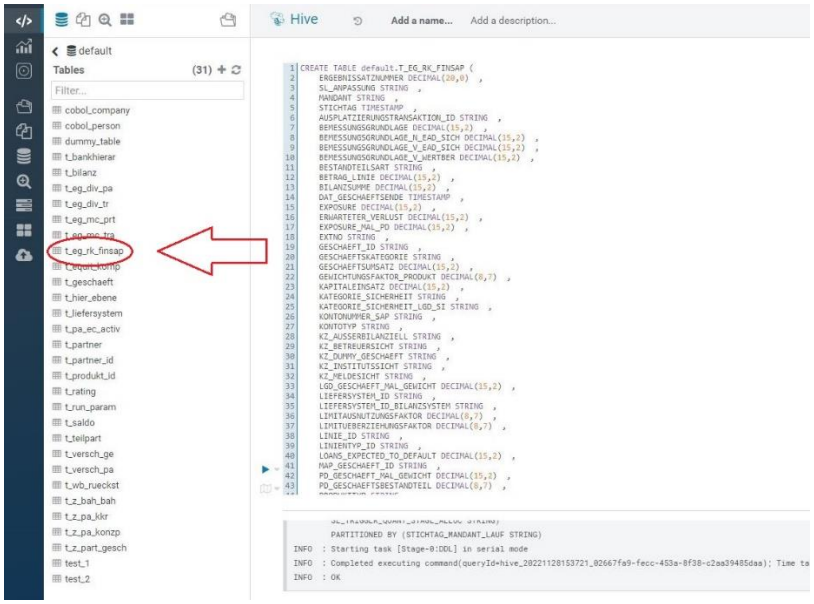


Abbildung 4: Erstellung der Testtabelle

Nach Erhalt der benötigten Zugriffsrechte konnte der Autor dann anschließend auf die Testumgebung der Cloud zugreifen. Der Zugriff selbst erfolgte über eine SSH-Verbindung mit dem Cluster. So konnte dann wahlweise via GUI oder über die Befehlszeile anhand von Linux Befehlen eine den Unternehmensrichtlinien entsprechende Projektdatenstruktur angelegt werden. Die Einhaltung dieser Struktur ist essenziell für den erfolgreichen Ablauf, da die

Initiierung der jeweiligen Spark-Instanzen mit bestimmten Parametern in Skripten erfolgt. Dort sind nicht nur die vorher angesprochenen Partitionsschlüssel angegeben, sondern beispielsweise auch die zugeteilten Ressourcen der jeweiligen Cluster. Wenn die dort verwendeten Verweise nicht übereinstimmen, kann es zu fehlerhaften oder fehlenden Ausführungen führen.

Nun fehlte nur noch das eigentliche Programm bzw. dessen Projekt, das implementiert werden musste.

4. Implementierungsphase

4.1. Erstellung des Quellcodes für das Spark Framework

4.1.1. Die Spark-Session

Zunächst galt es einen gültigen Spark Entry-Point einzurichten, der benötigt wird, um die nächsten Operationen auf dem Spark Framework auszuführen. Hierbei war zu beachten, dass die Methode „enableHiveSupport()“ benötigt wird, um das Hive Modul innerhalb der Spark Session mitabzurufen.

```
//Spark-Session Initiator
SparkSession spark = SparkSession.builder()
        .master("local[4]")
        .config("spark.hadoop.hive.exec.dynamic.partition", "true")
        .config("spark.hadoop.hive.exec.dynamic.partition.mode", "nonstrict")
        .enableHiveSupport()
        .getOrCreate();
```

Initiierung der Spark-Session inkl. Konfigurationen

Für die lokale Ausführung ist hier noch der Zusatz „master(„local[4]“) aufgeführt. Dieser regelt die Anzahl der Rechenkerne, die dem Spark-Job zugeteilt werden, in diesem Fall eben vier. Später fällt dieser Teil innerhalb des Quellcodes heraus, da dies in den zugehörigen Skripten auf der Cluster-Umgebung festgehalten und übernommen wird. Die zwei aufgeführten Konfigurationen ermöglichen dem Hive-Modul dynamisch Partitionen anzulegen, und

gegebenenfalls zu überschreiben ohne bereits existente Tabellen mit unterschiedlichen Partitionsschlüsseln zu modifizieren oder zu löschen.

4.1.2. Die JDBC API

Nun, da das Fundament für einen Spark-Job gelegt wurde, musste eine Verbindung zu der Quelle hergestellt werden. Die Quelltabelle ist die „T_EG_RK_FINSAP“-Tabelle der „TUKB101“ Datenbank. Diese wurde mit der String Variablen „oraTable“ versehen. Für die Verbindung zu der Tabelle wird die JDBC API verwendet.

Damit diese überhaupt erst eine Verbindung herstellen kann, benötigt sie eine Liste an Informationen wie die zugehörige URL der Tabelle, einen Treiber und Zugangsdaten wie Nutzername und Passwort. Diese wurden in einer externen Konfigurationsdatei bereitgestellt. Sie werden über einen Input Stream ausgelesen und über die Property Klasse zur Verfügung gestellt, ohne dass sie direkt dem Quellcode zu entnehmen sind, da dies streng vertrauliche Zugangsdaten des Unternehmens sind.

```
//Properties Input Stream
Properties props = new Properties();
try (FileInputStream jdbcInput = new FileInputStream(new File(jdbcConfig))) {
    props.load(jdbcInput);
}

//Dataset containing relevant Oracle table
Dataset<Row> finsapDS = spark.read() DataFrameReader
        .jdbc(props.getProperty("url"),
                "(SELECT * FROM " + oraTable + " WHERE STICHTAG_MANDANT_LAUF = '" + targetPartition + "') tmp", props) Dataset<Row>
        .limit(1000);
```

Bereitstellen der JDBC inkl. Properties

4.1.3. Das Dataset

Anschließend wird das Dataset erstellt, welches die Daten der Quelle beinhalten soll. Dataset ist die Schnittstelle in Spark, die zur Erstellung und Transformation von Datenbanktabellen in der Java-Umgebung dient. Sie bringt den großen Vorteil, dass sie die Operation zur Kompilierzeit durchführt und somit Fehler gefunden werden können, bevor die Applikationen gestartet sind. Über die „jdbc()“-Methode werden dann die bereits erwähnten nötigen Informationen bzgl. der Oracle-Quelle als Parameter übergeben.

Das Limit von 1000 wird hier angegeben, um nicht unnötige Ressourcen für die ersten Tests zu besetzen und wird im weiteren Verlauf dann einfach entfernt.

4.1.4. Das Hive-Modul

Der nächste Schritt ist die Logik in Bezug auf die Zielpartition im Hive Data-Warehouse. Zunächst soll überprüft werden, ob das eben erstelle Dataset eine Partition enthält, deren Partitionsschlüssel bereits in der Datenbank zu finden ist. Dafür wird von Spark das Interface Catalog angeboten, dass zur Handhabung von Metadaten gedacht ist. So kann in einem einfach zu lesendem Bedingungsblock, das Löschen der Partition veranlasst werden, sofern benötigt.

```
//Drop existing partition
if (spark.catalog().tableExists(targetDatabase, hiveTable)) {
    spark.sql(format(sql_drop_partition, hiveTable, targetPartition));
}

//Creating Hive Table
finsapDS.write()
        .format("hive")
        .partitionBy("STICHTAG_MANDANT_LAUF")
        .mode(SaveMode.Append)
        .saveAsTable(hiveTable);
```

Löschen und Erstellen der betroffenen Hive-Partitionen

Abschließend wird die Hive-Tabelle auf Grundlage des Datasets erstellt und mit dem angegebenen Schlüssel partitioniert, in diesem Fall eben die konkatenierte Spalte „STICHTAG_MANDANT_LAUF“. In der „.mode()“-Methode wird der Modus zum Speicherverhalten bei bereits vorliegenden Daten angegeben. Via dem „SaveMode.Append“-Parameter wird deklariert, dass diese Daten an bereits bestehende Partitionen mit anderer Partitionierung angehängt werden sollen und diese nicht ersetzen sollen.

Somit sind die Grundlegenden Funktionalitäten der Applikation implementiert und bereit für erste einfache Testläufe.

Leider zeigt sich zunächst ein Problem mit der Maven-Projektstruktur auf. Das Maven-Plugin für die Entwicklungsumgebung IntelliJ IDEA bietet ein übersichtliches GUI zur Handhabung des Projekts. So lässt sich z.B.: auch das Kompilieren der zugehörigen Java-Klassen einfach per Tastendruck starten und muss nicht umständlich über die Befehlszeile ausgeführt werden. So zeigt sich in der Konsole nach einem ersten Testlauf nur die nicht sehr aussagekräftige

Fehlermeldung „cannot find Symbol“, die zunächst für große Verwunderung sorgte. Nach einiger Recherche konnte der Autor jedoch feststellen, dass es sich hierbei um besagtes Problem mit der Projektstruktur handelt. Leider halfen allerlei Lösungsvorschläge nicht weiter und das Projekt musste neu aufgesetzt werden, was leider ungeplante Zeit in Anspruch nahm.

4.1.5. Anpassung an die dynamischen Anforderungen eines Workflows

Die Störung konnte allerdings recht zügig behoben werden und nach den ersten erfolgreichen kurzen Testläufen, stand der Integration des Workflows noch ein letzter Schritt im Weg. Nämlich die Anpassung der Variablen und Querverweise innerhalb des Codes an die Projektstruktur auf der Cloudumgebung. Sämtliche Angaben wie die Hardware-Allokation, der Partitionsschlüssel, das übergeordnete Projekt oder die gegebene Umgebung mussten den Skripten entnommen werden können.

Um den jeweiligen Daten aus dem Skript aussagekräftige Werte zuordnen zu können, wurde die „Options“-Klasse der Apache CLI Bibliothek verwendet. Mit der „addOption()“-Methode ist es dann möglich, einen Variablennamen festzulegen, den Parameter im Skript anzugeben, der ausgelesen werden soll und eine kurze Beschreibung dazu zu verfassen. Der Parameter „oraPartition“ ist in diesem Fall der Partitionsschlüssel, den die Event Engine im Produktionsumfeld angibt und der in der Variable „sml“ gespeichert werden soll.

```
// Creating necessary options
Options options = new Options();
options.addOption("sml", "oraPartition", true, "Target Partition in the format STICHTAG_MANDANT_LAUF");
options.addOption("le", "legalEntity", true, "Legal entity from the Event engine");

// Creating Commandline Parser
CommandLineParser parser = new GnuParser();
CommandLine cmd = parser.parse(options, args);

String sml = cmd.getOptionValue("sml").trim();
```

Anlegen der nötigen Options und des Parsers

Anschließend wurde ein CommandLineParser derselben Bibliothek erstellt. Dem Parser wird die zu parsende Option übermittelt und in einer eigenen String Variable darauf verwiesen, die wiederum der Methode der eigentlichen Applikation als Parameter übergeben wird. So weiß das Programm, welcher Partitionsschlüssel gefordert wird. Dies kann für weitere Variablen

natürlich erweitert werden, um problemlos in der dynamischen Umgebung der Event Engine zu funktionieren.

Zuletzt wurden noch die erwähnten lokalen Beschränkungen wie die Zuteilung der lokalen Hardware-Ressourcen entfernt und das Programm kompiliert. Das fertige .jar-File war somit bereit für den nächsten Schritt: das Aufbauen eines Workflows in der Testumgebung.

4.2. Integration des Workflows

Die Projektstruktur wurde in dem vorherigen Arbeitsschritt 3.4. bereits eingerichtet. So konnte das angefertigte .jar-File direkt übertragen werden. Nicht zu vergessen war noch das externe Config-File der JDBC API, welches zuvor benutzt wurde. Bevor diese beiden Dateien dann von der Clusterhardware auf das HDFS übertragen werden konnten, musste zunächst das Spark-Submit-Shellskript erstellt werden. Zur Veranschaulichung trägt der folgende Screenshot bei. Hierbei ist anzumerken, dass potenziell sensible Informationen aufgrund der Unternehmensrichtlinien unkenntlich gemacht wurden.

```
3  spark-submit \
4  --master yarn \
5  --queue batch \
6  --name test-gf-FinsapTest \
7  --deploy-mode cluster \
8  --num-executors 8 \
9  --executor-cores 2 \
10 --driver-memory 2G \
11 --executor-memory 2G \
12 --conf spark.sql.shuffle.partitions=16 \
13 --principal [redacted] \
14 --keytab /opt[redacted].keytab \
15 --files hdfs:///user[redacted]/jdbc-properties \
16 --class Demo \
17 --jars hdfs:///user/[redacted]/jars/* hdfs:///user[redacted]/FinsapTest-1.0-SNAPSHOT.jar \
18 --oraPartition 19123100001A
```

Inhalt eines Spark-Submit-Skripts

In der Zeile 6 ist der Name der Applikation zu sehen, der der Konvention „Umgebung-übergeordnetes Projekt-Workflowname“ folgt. „gf“ steht hierbei für das Projekt „group functions“. Die Zeilen 7-12 regeln verschiedene Hardware-Allokationen um auf dem Cluster mit den entsprechend zugeteilten Ressourcen ausgeführt zu werden.

Inhalt der Zeilen 13 und 14 sind Informationen für die Domain und die dafür benötigen Schlüssel, auf die Bezug genommen wird.

Abschließend befinden sich in den letzten vier Zeilen, die Parameter, die für die eigentliche Ausführung der Applikation ausschlaggebend sind, wie der Standort der benötigten JDBC-Konfigurationsdatei oder des auszuführenden .jar-Files im HDFS. Diese Dateien müssen nämlich zunächst in das HDFS übertragen werden, um anschließend vom Cluster genutzt werden zu können. Nach Erstellung dieses Skriptes, waren nun die ersten Testläufe innerhalb des Clusters bereit zur Ausführung.

5. Testphase

5.1. Durchführung der geplanten Testszenarien

Die Applikation wird über einen SSH-Zugriff anhand des Spark-Submit-Skriptes gestartet und kann dann mittels Workflownamen und der zugehörigen ID des entsprechendes Laufs (im folgenden Screenshot weiß hervorgehoben) in der GUI der Clusterumgebung zur Qualitätssicherung überprüft werden.

Auszug aus der Linux-Konsole

Erste Testläufe hatten zunächst mit Syntaxproblemen zu kämpfen, die durch die Anpassung des Codes zur Integration auf der Cloud auftraten. Diese konnten unproblematisch angepasst werden. Ein weiteres Problem war die falsche Formatierung der Zieltabelle.

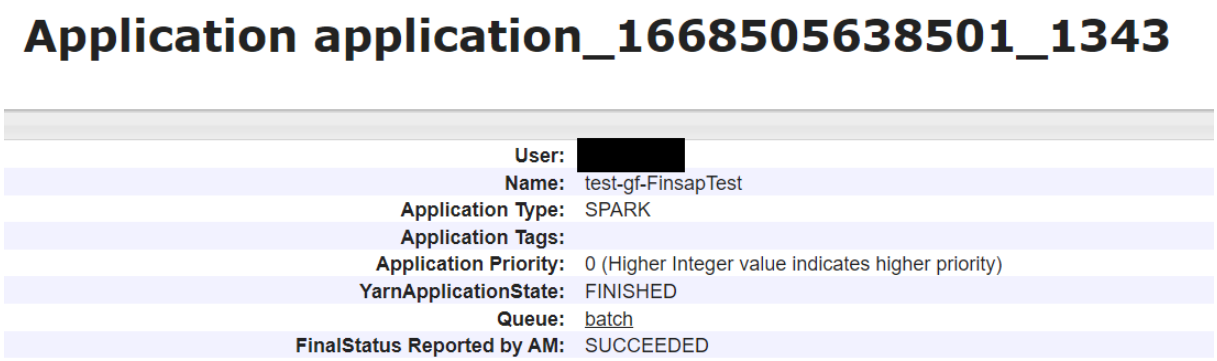
Auszug der Logs der Cluster GUI

Die betroffene Tabelle ist, die in Arbeitsschritt 3.4. erstellte „default.T_EG_RK_FINSAP“ Testtabelle, die befüllt werden sollte. Durch Anpassung des SQL-Skriptes zur Erstellung der Tabelle konnte auch dieses Problem behoben werden.

Die weiteren Testfälle wurden mit Bravour bestanden:

- Die verschiedenen Partitionsschlüssel konnten erfolgreich geparst werden
- Bereits angelegte Partitionen mit denselben Schlüsseln wurden zuerst gelöscht und anschließend neu angelegt
- Bestehende Partitionen mit anderen Schlüsseln sind bestehen geblieben
- Das Zielformat Hive für das HDFS ist erreicht worden

Somit war die Testphase auf der Testumgebung beendet und die Applikation bzw. der Workflow war bereit, dem Fachbereich zum Testen auf der QSU übergeben zu werden.



Erfolgreicher Abschluss des Workflows innerhalb der Management GUI

5.2. Abnahmetest durch den Fachbereich

Nachdem die Applikation auf der Testumgebung die letzten Tests bestanden hat, wurde sie dem Fachbereich zum Testen auf der QSU bereitgestellt. Lediglich die Parameter, die zuvor für die Testumgebung eingerichtet wurden, mussten an die Namensgebung der QSU angepasst werden.

Da der Ablauf des Programms sehr fixiert ist und die grundlegenden Problematiken bereits vorher beseitigt werden konnten, lief auch die Abnahme durch den Fachbereich problemlos. Somit steht der Integration des Workflows in der Live-Umgebung der ARAMIS Event Engine nichts mehr im Wege und sie kann in die dafür vorhergesehene Workflowgruppe DB_Finsap_TRANSFER_CLOUDERA eingebettet werden. Abschließend wurde mit dem fachlichen Ausbilder noch ein kurzes Code-Review zur Qualitätssicherung durchgeführt.

6. Abschlussphase

6.1. Soll- / Ist-Vergleich

Es folgt ein abschließender Vergleich über die ursprüngliche Zeitplanung des Projektes und dem tatsächlichen Verlauf. Wie in der folgenden Tabelle 2 zu sehen ist, war es dem Autor möglich bis auf wenige Ausnahmen die Zeitplanung einzuhalten. Somit konnte der vorgegebene Zeitrahmen von der IHK München von 80 Stunden eingehalten werden.

Projektphase	Geplante Zeit	Tatsächliche Zeit	Differenz
Analysephase	7 h	8 h	+1 h
Planungsphase	13 h	15 h	+2 h
Implementierungsphase	39 h	36 h	-3 h
Testphase	4 h	4 h	0 h
Wirtschaftlichkeitsbetrachtung	1 h	1 h	0 h
Übergabe an den Fachbereich	2 h	2 h	0 h
Erstellen der Dokumentation	14 h	14 h	0 h
Gesamt	**80 h**	**80 h**	**0 h**

Tabelle 2: Soll- / Ist-Vergleich Zeitplanung

Das Nutzen der vielen integrierten Frameworks mitsamt ihren Bibliotheken ersparte im Nachhinein zwar einiges an reinem Code-Aufwand, bedurfte aber aufgrund der schlichten Vielfalt und vielseitigen Interaktion doch eines höheren Planungsaufwands, um zu verstehen, wie kompatibel sie untereinander sind. Auch wurden durch den Zugriff auf viele verschiedene Umgebungen viele Anforderungen auf entsprechende Rechte fällig, die zusätzlich Zeit gekostet haben.

Zudem bedurfte das Arbeiten im Linux Umfeld für den Autor Einarbeitungszeit, da für die lokale Entwicklung kein Zugriff auf die Clusterumgebung bisher benötigt wurde. Mit etwas Hilfestellung, konnte jedoch auch diese Hürde überwunden werden und dann auch selbstständig angewandt werden.

6.2. LESSONS LEARNED

Durch die Umsetzung des Abschlussprojekts konnte der Autor wichtige Erfahrungen über die Arbeit an einem Projekt mitsamt den benötigten Arbeitsschritten sammeln. Vor allem der Wert einer akribisch geführten Planungs- und Entwurfsphase, der der weiteren Zeitplanung und dem erfolgreichen Verlauf des Projektes zugutekommt. So hätten die Diskrepanzen in der Planung der Implementierung verhindert werden können.

In Bezug auf die Umsetzung war in erster Linie der Einsatz vieler verschiedener Frameworks sehr interessant. Da für die unterschiedlichsten Anforderungen das Unternehmen bereits bewährte Frameworks zur Verfügung stellt, besteht die Herausforderung dabei mehr darin, die einzelnen Komponenten zu verbinden und so die jeweiligen Aufgabengebiete erfolgreich zu kombinieren.

Ebenso war Linux ein interessanter Berührungspunkt für den Autor. Linux galt vor allem im schulischen Bereich der Ausbildung immer eher als Bereich für die Fachinformatiker mit der Fachrichtung Systemintegration. Das zeigt eben jedoch wie fortgeschritten das Berufsfeld des Fachinformatikers geworden ist und wie sehr die einzelnen Fachrichtungen ineinander verwoben sein können.

Alles in allem war das Projekt eine sehr wertvolle Erfahrung, dessen Umsetzung viel Spaß gemacht hat und dennoch eine angenehme Herausforderung war.

A ANHANG

A.1. GLOSSAR

Workflow

Ein strukturierter Ablauf von einzelnen Arbeitsvorgängen, um ein bestimmtes Resultat oder Ziel zu erreichen.

Event Engine

Workflow-Management Software der ARAMIS Plattform zur Übersicht und Verwaltung aller integrierten Arbeitsabläufe

(Event) Trigger

Eine erfüllte Bedingung oder ein Ereignis, das ein bestimmtes darauffolgendes Ereignis auslöst

Cluster Computing

Ein Rechnerverbund, der logisch gebündelt wird, um mit den kombinierten Ressourcen gemeinsam Aufgaben auszuführen oder Daten abzuspeichern.

A.2. ABKÜRZUNGSVERZEICHNIS

Abkürzung	Bedeutung
ARAMIS	*Address Risk Analyse Management Information System*
FINSAP	*Financial Data for SAP*
HDFS	*Hadoop Distributed File System*
IFRS-9	*Internation Financial Reporting Standard 9*
JDBC	*Java Database Connectivity*
QSU	*Qualitätssicherungsumgebung*
SSH	*Secure Shell*

A.3. Detaillierte Zeitplanung

Projektphase	Geplante Zeit
Analysephase	**7 h**
• *Bestimmung der technischen Voraussetzungen*	3 h
• *Abgrenzung der betroffenen Schnittstellen*	3 h
• *Definition des Projektziels*	1 h
Planungsphase	**13 h**
• *Zeitplanung der Projektschritte*	1 h
• *Planung der benötigten Software-komponenten und Zugriffsrechte*	4,5 h
• *Entwurf der benötigten Tests*	2 h
• *Einarbeitung in die Testumgebung*	5,5 h
Implementierungsphase	**39 h**
• *Erstellung des Quellcodes für das Spark Framework*	25 h
• *Integration des Workflows*	6 h
• *Implementierung der Unit Tests*	8 h
Testphase	**4 h**
• *Durchführung der geplanten Testszenarien*	4 h
• *Abnahmetest durch den Fachbereich*	Fremdleistung (0 h)
Abschlussphase	**3 h**
• *Wirtschaftlichkeitsanalyse*	1 h
• *Übergabe an den Fachbereich*	2 h
Gesamt	**80 h**

A.4. Übersicht verwendeter Software

- IntelliJ IDEA Community Edition – Entwicklungsumgebung für Java
- Apache Spark – Framework zur Verarbeitung großer Datenmengen
- Apache Hadoop – Framework zur Verwaltung verteilt arbeitender Software
- Apache HDFS – Framework zur Speicherung von Daten in Computer Clustern
- Apache Maven – Projektmanagement Tool
- DBeaver – Datenbankmanagement Tool
- PuTTY – SSH-Client für Windows
- WINSCP – SFTP-Client für Windows
- JUnit – Framework zur Durchführung von Unit Tests
- HUE – Apache Hive Management Tool
- Draw.io – Diagramm Design Tool
- YARN – Clusterjobs Management Tool