I Inhaltsverzeichnis

II Tabellenverzeichnis

III Abbildungsverzeichnis

IV Abkürzungsverzeichnis

SFC	Salesforce Classic
SFL	Salesforce Lightning
CRM	Customer-Relationship-Management
CM	Content Management System
B2B	Business To Business
SE	Europäische Gesellschaft
[redacted]	[redacted]
CC	CloudCraze
z.B.	Zum Beispiel
FTP	File Transport Protocol
SFTP	Secure File Transport Protocol
OAuth	Open authorization
Id	Idendification Number
Bzw.	Beziehungsweise
SOQL	Structured Object Query Language
CSV	Comma-separated values
REST	Representational State Transfer
XML	Extensible Markup Language
JSON	JavaScript Object Notation
WSDL	Web Services Description Language
API	Application Programming Interfaces

1 Einleitung

1.1 Motivation für eine Datenmigration

In einer Welt, die zunehmend digitalisiert und vernetzt ist, spielt das Management von Kundenbeziehungen eine zentrale Rolle für den Erfolg von Unternehmen. Customer-Relationship-Management-Systeme (CRM) sind hierbei unverzichtbare Werkzeug, vor allem im E-Commerce, die es Unternehmen ermöglichen, ihre Beziehungen zu Kunden zu pflegen, zu analysieren und strategisch zu nutzen. Vor allem in Online-Shops kann die effektive Nutzung dieser Daten den entscheidenden Wettbewerbsvorteil ausmachen. Als eine führende CRM-Plattform hat sich Salesforce weltweit etabliert, nicht zuletzt durch ihre kontinuierliche Weiterentwicklung und Anpassung an die sich verändernden Bedürfnisse der Geschäftswelt.

Mit der Einführung von Salesforce Lightning, welche Salesforce Classic abgelöst hat, haben Unternehmen nun die Möglichkeit ihre Geschäftsprozesse effizienter und effektiver zu gestalten. Dabei bietet die neue Plattform viele neue Funktionen wodurch Unternehmen die Chance bekommen ihre Produktivität zu steigern und ihr Kundenmanagement zu optimieren.

In dieser Arbeit liegt der Fokus darauf, ein Teil der Daten von Salesforce Classic zu Salesforce Lightning zu migrieren. Dies soll anhand eines Beispiels geschehen: Ein Onlineshop für Saatgut welcher von [redacted] betrieben wird. Die Migration von Salesforce B2B-Commerce zu Salesforce B2B-Lightning ist ein komplexer Prozess, der eine gründliche Planung und Umsetzung erfordert. Dieses Projekt bietet dabei eine Gelegenheit, die Herausforderungen und Lösungsansätze bei einer Datenmigration in einem realen Unternehmenskontext zu untersuchen. Im Zentrum der Arbeit steht die Erstellung eines Datenmigrationskonzepts für einen Teil der Daten aus dem Online-Shop welches speziell auf die Bedürfnisse und Anforderungen des Saatgut-Shops von [redacted] zugeschnitten ist. Ein wesentlicher Vorteil für den Kunden in diesem Prozess ist der Zugriff auf alte Kundendaten im neuen System, wodurch die Nutzung neuer CRM-Funktionalitäten ermöglicht wird und wichtige Daten nicht verloren gehen. Auch für die dotSource SE entsteht hierdurch der Vorteil, dass Entwickler, welche vor einer ähnlichen Herausforderung stehen, diese Arbeit als Grundlage für ein Projekt nutzen können. Diese Migration führt außerdem zu einer besseren Anpassung an spezifische Geschäftsbedürfnisse, was wiederum die Prozesseffizienz und -effektivität des Onlineshops verbessert. Darüber hinaus bietet das neue System eine verbesserte Plattform für zukünftiges Wachstum und Anpassungen. Der Onlineshop kann so dynamisch auf Marktänderungen und

neue Geschäftsanforderungen reagieren, indem er seine vorhandenen Kundendaten effektiv nutzt. Dies stellt einen wesentlichen strategischen Vorteil dar, da der Onlineshop flexibel bleibt und sich an die sich ständig wandelnden Anforderungen des digitalen Handels anpassen kann.

1.2 Methodik und Herangehensweise

Die Durchführung einer erfolgreichen Datenmigration erfordert klar definierte Ziele und Methodik. In dieser Arbeit wird eine systematische Vorgehensweise zur Entwicklung eines Datenmigrationskonzepts aufgezeigt. Diese Methodik umfasst folgende Schritte:

Analyse der Ausgangssituation: Der bestehende Salesforce Classic-basierte Onlineshop für Saatgut des Kunden wird in Kapitel 2 analysiert, um die spezifischen Anforderungen und Datenstrukturen zu verstehen.

Identifikation von Herausforderungen: Die potenziellen Herausforderungen und Hindernisse bei der Datenmigration werden in Kapitel 3 identifiziert, einschließlich technischer und organisatorischer Aspekte.

Entwicklung eines Datenmigrationskonzepts: Basierend auf den Ergebnissen der Analyse wird in Kapitel 4 ein umfassendes Datenmigrationskonzept erstellt.

Umsetzung und Validierung: Die geplanten Schritte zur Datenmigration werden in Kapitel 5 durchgeführt, dokumentiert und die Ergebnisse validiert.

Das Ziel dieser Arbeit besteht darin, ein abgeschlossenes Konzept für die Datenmigration von Salesforce Classic zu Salesforce Lightning zu entwickeln.

2 Kundenprofil und Shop-Charakteristik

In diesem Kapitel werden die zwei Schlüsselbereiche Kundenprofil und Shop-Charakteristik behandelt. Beide Aspekte sollen dazu beitragen einen Überblick über die Leistungsbereiche des Kunden zu bekommen, um damit fundierte Entscheidungen für die Datenmigration zu treffen.

2.1 [redacted]

Die [redacted] wurde von [redacted] gegründet und ist ein weltweit führendes Unternehmen in der Pflanzenzüchtungs- und Biotechnologiebranche. Mit einem Umsatz aus landwirtschaftlichen Nutzpflanzen konnte sich [redacted] als vertrauenswürdiger Partner etablieren und damit zum viertgrößten Saatguthersteller der Welt aufsteigen.[1]

Ursprünglich auf Zuckerrübensaatgut spezialisiert, wurde der Betrieb bereits 1885 in eine Aktiengesellschaft umgewandelt. Im Jahr 1900 wurde die erste internationale Außenstelle in der ukrainischen Stadt Winnyzja gegründet. Im Jahr 1920 wurde das Sortiment durch die Aufnahme der Getreide-, Futterrüben- und Kartoffelzüchtung erweitert. Im Geschäftsjahr 2020/21 war [redacted] mit über 5.000 Mitarbeitern in über siebzig Ländern aktiv und ist auschlaggebender Teil der [redacted]. Mit ertragsstarkem Saatgut und umfangreichem Wissen trägt das Unternehmen dazu bei, Lösungen zur Ernährung einer stetig wachsenden Weltbevölkerung zu finden und zuletzt auch das Augenmerk auf die Umwelt zu lenken. Dabei erreichen Dinge wie der Input von Pflanzenschutzmitteln, Innovationen im Bereich alternative Energien und die effiziente Nutzung von verfügbaren Flächen einen hohen Stellenwert. Es finanziert die Grundlagenforschung sowie die Züchtung des Sortenspektrums der xxx und stellt seinen Tochtergesellschaften die notwendigen Ressourcen zur Verfügung.[2]

2.2 Überblick über den Shop

Der B2B-Online-Shop unter der Domain [redacted] ist das digitale Herzstück des Vertriebs des Kunden. Dieser wurde entwickelt, um dessen Kunden weltweit einen einfachen Zugang zu

[1] Vgl. [redacted]
[2] Vgl. Ebenda

hochwertigen den Saatgutprodukten des Unternehmens zu ermöglichen, welche bisher nur über die lokalen Verkaufswege Zugang zu diesen Waren hatten. Mit einer mehrsprachigen Benutzeroberfläche und umfangreichen Funktionen bietet der Shop eine optimale Plattform für Landwirte, Agrarbetriebe und andere Kunden.

Die dotSource SE hat maßgeblich zur Entwicklung dieses B2B-Commerce-Shops mittels Salesforce Classic beigetragen. Diese Partnerschaft stellt sicher, dass auch nun der neue Shop, welcher auf Salesforce Lightning und dessen modernen Technologien basiert, nicht nur die Standard E-Commerce-Funktionalitäten bietet, sondern auch auf die spezifischen Anforderungen und Bedürfnisse des Kunden zugeschnitten ist. Zu den Neuerungen in Salesforce Lightning gehören unteranderem eine neue E-Commerce-Plattform mit verbesserten und individualisierbareren Komponenten mittels der Lightning-Plattform in dem auch die zu migrierenden Daten zu finden sind. Dies wird aber näher in Kapitel 3.1 betrachtet.

Der Shop ermöglicht es Kunden Saatgut und verwandte Produkte bequem zu durchsuchen, auszuwählen und zu bestellen. Dabei stehen detaillierte Produktinformationen, Benutzerkonten mit Bestellhistorie und Rechnungsverwaltung sowie ein reibungsloser Bestellprozess im Vordergrund. Die Integration von Salesforce ermöglicht dem Kunden die Kundenbeziehungen zu optimieren und das Einkaufserlebnis kontinuierlich zu verbessern. Dieser Shop dient nicht nur als Vertriebskanal, sondern auch als Informationsquelle für Kunden, die stets auf dem neuesten Stand über die Produktentwicklungen und Innovationen des Kunden sein möchten.

3 Datenmigration von Salesforce Classic und Lightning

In diesem Kapitel geht es um das Schaffen einer Grundlage für die Datenmigration von Salesforce Classic zu Salesforce Lightning. Die Migration von Daten erfordert eine sorgfältige Planung und Anpassung, um die Datenintegrität zu gewährleisten. Daher ist die Analyse der zu migrierenden Daten essenziell und soll Teil des Kapitels sein. Weiter soll ein Überblick über Salesforce Classic und Salesforce Lightning gegeben werden und die relevanten Begriffe im Kontext der Migration erklärt, sowie welche Daten Teil der Migration sind. Der darauffolgende Teil soll das aktuelle System (Salesforce Classic) im Zusammenhang mit den zu migrierenden Daten analysieren. Der letzte Abschnitt beschreibt die Struktur der Daten im neuen System (Salesforce Lightning) sowie den Vergleich der beiden Systeme.

3.1 Einführung Salesforce Classic und Lightning

Die Evolution von Salesforce Classic (SFC) auf Salesforce Lightning (SFL) hat einen großen Meilenstein in der Entwicklung für die Salesforce-Plattform und dessen Nutzern gesetzt. Auch der Kunde, welcher sich im Laufe der Jahre als ein weltweit führendes Unternehmen in der Pflanzenzucht- und Biotechnologiebranche etabliert hat, ist von dieser Entwicklung betroffen. Die Entscheidung von SFC auf SFL zu migrieren hat sich als wichtigen Schritt erwiesen, um die Effizienz und Produktivität im Kontext der Geschäftsprozesse, vor allem im E-Commerce, weiter zu steigern.[3]

Als erstes sollte aber die Frage geklärt werden, wieso Salesforce diesen Schritt gegangen ist. Zurückzuführen ist dies zum Jahr 2018, als Salesforce das B2B-Commerce Produkt CloudCraze aufgekauft hat. Dieses wurde nach dem Kauf in Salesforce durch den Namen B2B-Commerce-Cloud mit Visualforce integriert. Dabei wurden die alten Strukturen der CloudCraze-Plattform direkt übernommen, welches in den Objektstrukturen mit der Abkürzung *CC-<ObjektName>* wiederzufinden ist (Siehe Kapitel 3.2).[4]

[3] Vgl. [redacted]
[4] Vgl. [BAR18]

Nun bestand das Problem, dass die alten Strukturen zwar erfolgreich in Salesforce implementiert, aber nicht direkt auf Salesforce zugeschnitten sind, sondern eher als eine Art Erweiterung dienten. Daher hat sich Salesforce 2020 entschieden das neue, eigene Produkt Salesforce B2B-Lightning auf den Markt zu bringen. Diese Entscheidung hat der B2B-Commerce-Plattform ein komplett neues Aussehen gegeben und implementiert nun einen neues Datenmodell welches direkt in das gesamte Ökosystem von Salesforce integriert ist. Dazu gehören unterandere neue Features und ein aktualisiertes Aussehen. Zu diesen leistungsstarken Aktualisierungen zählen ein neues userorientiertes Interface mit Drag & Drop und zum heutigen stand auch alle nötigen B2B-Standard-Funktionalitäten (Wie z.B. Coupons, Promotions oder Abonnements), welche vorher nur bedingt vorhanden waren. [5]

Auch besteht nun die Möglichkeit durch die Kombination von z.B. Headless E-Commerce und Lightning Web Components einen leistungsstarken individuellen Online-Shop zu entwickeln. Zudem eröffnen Automatisierungsmöglichkeiten wie die kürzlich hinzugefügten Einstein-Analysen neue Chancen im Bereich Content Management (CM) und Customer-Relationship-Management (CRM).[6]

Wieso sollte der Kunde nun die Entscheidung treffen das eigene System zu aktualisieren? Salesforce hat zwar 2020 angekündigt, dass B2B-Commerce auch weiterhin unterstützt wird, aber neue Funktionen auf die B2B-Lightning Plattform ausgelegt sein werden. Dies bedeutet, dass die Funktionalitäten und der Support für die alte Plattform eingeschränkt werden. Zu sehen ist dies schon durch die neu implementierten Features nach 2020. Hier wurde durch die neuen Lightning Web Components ein Schritt weiter in eine moderne Zukunft von B2B E-Commerce gegangen. Daher ist die Migration nicht nur eine Frage der Modernisierung, sondern auch der Wettbewerbsfähigkeit.[7] Im Kontext der Salesforce Commerce Cloud ist das Datenmodell das Herzstück der Commerce-Plattform. Dazu zählt vor allem das CC-Order-Objekt, welches typischerweise die oberste Ebene einer Bestellung widerspiegelt. Nicht nur dient dieses dazu Bestellungen hinter gegebenen Prozessen im Online-Shop zu erfassen und zu verarbeiten, sondern es repräsentiert auch eine Sammlung von Metadaten wie z.B. Bestellmengen, Preise oder Beziehungen zu Kind Objekten wie dem Warenkorb oder die bestellten Produkte. Dazu aber mehr in den folgenden Kapiteln.

---

[5] Vgl. [TEL20]
[6] Vgl. Ebenda
[7] Vgl. [SAL24]

Somit werden in diesem Objekt wichtige Kundendaten gespeichert, die für den Kunden und ihre Kunden des Online-Shops von großem Nutzen sind. So können Benutzer beispielsweise die Daten analysieren und für die Berichtserstellung verwenden. Es können Verkaufstrends, Umsatzprognosen und personalisierte Inhalte erzeugt werden, um Entscheidungsprozesse oder Marketinginitiativen strategisch zu unterstützen. Auch besteht die Möglichkeit das alte Bestellungen nachvollzogen und abgerufen werden können. Somit ist es essenziell diese Daten nicht zu verlieren und bei einer Migration zu berücksichtigen.

3.2 Analyse des aktuellen Systems (Salesforce Classic)

Die Analyse des SFC-Systems ist ein entscheidender Schritt im Prozess der Datenmigration. Dieses Kapitel konzentriert sich auf die Untersuchung des Objektes CC-Order innerhalb von SFC. Wie bereits beschrieben spielt dieses Objekt eine zentrale Rolle im Bestellmanagement und spiegelt die vielschichtigen Anforderungen des E-Commerce hinsichtlich der Bestellverwaltung und Kundeninteraktionen wider. Die Prüfung der Objektstruktur ist wichtig, um eine effektive und effiziente Datenmigration zu gewährleisten sowie um später zu entscheiden, welche Felder in die SFL-Umgebung übernommen werden müssen.

Die Prüfung der Struktur des CC-Order-Objektes wurde mit dem Salesforce Schema Builder und der internen Dokumentation der dotSource SE durchgeführt. Der Schema Builder ist ein Werkzeug innerhalb der Salesforce-Plattform, welches eine visuelle Darstellung der Objekte und ihrer Beziehungen bietet. Er ermöglicht Benutzern, die Struktur ihrer Objekte und dessen Beziehungen leicht zu verstehen und zu modifizieren. Dabei kann über eine interaktive Oberfläche der Benutzer Objekte und Felder hinzufügen, bearbeiten und verschieben (zu sehen in Abbildung 1).[8]

[8] Vgl. [SFH23]

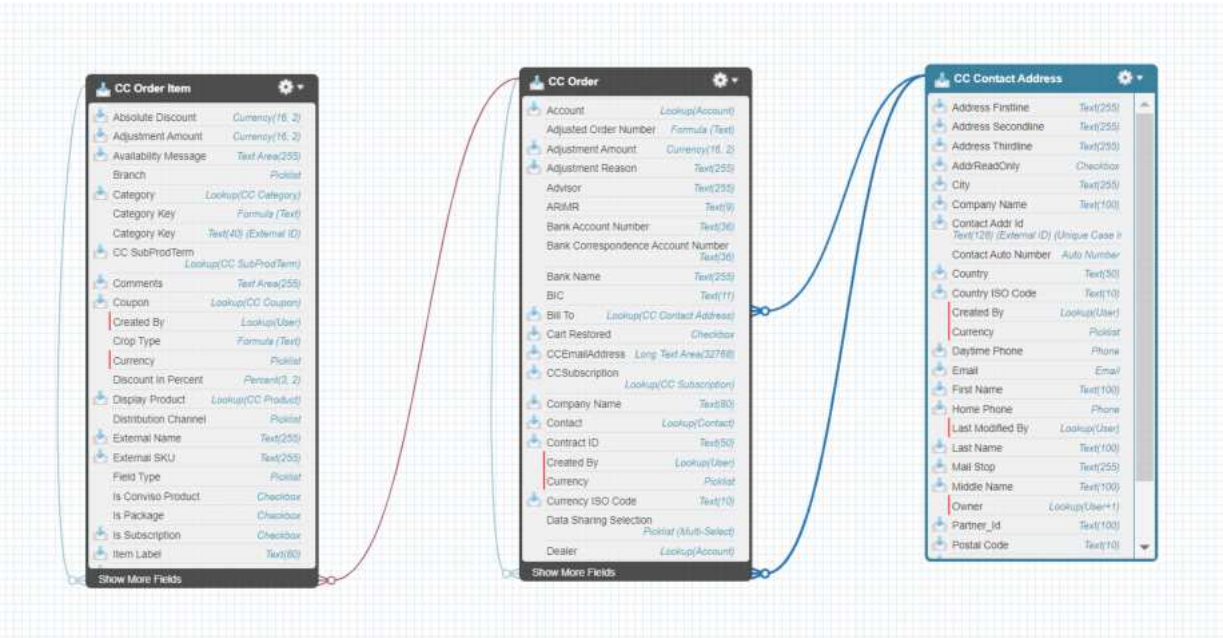


Abbildung 1: Ausschnitt CC-Order Beziehungen

Quelle: Screenshot aus Kunden-Sandbox

Relationale Struktur des „CC-Order"-Objektes:

1. *Account*: Dieses Feld etabliert eine Verbindung zum Kundenkonto.
2. *CC-Cart*: Bildet die Beziehung zum temporären Warenkorb ab.
3. *Owner, CreatedById* und *LastModifiedById*: Diese Felder repräsentieren die Beziehungen zu den Verantwortlichen innerhalb des Salesforce-Systems. Der *Owner* ist der Besitzer des Auftrags, *CreatedById* zeigt an, wer den Auftrag ursprünglich erstellt hat, und *LastModifiedById* gibt Aufschluss darüber, wer den Auftrag zuletzt modifiziert hat.
4. *Contact*: Stellt die Verbindung zum Kunden her, der die Bestellung aufgegeben hat.
5. *CC-OrderItem*: In diesem Zusammenhang werden sowohl die Summe der Steuern im Feld SumTax als auch der Gesamtpreis im Feld SumTotals der Bestellung abgebildet.
6. *CC-Seller*: Dieses Feld identifiziert, welcher Verkäufer oder welches Team die Order verkauft hat.
7. *CC-Subscription*: Gibt an, ob die Bestellung im Rahmen einer Dauerbestellung oder eines Abonnements erfolgt.
8. *CC-ContactAddress*: Enthält Informationen zu den Adressen für die Lieferung im Feld ShipTo, Rechnungsstellung in BillTo und externe Lieferungen in ExtShipTo.

Um das Order-Objekt erfolgreich zu migrieren, müssen die Objekte aus den oben genannten Beziehungen teilweise mit übernommen werden. Diese Objekte sind entscheidend für eine erfolgreiche Übertragung der Bestell- und Verwaltungsprozesse des Shops. Im Folgenden wird eine Übersicht über die wichtigsten relationalen Objekte und ihre gemeinsamen Merkmale gegeben:

CC-OrderItem und Verwandte Objekte:
Diese Gruppe umfasst Objekte wie *CC-OrderItemGroup*, *CC-Product*, *CC-Attribute*, und *User*. Sie alle stehen in direkter Beziehung zum CC-Order-Objekt und tragen Informationen über Bestellartikel, Produktinformationen, Benutzeraktionen und spezifische Attribute in sich. Sie bilden gemeinsam die Basis für die Darstellung und Verwaltung der Bestellungen.

CC-Coupon und Verwandte Objekte:
Objekte wie *CC-Rule*, *CC-Product*, und *User* sind relevant für die Handhabung von Coupons und Rabatten. Sie spielen eine wichtige Rolle bei der Definition von Rabattregeln, der Zuordnung der Produkte und der Nachverfolgung der Coupon-Nutzung durch Benutzer.

CC-Seller und Verwandte Objekte:
Hierzu zählen *User* und *Account*, die Informationen über die Verkäufer, Verkaufsteams und die zugehörigen Kundenkonten bereitstellen.

CC-ContactAddress und Verwandte Objekte:
Diese Objekte speichern die Informationen zu Adressen, die für Lieferungen, Rechnungsstellungen und andere logistische Aspekte wichtig sind. Sie sind eng mit *User* und anderen Bestellungsbezogenen Daten verknüpft.

CC-Subscription und Verwandte Objekte:
Diese umfassen eine Vielzahl von Objekten wie *CC-OrderItem*, *CC-ContactAddress*, *CC-StoredPayment*, und *User*.

Weitere Objekte:

Zusätzlich zu den oben genannten gibt es weitere Objekte wie *CC-SubProdTerm*, *CC-Category,* und *CC-Attribute*, die in spezifischen Szenarien eine Rolle spielen. Sie sind wichtig für die Verwaltung von Produktkategorien, Abonnementbedingungen und Attributen.

Diese Objekte haben gemeinsam, dass sie alle wesentlichen Beziehungen und Informationen enthalten, die für die effektive Verwaltung von Bestellungen in SFC wichtig sind. Die genauen Details zu jedem dieser Objekte und deren relationalen Feldern sind im Anhang 1 aufgeführt. Hierbei wurde aber nicht eine die Darstellung des Schema Builders verwendet, sondern ein Entity-Relationship-Diagramm. Dies vereinfacht die Struktur des Objektes und die Beziehungen werden sichtbarer.

Wichtig zu erwähnen sind auch die Prozesse, welche mit *CC-Order* zusammenhängen. Der Kunde benutzt für die Verwaltung seiner Bestellungen eine Lösung des Unternehmens SAP. Sobald die Bestellungen von Salesforce erstellt werden, stellt ein Skript im Backend sicher, dass die Bestellung in mehrere Unterobjekte sogenannte *VarietyEntrys* umgewandelt werden. Dies geschieht auf Grundlage der bestellen Kategorien bzw. Saattypen. Wird beispielsweise eine Bestellung von Rübensaatgut und Sommerhybridroggen aufgegeben wird diese Bestellung in zwei separate Bestellungen mit separaten *OrderItems* aufgeteilt (Siehe Abbildung 2).

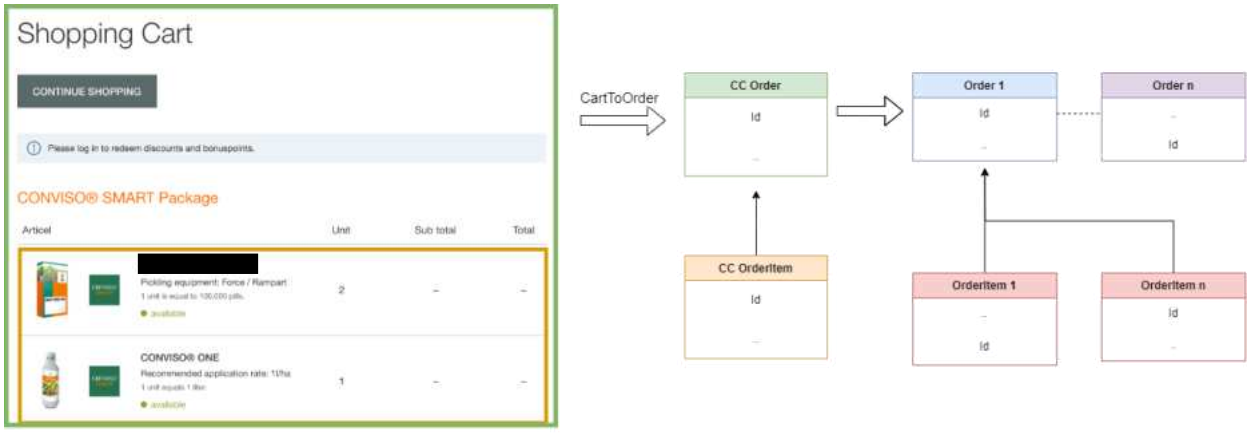


Abbildung 2: OrderSplit

In diesem Kapitel wurde nicht näher auf die nicht relationalen Felder der Objekte eingegangen. Der Hauptgrund hierfür ist, dass diese Felder einfacher zu migrieren sind, da sie keine komplexen Beziehungen oder Abhängigkeiten zu anderen Objekten aufweisen. Die Entscheidung, welche dieser Felder migriert werden, basiert daher auf ihrer Relevanz und Nützlichkeit im neuen System und wird zu einem späteren Zeitpunkt des Datenmigrationsprozesses (Kapitel 4.3) getroffen. Dies gilt auch für die dahinterliegenden Prozesse.

3.3 Analyse des neuen Systems (Salesforce Lightning)

Mit dem Übergang zu B2B-Lightning wurde *CC-Order* aus B2B-Classic durch das standardisierte *Oder*-Objekt ersetzt. Diese Änderung bringt nicht nur technologische Verbesserungen und eine engere Integration in das Salesforce-Ökosystem mit sich, sondern auch die Möglichkeit, spezifische Anforderungen der Kunden effizienter zu adressieren.

Die Basis für das Verständnis der Struktur der *Order* bildet die offizielle Salesforce-Dokumentation sowie auf einer projektinternen Datenspezifikation, welche das Objekt festgelegt und speziell angepasst hat[9]. Laut der Salesforce Object Reference bietet die *Order* in Salesforce Lightning eine vielseitige und anpassungsfähige Grundlage, die für eine breite Palette von Geschäftsprozessen geeignet ist. Im Gegensatz zum alten *CC-Order*, das für spezifische Anforderungen in B2B-Commerce entwickelt wurde, bietet das neue *Oder*-Objekt eine modernere Basis mit erweiterten Funktionen, die eine bessere Automatisierung, verbesserte Reporting-Möglichkeiten und eine tiefere Integration in andere Salesforce-Module.[10]

Einerseits wird eine bessere Skalierbarkeit erreicht, welches die *Order* in Lightning nativ in die Plattform integriert und somit nahtlos mit anderen Salesforce-Funktionen zusammenarbeitet. Dies erleichtert die Implementierung von Cross-Functional-Workflows und verbessert die Datengenauigkeit durch eine konsistentere Nutzung von Salesforce-Standards.[11]

Zunächst ist festzuhalten, dass die grundlegenden Felder wie *Id*, *OwnerId*, *Name* und *CreatedDate*, die bereits vorher in SFC vorhanden waren, von Salesforce beibehalten wurden.

---

[9] Vgl. [DSL23]
[10] Vgl. [SFO23]
[11] Vgl. Ebenda

Die Einführung neuer Felder wie *ContractId*, *Pricebook2Id*, *OriginalOrderId* und *OpportunityId* erweitert jedoch die Funktionalität der neuen Objekte. Besonders hervorzuheben sind die Felder, die zuvor in *CC-ContactAddress* (BillTo) enthalten waren, wie *BillingStreet*, *BillingCity* und *BillingAddress*. Diese wurden nun als separate nicht relationale Felder direkt zur *Order* hinzugefügt, was eine differenziertere und genauere Adressverwaltung ermöglicht.

Die Reduktion von Relationen, wobei wichtige Beziehungen wie zum *PriceBook*, dem *Store* sowie zu *User/Accounts* erhalten bleiben, sorgt für eine vereinfachte und übersichtlichere Struktur. Die Einführung von neuen Objekten wie *OrderDeliveryGroup* und *OrderDeliveryMethod* trägt zur Vereinfachung der Bestellprozesse bei, indem sie eine flexiblere Handhabung von Liefermethoden und -adressen ermöglichen. Beispielsweise sind unter anderem in der *OrderDeliveryGroup* alle *OrderItems* sowie die alte *ShipTo*-Relation hinterlegt, welche die gleiche Liefermethode teilen. Zu all diesen Objekten wurden auch viele eigene nicht relationale Felder hinzugefügt, um die spezifischen Anforderungen des Kunden zu erfüllen. Auch liegen hier wieder viele Prozesse im Hintergrund, welche zusätzliche Funktionen der Bestellungen erfüllen. So auch wie im alten System die Integration zum SAP-System. Diese funktioniert ähnlich wie in SLC. Da sich die Objektstruktur aber von SFC zu SFL unterscheidet mussten hier die Prozesse angepasst werden. Die Bestellung wird ähnlich wie im alten System, für jede Kategorie aufgeteilt in mehrere *VarietyEntrys* mit einer dazugehörigen *OrderDeliveryEntry* und dessen *OrderItemEntry*. Ist die Anlegung der Daten in SAP fertiggestellt werden in Salesforce direkt aus diesen *VarietyEntrys* neue Salesforce interne Objekte erzeugt. Aus einer Bestellung mit 2 Produkten aus verschiedenen Kategorien werden erst von Salesforce ein einziges *Order*-, *Orderitem*- und *OrderDeliveryGroup*-Objekt erstellt. Dies wird dann in jeweils 2 *VarietyEntrys* aufgeteilt (aber noch nicht direkt in Salesforce). Ist dies vom SAP geschehen, werden nun aus der eigentlichen *OrderDeliveryGroup*/*OrderItems* gleichwertig 2 einzelne erstellt. Weitere Prozesse sind durch z.B. definierte Formular-Felder wie *TotalTaxAmount* oder *SubTotalAmount* welche Zahlenberechnungen durchführen definiert.

Im Vergleich zur alten *CC-Order* in SFC, dass eher spezifische und begrenzte Funktionen aufwiesen, zeigt das neue Order-Objekt in SFL eine deutlich breitere Anwendbarkeit und Flexibilität. Durch diese maßgeschneiderten Anpassungen wird nicht nur eine präzisere Abbildung der Geschäftsprozesse erreicht, sondern auch die Datenqualität und

Prozesseffizienz wesentlich verbessert. Die Migration und Anpassung des Oder-Objekts in Salesforce Lightning ist somit ein entscheidender Schritt in der digitalen Transformation des Onlineshops des Kunden. Sie spiegelt nicht nur die technische Weiterentwicklung der Plattform wider, sondern stellt auch eine strategische Verbesserung dar, die das Bestellmanagement effizienter und kundenorientierter gestaltet.

4 Datenmigrationskonzept

Dieses Kapitel beschäftigt sich dem Konzept der eigentlichen Datenmigration. Es zielt darauf ab, einen methodischen Ansatz für diese zu definieren und die verschiedenen Aspekte des Prozesses zu beleuchten. Die Definition klarer Ziele und Erfolgskriterien bildet den Grundstein dieses Kapitels. Diese Zielsetzungen sind von essenzieller Bedeutung, um sicherzustellen, dass die Datenmigration die geschäftlichen Anforderungen und Erwartungen des Kunden erfüllt. Dies wird anhand einer globalen Definition of Done geschehen.

Die Auswahl der geeigneten Migrationsmethode stellt einen weiteren wichtigen Schritt dar. Salesforce bietet diverse Methoden zur Datenmigration an und daher wird dieses Kapitel die Auswahlkriterien der jeweiligen Methode erörtern, um die geeignetste auszuwählen. Ist die Methode ausgewählt kann die Planung der Datenmigration stattfinden. Dabei wird unteranderem definiert, welche Daten aus dem bestehenden System übernommen werden. Abschließend wird eine umfassende Teststrategie entwickelt, um sicherzustellen, dass die Datenmigration reibungslos verläuft. Dies umfasst die Planung und Durchführung von Testszenarien, um die Integrität der übertragenen Daten und die Funktionalität der neuen Salesforce Umgebung sicherzustellen.

4.1 Ziele und Erfolgskriterien der Migration

Die Durchführung einer Datenmigration ist ein entscheidender Schritt bei der Einführung eines neuen Systems oder der Aktualisierung einer bestehenden Plattform. Um sicherzustellen, dass dieser Prozess mit möglichst wenig Problemen verläuft und den angestrebten Nutzen bringt, ist es von entscheidender Bedeutung, klare Ziele und Erfolgskriterien festzulegen. Dieses Kapitel widmet sich der Definition dieser Ziele und der Festlegung der Kriterien, die erfüllt sein müssen, um den erfolgreichen Abschluss der Migration zu definieren. Dabei sollen der Migration folgende Ziele zugrunde liegen:

Ziel 1: Vollständigkeit der Daten

Das Hauptziel ist die Gewährleistung der Vollständigkeit der übertragenen Daten. Alle relevanten Daten (Felder und Objekte), welche in Kapitel 4.3 definiert werden, aus dem bestehenden System müssen in Salesforce migriert werden, um sicherzustellen, dass sämtliche Informationen für zukünftige Anforderungen zur Verfügung stehen.

Ziel 2: Datenqualität und Integrität

Die Datenqualität und -integrität müssen während des Migrationsprozesses sichergestellt werden. Dies beinhaltet die Überprüfung auf Datenfehler, Duplikate und Inkonsistenzen, um sicherzustellen, dass die in Salesforce gespeicherten Daten genau, konsistent und verlässlich sind.

Nun soll eine globale[12] Definition of Done für diese Datenmigration definiert werden. Durch diese kann exakt festgehalten werden, ob die Datenmigration vollständig abgeschlossen ist. In der folgenden Tabelle werden die Kriterien dieser ausgelistet:

Titel	Kriterium
Übernahme aller relevanten relationalen Objekte	_ relationale Objekte wurden übernommen (3)
Übernahme aller relevanten Daten	_ % der Daten wurden übernommen (>75%)
Mindestens einen Test geschrieben	_ Tests wurden geschrieben (>0)
Dokumentation auf Confluence	○ Die Seite ist öffentlich zugänglich ○ Alle Schritte des Ablaufdiagramms sind beschrieben und dargestellt

Tabelle 1: Kriterien der Definition of Done

Das erste Kriterium trägt dazu bei der Verknüpfung aller wichtigen relationalen Objekte mit dem neuen Objekt zu erhalten. Es ist entscheidend für die Aufrechterhaltung der Datenbeziehungen und -integrität. Außerdem sorgt es dafür, dass das neue System die notwendigen Verbindungen zu anderen relevanten Datenpunkten aufweist, was für die Gesamtfunktionalität und Effizienz des Systems unerlässlich ist. Auch das zweite Kriterium dient dazu, die Datenintegrität beizubehalten. Welche Felder und Objekte übernommen werden sind in Kapitel 4.3 beschrieben. Dabei können grundsätzlich zwei Szenarien eintreten. Entweder werden die wichtigsten Felder für eine Order-Historie übernommen, welche nur für die Nutzer des Onlineshops nötig wären oder es werden so gut wie alle nötigen Informationen übernommen, um diese in die Prozesse des Ordermanagements zu integrieren.

[12] Eine globale Definition of Done gilt nicht für eine Aufgabe, sondern gesammelt für ein Projekt um die Fertigstellung zu Messen

Abschließend soll eine Dokumentation in Confluence erstellt werden, welche ein Ablaufdiagramm inklusive der Beschreibung beinhaltet. Diese dient dazu einen zentralen, leicht zugänglichen Informationspunkt für den Kunden und Mitarbeiter, welche vor ähnlichen Herausforderungen stehen, zu bieten. Das detaillierte Ablaufdiagramm des Skripts auf dieser Seite hilft dabei, den Prozess für alle Beteiligten transparent und nachvollziehbar zu machen. Es dient als visuelle Unterstützung, um den Workflow und eventuelle Abhängigkeiten klar zu verstehen.

4.2 Auswahl der Migrationsmethode

In diesem Kapitel soll die Auswahl der Migrationsmethode im Mittelpunkt stehen. Wie bereits erwähnt ist die Migration auf Grund des neuen Projektes für die dotSource SE bereits erfolgt. Dabei wurden das zu Migrierende Objekt *CC-Order* und dessen Relationen bereits in die neue Salesforce Organisation übernommen und gespeichert. Nun müssen die bereits migrierten Objekte in die neuen Objektstrukturen umgewandelt werden.
Solche Fälle treten häufig auf, wenn Organisationen ihre Salesforce-Instanz über Jahre hinweg weiterentwickeln und aktualisieren. Mit dem Aufkommen neuer und effizienteren Objektstrukturen, wie sie durch die Einführung von Lightning bereitgestellt werden, wird die Überführung dieser Daten in neue Objekte notwendig, sobald sich für eine Migration entschieden wird. Die Herausforderung liegt darin, die Daten so zu transformieren und zu integrieren, dass sie nicht nur in den neuen Strukturen gespeichert, sondern auch für aktuelle und zukünftige Anwendungen genutzt werden können. In den folgenden Abschnitten werden verschiedene Methoden einer Datenmigration in Salesforce vorgestellt und bewertet, um die geeignetste Lösung für den Kunden zu finden.

4.2.1 **Data Import Wizard**

Der Data Import Wizard ist ein in Salesforce integriertes Werkzeug, welches für die Datenmigration entworfen wurde. Dieses Tool zeichnet sich dadurch, dass es ohne eine zusätzliche Konfiguration direkt über das Salesforce Setup zugänglich ist. Die Hauptfunktionen ermöglichen es den Benutzern:

- Das Einfügen von neuen Datensätzen (Insert)
- Das aktualisieren bestehender Datensätze (Update)
- Durch eine Kombination dieser beiden, wobei mittels eines Primär- oder Fremdschlüssels geprüft wird, ob der gesuchte Datensatz bereits existiert (Upsert)

Der Data Import Wizard unterstützt dabei eine Vielzahl von Standard- aber auch benutzerdefinierte Objekte. Es ist jedoch wichtig zu beachten, dass der Wizard nicht für alle Standardobjekte, wie beispielsweise Opportunities, geeignet ist (Abbildung 3).[13]

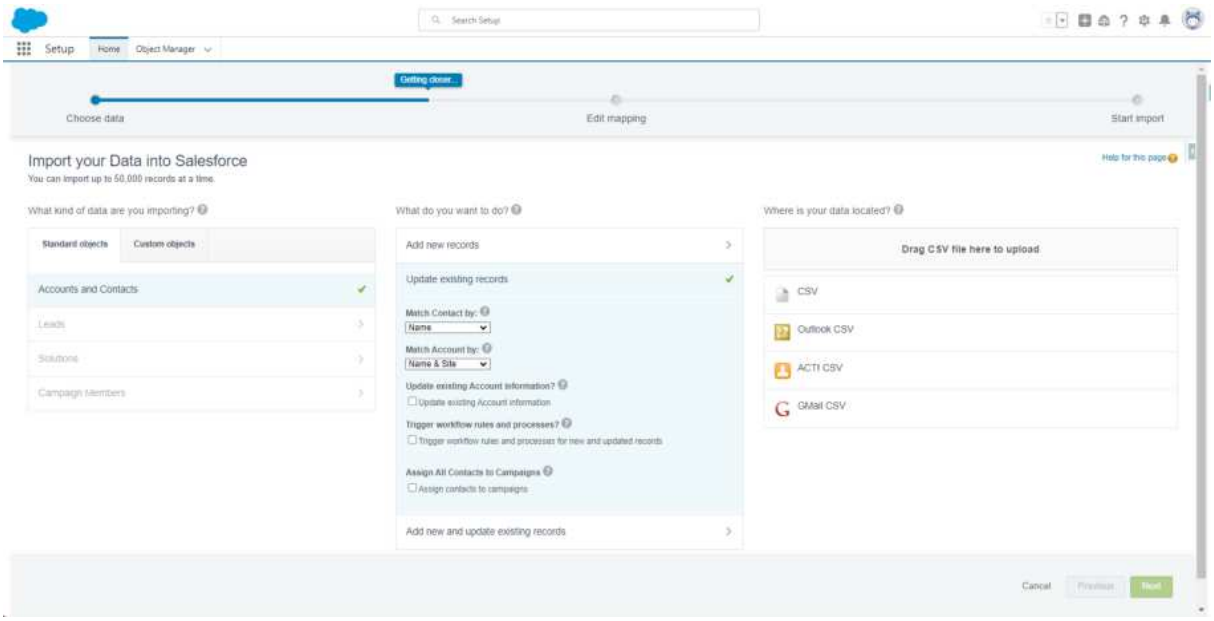


Abbildung 3: Dataimport Wizard
Quelle: Screenshot aus Kunden-Sandbox

Dabei eignet sich das Tool besonders für das Importieren von Datensätzen in kleinerem Umfang. Er kann bis zu 50.000 Datensätze auf einmal verarbeiten, was ihn zu einer guten Lösung für kleinere bis mittelgroße Datenmigrationsprojekte macht. Zudem bietet der Wizard nützliche Features wie die Vermeidung von Duplikaten durch Auswahl passender Abgleichkriterien (z.B. E-Mail oder ID) und die Möglichkeit, Prozesse während eines Imports, Updates oder Upserts zu deaktivieren.[14]

Für Organisationen, die eine schnelle und einfache Lösung zur Datenmigration benötigen, ohne die Komplexität externer Tools oder umfangreicher Konfigurationen, stellt der Data Import Wizard eine praktikable und effiziente Option dar. Er ist besonders nützlich für Standardmigrationsszenarien, in denen eine begrenzte Anzahl von Datensätzen betroffen ist und die Datenstrukturen gut zu den vom Wizard unterstützten Objekttypen passen.

[13] Vgl. [SFH23]
[14] Vgl. Ebenda

4.2.2 **Data Loader**

Der Data Loader ist ein externes Tool, das speziell für Salesforce-Datenmigrationen entwickelt wurde. Im Gegensatz zum Data Import Wizard, der direkt in Salesforce integriert ist, muss der Data Loader lokal auf einem Gerät installiert werden. Dabei wird es ermöglicht eine Verbindung zur jeweiligen Salesforce Organisation herzustellen. Über das Setup-Menü in Salesforce kann dieser heruntergeladen und installiert werden, mit separaten Versionen für Windows und Mac.[15] Das Tool bietet eine breitere Palette an Interaktionsmöglichkeiten mit Salesforce-Daten im Vergleich zum Data Import Wizard. Zu den Hauptfunktionen des Tools gehören:

- Das Einfügen von neuen Datensätzen (Insert)
- Das Aktualisieren bestehender Datensätze (Update)
- Durch eine Kombination dieser beiden, wobei mittels eines Primär- oder Fremdschlüssels geprüft wird, ob der gesuchte Datensatz bereits existiert (Upsert)
- Das Entfernen von Datensätzen (Delete)
- Das Exportieren von Datensätzen in eine CSV-Datei (Export)

Er eignet sich für den Umgang mit großen Datenmengen und komplexeren Migrationsszenarien (Bis zu 5 Millionen Datensätze). Im Gegensatz zum Data Import Wizard bietet Er eine robustere Lösung für umfangreichere Datenmigrationen mit vielen Feldern oder Objekten. Das Tool ist besonders geeignet für Szenarien, in denen eine umfassende Kontrolle über den Migrationsprozess erforderlich ist, wie zum Beispiel bei der Migration von komplexen Datenstrukturen oder bei der Notwendigkeit, Daten vollständig aus dem System zu entfernen. Auch unterstützt Er, anders als bei dem Import Wizard, alle Standard- und Benutzerdefinierte Objekte.

Der Data Loader eine gute Wahl für Organisationen, die eine leistungsstarke, vielseitige und anpassbare Lösung für ihre Salesforce-Datenmigrationsbedürfnisse benötigen, insbesondere wenn diese Bedürfnisse über die Grundfunktionalitäten des Data Import Wizards hinausgehen.[16]

[15] Vgl. [SFO23]
[16] Vgl. Ebenda

4.2.3 **Apex-Skripte in Verbindung mit Salesforce API**

Apex kombiniert mit den Salesforce APIs, bieten eine flexible und leistungsstarke Möglichkeit zur Datenmigration und -manipulation. Diese Methode beinhaltet die Erstellung von benutzerdefinierten Skripten und deren Ausführung über die Salesforce APIs. Apex ist eine objektorientierte Programmiersprache, die es Entwicklern ermöglicht, komplexe Geschäftslogik innerhalb der Salesforce-Plattform auszuführen. Durch die Verwendung von Apex in Kombination mit der Salesforce API können Entwickler maßgeschneiderte Datenmigrations- und Manipulationsprozesse erstellen. Diese Skripte können direkt in der Salesforce-Umgebung ausgeführt werden, wodurch ein hohes Maß an Kontrolle und Flexibilität bei der Handhabung von Daten gewährleistet wird. Folgende API-Schnittstellen bietet Salesforce an:[17]

REST-API

- Format: Datenübertragung erfolgt im XML- oder JSON-Format
- Funktionalitäten: Unterstützt Create, Read, Upsert und Delete
- Einsatzbereiche: Besonders geeignet für mobile Anwendungen und Websites

SOAP-API

- Format: Datenaustausch über eine WSDL-Datei (XML-basiert)
- Anwendungsbereich: Ideal für dauerhafte System-zu-System-Integrationen

Bulk-API

- Funktionsweise: Ähnlich der REST-API, jedoch für das Laden und Suchen großer Datenmengen innerhalb einer Transaktion konzipiert
- Datenvolumen: Geeignet für mehr als 50.000 Datensätze
- Einsatzgebiete: Besonders nützlich für große Aufgaben wie Datenmigrationen

Streaming-API

- Konzept: Basierend auf dem Publish-and-Subscribe-Prinzip
- Anwendungsbereiche: Ideal für die Interaktion zwischen Systemen bei Datenänderungen und für entkoppelte Anwendungen

[17] Vgl. [DIP22] S.230 ff.

4.2.4 **DataLoader.io**

DataLoader.io ist eine Cloud-basierte Lösung von Salesforce, die sich durch eine benutzerfreundliche Oberfläche und vielseitige Datenmanipulationsfunktionen auszeichnet. Dabei bieten diese folgenden Eigenschaften:

- DataLoader.io ermöglicht mittels bestehenden Salesforce-Anmeldeinformationen über OAuth 2.0 einzuloggen, ohne eine Anwendung herunterladen zu müssen
- Auto-Mapping, Tastaturkürzel und Suchfilter, die den Zeitaufwand für die Zuordnung von Daten aus der Quelldatei zu den Salesforce-Feldern reduzieren
- Ermöglicht den Export verwandter Objekte durch einen einzelnen Zugriff, wodurch manuelle und redundante Arbeit, wie das mehrfache Ziehen von Datensätzen und deren Neuzuordnung z.B. in Excel, entfällt
- Unterstützung von Import und Export von Daten direkt aus z.B. Dropbox, FTP- und SFTP-Repositories, was die Handhabung und Zugänglichkeit von Daten vereinfacht
- Aufgaben für den Import und Export von Daten können automatisch auf stündlicher, täglicher, wöchentlicher oder monatlicher Basis geplant werden

Data Loader.io ist besonders geeignet für Anwender, die eine Cloud-basierte Lösung bevorzugen und einfache bis mittelschwere Datenmigrationen durchführen möchten. Es ist ideal für Situationen, in denen Benutzer eine schnelle, einfache und automatisierte Möglichkeit suchen, ohne die Notwendigkeit lokaler Installationen oder komplexer Konfigurationen.
Die Seite eignet sich für eine breite Palette von Anwendungen und bietet dabei eine intuitive Benutzeroberfläche, die die Handhabung von Datenmigrationen vereinfacht und effizient gestaltet.[18]

Nun soll mittels einer Nutzwertanalyse entschieden werden, welche der Methoden am besten für den spezifischen Fall des Kunden geeignet ist. Im Folgenden soll die Auswahl der Kriterien näher betrachtet werden. Das erste Kriterium *Asynchronität* stellt die Möglichkeit einer asynchronen Arbeitsweise sicher. Dies ist wichtig, da bei vielen tausenden Einträgen das System z.B. überlastet oder für einige Zeit nicht nutzbar werden kann.
Um dies zu verhindern, sollten aufwendige Prozesse asynchron ausgeführt werden, damit das Hauptsystem nicht beeinträchtigt wird.

[18] Vlg. [SDL23]

Das zweite Kriterium überprüft die *Komplexität* der jeweiligen Tools. Ist ein Tool sehr komplex und muss gleichzeitig eine sehr komplexe Aufgabe erfüllen, kann sich die Zeit für die notwendige Auseinandersetzung für das Tool um ein Vielfaches erhöhen. Dann wäre es besser ein Tool zu verwenden, was zwar einfach in der Bedienung ist, aber auch komplexe Aufgaben schnell erfüllt. Das dritte Kriterium stellt die *Individualisierbarkeit* sicher. Durch die sehr spezifischen Anforderungen des Kunden ist es wichtig, dass das Tool anpassbar ist, um diesen Anforderungen zu entsprechen. Dazu gehören z.B. Regeln oder Strukturen, welche nicht direkt aus Salesforce stammen. Das letzte Kriterium ist dabei das wichtigste. Dieses stellt sicher das die Salesforce internen *Recordlimits* nicht überschritten werden. Da die zugrundeliegende Datenmigration mehrere Tausend daten beinhaltet muss das Tool mit diesen auch umgehen können, ohne diese Recordlimits zu verletzen. Durch diese vier Kriterien ergeben sich folgende Gewichtungen: Asynchronität 25%, Komplexität 20%, Individualisierbarkeit 25% und Recordlimits 30%. In der folgenden Tabelle wird die Nutzwertanalyse dargestellt und ausgewertet. Die Bewertungsskala wurde 1 (geringste Erfüllung) bis 10 (höchste Erfüllung) festgelegt.

Tool	Asynchronität	Komplexität	Individualisierbarkeit	Recordlimits
Data Import Wizard	5	4	3	4
Data Loader	6	7	6	7
Apex	9	9	10	10
DataLoader.io	7	6	7	8

Tabelle 2: Nutzwertanalyse der Salesforce-tools

Wie in Kapitel 4.2.1 erwähnt ist der Data Import Wizard vor allem für kleinere Migrationsprojekte konzipiert. Seine direkte Integration in Salesforce ermöglicht zwar eine einfache Handhabung, schränkt jedoch seine Fähigkeit zu asynchronen Operationen ein, was zu einer mittleren Bewertung führt (5 Punkte). In Bezug auf die Komplexität bietet der Wizard Unterstützung für eine Vielzahl von Standard- und benutzerdefinierten Objekten, stößt aber bei komplexeren Migrationsszenarien an seine Grenzen (4 Punkte). Die Individualisierbarkeit ist durch die einfache und direkte Integration begrenzt (3 Punkte).
Mit einer Kapazität von bis zu 50.000 Datensätzen ist der Wizard für kleine bis mittelgroße Projekte ausreichend, erreicht aber bei sehr großen Datenmengen keine Möglichkeit (4 Punkte). Daher erreicht dieser eine Gesamtwertung 4.0 von 10 Punkten.

Der Data Loader ist ein externes Tool (Nachzulesen in Kapitel 4.2.2), dass eine breitere Palette an Funktionen für Salesforce-Datenmigrationen bietet und sich besonders für den Umgang mit großen Datenmengen (Bis zu 5 Millionen) und komplexeren Migrationsszenarien eignet. Dies spiegelt sich in einer höheren Bewertung in den Bereichen Komplexität (7 Punkte) und Record Limits (7 Punkte) wider. Auch bietet der Data Loader auch eine bessere asynchrone Verarbeitung da dieser die Salesforce Bulk Api nutzt (6 Punkte). Zudem ermöglicht seine robuste Funktionalität eine höhere Anpassungsfähigkeit, da er alle Standard- und benutzerdefinierten Objekte unterstützt (6 Punkte). Daher erreicht dieser eine Gesamtwertung 6.5 von 10 Punkten.

Die Kombination von Apex-Skripten mit der Salesforce API bietet das höchste Maß an Flexibilität und Kontrolle über den Migrationsprozess. Dies spiegelt sich in hohen Bewertungen in den Bereichen Asynchronität (9 Punkte), Komplexität (9 Punkte) und Individualisierbarkeit (10 Punkte) wider. Apex ermöglicht die Handhabung komplexer Migrations- und Manipulationsszenarien und bietet maximale Anpassungsfähigkeit, da die Skripte vollständig an spezifische Anforderungen angepasst werden können. In Bezug auf Record Limits können mit Apex und der Salesforce API sehr große Datenvolumen effizient gehandhabt werden, was sie ideal für umfangreiche Datenmigrationen macht (10 Punkte). erreicht dieser eine Gesamtwertung 9.55 von 10 Punkten.

DataLoader.io als Cloud-basierte Lösung bietet eine gute Unterstützung für asynchrone Operationen (7 Punkte) und ist besonders für Situationen geeignet, in denen Benutzer eine schnelle und einfache Lösung für Datenmigrationen suchen (6 Punkte). Die Funktionen wie Auto-Mapping und die Unterstützung verschiedener Datenquellen bieten eine gute Anpassungsfähigkeit (7 Punkte), erreichen aber nicht das Niveau von Apex. In Bezug auf Record Limits ist DataLoader.io für kleine bis mittelgroße Migrationen geeignet (8 Punkte). Daher erreicht dieser eine Gesamtwertung 7.1 von 10 Punkten.

Damit spiegelt sich eindeutig wider, dass die Migration mittels eines Apex-Skriptes erfolgen soll. Zwar ist der Aufwand ein solches Skript zu schreiben deutlich höher als bei den vorgefertigten Tools, aber dafür ist die Individualisierbarkeit, Testbarkeit und Kontrolle im deutlichen Vorteil.

4.3 Planung der Migration

Vorerst musste mit Angehörigen des Teams geklärt werden welche Felder und Objekte zu migrierenden sind. Da dies aber zu unregelmäßigen Zeitpunkten kleinere spontane Gespräche waren, wurde dies nicht transkribiert. Dabei wurde festgelegt, dass die Felder übernommen werden sollen, welche im Prozess der Erzeugung der bereits erwähnten *VarietyEntry* übertragen werden. Diese Felder sind dem Skript des alten SFC-Systems zu entnehmen und sind im Anhang 2 beigefügt. Diese Entscheidung wurde auf der Grundlage getroffen, da nur diese Felder auch für den Kunden wichtig sind und in den Prozessen von Salesforce beschrieben werden. Ausgeschlossen davon sind die Felder, die für die Salesforce Prozesse notwendig sind (z.B. *Id* oder *CreatedById*). Dabei soll geschaut werden, inwieweit sich diese Felder befüllen lassen, da diese von Salesforce standardmäßig automatisch befüllt werden. Eine weitere Differenzierung erfolgte durch die Farbkodierung der Felder: Blau markierte Felder sollten von automatischen Prozessen entkoppelt werden um unerwünschte Berechnungen, wie die Aktualisierung des Lagerbestands, zu vermeiden und Rote werden nicht übernommen bzw. sind nicht notwendig. Für die Migration wurde außerdem entschieden, dass nur die relationalen Objekte *CC-OrderItem*, *CC-Order* und *CC-ContactAddress* nötig sind. In den anderen relationalen Objekten werden keine relevanten Daten gespeichert oder diese werden nicht aktiv vom System genutzt bzw. wurden schon bei der Migration anderer Daten übernommen. Dazu zählt beispielsweise das *Contact* oder *Account* Objekt.

Somit ist die Menge der zu übernehmenden Felder für die Definition of Done auf die Anzahl der in Anhang 2 hinterlegten Felder und die zu übernehmende Objekte auf 3 festgelegt. Außerdem wird festgelegt, dass das Skript dynamisch und anpassbar sein sollte. Dies ist auf die Ungewissheit der Datenlage und Anforderungen seitens des Kunden zurückzuführen. Damit wird sichergestellt das das Konzept auf diese reagieren kann. Auf Basis dieser Erkenntnisse wird nun ein vorläufiger Plan für das Migrations-Skript erstellt, welcher im Ablaufdiagramm visualisiert ist. Die Hauptkomponenten dieses Plans umfassen die Erstellung einer *OrderConversion*-Klasse zur Speicherung der abgefragten Daten, das Mapping dieser Daten auf die neue Struktur in Salesforce Lightning und schließlich die Speicherung bzw. das Einfügen der neuen Objekte in das System. Das Ablaufdiagramm kann im Anhang 3 angesehen werden.

Der Prozess der Implementierung wird in Kapitel 5.1 genau beschrieben. Ist das Skript erstellt, soll dieses samt dem *ConversionObjekt* ausführlich getestet werden. Dieser Schritt ist entscheidend, um die Funktionalität und Korrektheit des Migrationsskriptes zu gewährleisten. Nach erfolgreichem Abschluss der Tests wurde das Skript in der Salesforce Lightning Sandbox-Umgebung implementiert. Damit ist die Datenmigration vorbereitet und kann nach Wunsch des Kunden durchgeführt werden.

5 Implementierung des Konzeptes

In diesem Teil der Arbeit wird das Skript der Datenmigration und die Konvertierung von *CC-Order* in Salesforce behandelt. Beginnend mit der Entwicklung einer Klasse zur Konvertierung. Danach soll eine Apex-Methode zur Datenerfassung implementiert werden, welche alle relevanten Objekte und Felder aus Salesforce extrahiert.

Anschließend folgt eine Konvertierung der Daten mittels der zuvor erstellen *OrderConvertionHelper*-Klasse. Danach erfolgt das Mapping der Daten, bei dem die Felder der *CC-Order* den entsprechenden *Order*-Feldern und relationalen Objekten zugeordnet werden.

Das Skript endet mit dem Einfügen der transformierten Daten in Salesforce und deren Validierung durch umfassende Tests. Abschließend werden Herausforderungen und Probleme, die während der Migration aufgetreten sind, diskutiert und Lösungsansätze aufgezeigt, was wichtige Einblicke für zukünftige Projekte in diesem Bereich bietet.

5.1 Umsetzung

Zu Beginn der Umsetzungsphase der Datenmigration wurde ein neuer Branch erstellt und eines speziell dafür vorgesehenen Ordners im Lightning-System angelegt. Die Entwicklung des *OrderConversionObjekts* stellt den ersten Schritt in der technischen Umsetzung dar. Die Notwendigkeit eines solchen spezialisierten Objekts ergibt sich aus mehreren Gründen. Erstens verhindert das *OrderConversionObjekt* die Limitationen von SOQL-Abfragegrenzen in Salesforce. Durch die Vermeidung von Abfragen innerhalb von Schleifen wird eine effizientere und ressourcenschonendere Datenverarbeitung ermöglicht. Zweitens dient das Objekt als zentraler Speicherort für die migrierten Daten mit diesen dann einfach ohne Abfragen gearbeitet werden kann. Diese Architektur ermöglicht es, Änderungen und Anpassungen im Migrationsverlauf präzise zu verfolgen und bei Bedarf einfache Maßnahmen einzuleiten.

OrderConvertionHelper

Die Beschreibung der Struktur erfolgt anhand der Abbildung 4.

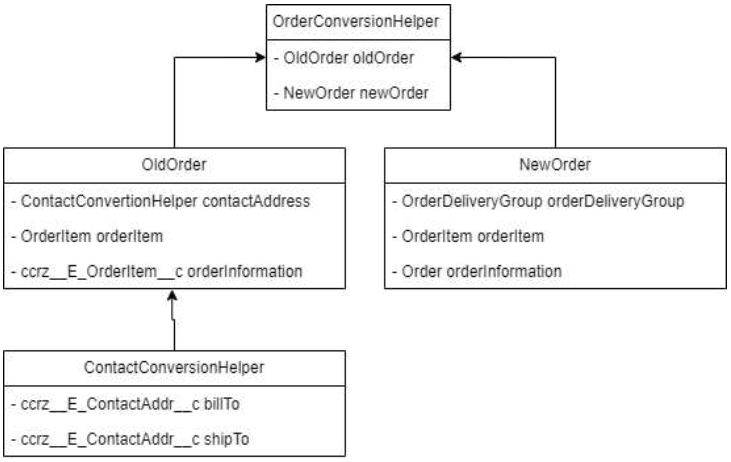


Abbildung 4: OrderConvertionHelper

Im Detail besteht das *OrderConversionObjekt* aus den Attributen *NewOrder* und *OldOrder*, welche beide vom Typ derselben Klasse sind. Damit wird die alte und neue Order voneinander entkoppelt und kann direkt zugeordnet werden. Innerhalb von *NewOrder* werden Attribute wie *OrderDeliveryMethod*, *OrderItem* und *OrderInformation* (Also die eigentliche Order) definiert. Die Klasse *OldOrder* beinhaltet ähnlich auch die *OrderInformation*, das *OrderItem* und die zusätzliche Klasse *OrderDeliveryGroupConverter*. Letztere umfasst die Attribute *Billing-* und *ShippingAddress* des Typs ContactAddress.

Die Entscheidung, ContactAddress nicht in einem einzelnen Objekt zusammenzufassen, basiert auf der spezifischen Architektur von Salesforce Lightning. In dieser Umgebung werden ContactAddress-Informationen auf verschiedene Objekte aufgeteilt. Diese Strukturierung erleichtert den Zugriff auf die Daten und dessen spätere Zuordnung.

Selektieren der Objekte

Nun wurde eine Methode implementiert, welche alle nötigen Objekte mit deren Relationen und Feldern über eine SOQL-Query selektiert. Diese nutzt zwei Maps mit dem Typ *Map<String, String>*, mit dem Bezeichner o*rderFieldMapping* und *orderItemFieldMapping*. Der Key beinhaltet jeweils den ursprünglichen Feldnamen (z.B. ccrz__BuyerEmail__c), während die Werte die neuen Feldnamen darstellen (z.B. OrderEmail__c). Die Zuordnung ist in Anlage 3 der Arbeit dokumentiert. Diese basieren auf in Salesforce existierenden Beschreibungen, welche durch die dotSource SE im Rahmen des Kundenprojektes angelegt wurden. Der Grund für die Verwendung des alten Feldnamens als Schlüssel in den Maps liegt darin, dass es dadurch möglich wird, mittels des Schlüssels durch die Map zu iterieren und über die Methode *getValue(key)* direkt auf den neuen Feldnamen zuzugreifen.

Dies bietet einerseits eine hohe Flexibilität und erlaubt eine dynamische und einfache Erweiterung des Mappings, falls eine Anpassung an die Felder vorgenommen werden muss.

Die eigentliche Abfrage der Daten erfolgt über *Database.query()* Methode in Apex. Der Query-String wird dynamisch aus den Schlüssel-Wert-Paaren der KeyMap generiert. Dieser Ansatz ermöglicht es, die Abfrage flexibel und anpassungsfähig zu gestalten. Als Ergebnis liefert die Methode eine Liste von *CC-Order*-Objekten, einschließlich der zugehörigen *OrderItem*-Relationen (Siehe unten). Dies bedeutet, dass alle relevanten Informationen in einem einzigen Objekt zusammengefasst sind, wodurch separate Abfragen überflüssig werden. Tritt während der Abfrage ein Fehler auf, gibt die Methode eine leere Liste zurück und wirft eine Exception. Dieses Vorgehen erlaubt es, den Migrationsprozess bei Bedarf abzubrechen und bietet eine kontrollierbare Fehlerbehandlung. Im Erfolgsfall liefert die Methode eine Liste von CC-Orders, die direkt aus der Methode extrahiert werden können.

```
private static List<ccrz__E_Order__c> selectAllObjects()
{
    List<ccrz__E_Order__c> orderList;

    try {
        String orderFieldListString =
        String.join(orderFieldMapping.keySet(), ',');

        String orderItemfieldListString =
         String.join(orderItemFieldMapping.keySet(), ',');

        String soqlQuery = 'SELECT ' + orderFieldListString         +
                           ', (SELECT ' + orderItemfieldListString +
                           ' FROM ccrz__E_OrderItems__r)'           +
                           'FROM ccrz__E_Order__c';

        orderList = Database.query(soqlQuery);
    } catch (Exception e)
    {
        System.debug('Failed OrderQuery : ' + e.getMessage());
        orderList = new List<ccrz__E_Order__c>();
    }
    return orderList;
}
```

Konvertierung zum OrderConvertionHelper

Nun wird die Methode zur Konvertierung in das *OrderConvertionHelper*-Objekt beschrieben (Siehe nächste Seite). Die Methode beinhaltet als Parameter einer Liste von *CC-Order*-Objekten, die durch die vorherige Methode erstellt wurde.

Zunächst wird überprüft, ob die Liste der Orders leer oder null ist. Ist dies der Fall, wird eine Exception ausgelöst, um zu gewährleisten, dass nur valide Daten verarbeitet werden. Während der Iteration über die Liste der *CC-Orders* wird für jedes Objekt ein neuer *OrderConversionHelper* erstellt. Innerhalb jedes dieser Objekte wird die *OldOrder* festgelegt, welche die *BillTo*- und *ShipTo*-Relationen der ursprünglichen *CC-Order* beinhaltet.

```
private static List<OrderConversionHelper> createHelperObjects(List<ccrz__E_Order__c>
oldOrders)
{

    if (oldOrders == null)
    {
        throw new IllegalArgumentException('CC-Orders are null');
    }

    List<OrderConversionHelper> convertedOrders = new List<OrderConversionHelper>();

    for (ccrz__E_Order__c oldOrder : oldOrders)
    {
    try
     {
      // Create ContactConversionHelper + Select Orderitems
      OrderConversionHelper.ContactConversionHelper contactHelper =
        new OrderConversionHelper
        .ContactConversionHelper(oldOrder.ccrz__BillTo__r, oldOrder.ccrz__ShipTo__r);

      ccrz__E_OrderItem__c[] orderItems = oldOrder.ccrz__E_OrderItems__r;

      // Create OldOrder
      OrderConversionHelper.OldOrder oldOrderObj =
       new OrderConversionHelper.OldOrder(contactHelper, orderItems, oldOrder);

      // Add NewOrder
      OrderConversionHelper.NewOrder newOrderObj =
      new OrderConversionHelper.NewOrder(new OrderDeliveryGroup(),
      new List<OrderItem>(),
      new Order());

      // Add ConvertionHelper
      OrderConversionHelper conversionHelper =
      new OrderConversionHelper(oldOrderObj, newOrderObj);

      convertedOrders.add(conversionHelper);

      }  catch (Exception e)
      {
       System.debug('Failed to Create a HelperObject : ' + e.getMessage());
      }

      }
          return convertedOrders;
      }
```

Wird in Salesforce ein SObject wie *ContactAddress* mit einer Relation wie *ccrz__E_BillTo__r* befüllt, werden die entsprechenden Felder automatisch in das Objekt übernommen. Der Konstruktor *OldOrder* des *OrderConversionHelper*-Objekts nimmt ein *ContactConversionObjekt*, die *OrderItems__r* und das aktuelle *CC-Order*-Objekt als Parameter. Der *NewOrder*-Konstruktor erstellt jeweils ein neues Objekt (Order, List<OrderItem> und DeliveryGroup), sodass diese zu einem späteren Zeitpunkt bereits befüllt sind.

Sobald das Helper-Objekt vollständig aufgebaut ist, wird dieses in eine Liste hinzugefügt. Falls während des Prozesses ein Fehler auftritt, wird eine entsprechende Nachricht im Debug-Protokoll ausgegeben und eine Exception geworfen. Dies ermöglicht eine detaillierte Nachverfolgung und Fehlerbehebung.

Mapping CC-Order zu Order

Nun wird das eigentliche Datenmapping zu dem neuen Order-Objekt behandelt. Wie im Ablaufdiagramm (Anhang 3) dargestellt, beginnt der Prozess mit einer Liste von *OrderConversionHelper* als Parameter. Es folgt die Iteration über diese Liste, um die relevanten Daten für die Konvertierung zu extrahieren.

Für jeden *OrderConversionHelper* in der Liste werden die *oldOrder*, *newOrder*, *oldOrderItems* und *newOrderItems* mittels entsprechender Getter-Methoden in Variablen gespeichert. Diese tragen zur Übersichtlichkeit des Codes bei. Der nächste Schritt ist das Mapping der Order-Felder. Dies geschieht durch die Iteration über die Schlüssel der zuvor erwähnten Map *orderFieldMapping*. Felder, deren Namen einen Punkt enthalten, werden vorerst übersprungen, da es sich hierbei um ein Relationales-Feld handelt. Also werden zuerst alle regulären Felder bearbeitet. Mittels *SObject.Get(Fieldname)* wird der aktuelle Wert eines Feldes in einer Objekt-Variable gespeichert. Ist dieser Wert nicht befüllt, also null, wird das Feld übersprungen und im Log gespeichert. Ist ein Wert vorhanden, wird das Feld mit *SObject.Put(FieldName, ObjectValue)* befüllt. Bei Feldern wie ccrz__E_*orderDate__c*, die im alten System nur einmal existieren, aber im neuen System zwei Felder füllen müssen (hier *OrderedDate* und *OrderDate*), erfolgt eine spezielle Verarbeitung. Wenn dieses Feld angesprochen wird, werden beide entsprechenden Felder im neuen System mit demselben Wert befüllt. Dies ist nötig, da mittels der Map auch die SOQL-Abfrage ausgeführt wird und hierbei keine Felder doppelt vorhanden sein dürfen.

Die neue Order benötigt zusätzlich eine *SalesStoreId*, die aus einer Konstante abgeleitet wird, welche zuvor über eine Query abgefragt wird, da sie immer gleich ist und somit direkt eingefügt werden kann. Die Felder *Status* und *OrderStatus* werden auf "Draft" gesetzt. Im alten System gibt es nur ein Statusfeld, während im neuen zwei erforderlich sind. Da keines der beiden neuen Felder die alten Werte direkt übernehmen kann, wurde entschieden, beide auf "Draft" zu setzen. Falls nun ein relationales Feld auftaucht, wird geprüft, ob es sich um *BillTo* oder *ShipTo* handelt, da diese Informationen im *ContactConversion*-Objekt gespeichert sind. Je nachdem, ob *BillTo* oder *ShipTo* angesprochen werden, werden unterschiedliche Felder in der neuen Order befüllt.

Abschließend wird das OrderItem-Mapping durchgeführt. Für jedes *oldOrderItem* in der Liste wird ein neues *OrderItem* zur Liste der *newOrderItems* hinzugefügt. Dies ermöglicht eine parallele Iteration über beide Listen, ohne Laufzeitfehler zu verursachen. Die Zuordnung der Werte erfolgt ähnlich wie zuvor über Get- und Put-Methoden. Auch hier wird nach der *SObject.get(fieldname)* Methode eine Überprüfung auf den Wert durchgeführt, um sicherzustellen, dass dieser nicht null ist. Bei zwei benötigten Feldern (Quantity und Price) müssen jeweils Werte vorhanden sein. Daher wird, falls das Feld null ist, der Wert manuell mit 1 befüllt. Danach erfolgt der gleiche Ablauf wie zuvor. Da diese Methode größer als die anderen ist, kann der gesamte Code im Anhang 4 betrachtet werden.

Nun müssen die Objekte in Salesforce in richtiger Reinfolge angelegt werden (Siehe nächste Seite). Dabei wird zuerst die *Order* eingefügt, da die Id für das Einfügen der *OrderDeliveryGroup* benötigt wird. Inserts in die Salesforce Datenbank nicht in Schleifen erfolgen sollte, wird jede neue Order und *OrderDeliveryGroup* aus dem *OrderConversionHelper* einer Liste hinzugefügt. Dies entlastet das System und verhindert das die Datenbank-operation-limits nicht erreicht werden.

```
Product2 dummyProduct = [SELECT Id FROM Product2 WHERE Name = 'MigrationDummyProduct'
LIMIT 1];

for (OrderConversionHelper helper : orderConversionHelpers)
{
    Order order = helper.getNewOrderObj().getOrderInfos();
    newOrders.add(order);
    newOrderDeliveryGroups.add(helper.getNewOrderObj().getOrderDeliveryGroup());
}

Database.SaveResult[] saveResultsNewOrders = Database.insert(newOrders, false);

// Map OrderId to OrderDeliveryGroups
for (Integer i = 0; i < newOrders.size(); i++)
{
    newOrderDeliveryGroups[i].OrderId = newOrders[i].Id;
}

Database.SaveResult[] saveResultsDeliveryGroups =
Database.insert(newOrderDeliveryGroups, false);

// Map OrderDeliveryGroupId and OrderId to OrderItems and insert them
for (Integer i = 0; i < orderConversionHelpers.size(); i++)
{
    Order newOrder = newOrders[i];
    OrderDeliveryGroup orderDeliveryGroup = newOrderDeliveryGroups[i];
    List<OrderItem> orderItems =
        orderConversionHelpers[i].getNewOrderObj().getOrderItems();

    for (OrderItem item : orderItems) {
        item.OrderId = newOrder.Id;
        item.OrderDeliveryGroupId = orderDeliveryGroup.Id;
        item.Product2Id = dummyProduct.Id;
        item.ListPrice = item.UnitPrice;

    }

    // Insert OrderItems
    Database.SaveResult[] saveResultsNewOrderItems = Database.insert(orderItems,
         false);
    System.debug(saveResultsNewOrderItems);

}

System.debug(saveResultsNewOrders);
System.debug(saveResultsDeliveryGroups);

}
catch (Exception e)
{
    System.debug(e.getMessage());
}
```

Danach können diese alle kompakt eingefügt werden. Ist dies geschehen kann über eine erneute Schleife die jeweilige *OrderId* der zugehörigen *OrderDeliveryGroup* eingefügt und danach in die Datenbank integriert werden. Als letztes müssen die *OrderItems* behandelt werden. Erst wird die *OrderId* und die *OrderDeliveryGroupId* in einem Objekt gespeichert, da diese für den Insert der *OrderItems* benötigt werden. Es wird über jedes *OrderItem* der *NewOrder* iteriert und die beiden Ids gesetzt, sowie einige Felder, für die eine gesonderte Anforderung benötigt wird, angepasst. Zu diesen Feldern gehört zum Beispiel der *UnitPrice* dieser existierte zwar im alten System, aber im neuen kommt ein *ListPrice* hinzu. Hier wurde festgelegt, dass dieser denselben Wert die der *UnitPrice* bekommt. Auch gehört zu jedem *OrderItem* ein *Product*, also das, was eigentlich gekauft wurde. Es wurde festgelegt das die Produkte besondere Eigenschaften aufweisen sollten, welche aber durch den Kunden noch nicht aufgezeigt wurden. Daher wird für das Konzept ein Platzhalter eingefügt, welcher das bestellte Produkt repräsentiert. Dieses kann später, wenn die Anforderungen geklärt sind, aktualisiert werden. Daher ist dieser Prozess auch nicht Teil der Arbeit. Außerdem werden alle Datenbankoperationen werden in einem *Database.SaveResult* Objekt gespeichert. Dadurch können die Objekte nachverfolgt werden, bei denen ein Fehler aufgetreten ist.

Nun muss noch sichergestellt werden, dass die Prozesse hinter dem Anlegen einer Order ausgeführt werden. Wie bereits in Kapitel 3.3 erwähnt, passiert dies im neuen Lightning System durch sogenannte Trigger am OrderItem und Formular-Felder an der Order und dem OrderItem. Durch die bereits Implementierte Lösung kann hier eine Methode Namens *ArlClsTriggerControll.setTriggerDisabled(OrderItem.SObjectType)* aufgerufen werden. Dies verhindert, dass die Logik im Trigger auf die OrderItems ausgeführt wird.

Damit ist das Mapping zum neuen Order-Objekt abgeschlossen. Nun muss noch die Asynchronität behandelt werden. Dafür wurde eine neue Klasse angelegt, welche das *Database.Batchable* Interfaces implementiert. Über die drei vorgegebenen Methoden Start, Finish und Execute können nun die einzelnen Daten und Schritte bearbeitet werden. In der Start-Methode werden die einzelnen Datensätze abgefragt (Also hierbei die Methode selectAllObjects und danach das Mapping zum OrderConvertionHelper). Das Ergebnis kann dann in die Execute-Methode übergeben werden und dort die Funktionen *mapToNewOrderObject* und anschließend *insertNewOrders* aufgerufen werden. Diese Methode verarbeitet die Daten in sogenannten Batches und stellt die Asynchronität sicher. Damit ist das Konzept inklusive des Skriptes für die Datenmigration abgeschlossen.

Nun muss durch einen Integration-Test die Datenintegrität und Funktionalität der geschriebenen Methoden sichergestellt werden. Dies wird im anschließenden Kapitel behandelt.

5.2 Testen

Nun sollen die bisher geschriebenen Methoden für die OrderMigration in einem Integration Test getestet werden. Da dies ein zusammengeführter Prozess ist, welcher nur mit den gemeinsam gekoppelten Funktionen funktioniert, sollte daher sichergestellt das die einzelnen Methoden gekoppelt funktionieren. Der Vorteil von Tests in Apex liegt in ihrer Fähigkeit, die Funktionalität von Codeblöcken zu testen, ohne abhängig von externen Systemen zu sein. Daher wurde eine Methode *TestSetup()* erstellt, welche vor jedem Test der verschiedenen Methoden Testdaten in die Salesforce-Umgebung einfügt. Dies sollte gemacht werden, da standardmäßig die Tests in einer eigenen Umgebung ohne die Daten der jeweiligen Organisation durchgeführt werden. Die eingefügten Daten sind realen Daten nachempfunden. Für das Testen reicht es aus eine *CC-Order* mit Ihren relationalen Objekten *CC-ContactAddress*, *CC-OrderItem*, *Account* und *Contact* erstellt wird. Der Integration Test prüft zuerst anhand einer SOQL-Abfrage, ähnlich der zuvor implementierten *selectAllObjetcs()-Methode*, ob es bereits Order-Objekte und deren dazugehörige relationale Objekte existieren. Es sollten 2 alte Order-Objekte mit den dazugehörigen OrderItems und ContactAddresses in der Datenbank, aber keine neue Order sein. Danach werden die OrderMigration-Methoden gekoppelt aufgerufen. Anschließend kann nun über eine Schleife über das *orderFieldMapping* iterieren und mittels *System.AssertEquals(<expected>, <acutal>)* vergleichen, ob der Wert in der alten Order, bzw. relationalen Objekte, dem in der neuen Order entspricht (Siehe nächste Seite). Durch die Überprüfung jedes Feldes wird sichergestellt, dass alle benötigten Daten übernommen werden. Anschließend werden die OrderItems ähnlich geprüft. Ist der Test erfolgreich, es also keine Fehler gibt, ist somit sichergestellt das alle Werte erfolgreich übernommen wurden.

```
for(String fieldName : orderFieldMapping.keySet())
{
    Object oldFieldValue;
    Object newFieldValue;

    if (fieldName.startsWith('ccrz__ShipTo__r.'))
    {
        oldFieldValue = oldOrders.get(0).ccrz__ShipTo__r.get(fieldName.substring(16));
        newFieldValue =
       newOrders.get(0).OrderDeliveryGroups.get(0).get(fieldName.substring(16));
    }
    else
    {
        if(fieldName.startsWith('ccrz__BillTo__r.'))
        {
            oldFieldValue =
           oldOrders.get(0).ccrz__BillTo__r.get(fieldName.substring(16));
            newFieldValue = newOrders.get(0).get(orderFieldMapping.get(fieldName));
        } else
        {
            oldFieldValue = oldOrders.get(0).get(fieldName);
            newFieldValue =  newOrders.get(0).get(orderFieldMapping.get(fieldName));
        }

    }
    System.assertEquals(oldFieldValue,newFieldValue);

}
```

5.3 Auswertung

Ziel des Konzeptes war einerseits das alle relevanten und genannten Felder und relationalen Objekte (CC-Order, CC-OrderItem und CC-ContactAddress) in das neue Order-Objekt übernommen werden. Außerdem soll während der Migration sichergestellt werden, dass Datenfehler, Duplikate oder Inkonsistenzen nicht die Integrität der Daten gefahren. Auf Grundlage der vorher definierten Definition of Done kann festgestellt werden, ob diese Ziele erreicht wurden (Siehe Tabelle 3). Das erste Kriterium sah vor, dass die geforderten 3 Objekte übernommen wurden. Anhand des vorher geschriebenen Skriptes sind diese 3 Objekte mit Implementiert, somit ist das erste Kriterium erfüllt. Über das definierte OrderFieldMapping wurden alle Felder, welcher in der VaretyEntry definiert waren integriert. Durch die Integration und ausführen des Tests wurde sichergestellt, dass alle befüllten Felder übernommen werden. Dies wird anhand der System.AsserEquals()-Methode überprüft. Da hierbei kein Fehler auftritt, kann festgelegt werden, dass 100 Prozent der Felder übernommen wurden.

Damit ist auch das zweite Kriterium und dritte erfüllt. Letztlich wurde das Skript samt Ablaufdiagramm auf der projektinternen Confluence Seite dokumentiert und beschrieben.

Nun ist auch das letzte Kriterium erfüllt. Damit können auch beide zuvor definierten Ziele als erfüllt betrachtet werden.

Titel	Kriterium
Übernahme aller relevanten relationalen Objekte	3 relationale Objekte wurden übernommen (3)
Übernahme aller relevanten Daten	100 % der Daten wurden übernommen (>75%)
Mindestens einen Test geschrieben	1 Test wurden geschrieben (>0)
Dokumentation auf Confluence	× Die Seite ist öffentlich zugänglich × Alle Schritte des Ablaufdiagramms sind beschrieben und dargestellt

Tabelle 3: Vergleich der Definition of Done

5.4 Herausforderungen

Welche Herausforderungen während der Datenmigration aufgetreten sind soll in diesem Kapitel betrachtet werden. Ein wichtiges Problem war das Verständnis und die Entkopplung der Prozesse, welche hinter der Erstellung von Orders liegen. Darin lag die Herausforderung bei der Identifizierung der Prozesse, welche in B2B-Lightning beteiligt sind. Basierend auf der Dokumentation des B2B-Lightning-Shops wurden überprüft, wie diese Prozesse umgangen werden konnten. Darin wurde beschrieben, wie die spezifischen OrderItem-Trigger deaktiviert werden. Eine Entkopplung der Blau maskierten Felder ist aufgrund der Komplexität nicht geschehen. Da die Felder mittels einer Formel befüllt werden, können diese nicht händig befüllt werden. Die Recherche wie man dieses Problem lösen kann gab leider keine Ergebnisse. Daher wird dieses Problem an einer anderen Stelle außerhalb dieser Arbeit noch einmal betrachtet. Auch erschwerte das Fehlen einer detaillierten Beschreibung des alten Systems die Identifizierung relevanter Daten, welche für die Übernahme wichtig waren. Diese Herausforderung wurde durch Kommunikation mit Mitarbeitern, welche das alte System aufgestellt hatten, bewältigt. Hierbei wurde festgelegt und erklärt, welche Felder den jeweiligen Feldern aus dem B2B-Commerce Systems entsprechen. Dadurch konnte eine genaue Zuordnung erreicht werden. Auch stellten die Unentschlossenheit und Unklarheit des Kunden hinsichtlich seiner Anforderungen eine zusätzliche Herausforderung dar. Es ist bis zu Ende dieser Arbeit nicht klar, ob einige Daten wie die VaretyOrder oder anderer relationaler Objekte übernommen werden.

6 Zusammenfassung und Fazit

Diese Arbeit beschäftigte sich mit der Datenmigration des Order-Objektes des alten B2B-Commerce (Classic) zum neuen B2B-Lightning Systems von Salesforce für einen Online-Shop für Saatgut. Im Rahmen dieser Arbeit wurde zunächst das vorhandene Datenmodell in Salesforce B2B-Commerce geprüft, um spezifische Anforderungen für das neue System zu nennen und festzulegen. Basierend auf dieser Analyse wurde eine globale Definition of Done und Ziele festgelegt, welche das danach erstelle, Konzept messbar machen. Dieses Konzept wurde danach in praktischen Schritten umgesetzt und durch Tests validiert. Die erfolgreiche Umsetzung dieses Konzeptes führte zur Übernahme aller vorher benannten, relevanten relationalen Objekte und Daten in das neue System. Zwar konnten einige wenige Felder im neuen System nicht von laufenden Prozessen entkoppelt werden, aber über die durchgeführten Tests kann belegt werden, dass 100% der angelegten Daten erfolgreich migriert werden können. Zudem wurde eine umfassende Dokumentation auf Confluence, inklusive eines Ablaufdiagramms erstellt, die den Migrationsprozess detailliert beschreibt und für den Kunden zugänglich macht.

Die Datenmigration von B2B-Commerce zu B2B-Lightning demonstriert, wie wichtig eine sorgfältige Planung und Durchführung von Datenmigrationsprojekten ist. Trotz der Herausforderungen, wie der mangelnden Klarheit in den Kundenanforderungen und der unzureichenden Beschreibung des alten Systems, wurde erfolgreich eine Grundlage für die Datenmigration geschaffen. Diese kann nun nach Bedarf des Kunden angepasst und später durchgeführt werden. Auch unterstreichen die Ergebnisse die Bedeutung eines systematischen Ansatzes bei der Datenmigration, um sowohl die Integrität der Daten zu bewahren als auch die Nutzung neuer Systemfunktionen zu ermöglichen. Die Umstellung auf B2B-Lightning bedeutet nicht nur, dass der Kunde von den verbesserten Funktionen von Salesforce Lightning profitieren kann, sondern auch eine robuste Plattform für zukünftiges Wachstum und Anpassungsfähigkeit in einem sich ständig wandelnden digitalen Handelsmarkt zur Verfügung hat. Abschließend lässt sich sagen, dass dieses Projekt nicht nur für den spezifischen Fall des Saatgut-Shops von Bedeutung ist, sondern auch wertvolle Erkenntnisse für ähnliche Migrationsprojekte in anderen Commerce-Projekten liefert. Das entwickelte Skript, die Dokumentation in Confluence und Lösungsansätze können als Leitfaden für zukünftige Datenmigrationsprojekte dienen und zur Steigerung der Effizienz und Effektivität in solchen Projekten beitragen.

Literaturverzeichnis

1. **[DSL23] dotSource SE.** B2B Lightning Shop. [Online] [Cited: November 2023, 30.]

2. [redacted]

3. [redacted]

4. [redacted]

5. **[DIP22], Jyoti, Dipanker und Hutcherson, James A.** *Handbuch für Salesforce-Architekten: Ein umfassender Leitfaden für End-to-End-Lösungen.* s.l. : Springer, 2022. 978-3-662-66534-3.

6. **[SFO23] Salesforce Inc.** [Online] [Cited: November 2023, 30.] https://developer.salesforce.com/.

7. **[SFH23] Salesforce Inc.** [Online] [Cited: November 29, 2023.] https://help.salesforce.com/.

8. **[SDL23] Salesforce Inc.** Dataloader. [Online] [Cited: Dezember 5, 2023.] https://dataloader.io/documentation.

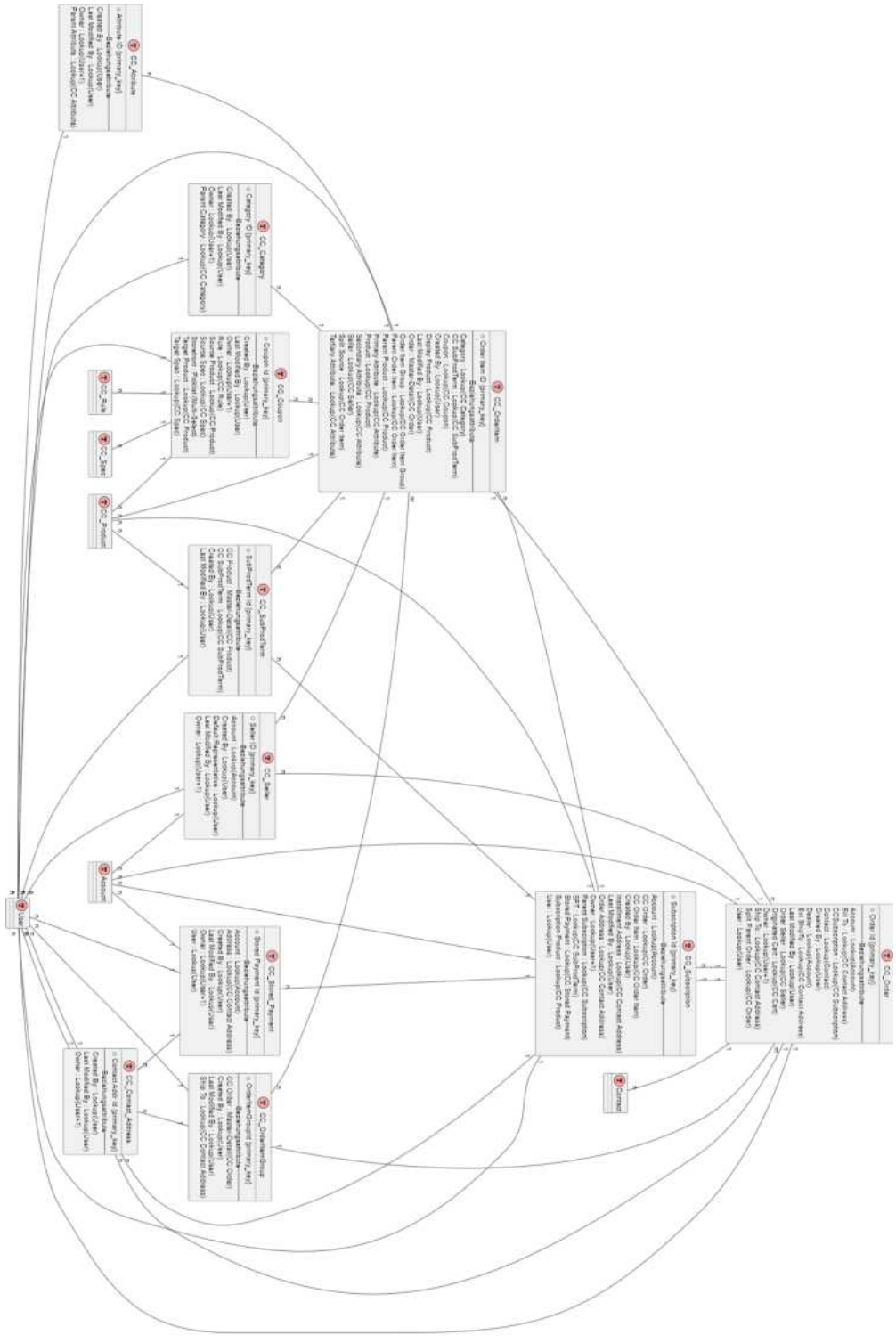
9. **[SAL24] Salesfroce Inc.** [Online] 08. Januar 2024. https://trailhead.salesforce.com/de/content/learn/modules/lex_migration_rollout.

10. **[BAR18] Barry Levine.** MarTech. [Online] 02. Mai 2018. [Zitat vom: 29. Januar 2024.] https://martech.org/how-salesforces-acquisition-of-cloudcraze-expands-its-commerce-cloud/.

11. **[TLK20] Anna-Luisa Becke.** [Online] 26. November 2020. [Zitat vom: 29. Januar 2024.] https://digital-commerce.telekom-mms.com/blog/unterschiede-salesforce-classic-vs-lightning.html.

Anhang

1. Struktur CC-Order-Objekt



CC Order	
Feldname	**Datentyp**
Adjusted Order Number	Formula (Text)
Adjustment Amount	Currency(16, 2)
Adjustment Reason	Text(255)
Advisor	Text(255)
ARiMR	Text(9)
Bank Account Number	Text(36)
Bank Correspondence Account Number	Text(36)
Bank Name	Text(255)
BIC	Text(11)
Cart Restored	Checkbox
CCEmailAddress	Long Text Area(32768)
Company Name	Text(80)
Contract ID	Text(50)
Currency ISO Code	Text(10)
Data Sharing Selection	Picklist (Multi-Select)
Dealer Name	Text(255)
Early Shipping	Checkbox
Effective Account ID	Text(20)
Email	Email
Encrypted Id	Text(250) (Unique Case Insensitive)
Energg Beet	Checkbox
Ext Carrier	Text(50)
Ext ShipTrack No	Text(250)
Feedbeet	Checkbox
First Name	Text(40)
Grand Total Amount	Formula (Currency)
Has Conviso Product	Checkbox
Hide Prices	Checkbox
INN	Number(10, 0)
Is Anonymous	Checkbox
Is Loyalty Member	Checkbox
KPP	Number(9, 0)
Last Name	Text(80)
Mobile Phone	Phone
Name	Auto Number
Note	Long Text Area(1000)
Number of All Order Items	Roll-Up Summary (COUNT CC Order Item)
Number of Major Order Items	Roll-Up Summary (COUNT CC Order Item)
OGRN	Number(13, 0)
Order Date	Date
Order Name	Text(255)
Order Name (Duplicated)	Formula (Text)
Order Number	Number(18, 0) (Unique)
Order Number (External ID)	Auto Number
Order Number Link	Formula (Text)
Order Source	Text(255)
Order Status	Picklist
Patronymic Name	Text(255)
PaymentMethod	Text(50)
Phone	Phone
Points Total	Number(18, 0)
PO Number	Text(30)
Redeemed Points	Number(18, 0)
Redemption Value	Currency(16, 2)
REGON	Text(9)
Request Date	Date
Resend MC Order Confirmation Mail	Checkbox
SDI Code	Text(7)
Ship Amount	Currency(16, 2)
Ship Complete	Checkbox
Ship Discount Amount	Currency(16, 2)
Ship Method	Text(50)
Shipping Method	Picklist
Ship Structure	Text(128)
Skip Tax Calculation	Checkbox
Split Processed	Checkbox
Storefront	Text(255)
Subtotal Amount	Roll-Up Summary (SUM CC Order Item)
Sugar Beet	Checkbox
Sugar Factory	Text(255)
Sugarfactory Conviso	Text(255)
Tax Amount	Currency(16, 2)
Tax Exempt Flag	Checkbox
Tax Number	Text(255)
Tax SubTotal Amount	Roll-Up Summary (SUM CC Order Item)
Total Amount	Formula (Currency)
Total Discount	Currency(16, 2)
Total Surcharge	Currency(16, 2)
VAT Percentage	Number(3, 2)
VerifyExemptionFlag	Checkbox

CC Order Item	
Feldname	**Datentyp**
Absolute Discount	Currency(16, 2)
Adjustment Amount	Currency(16, 2)
Availability Message	Text Area(255)
Branch	Picklist
Category Key	Formula (Text)
Category Key	Text(40) (External ID)
Comments	Text Area(255)
Crop Type	Formula (Text)
Discount In Percent	Percent(3, 2)
External Name	Text(255)
External SKU	Text(255)
Field Type	Picklist
Is Conviso Product	Checkbox
Is Package	Checkbox
Is Subscription	Checkbox
Item Label	Text(80)
Item Status	Picklist
Item Total	Formula (Currency)

CC Order Item Group	
Group Name	Text(255)
Name	Auto Number
Note	Long Text Area(1000)
Request Date	Date
Ship Amount	Currency(16, 2)
Ship Method	Text(50)

CC Attribute	
Attribute ID	Text(255) (External ID)
CC Attribute Name	Text(80)
Display Name	Rich Text Area(1024)
Effective End	Date/Time
Effective Start	Date/Time
Sequence	Number(8, 0)
SKU Fragment	Text(255)

CC Coupon	
Cart Total Amount	Currency(16, 2)
Coupon Auto Number	Auto Number
Coupon Code	Text(20) (Unique Case Insensitive)
Coupon Name	Text(128)
Coupon Notes	Text Area(255)
Coupon Type	Picklist
Currency ISO Code	Text(10)
Discount Amount	Currency(16, 2)
Discount Type	Text(30)
Enabled	Checkbox
End Date	Date
Max Use	Number(18, 0)
Minimum Quantity	Number(18, 0)
Rule Name	Formula (Text)
Rule Type	Picklist
Source Minimum Quantity	Currency(16, 2)
Source Relation	Picklist
Source Spec Minimum Quantity	Number(18, 0)
Source Spec Value	Text(100)
Start Date	Date
Target Quantity	Number(18, 0)
Target Relation	Picklist
Target Spec Quantity	Number(18, 0)
Target Spec Value	Text(100)
Total Used	Number(18, 0)

CC Category	
Category ID	Text(128) (External ID) (Unique Case Insensitive)
Category Name	Text(80)
Child Category Count	Number(4, 0)
DataId	Text(255) (External ID) (Unique Case Insensitive)
End Date	Date
Long Desc	Long Text Area(1000)
LongDescRT	Rich Text Area(5000)
Owner	Lookup(User+1)
Product Count	Roll-Up Summary (COUNT CC Product Category)
SEO Id	Text(255) (External ID)
Sequence	Number(8, 0)
Short Desc	Text(255)
ShortDescRT	Rich Text Area(2000)
Start Date	Date

CC Seller	
Feldname	**Datentyp**
Effective End	Date/Time
Effective Start	Date/Time
Enabled	Checkbox
Location	Geolocation
Logo	URL(255)
Long Description	Rich Text Area(32768)
Search Tokens	Long Text Area(32768)
Seller Name	Text(80)
Short Description	Rich Text Area(32768)
Storefront	Text(255)
WebsiteURL	URL(255)

CC Stored Payment	
Account Accessible	Checkbox
Account Number	Text(255)
Account Type	Text(255)
CC Stored Payment Name	Text(80)
Default	Checkbox
Display Name	Text(255)
Effective Account ID	Text(20)
Enabled	Checkbox
End Date	Date
Exp Month	Number(2, 0)
Exp Year	Number(4, 0)
Name	Auto Number
Payment Type	Text(255)
Read Only	Checkbox
Sequence	Number(18, 0)
Start Date	Date
Stored Payment Id	Text(255) (External ID) (Unique Case Insensitive)
Storefront	Text(255)
Subaccount Number	Text(255)
Token	Text(255)

CC Contact Address	
Address Firstline	Text(255)
Address Secondline	Text(255)
Address Thirdline	Text(255)
AddrReadOnly	Checkbox
City	Text(255)
Company Name	Text(100)
Contact Addr Id	Text(128) (External ID) (Unique Case Insensitive)
Contact Auto Number	Auto Number
Country	Text(50)
Country ISO Code	Text(10)
Created By	Lookup(User)
Daytime Phone	Phone
Email	Email
First Name	Text(100)
Home Phone	Phone
Last Modified By	Lookup(User)
Last Name	Text(100)
Mail Stop	Text(255)
Middle Name	Text(100)
Owner	Lookup(User+1)
Partner_Id	Text(100)
Postal Code	Text(10)
Shipping Comments	Long Text Area(32000)
State	Text(255)
State ISO Code	Text(10)

CC Subscription	
Feldname	**Datentyp**
Auto Renew	Checkbox
CC Subscription Name	Auto Number
Comments	Long Text Area(32768)
Currency ISO Code	Text(10)
Effective Account ID	Text(20)
Ignore Installment Frequency Fields	Checkbox
Installment Count Remaining	Number(10, 0)
Installment Last Date	Date/Time
Installment Next Date	Date/Time
Installment Status	Text(255)
Last Recurring Failure	Long Text Area(32768)
Order Count Remaining	Number(10, 0)
Order Last Date	Date/Time
Order Next Date	Date/Time
Order Status	Text(255)
Quantity	Number(18, 0)
Recurring Price	Number(16, 2)
Recurring Price Sub Amount	Currency(16, 2)
Sequence	Number(8, 0)
SPT Installment Count	Number(10, 0)
SPT Installment Frequency	Number(10, 2)
SPT Installment Frequency UOM	Text(255)
SPT Order Count	Number(10, 0)
SPT Order Frequency	Number(10, 2)
SPT Order Frequency UOM	Text(255)
SPT Recurring Price Percentage	Number(16, 2)
SPT Upfront Price Percentage	Number(16, 2)
Storefront	Text(255)
Subscription Id	Text(255) (External ID) (Unique Case Insensitive)
Subscription Start Date	Date/Time
Subscription Status	Text(255)

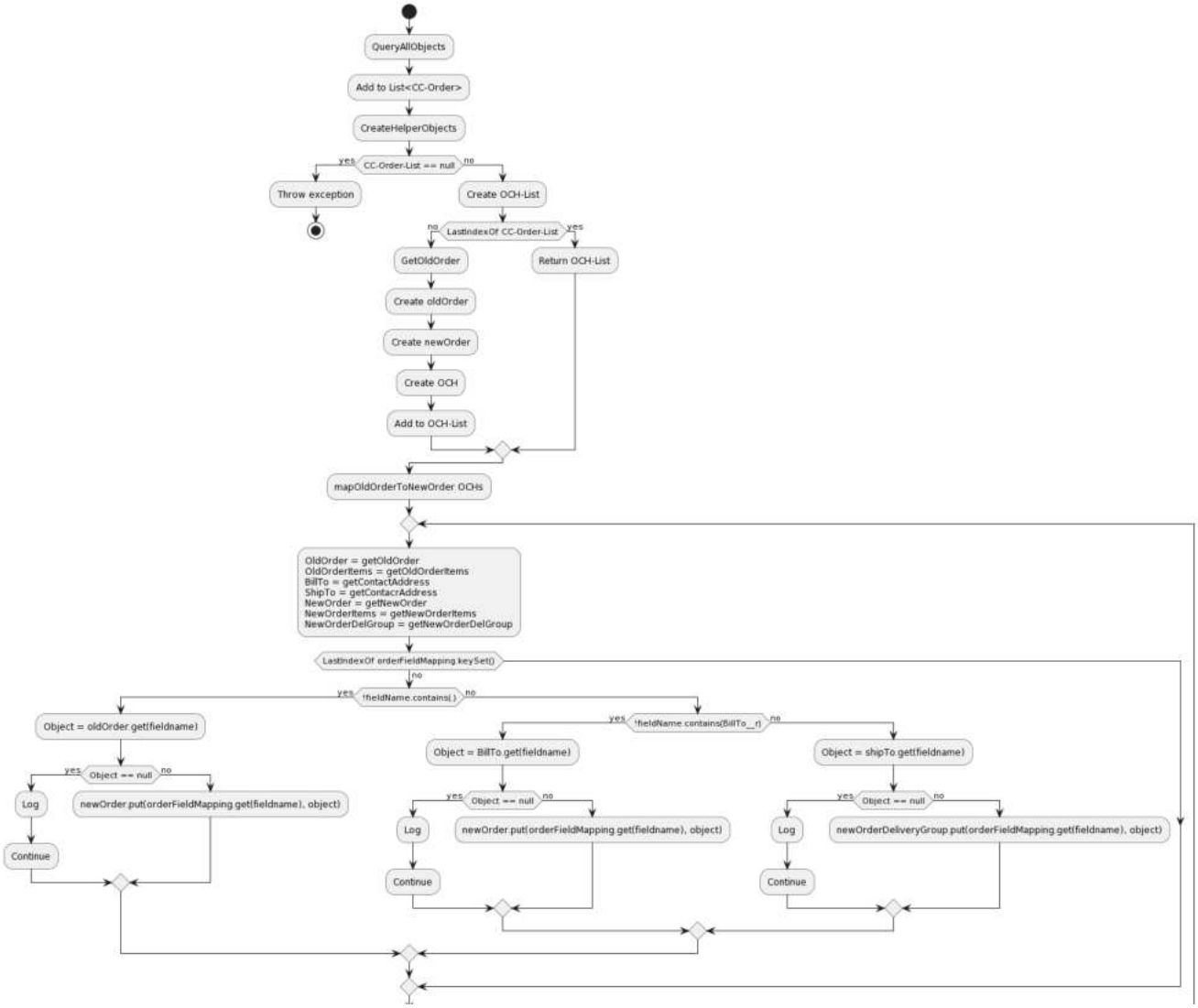
CC SubProdTerm	
Auto Renew	Checkbox
Cart Display Name	Text(255)
Cart Display Name RT	Rich Text Area(5000)
CC SubProdTerm Name	Text(80)
Display Name	Text(255)
Display Name RT	Rich Text Area(256)
Effective End	Date/Time
Effective Start	Date/Time
Email Display Name	Text(255)
Email Display Name RT	Rich Text Area(5000)
Enabled	Checkbox
Ignore Installment Frequency Fields	Checkbox
Installment Count	Number(10, 0)
Installment Frequency	Number(10, 2)
Installment Frequency UOM	Text(255)
Name	Auto Number
Order Count	Number(10, 0)
Order Display Name	Text(255)
Order Display Name RT	Rich Text Area(5000)
Order Frequency	Number(10, 2)
Order Frequency UOM	Text(255)
Product Detail Display Name	Text(255)
Product Detail Display Name RT	Rich Text Area(5000)
Product List Display Name	Text(255)
Product List Display Name RT	Rich Text Area(5000)
Recurring Price Percentage	Number(14, 4)
Sequence	Number(8, 0)
Storefront	Text(255)
SubProdTerm Id	Text(255) (External ID) (Unique Case Insensitive)
Upfront Price Percentage	Number(14, 4)

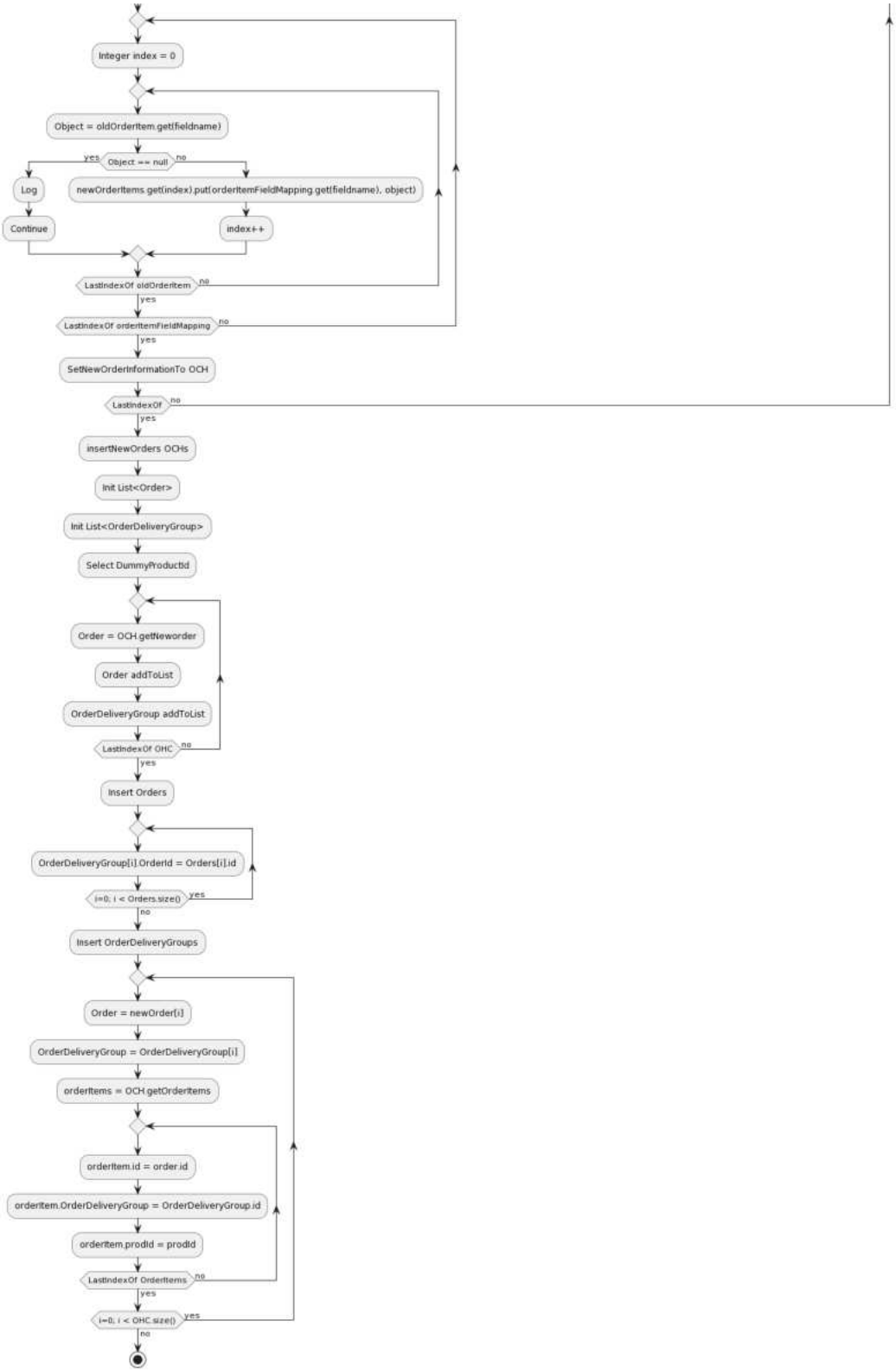
2. Zu Übernehmende Felder

Order	CC-Order		
CategoryKey__c	categoryKey	Order_PSP_Token__c	paymentDetails.size() == 1 ? paymentDetails[0].ccrz__Token__c : ''
SalesOrganization__c	keys [0]	Order_Mobile__c	order.ccrz__BuyerMobilePhone__c
DistributionChannel__c	keys [1]	Order_ID__c	order.DOT_Adjusted_Order_Number__c
Branch__c	keys [2]	Usage__c	DotVarietyEntryServiceHelper.getUsagePicklist(order)
Farmer_Customer__c		Early_Shipping__c	order.DOT_Early_Shipping__c
Order_Email__c	order.ccrz__BuyerEmail__c	Tax_Number__c	order.DOT_Tax_Number__c
Order_First_Name__c	order.ccrz__BuyerFirstName__c	Advisor__c	order.BTS_Advisor__c
Order_Last_Name__c	order.ccrz__BuyerLastName__c	ARIMR__c	order.DOT_ARIMR__c
Order_Phone__c	order.ccrz__BuyerPhone__c	REGON__c	order.DOT_REGON__c
Contact__c	order.ccrz__Contact__c	Sugar_Factory__c	order.DOT_Sugar_Factory__c
Notes__c	order.ccrz__Note__c	Shipping_Method__c	order.DOT_Shipping_Method__c
Order_Date__c	order.ccrz__OrderDate__c	INN__c	order.DOT_INN__c
Order_Storefront__c	order.ccrz__Storefront__c	KPP__c	order.DOT_KPP__c
Order_User__c	order.ccrz__User__c	OGRN__c	order.DOT_OGRN__c
RecordTypeId	recordTypeId	BIC__c	order.DOT_BIC__c
Season__c	order.ccrz__OrderDate__c	Bank_Name__c	order.DOT_Bank_Name__c
CC_Order__c	order.id	Bank_Account_Number__c	order.DOT_Bank_Account_Number__c
Billing_City__c	order.ccrz__BillTo_r.ccrz__City__c	Bank_Correspondence_Account_Number__c	order.DOT_Bank_Correspondence_Account_Number__c
Billing_Country__c	order.ccrz__BillTo_r.ccrz__CountryISOCode__c	Order_Patronymic_Name__c	order.DOT_BuyerPatronymicName__c
Billing_Postal_Code__c	order.ccrz__BillTo_r.ccrz__PostalCode__c	Adjustment_Amount__c	order.ccrz__AdjustmentAmount__c
Billing_State__c	order.ccrz__BillTo_r.ccrz__StateISOCode__c	Subtotal_Amount__c	order.ccrz__SubtotalAmount__c
Billing_Street__c	billingStreet	Tax_Amount__c	order.ccrz__TaxAmount__c
Invoice_Email__c	order.ccrz__BillTo_r.ccrz__Email__c	TotalAmount	order.DOT_Grand_Total_Amount__c
Billing_Company__c	order.ccrz__BillTo_r.ccrz__CompanyName__c	Total_Points__c	DotVarietyEntryServiceHelper.getVarietyEntryPoints(order, countryIsoCode)
Billing_First_Name__c	order.ccrz__BillTo_r.ccrz__FirstName__c	Transaction_ID_CC__c	paymentDetails.size() == 1 ? paymentDetails[0].ccrz__TransactionCode__c : ''
Billing_Last_Name__c	order.ccrz__BillTo_r.ccrz__LastName__c	Loyalty_Redeemed_Points__c	order.DOT_Redeemed_Points__c
Shipping_City__c	order.ccrz__ShipTo_r.ccrz__City__c	Loyalty_Redeemed_Value__c	order.DOT_Redemption_Value__c
Shipping_Country__c	order.ccrz__ShipTo_r.ccrz__CountryISOCode__c	Loyalty_Voucher_Discount__c	couponOrderItem != null
Shipping_Postal_Code__c	order.ccrz__ShipTo_r.ccrz__PostalCode__c	Loyalty_Voucher_Code__c	couponOrderItem != null__c
Shipping_Street__c	shippingStreet	Partner_Customer__c	order.DOT_Dealer__c
Shipping_Company__c	order.ccrz__ShipTo_r.ccrz__CompanyName__c	Partner_Customer_Name__c	order.DOT_Dealer_Name__c
Shipping_First_Name__c	order.ccrz__ShipTo_r.ccrz__FirstName__c	Dot_Sugarfactory_Conviso__c	order.DOT_Sugarfactory_Conviso__c
Shipping_Last_Name__c	order.ccrz__ShipTo_r.ccrz__LastName__c	DOT_SDI_Code__c	order.DOT_SDI_Code__c
Daytime_Phone__c	order.ccrz__ShipTo_r.ccrz__DaytimePhone__c		
Shipping_State__c	order.ccrz__ShipTo_r.ccrz__StateISOCode__c		
Payment_Type__c	order.ccrz__PaymentMethod__c		

OrderItem	CC-OrderItem
Order_Item_ID__c	orderItem.ccrz__OrderItemId__c
PriceSetByApp__c	true
TotalLineAmount	orderItem.ccrz__SubAmount__c
TotalPrice	orderItem.ccrz__ItemTotal__c
Material__r	new Material__c(CategoryKey__c = categoryKeyOfMaterial)
RecordTypeId	recordTypeId
CC_Order_Item__c	orderItem.Id
DOT_Is_Conviso_Product__c	orderItem.ccrz__Product__r.DOT_Is_Conviso__c
Parent_Item_Order__c	getParentItemOrderSequenceNumber(orderItem)
UnitPrice	orderItem.ccrz__Price__c
Quantity	orderItem.ccrz__Quantity__c
TotalAdjustmentAmount	orderItem.ccrz__AdjustmentAmount__c

3. Ablaufdiagramm


QueryAllObjects
Add to List<CC-Order>
CreateHelperObjects
CC-Order-List == null
yes
no
Throw exception
Create OCH-List
LastIndexOf CC-Order-List
no
yes
GetOldOrder
Return OCH-List
Create oldOrder
Create newOrder
Create OCH
Add to OCH-List
mapOldOrderToNewOrder OCHs
OldOrder = getOldOrder
OldOrderItems = getOldOrderItems
BillTo = getContactAddress
ShipTo = getContacrAddress
NewOrder = getNewOrder
NewOrderItems = getNewOrderItems
NewOrderDelGroup = getNewOrderDelGroup
LastIndexOf orderFieldMapping.keySet()
!fieldName.contains(.)
!fieldName.contains(BillTo__r)
Object = oldOrder.get(fieldname)
Object = BillTo.get(fieldname)
Object = shipTo.get(fieldname)
Object == null
Log
Continue
newOrder.put(orderFieldMapping.get(fieldname), object)
newOrderDeliveryGroup.put(orderFieldMapping.get(fieldname), object)



4. Methode zum Order Mapping

```
private static List<OrderConversionHelper>
mapOldOrderToNewOrder(List<OrderConversionHelper> orderConversionHelpers){

    for(OrderConversionHelper orderConversionHelper : orderConversionHelpers)
    {
        ccrz__E_Order__c oldOrder =
        orderConversionHelper.getOldOrderObj().getOrderInfos();
        List<ccrz__E_OrderItem__c> oldOrderItems =
        orderConversionHelper.getOldOrderObj().getOrderItems();
        ccrz__E_ContactAddr__c billTo =
        orderConversionHelper.getOldOrderObj().getContactAddress().getBillTo();
        ccrz__E_ContactAddr__c shipTo =
        orderConversionHelper.getOldOrderObj().getContactAddress().getShipTo();

        Order newOrder = orderConversionHelper.getNewOrderObj().getOrderInfos();
        List<OrderItem> newOrderItems =
        orderConversionHelper.getNewOrderObj().getOrderItems();
        OrderDeliveryGroup newOrderDeliveryGroup =
        orderConversionHelper.getNewOrderObj().getOrderDeliveryGroup();

        // OrderMapping
        for(String fieldName : orderFieldMapping.keySet())
        {
            //Field with __r are skipped -> Value already in ConversionObject
            //Map normal fields
            if(!fieldName.contains('.'))
            {
                try
                {
                    Object oldFieldValue = oldOrder.get(fieldName);

                    if (oldFieldValue == null) {
                        System.debug('Field is Empty : '+ fieldName);
                        continue;
                    }

                    newOrder.put(orderFieldMapping.get(fieldName), oldFieldValue);

                    // oldOrder 1 orderDate, newOrder 2 orderDates -> both same value
                    if(fieldName.equals('ccrz__OrderDate__c')) {
                        newOrder.OrderedDate = (Date) oldFieldValue;
                    }

                    newOrder.SalesStoreId = WEBSTOREID;

                    //Set required Status
                    //Status required -> activated & draft possible, activated is
blocked
                    newOrder.Status = 'Draft';
                    newOrder.OrderStatus__c = 'Neu';


                } catch (Exception e) {
                    System.debug('Order put Error : ' + e.getMessage());
                }
            }
```

```
// Map AddressConversionField
            else {
                try {

                    //Map all BillingFields
                    if(fieldName.contains('BillTo__r'))
                    {
                        Object oldFieldValue =
billTo.get(fieldName.substring(16));

                        if (oldFieldValue == null) {
                            System.debug('Field is Empty : '+ fieldName);
                            continue;
                        }
                        if(fieldName.contains('ccrz__CountryISOCode__c'))
                        {
                            String matchingPicklistValue =
        getMatchingPicklistValue(oldFieldValue.toString(),
newOrder.getSObjectType(),
        orderFieldMapping.get(fieldName));

newOrder.put(orderFieldMapping.get(fieldName),matchingPicklistValue);
                        }


newOrder.put(orderFieldMapping.get(fieldName),oldFieldValue);

                    }

                    //Create & Map OrderDeliveryMethod
                    else if(fieldName.contains('ShipTo__r'))
                    {
                        Object oldFieldValue
=shipTo.get(fieldName.substring(16));

                        if (oldFieldValue == null) {
                            System.debug('Field is Empty : '+ fieldName);
                            continue;
                        }


newOrderDeliveryGroup.put(orderFieldMapping.get(fieldName),oldFieldValue);

                        //DeliverToName Required -> Name = LastName?
                        if(fieldName.contains('ccrz__LastName__c'))
                        {
                            newOrderDeliveryGroup.DeliverToName  =
oldFieldValue.toString();
                        }

}

            } catch (Exception e){
                System.debug('Error on getValue' + e.getMessage());
            }
            }
        }

        //Create new OrderItem for each CC-OrderItem
        for(ccrz__E_OrderItem__c item : oldOrderItems) {
            newOrderItems.add(new OrderItem())
```

```
}
        //OrderItemMapping
        for(String fieldName : orderItemFieldMapping.keySet())
        {
            try
            {
                Integer index = 0;

                for(ccrz__E_OrderItem__c item : oldOrderItems)
                {
                    Object oldFieldValue = item.get(fieldName);

                    if (oldFieldValue == null)
                    {
                        if (fieldName.contains('ccrz__Quantity__c') ||
                               fieldName.contains('ccrz__Price__c'))
                        {
                            oldFieldValue = 1;
                            System.debug(oldFieldValue);
                        } else {
                            System.debug('Field is Empty : '+ fieldName);
                            continue;
                        }

                    }

newOrderItems.get(index).put(orderItemFieldMapping.get(fieldName),
                           oldFieldValue);
                    index++;
                }


            } catch (Exception e)
            {
                System.debug('Error on getValue  :  ' + e.getMessage());
            }
        }


orderConversionHelper.getNewOrderObj().setOrderItems(newOrderItems);
        orderConversionHelper.getOldOrderObj().setOrderInfos(oldOrder);
        orderConversionHelper.getNewOrderObj().setOrderInfos(newOrder);
        orderConversionHelper.getNewOrderObj()
        .setOrderDeliveryGroup(newOrderDeliveryGroup);

    }


    return orderConversionHelpers;

}
```