Enhancing Efficiency in Cell & Gene Therapy Shipments: A Pathway to Scalability and Reliability

by

Kevin-Alexandre Jacquot
Master of Mechanical Engineering at Institut Supérieur de l'Automobile et des Transports, 2012

SUBMITTED TO THE PROGRAM IN SUPPLY CHAIN MANAGEMENT
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE IN SUPPLY CHAIN MANAGEMENT
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Signature of Author: _____
Department of Supply Chain Management
May 10, 2024

Certified by: _____
Elenna Dugundji
Research Scientist at the Center for Transportation and Logistics (CTL)
Capstone Advisor

Certified by: _____
Thomas Koch
Postdoctoral Associate at the Center for Transportation and Logistics (CTL)
Capstone Co-Advisor

Accepted by: _____
Prof. Yossi Sheffi
Director, Center for Transportation and Logistics
Elisha Gray II Professor of Engineering Systems
Professor, Civil and Environmental Engineering

Enhancing Efficiency in Cell & Gene Therapy Shipments: A Pathway to Scalability and Reliability

by

Kevin-Alexandre Jacquot

Submitted to the Program in Supply Chain Management
on May 10, 2024 in Partial Fulfillment of the
Requirements for the Degree of Master of Applied Science in Supply Chain Management

ABSTRACT

The logistics supporting life-saving Cell & Gene Therapy treatments, such as autologous CAR-T, face significant challenges due to strict constraints like time sensitivity, temperature control, and regulatory compliance. These constraints make the supply chain vulnerable to disruptions that could result in the loss or damage of these delicate, high-value therapies during global shipments handled by specialty couriers third-party couriers. To address this issue, a study was conducted to analyze historical shipment data from both qualitative and quantitative perspectives. The goal was to create a model that could predict possible disruptions by calculating "validation points" throughout the shipment process and updating them continuously for each new input. As a result, the sponsors' planning team would be informed in advance of any potential disruptions, allowing them to proactively reach out to their couriers for immediate action. Although still in its early stages, the model provides more visibility into current processes and lays the foundation for a scalable solution to be implemented in the sponsor operating system. Additionally, this study has allowed the sponsor to review their current process with their couriers and identify areas for improvement in terms of process, data gathering, and data quality. A proposal for a roadmap is also included, outlining possible enhancements that could leverage Machine Learning and AI techniques.

Capstone Advisor: Elenna Dugundji
Title: Research Scientist at the Center for Transportation and Logistics (CTL)

Capstone Co-Advisor: Thomas Koch
Title: Postdoctoral Associate at the Center for Transportation and Logistics (CTL)

ACKNOWLEDGMENTS

I would like to extend my deepest gratitude to the advisory team at my sponsor's organization for their invaluable patience and insightful feedback throughout this project. My profound appreciation also goes to my capstone advisors, Elenna Dugundji and Thomas Koch, whose support has been fundamental to this endeavor.

I am thankful to the MIT staff at the CTL lab for providing me with the opportunity to fulfill my dream of studying at this prestigious university. Special thanks are due to Pamela Siska for her meticulous editing and review of the various drafts of this final document.

Finally, I must express my heartfelt thanks to my partner, Adélia, and her two sons, Vitoria and Lucca, for their enduring belief in me and their steadfast support over these past five months.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Motivation

CAR-T, also known as Chimeric Antigen Receptor T-cell therapy, is a groundbreaking and personalized approach to treating specific types of lymphoma and leukemia cancers with remarkable success rates. This form of immunotherapy works by utilizing a patient's immune system to target and destroy cancer cells. T-cells are collected and genetically modified to express a Chimeric Antigen Receptor (CAR) on their surface. This CAR is designed to specifically target a protein (antigen) found on the surface of the patient's cancer cells. The CAR recognizes and binds to the cancer cells, which then triggers the patient's immune system to fight the cancer (Papathanasiou et al., 2020).

This form of immunotherapy, part of the Cell & Gene Therapy (C&GT) industry, has shown considerable progress, especially since the FDA approved the initial CAR-T therapies, Kymriah (*tisagenlecleucel*) and Yescarta (*axicabtagene ciloleucel*) in 2017. These treatments are used for refractory/relapsed B-cell precursor acute lymphoblastic leukemia (ALL) in patients up to 25 years old and adult patients with relapsed or refractory diffuse large B-cell lymphoma (DLBCL) (EMA, 2018). These treatments have inspired further research and clinical trials, including both autologous (where cells or tissue are obtained for an individual's own use) and allogeneic (where cells or tissue are obtained for use in others) products.

Despite the initial triumphs, there are challenges to the manufacturing and supply chain processes of CAR-T therapies. The manufacturing process is complex, with multiple steps that are difficult to execute and synchronize across various locations. Additionally, as we progress toward establishing CAR-T cell therapies as pivotal options in cancer treatment, the current supply chain model can accommodate only a limited number of patients, which creates scalability issues and necessitates substantial enhancements (Harrison et al., 2019).

Furthermore, the prohibitive costs of developing, manufacturing, and administering CAR-T therapies significantly influence their list prices. For instance, in Europe, Kymriah is priced around €320,000 and Yescarta at €327,000 (Jørgensena, Hannab & Kefalasa, 2020), while in the US, they are respectively sold for $475,000 and $373,000 (Chen, Abila & Kamel, 2021).

Moreover, regulatory directives from federal agencies (Kaiser et al., 2015) and complex reimbursement procedures (Papathanasiou et al., 2020) further compound the challenges of CAR-T therapies, which already involve intricate custodial procedures and substantial expenses.

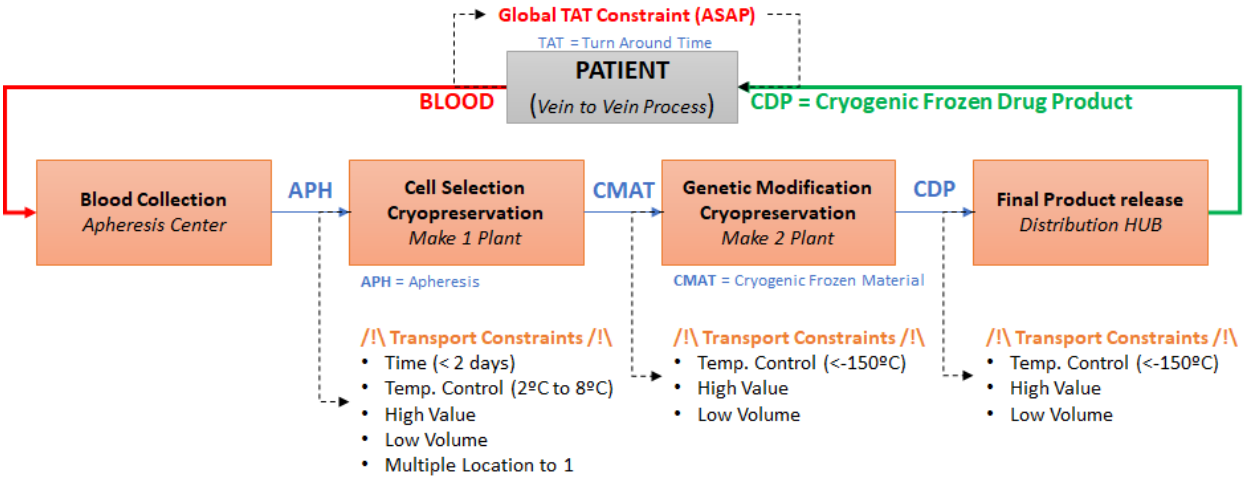
Our sponsor, one of the largest pharmaceutical companies globally, provides some of the CAR-T-approved therapies. They are actively engaged in constructing a robust and expandable supply chain model to effectively address the increasing demand.

1.2 Problem statement and research questions

As mentioned, our sponsor company distributes authorized CAR-T therapies worldwide, where the patient is their own donor. This process is commonly called "vein to vein" (Papathanasiou et al., 2020). It begins with the cell collection at a specialized clinic known as an apheresis center. The cells collected through this process, referred to as apheresis (APH), are then transported to the initial manufacturing site, "Make 1," for activation and cryopreservation. The semi-finished goods called "Cryogenic Frozen Material" (CMAT), are then transferred to the second manufacturing site, "Make 2," where various processes such as genetic modification, expansion, formulation, cryopreservation, and final Quality Control/Quality Assurance (QC/QA) are conducted. The final product, known as Cryogenic Frozen Drug Product (CDP), is then distributed directly to the infusion center, or through a hub if any are present in the region, where the patient receives customized treatment based on their own cells (refer to Figure 1).

Figure 1

Vein to Vein Process



Note: Represent the Vein-to-Vein Process, from collection to application. Own work.

Currently, third-party logistics couriers known as "specialty couriers," handle the shipping operations between facilities. However, due to the constraints presented in Figure 1 for each leg, it is

crucial to closely monitor time and temperature during transit to prevent any risks of excursion, which could lead to the loss of collected T-cells (APH) from the patient, the semi-finished goods (CMAT) or the finished goods (CDP) before patient treatment at the infusion center.

To ensure accurate tracking of shipment statuses during transit, our sponsor utilizes a tracking system fed with real-time data from various couriers. These data inputs are gathered from fragmented sources, i.e. the external partners responsible for the shipments. However, the system lacks granularity for effective planning and timely shipments and an alert management system crucial for preventing disruptions. These disruptions, ranging from traffic congestion to airline delays due to external factors like storms or strikes, can have lethal consequences given the criticality of the products being managed. Consequently, the sponsoring organization has expressed concerns about the third-party logistics providers' failure to integrate risk management into their daily operations. Despite communication of primary shipment milestones, there is no comparison with non-shared planning, and inadequate monitoring negatively impacts material tracking. This lack of visibility in the shipping process leads to the sponsor and courier not being able to proactively react to shipping issues with the end user having to report them to the sponsor upon their discovery. This poses risks of delays at critical touchpoints with downstream ramifications.

In addition, mainly due to the specialized and relatively new nature of the Cell & Gene Therapy industry, the sponsor has not found reliable and resilient digital Control Tower tools on the market, a sentiment shared by other pharmaceutical companies providing the same kind of treatment. While some entities in the C&GT space have begun to utilize AI (Artificial Intelligence) and ML (Machine Learning) to process diverse data sources, these tools are still in their early stages of development and do not meet the sponsor's requirements.

Furthermore, the sponsor foresees a five-fold increase in demand for CAR-T therapies in the coming years. While the current have a 95% accuracy in term of time delivery, it currently requires extensive and time-consuming manual follow-up. They recognize that their existing Tracking/Alert Management system falls short of offering the desired level of control, leaving them vulnerable to disruptive factors. Without proper automation, they anticipate challenges in meeting the expected growth.

In response to all these challenges, the sponsor requires a robust alert system that can detect events and enable their team to take proactive measures. They also seek a scalable solution that can monitor shipments across the full vein-to-vein process, from collection to delivery, on a global scale.

Moreover, the system should have a solid foundation to accommodate future enhancements, including alternative shipping plans based on the product's location, within a short time frame.

In that context, the questions to be answered were:

- How can information be captured from multiple sources and aggregated into a unique 'source of truth'?
- How can the entire shipment cycle be monitored, and potential disruptions predicted, while managing constraints?"

1.3 Project goals and expected outcome

Considering the challenges highlighted above, the primary goal of this project was to craft a scalable Proof of Concept (PoC) for our Sponsor, encompassing tracking, monitoring, and alert functionalities. This model initiative aims to consolidate data from diverse internal and external sources and create the foundation to harness the potential of AI and ML.

The main benefit of our project was the development of a robust solution that will support the sponsor in their decision-making by leveraging existing data analytics capabilities, leading to better risk mitigation, while being scalable and in adherence to regulatory and quality standards. Our Sponsor envisions this tool as an alternative to their current system, aspiring for it to become a cornerstone of their supply chain infrastructure. This strategic tool shall have the potential to support the success of CAR-T therapies and any potential Cell & Gene Therapies, in both clinical and commercial stages.

Within this context, the deliverables for the Sponsor are:

- A functional POC able to:
    - Record, in a dedicated database, the historical and current main events sent by the couriers.
    - Dynamically calculate “validation points” by looking at WHEN the sponsor should have received those events.
    - In case of a missing event in the time frame defined previously, send an alert for proactive intervention.
- A detailed roadmap outlining potential future enhancements, emphasizing scalability for other products and regions.

2 STATE OF THE ART

As described in the previous chapter, shipments involving CAR-T therapies and any potential Cell & Gene Therapies present a multifaceted set of challenges and constraints:

- *Time and Temperature Sensitivity*: These shipments are extremely sensitive to time and temperature, increasing the risk of obsolescence (Meacle et al., 2016).
- *High Value Shipments*: Besides the significant cost associated with these therapies, they carry an important human factor, being a potential life-saving solutions for cancer patients.
- *Global Low Volume Shipments*: Products are shipped unitary from several locations, concentrating their destination to only a few manufacturing facilities around the globe (multiple-to-one), then back to the hundreds of infusion centers, where the product originated (one-to-multiple) (Hanley et al., 2022).
- *Third-party Courier Dependence*: Shipments are managed and overseen by third-party couriers due to the global scope, resulting in reduced immediate control from the pharmaceutical companies regarding the ongoing activities.

In this chapter we analyze the most significant works done by other professionals or scholars on how those constraints are usually tackled in the C&GT market or in other markets confronted with the same stringent requirements. We also go in more depth into the principle of the "Control Tower," looking at the benefits and challenges such technology can bring.

2.1 Inherent Supply Chain challenges CAR-T Therapy faces

2.1.1 End-to-End Versus Vein-to-Vein

In the previous chapter, we provided a comprehensive overview of the C&GT industry. This section delineates the comprehensive lifecycle of Autologous CAR-T cell therapy, encompassing the process from collection to administration (vein-to-vein), and underscores the distinct attributes that set it apart from the traditional pharmaceutical supply chain (Rutherford, Barry, Campbell & Turner, 2017).

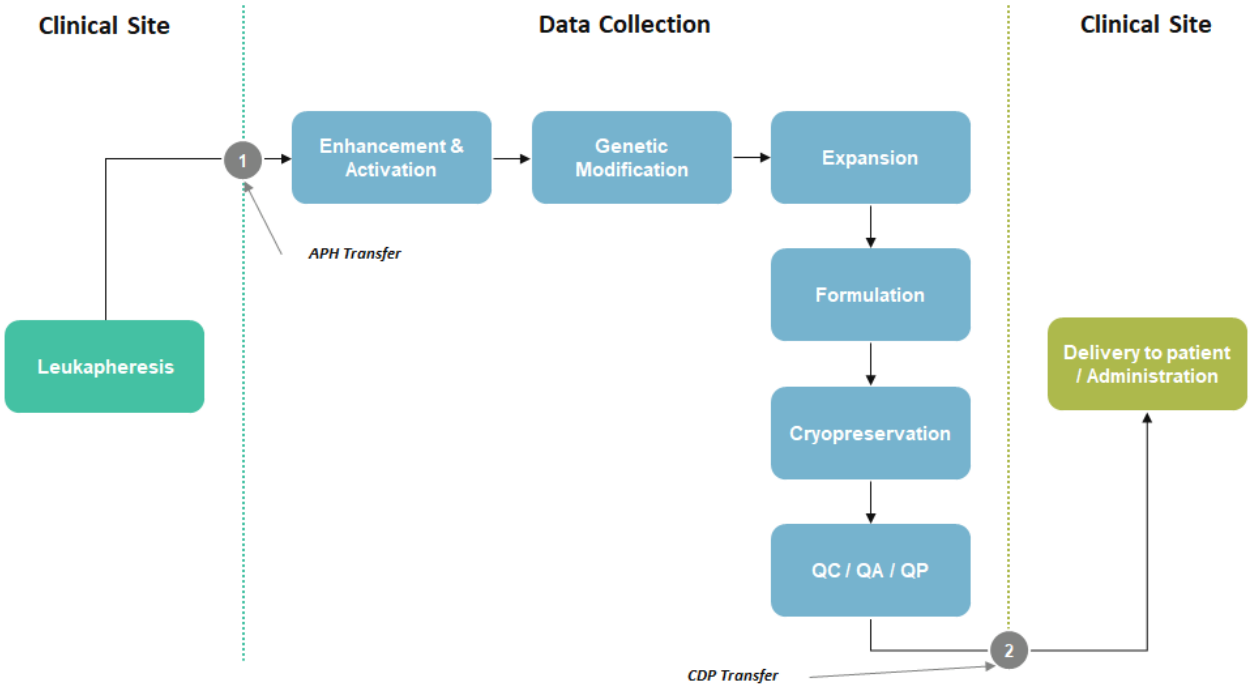
Shah (2004) describes the conventional "*end-to-end*" pharmaceutical supply chain structure, which consists of established warehouses or distribution centers tasked with the storage and distribution of manufactured drugs to retail outlets. The typical stages include primary manufacturing (production of the active ingredient), and secondary manufacturing (assembly of the final product in SKU format), followed by market warehouses/distribution centers, wholesalers, and ultimately retailers/hospitals.

Conversely, "*vein-to-vein*" autologous CAR-T cell therapies are characterized by a patient-centric supply chain model, emphasizing a bidirectional flow where the patient is pivotal. The pathway of materials is as follows:

- Apheresis Center (Clinical site for leukapheresis or "cell collection").
- Manufacturing site for therapy production.
- Infusion Center for therapy administration.

Figure 2

Current CAR-T cell process steps



Note: Represent the main flow of the manufacturing of CAR-T. Adapted from "Autologous CAR T-cell therapies supply chain: challenges and opportunities?" by M. M. Papathanasiou, C. Stamatis, M. Lakelin, S. Farid, N. Titchener-Hooker and N. Shah, 2020, *Cancer Gene Therapy 27*, p. 799–809. Copyright 2020 by Springer Nature America, Inc.

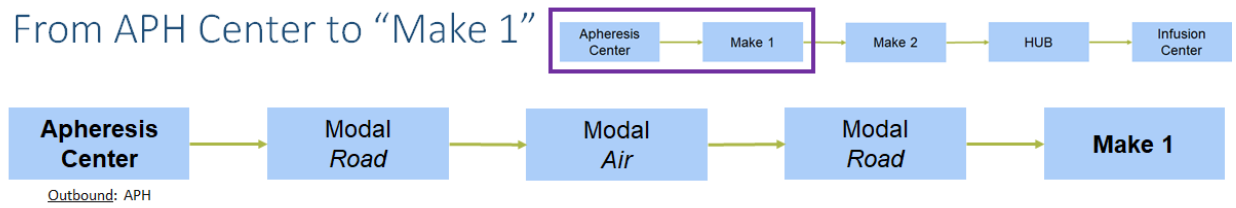
Apheresis Center

The leukapheresis procedure occurs at a specialized clinical site, extracting patient blood to separate the leucocytes, and returning the remaining blood to the patient's circulation (Levine, Miskin, Wonnacott, Keir, 2017). Subsequently, the sample is transported to the manufacturing site for further

processing. Depending on regulatory authorization and manufacturer preference, the sample can be transferred fresh (2ºC to 8ºC), frozen (−80 ºC), or cryopreserved (−180 ºC). In our capstone, the product of this phase is referred to internally as APH and is shipped "fresh" using a Single-Use Credo Cube for packaging, maintaining the product between 2ºC to 8ºC for 72 hours.

Figure 3

Considered modals from "Apheresis Center" to "Make 1"



Note: Represent the modals considered from the Apheresis Center to Make 1. Own work.

Manufacturing site

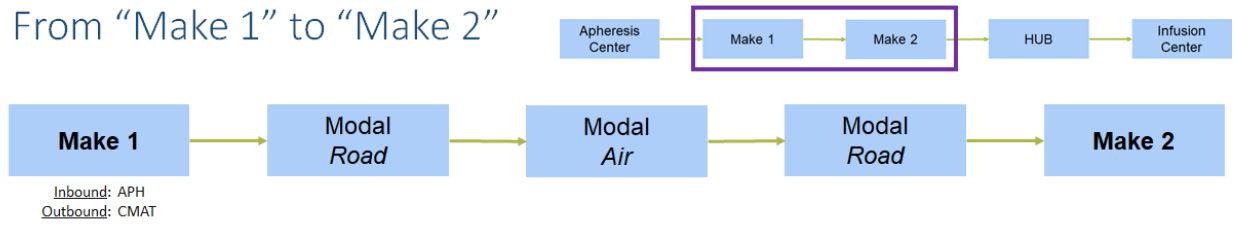
At the manufacturing site, cells undergo several processes including enrichment, activation, genetic modification, expansion, formulation, and cryopreservation, until the final therapeutic product is prepared for hospital delivery.

A key process in CAR T cell therapy production is the genetic modification, where patient T cells are transduced with the CAR receptor, typically through viral gene transfer systems. Quality Control/Assurance is executed after this modification, with the final product being cryopreserved and dispatched to the administration site. It is important to note that QC/QA might be conducted in-house or outsourced (Papathanasiou et al., 2020).

Per our sponsor company's protocols, manufacturing is generally executed at two distinct facilities named "Make 1" and "Make 2." "Make 1" involves cell enrichment and activation, resulting in Cryogenic Frozen Material (CMAT) that is shipped in a specialized dry vapor container, keeping the material below -150ºC for approximately 8 days. Subsequent processes including genetic modification, expansion, formulation, cryopreservation, and QC/QA occur at "Make 2," from whence it is then sent to a HUB. Based on the scope of our project, we focus on the reception of the APH in one specific plant, acting as Make 1 and Make2.

Figure 4

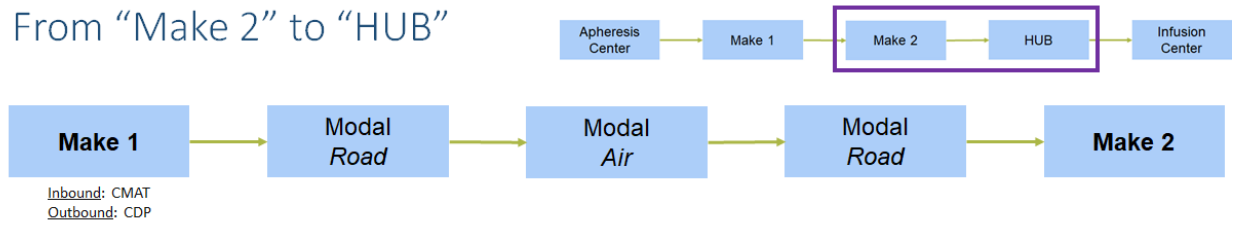
Considered modals from "Make 1" to "Make 2"



Note: Represent the modals considered from Make 1 to Make 2. Own work.

Figure 5

Considered modals from "Make 2" to "HUB"



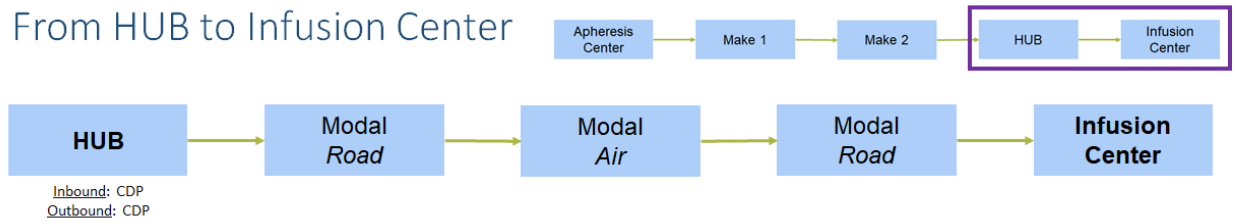
Note: Represent the modals considered from Make 2 to Hub. Own work.

Infusion Center

Following successful release, the therapy is transported from the HUB to the hospital, where it is thawed and administered to the patient. This step typically requires about one week of pre-conditioning in alignment with the patient’s medical status prior to therapy administration (Papathanasiou, Stamatis, Lakelin, Farid and Titchener-Hooker, 2020).

Figure 6

Considered modals from "HUB" to "Infusion Center"



Note: Represent the modals considered from Hub to the Infusion Center. Own work.

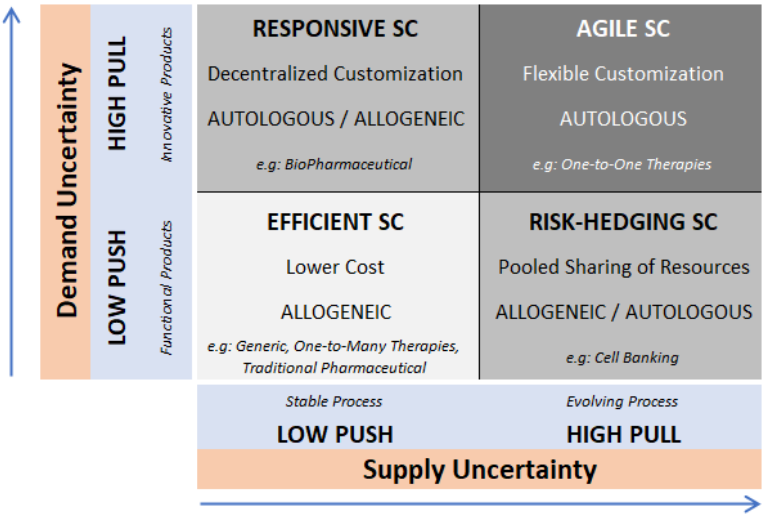
2.1.2 Leading with Uncertainty

In the previous section, we compared the supply chain of autologous CAR-T therapy to traditional pharmaceuticals from the point of view of Distribution and Manufacturing. This section will elaborate on this comparison, focusing on the uncertainty in supply and demand, again largely due to the need for a human donor and the complex, variable nature of the product.

In 2013, Teng et al. proposed a classification framework for cellular therapies, which has been referenced in subsequent research addressing supply chain uncertainties in the C&GT industry (e.g., Rutherford et al., 2017). In their study, Teng et al. separated these therapies into two main groups based on the source of the cells: either one-to-one (autologous and/or allogeneic) or one-to-many (allogeneic) using a single donor. They further categorized these based on the demand (high or low) and the method of administration (product or procedure), using a 2x2 matrix to map out demand uncertainty and supply uncertainty (Figure 7).

Figure 7

Therapeutic Medicine Supply Chain Uncertainty Framework



Note: Represent the different strategies to manage the Supply Chain of therapeutic medicine to enhance the effectiveness and efficiency. Adapted from "An analysis of supply chain strategies in the regenerative medicine industry—Implications for future development" by C. W. Teng, L. Foley, P. O'Neill and C. Hicks, 2013, *International Journal of Production Economics 149*, p. 211–225. Copyright 2013 by Elsevier B.V.

The framework identifies four key supply chain strategies according to levels of uncertainty:

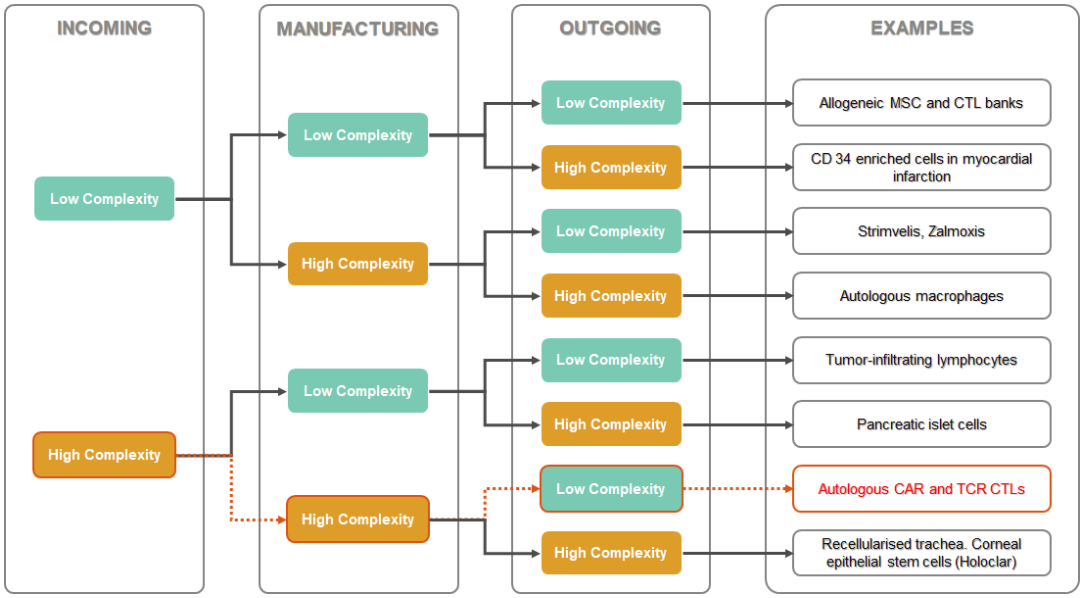
- *Efficient Supply Chain (Push-Push)*: Features stable, predictable demand, focusing on economies of scale to reduce costs, crucial for low-margin treatments.
- *Responsive Supply Chain (Push-Pull)*: Maintains raw materials at a decoupling point until a specific customer order or patient requirement is confirmed.
- *Agile Supply Chain (Pull-Pull)*: Follows a make-to-order approach, initiating production only when customer demand is confirmed, similar to autologous CAR-T therapies, where the donor is the critical point of material decoupling.
- *Risk-Hedging Supply Chain (Pull-Push)*: Utilizes a pull strategy in markets with unpredictable supply, accumulating materials in anticipation of future needs.

While this framework effectively maps the uncertainties in cellular therapies, we believe that the approach by Rutherford et al. (2017) more accurately reflects the complexities involved by examining the entire process from start to finish:

- *Incoming*: Focuses on the challenges of obtaining donor material, potentially requiring specialized donation facilities, and considering geographical limitations.
- *Manufacture*: Highlights the need for timely transportation to the manufacturing site, especially important for perishable materials requiring controlled conditions.
- *Outgoing*: Addresses issues such as product perishability, shelf life, advanced processing needs, tracking and tracing, chain of custody, specialized logistics, and timing of process execution.

Figure 8

Cellular Therapeutics Supply Chain complexity breakdown



Note: Represent the levels of complexities from the Donation (collection and Shipment of the human material) to the Manufacturing and Distribution back to the patient. Adapted from "The Importance of Understanding & Designing Cellular Therapy Supply Chains" by C. Rutherford, J. Barry, J. DM. Campbell and M. Turner, 2017, *Cell and Gene Therapy Insights 3*(10), p. 873–889. CC-BY-NC-ND Open Access license by BioInsights.

This approach highlights the need for a cellular therapeutic supply chain strategy that accounts for complexities and constraints throughout the incoming, manufacturing, and outgoing stages. Excluding the manufacturing aspect for now, the incoming process for autologous CAR-T is considered highly complex, while the outgoing process is somewhat less complex, mainly due to the less rigorous time constraints.

2.1.3 Temperature and Time: Characteristics and Challenges

In preceding discussions, we have underscored the scarcity (low volume) and high cost associated with autologous CAR-T cell therapies as primary constraints, and outlined optimal supply chain strategies to address these challenges. This section will focus on other critical factors, namely time and temperature, which are essential in formulating an effective supply chain model for these therapies.

Given the patient-specific customization of each therapy in the one-to-one model, the loss of any sample or therapy becomes significantly detrimental due to its irreplaceable nature. These therapies are subject to strict shelf-life limitations, necessitating rigorous control over the duration of processing, storage, and transportation.

Similarly, temperature fluctuations and mishandling also present substantial risks. Owing to the fragile nature of CAR-T cells, which are prone to damage under suboptimal conditions, it is crucial to maintain storage and transport under precisely regulated environments. This necessitates the use of specialized packaging systems and meticulous handling protocols.

To mitigate these risks, therapies are frequently transported using dry shippers that uphold necessary temperature conditions throughout transport. Papathanasiou et al. (2020) advocate for cryopreservation as a superior method owing to its flexibility in extending the transport and treatment window compared to fresh products, which are constrained to a maximum storage duration of 24-72 hours depending on the packaging system utilized. Cryo-transport systems are capable of maintaining both temperature and product integrity for periods of 10–14 days. Selecting an appropriate shipping

package that can preserve the required temperature for the necessary duration is crucial in reducing the risk of product loss due to mishandling.

The following are descriptions of two shipping containers used by our sponsor company, differentiated by the type of material transported and their respective capabilities in maintaining controlled temperatures and holding times.

Figure 9

Type of Packaging Consider

Note: Represent the packaging used by the sponsor company depending on the type of products shipped. Image on the left side from *Credo Cube™ Series 4 Product Sheet*, by Peli BioThermal™, n.d. (https://www.pelibiothermal.com/sites/default/files/202212/Credo%20Cube%20Series%204%20Product%20Sheet.pdf). Copyright 2024 by Peli BioThermal Limited. Image on the right side from Advanced Therapy ShipperTM, by Cryoport Systems, n.d. (https://cryoport.com/solutions/shipping-systems/cryoport-express/). Copyright 2024 by Cryoport Systems, LLC.

2.1.4 Cell & Gene Therapy Track and Trace Necessity

Another pertinent characteristic in the supply chain model of CAR-T cell therapies pertains to what is termed Track & Trace, which involves the tracking of both samples/products and patient identification. The global production and distribution of medicines necessitate rapid information exchange among regulatory authorities to uphold supply chain integrity and ensure patient safety. Track and trace systems are recognized as valuable tools in mitigating risks such as shortages and the circulation of counterfeit medicines by offering visibility into the medicine supply chain at any given point (EMA, 2021).

On March 9, 2022, the Center for Drug Evaluation and Research (CDER) of the U.S. Food and Drug Administration (FDA) released a revised draft guidance for industry titled Verification Systems Under the Drug Supply Chain Security Act for Certain Prescription Drugs. This updated guidance addresses the verification systems essential for manufacturers, repackagers, wholesale distributors, and dispensers to comply with the Federal Food, Drug, and Cosmetic Act (FD&C Act) as amended by the Drug Supply Chain Security Act (DSCSA). The DSCSA outlines crucial steps toward establishing an electronic, interoperable Track & Trace system for specific prescription drugs distributed in the US. Additionally, it delineates definitions, requirements for supply chain participants, standards for licensing wholesale drug distributors and third-party logistics providers, and mandates manufacturers to test electronic connections with trading partners, to be operational on November 2023.

Although autologous CAR-T cell therapies are exempt from this new regulation in both the US and EU, this exemption does not absolve these markets from adhering to strict guidelines and compliance requirements outlined by the US FDA and other global regulatory agencies (Meacle et al., 2016).

In the context of CAR-T cell therapies, bi-directional tracking is essential to ensure the correct therapy reaches the right patient at the culmination of the product cycle. Efficient tracking of each product from the Apheresis Center to the Infusion Center is crucial. Equally significant is patient identification during cell harvesting, product release, and treatment stages. Patient-specific identity linking to the sample is imperative for sample-specific identity requirements during cell harvesting and to ensure that the right therapy is administered to the appropriate patient during product release and treatment.

Furthermore, addressing chain-of-custody documentation remains a challenge, necessitating the recording of location, security, and temperature details throughout various facilities, involving diverse personnel and organizations.

2.1.5 Transportation

Time serves as a pivotal constraint that influences not only the duration between two destinations but also dictates the selection of transportation modes, a factor hitherto unaddressed. To streamline the complex supply network and reduce the number of hand-offs, firms within the G&GT sector collaborate with premier specialty couriers services that specialize in the meticulous handling of these delicate shipments. Additionally, the unique characteristics of the C&GT field previously discussed require sophisticated management through technological innovations, process improvements, and shifts in organizational mindset.

The transportation of C&GT products, such as APH or drug products, to and from manufacturing sites often necessitates intermodal transport solutions, incorporating both ground transportation and aviation (Yang, 2006). Commercial airlines are frequently utilized owing to their widespread availability and extensive network. However, in instances where delays could critically impair product integrity, charter flights may be employed to mitigate risk.

Airline disruptions are just one of several factors that contribute to delays across the entire shipment process, often precipitating significant cascading effects and prolonged recovery times, which can severely disrupt product timelines. To tackle these challenges, our analysis shifted from a narrow focus on cargo shipments to a broader examination of passenger disruption recovery strategies, due to the more readily available data in this area. The airline industry has made substantial investments in enhancing recovery protocols following significant disruptions, primarily those caused by adverse weather conditions. Through the utilization of advanced technology, analytical tools, dedicated control centers, improved processes, structured recovery methodologies, mindset adjustments, and comprehensive strategies at the industry level, airlines strive to minimize delay impacts and expedite passenger arrivals to their destinations (Gershkoff, 2016).

2.2 Control Tower

According to Shekarian and Mellat Parast (2021), Supply Chain Resilience (SCRES) is designed to alleviate the impact of disruptions by identifying strategic actions that enable an effective response to incidents. Central to enhancing supply chain resilience are the principles of visibility, redundancy, and network simplification, which collectively strengthen the ability of organizations to manage various types of disruptions—including demand, supply, process, control, and environmental issues. Our capstone project focuses on the latter three categories (process, control, and environmental disruptions) and aims

to improve visibility through the implementation of Artificial Intelligence-driven data analytics, encapsulated within the framework of a control tower.

A prior procurement control tower, in a capstone project by Kumar & Gomez (2023), emphasized that control towers commence operations by integrating data from multiple sources. These systems are optimally cloud-based, leveraging real-time data to enhance transparency into supply chain activities, disruptions, and support decision-making processes. The disruptions spurred by the Covid-19 pandemic have accelerated investments and shifted organizational priorities toward the deployment of control towers that manage end-to-end business processes, grounded in three foundational pillars outlined by Siddharth (2020):

- Comprehensive, cross-organizational, proactive approaches focused on exception management.
- Fostering a culture of organizational change and learning, with a profound grasp of business operations to react dynamically to events and optimize processes.
- A sophisticated data platform that continuously monitors transactional data from both internal and external sources, capable of autonomously detecting issues and initiating alerts for stakeholders.

Hofman (2014) described the structure of a multi-modal and synchronized control tower for logistics that operates on real-time data, necessitating open data exchange. The architecture of the control tower includes a "rules handler" skilled in spotting deviations from planned scenarios by monitoring data changes.

Our sponsor does not have a dedicated control tower and instead, relies on the one operated by their designated specialty couriers service provider, which, in turn, shares critical data, feeding into a third-party system utilized by the sponsor to oversee shipments. Despite the existing data exchange between the two entities, there remains a gap in establishing necessary "Checkpoints" and Alerts that could preemptively address disruptions during transit.

3 METHODOLOGY

This section outlines the methodology adopted for this research, which was underpinned by an extensive review of relevant literature and continuous consultations with our project sponsor. It was determined that a sequential approach to addressing the research questions would optimally satisfy the sponsor’s needs.

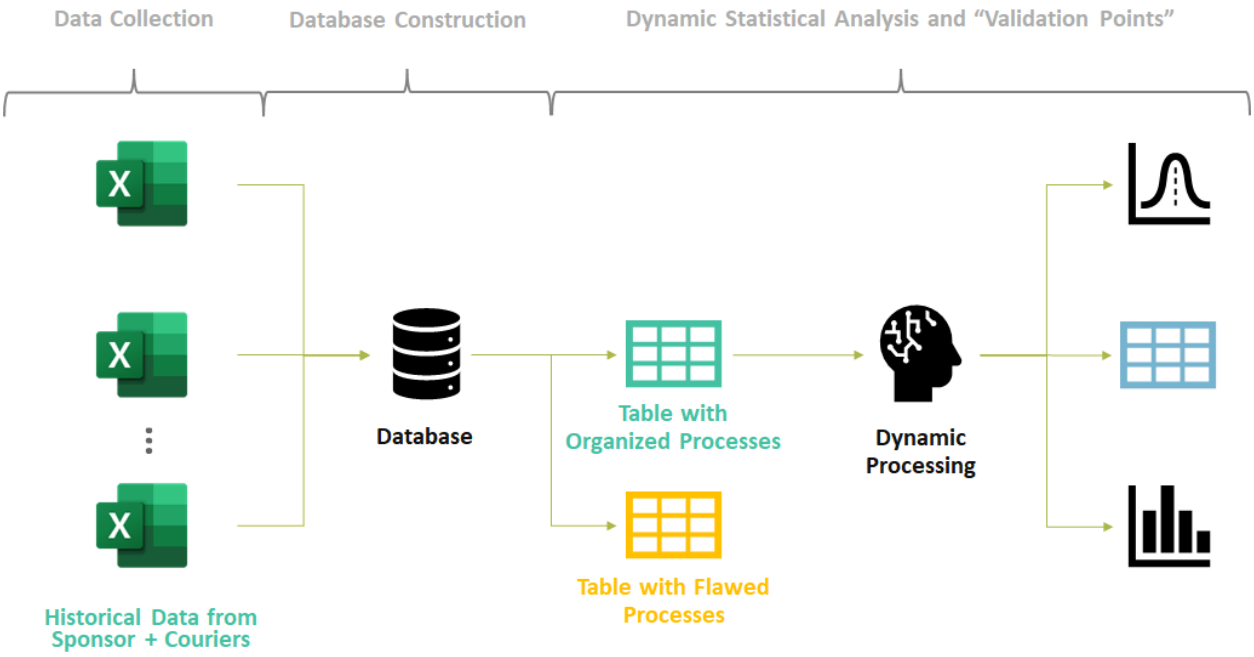
The research began by addressing the query: "How can information be captured from multiple sources and aggregated into a unique 'source of truth'?" This involved the creation of a centralized data repository, primarily populated with information provided by suppliers, which served as the authoritative reference for the development of our Proof of Concept (PoC). The construction of this PoC adhered to the methodology illustrated in Figure 10, focusing on:

- Data Collection
- Database Construction
- Dynamic Statistical Analysis and "Validation Points"

Following the establishment of this foundational repository, the research explored this question: "How can the entire shipment cycle be monitored, and potential disruptions predicted, while managing constraints?" This question has been addressed through sophisticated data analysis and predictive analytics.

Figure 10

Methodology



Note: Represent the methodology applied. Own work.

3.1 Data Collection

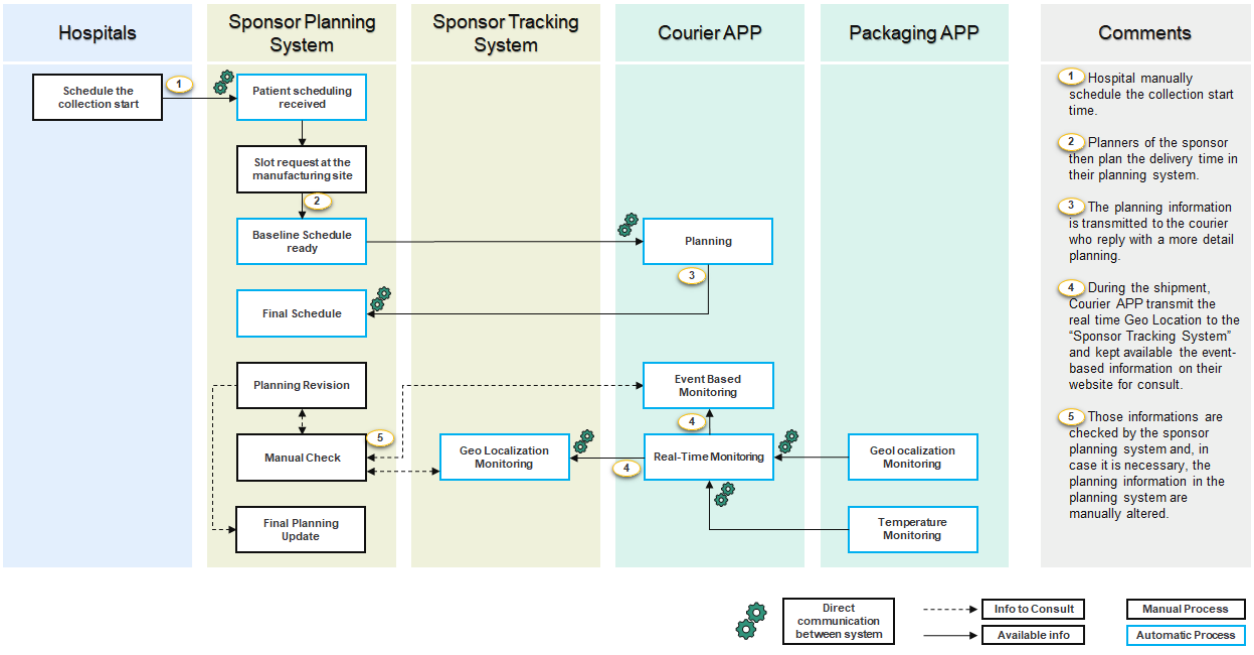
Initially, it was imperative to comprehend the source, relevance, and extraction methods of each data set. This critical step involved an in-depth examination of the sponsor's existing processes and systems.

3.1.1 Process Overview

At the Apheresis Center, practitioners schedule patient treatments through a system proprietary to the sponsor. This information is internally processed by the sponsor's scheduling team, who generate a unique identifier serving as the primary reference. They also designate key milestones to be met by couriers to ensure timely delivery to the subsequent phase (Make 1) within a 48-hour window. This data is input into a "Planning System," linked to the "Couriers App." Upon receipt of this data, the courier planning team initiates their scheduling activities and conveys a planned schedule back to the sponsor. This schedule then becomes the baseline for the entire process in the sponsor's system.

Figure 11

Interactions between system



Note: Flow-Chart representing the communication flows between the company, the hospitals and the couriers to manage the shipments. Own work.

Figure 11 depicts a simplified schematic of the systems interactions, highlighting the varying degrees of information consistency due to differences in integration levels with the sponsor and internal processes of the couriers.

In the absence of unforeseen events (e.g., delays or cancellations by the Apheresis Center or the sponsor company), couriers are expected to collect and deliver cells according to the schedule set by the Apheresis Center and planned by the sponsor. The information flow from the courier to the tracking system owned by the sponsor is critical for monitoring ongoing processes. The sponsor's planning team then needs to manually verify the existence of any discrepancies or gaps in information, and if encountered contact the courier planning team or directly access flight information from the airline's website.

For this study, the focus is specifically on one courier, designated as "Courier A" for the remainder of this document.

3.1.2 Data Sources

This section delves into the principal sources of information that were utilized as the primary "source of truth" for our Proof of Concept (PoC).

Planning System (Sponsor)

This planning system is designed to serve as the central data repository, aggregating information from various sources, including different couriers and internal processes such as manufacturing and distribution. While it acts as the main repository and is instrumental in planning and tracking current shipments, it also facilitates logistics planning for couriers by providing essential details (Process ID, Product ID, Pickup Timestamp and origin, Planned delivery time and destination, etc.). However, multiple integrations over time have led to information duplication with slight variations in naming, complicating data management in the event of unexpected changes. Furthermore, the system permits manual adjustments, which can result in the loss of historical data, complicating the preservation of a shipment's planning baseline.

Key challenges include:

- Extensive lines of mixed information from various systems,
- Replicated information under different names,
- Difficulties in extracting valuable data,
- Dual functionality for planning and tracking,

- Predominantly serving as the main repository,
- Manual modifications that disrupt the “FACT_Shipment_Baseline”.

Courier App

As previously mentioned, information standardization among couriers is lacking; therefore, we narrow our focus to data provided by Courier A. This courier sends planning details based on the sponsor's data and updates for each shipment milestone. These updates feed directly into the sponsor's planning and tracking systems. The data undergo manual updates during the process, which replaces older entries with new ones. For planning purposes, Courier A provides detailed information about the shipment, including flight departure and arrival times, which supplements the basic information provided by the sponsor.

Throughout the shipment, Courier A utilizes real-time GPS tracking for internal purposes but communicates only fixed milestones to the sponsor via the courier website, which include:

1. *Pickup* – Timestamp of cell collection at the Apheresis Center.
2. *Tender to Airline* – Timestamp when the product and documentation are handed over to the airline.
3. *Flight Departure* – Timestamp of the flight's departure.
4. *Flight Arrival* – Timestamp of the flight's arrival.
5. *Recovery* – Timestamp when the courier retrieves the product from the airline.
6. *Delivery* – Timestamp when the courier delivers the product to the manufacturing site.

Key challenges include:

- Dependency of the Courier to gather the data,
- Predominantly serving as the main source of truth for the shipments information,
- Manual modifications that disrupt the “FACT_Segments_Planned”.

Tracking System

The tracking system collects the geolocalization data from the courier. It is utilized by the sponsor's planners to monitor ongoing shipments. Although the real-time geolocation data of the ongoing shipment is transmitted to this system, it is not linked to the planning system which contain all the initial planning informations. The planning team then need to manually look at both system to check the real status of the shipment. No analytical processing is performed that could assist planners in determining whether a shipment is delayed, on schedule, following the planned route, or has been rerouted.

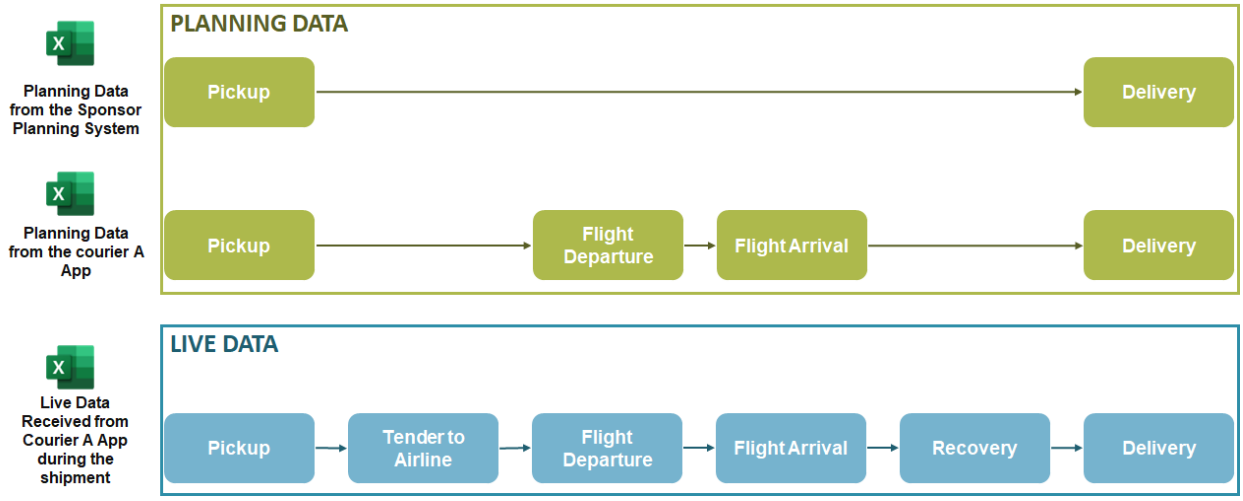
Key challenges include:

- Difficulty to retrieve the geolocalization from the system
- Not used in this project.

In summary, a diverse array of data circulates among various companies and systems. Besides fundamental data concerning the product and therapy from the sponsor, Figure 12 illustrates the aggregation of this information and its origins.

Figure 12

Source and Data to be considered in the database



Note: Represent the main information and sources provided during the process of planning and shipment. Own work.

Concerns about Data Integrity

It is noteworthy that the ability to manually alter data during processing, particularly in ways that overwrite older information with newer updates, introduces "mutable" data issues. Our review has revealed that initial planning data are often superseded by more recent information, resulting in a disconnection between planned and actual events from both the sponsor's and Courier A's perspectives. This issue of "data loss" could also be framed as a problem of "version control," where the absence of an effective versioning system means only the most current data version is retained, and historical data is lost.

This situation highlights the critical need for maintaining an audit trail, version history, or change log to ensure data integrity and traceability.

3.2 Database Construction

Following the identification of data sources, our focus shifted to the development of a database that would structure the input data across various tables to facilitate further analysis.

3.2.1 Database Development Process

Constructing an online database encompassed several critical stages designed to ensure robustness, security, and efficiency:

1. **Requirements Analysis**:
   - *Gather Requirements*: Engage with stakeholders such as developers, business managers, and end users to ascertain their needs.
   - *Define Objectives*: Establish the database's purpose, detailing the types of data it will store, access methods, and specific performance or security criteria.
2. **Design Phase**:
   - *Conceptual Design*: Develop a high-level model using Entity-Relationship (ER) diagrams to depict data storage and entity interactions.
   - *Logical Design*: Convert the conceptual design into a logical model, specifying the database structure through data tables, fields, keys, and relational constraints.
3. **Implementation**:
   - *Database Creation*: Utilize a Database Management System (DBMS) to establish the database per the conceptual design.
   - *Data Definition*: Employ Data Definition Language (DDL) within the DBMS to create tables, relationships, constraints, and other database elements.
   - *Data Manipulation*: Populate the database using Data Manipulation Language (DML) commands such as INSERT, UPDATE, and DELETE.
4. **Testing and Evaluation**:
   - *Test the Database*: Execute various tests to verify that the database meets all specified requirements, including functionality, performance, and security assessments.
   - *Optimization*: Enhance database performance through techniques such as indexing, query optimization, and configuration adjustments.
5. **Maintenance**:

- *Monitor and Tune*: Continuously monitor database performance and make necessary adjustments to accommodate new demands or integrate additional features.
- *Backup and Recovery*: Establish a routine backup schedule and formulate data recovery procedures to mitigate data loss.
- *Security Updates*: Implement security patches and updates to safeguard against vulnerabilities.

6. **Documentation**:
    - *Documentation*: Produce detailed documentation of the database architecture, operational guidelines, and maintenance protocols to support future development and upkeep.

For the purposes of this Proof of Concept (PoC), only the initial three stages—Requirements Analysis, Design Phase, and Implementation—were essential. Subsequent phases, such as "Testing and Evaluation", are detailed in Chapter 4. Maintenance protocols are not discussed, as this PoC is not intended for direct deployment in a production environment but rather as a replicable model. Comprehensive documentation has been provided to the capstone partner but is not included in this document due to its restricted and confidential nature.

3.2.2 Requirements Analysis

The sponsor's existing tracking system lacks comprehensive functionalities needed to manage daily operations effectively, particularly in terms of alert management for preventing deviations and delays during transit. Consequently, the PoC incorporates the following features:

- **Event-Based Tracking**: Chosen over real-time tracking due to current data reception limitations and the project's focus not being on tracking technology, but rather on the alert management system.
- **Alert Management**: An enhanced tracking system will be established, employing a rule-based engine to calculate potential deviations and classify issues during transit as "validation points" for effective alert generation.

3.2.3 Design Phase

The design phase entailed establishing a database capable of capturing essential information to effectively replicate the sponsor company's current system and enable dynamic data analysis while preserving data integrity. To achieve this, a framework comprising dimension tables and fact tables was developed.

In the realms of data warehousing and business intelligence, dimension tables (DIM tables) and fact tables serve distinct purposes and play crucial roles in data analysis. These tables are typically arranged in a star or snowflake schema, facilitating efficient data querying and reporting:

- **Dimension Tables (DIM Tables)**
    - Purpose: Store descriptive attributes about business entities, such as time periods, products, and customers.
    - Characteristics:
        - *Descriptive*: Include data such as product ID, name, category, and manufacturer.
        - *Stability*: Change infrequently, usually only updated to reflect changes in business descriptions or hierarchies.
        - *Low Volume*: Contain fewer rows than fact tables, capturing only entity characteristics.
        - *Keys*: Linked to fact tables via foreign keys to establish relationships.
- **Fact Tables**
    - Purpose: Record quantitative metrics essential for business process analysis.
    - Characteristics**:**
        - *Numerical*: Predominantly contain numerical data like counts, sums, and other metrics.
        - *High Volume*: Log extensive data amounts, recording each business event or transaction.
        - *Keys*: Feature foreign keys that match primary keys in dimension tables, integrating descriptive data with quantitative measures.
        - *Rapid Changes*: Frequently updated to incorporate new transactional data.

These tables form the structural foundation of the tracking system, allowing for comprehensive storage and analysis of shipment processes. Dimension tables provide contextual and descriptive details about shipments and products, whereas fact tables capture intricate transactional data, including:

- *Planning Information*: Establishes the baseline for shipments, facilitating comparisons with actual events. This encompasses essential data regarding the product/therapy and planned timing and locations.
- *Tracking Process Information*: Details each step of the tracking process, from ground transportation at the origin to air and ground transportation en route to the final destination.

Advanced statistical calculations will be dynamically performed and recorded, based on this structured data. These calculations include:

- *Planned Position Calculation*: Determines planned positions based on the current location and expected transit times for each journey segment.
- *Alerting System*: Defines conditions under which alerts are triggered, such as significant delays or deviations from the planned route.

To visually articulate the relationships between all database components, an Entity-Relationship (ER) diagram was constructed using "Visual Paradigm." This diagram (Figure 13) delineates the database structure in terms of tables, fields, keys, and relational constraints, providing a clear representation of data interactions within the system.

DIM Tables:

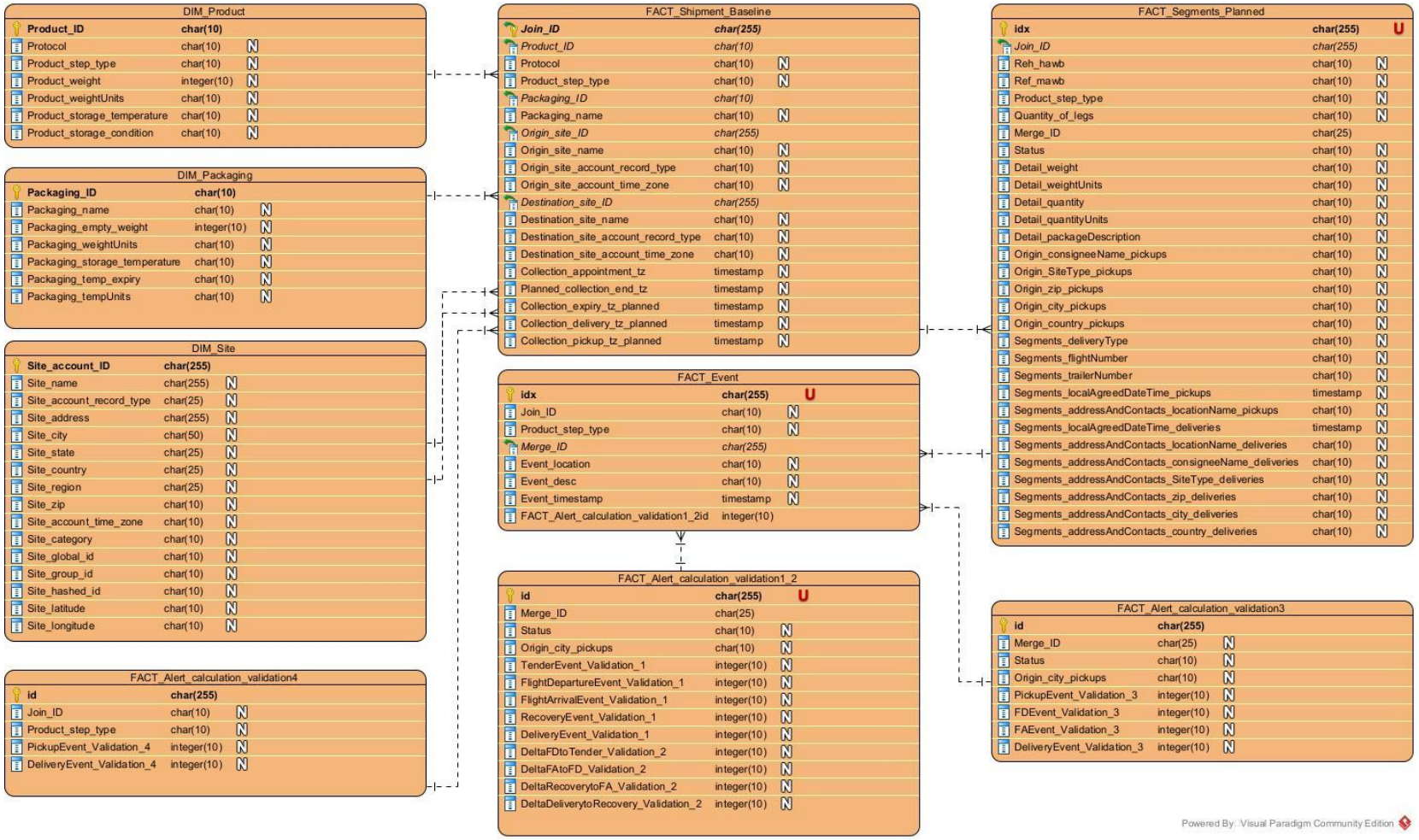
- DIM_Site: Static Table containing all the Sites, from the Apheresis Centers to the Infusion Centers and the Manufacturing Site.
- DIM_Product: Static Table containing all the Product information.
- DIM_Packaging: Static Table containing all the Packaging information.

FACT Tables (populated with Historical Data):

- FACT_Shipment_Baseline: Dynamic Table containing all transactional data. Represent the main information's sent by the sponsor to the Courier to plan the legs of the shipment. Historical data are used to populate this Table. In the Production environment, this table must be filled dynamically for each new process created.
- FACT_Segments_Planned: Dynamic Table containing all the planning information transmitted by the courier to the Sponsor Company. Historical data are used to populate this Table. In the Production environment, this table must be filled dynamically for each new process created.
- FACT_Event: Dynamic Table containing all the events send by the Courier to the Sponsor to follow up each shipment. Historical data are used to populate this Table. In the Production environment, this table must be filled dynamically for each new process created.

Figure 13

Entity-Relationship (ER) of the PoC Database



Note: Represent Entity-Relationship Diagram of the PoC. Own work.

FACT Tables (containing the "Validation Points" dynamically calculated, used to alert the sponsor in case of disruption):

- FACT_Alert_calculation_validation1_2: Table containing the Validation Points 1 & 2:
    - *Validation 1*: Used to measure the Average time from the origin (Pickup) to a certain event.
    - *Validation 2*: Used to measure the Average time of a specific Event.
- FACT_Alert_calculation_validation3: Table containing the Validation Points 3, used to measure the difference between the Real Time vs Planned (by the courier).
- FACT_Alert_calculation_validation4: Table containing the Validation Points 4, used to measure the difference between the Real Time vs Planned (by the sponsor).

3.2.4 Implementation

With the requirements specified and the design architecture established, the construction of the database commenced. This phase involved populating the database with the predefined historical data. For this purpose, "SQLite" was selected as the Database Management System (DBMS), and "SQLAlchemy" was utilized as the SQL toolkit and Object-Relational Mapping (ORM) library for the Python programming language. Programming tasks were executed within "Google Colab".

SQLite

From their homepage (https://www.sqlite.org/about.html), SQLite is recognized for its lightweight, self-contained characteristics, making it an ideal choice for applications that necessitate a straightforward, efficient, and minimally configured database solution. Unlike traditional DBMSs such as MySQL or PostgreSQL, SQLite operates without the need for a separate server process. It interacts directly with disk files, simplifying the data management process. Key features of SQLite include:

- Serverless Architecture: SQLite functions without a dedicated server process, integrating seamlessly with applications that access the database directly from the file system.
- Zero Configuration: Requiring no initial setup or ongoing administration, SQLite is designed for ease of deployment and use.
- Cross-Platform Compatibility: Available on multiple operating systems such as Windows, macOS, Linux, and various embedded systems, SQLite supports a wide range of applications.
- Compact Size: Known for its minimal resource requirements, SQLite is particularly well-suited for devices with limited hardware capabilities, including mobile phones and embedded systems.

SQLAlchemy

SQLAlchemy is not a standalone DBMS but serves as a comprehensive SQL toolkit and ORM library for Python. It enhances database interaction by abstracting complex SQL operations into Python-based objects and operations (SQLAlchemy Homepage, https://www.sqlalchemy.org/). Notable attributes of SQLAlchemy include:

- SQL Toolkit: Offers a robust suite of enterprise-level persistence patterns, ensuring efficient and high-performing database access.
- ORM Layer: Facilitates database operations through Python classes and objects, significantly simplifying the development process by abstracting raw SQL queries.
- Database Agnostic: Compatible with various database systems, including SQLite, PostgreSQL, MySQL, Oracle, and MS SQL Server, SQLAlchemy provides a uniform interface across these platforms.
- Flexibility: Accommodates a broad spectrum of use cases, from simple applications to complex systems, supporting both high-level ORM techniques and direct SQL scripting.

SQLAlchemy serves as a powerful tool that simplifies database management for Python developers, offering both high-level ORM functionalities and low-level SQL capabilities.

3.2.5 Data Population

Finally, the database tables, including *DIM_Site*, *DIM_Product*, *DIM_Packaging*, *FACT_Shipment_Baseline*, *FACT_Segments_Planned*, and *FACT_Event*, are populated extensively using data templates structured in Excel. Dynamic tables such as *FACT_Alert_calculation_validation1_2*, *FACT_Alert_calculation_validation3*, and *FACT_Alert_calculation_validation4* are updated dynamically with each process to ensure real-time accuracy and responsiveness. The Python scripts used for creating and populating these tables are detailed in Appendix A, providing a comprehensive guide to the technical implementation of the database.

3.3 Dynamic Statistical Analysis and "Validation Points"

With the database established, we delved into the operational capabilities of our Proof of Concept (PoC). This section describes the objectives and methodologies behind the validation points, the data analysis and processing conducted on these points, and details the rules these dynamic statistical markers follow to forecast potential disruptions in the CAR-T cell therapy supply chain.

3.3.1 Validations Points – Objective and Calculation

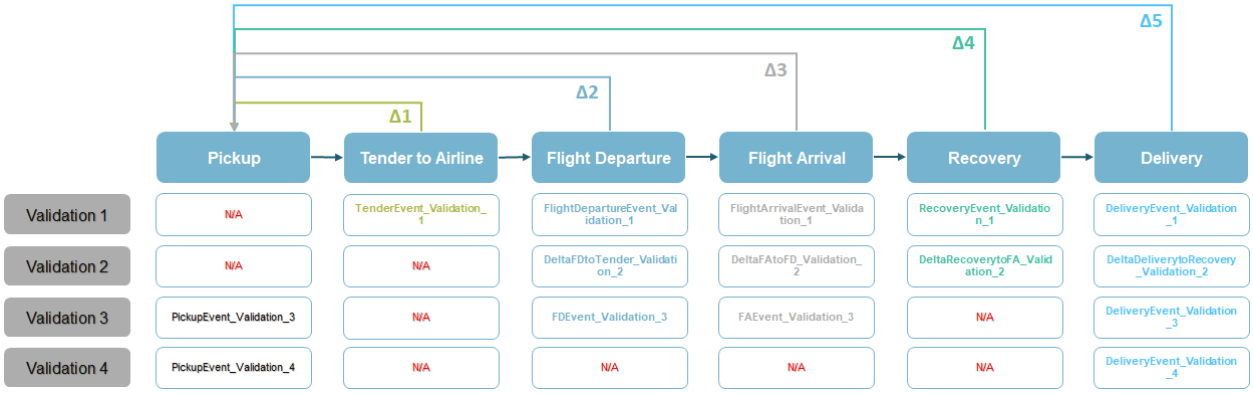
As previously outlined, the tables FACT_Alert_calculation_validation1_2, 3, and 4 encompass four principal validation points:

- *Validation Point 1*: Measures the average time from the origin (Pickup) to a specific event.
- *Validation Point 2*: Calculates the average duration of a specific event.
- *Validation Point 3*: Assesses the difference between the actual time and the planned time (by the courier).
- *Validation Point 4*: Determines the discrepancy between the actual time and the planned time (by the sponsor).

These validation points were designed to identify specific moments when the sponsor should ideally receive event notifications from the courier within a predetermined confidence interval, based on the event's criticality. If no notification is received by this designated time, the system issues an alert to the sponsor's planning team indicating a potential disruption. Conversely, if an event is reported before reaching the validation point threshold, the system acknowledges the milestone and shifts focus to the subsequent event. Figure 14 displays the calculated set of validation points.

Figure 14

Validation Points



Note: Represent the list of the 15 validation points created. The "N/A" are due to a lack of information or redundancy. Own work.

These points were computed across all processes, yielding a total of 15 "checkpoints" to monitor whether the shipment adheres to the original plan or begins to deviate.

Validations 1 and 2 were computed for each actual event received—Validation 1 focusing on the time from the origin to a specific event, and Validation 2 on the duration of that event. Conversely, Validations 3 and 4 compare the actual times of events against their planned times, providing a binary true/false output—for instance, determining whether a pickup occurred on time.

The "N/A" represent points where no calculation were possible. For the *Pickup* event for example, we cannot calculate Vaidation 1 or 2, as pickup is already our point of origin. For the *Tender to Airline* and *Recovery*, as we don't receive the planning information relative to both event, we cannot calculate validation points 3 and 4. Although for the Tender to Airline we could calculate validation 2 but it's value would be equal to validation 1 as it will be the time between the *Tender to Airline* and the *Pickup* events. Finally, for both *Flight Departure* and *Flight Arrival*, as we don't have the information planned by the sponsor referent to both event, we cannot calculate validation 4.

These calculations are executed through queries within the newly established database. The Python code utilized for these calculations is included in Appendix A.

3.3.2 Data Analysis

With the database populated with historical shipment information and calculated validation points, we initiated data analysis. The first step involved extracting data from specific tables for detailed examination. For illustration, we focus on the table FACT_Alert_calculation_validation1_2, containing validation points 1 and 2. The extraction process is structured as follows:

1. **Check the Status "Confirmed"** – Our dataset distinguishes between "Confirmed" and "Job Cancelled" statuses, with no "Ongoing" status present. We concentrate on completed processes to evaluate our PoC and propose the inclusion of an "Ongoing" status in future enhancements in consultation with our sponsor.
2. **Check the kind of product "APH"** – Initially, our database includes shipment data from the Apheresis Center to Make 1 (APH Product) and back to the patient (CDP, which is the final product). This study focuses on the Apheresis leg, which is considered the most critical leg due to the stringent time (< 2 days) and temperature (between 2ºC and 8ºC) constraints.
3. **Check for Potential Negativity** – It is crucial to ensure all validation points are positive. Negative values indicate data transfer errors between the courier and the sponsor, such as events being recorded out of sequence (e.g., "recovery" logged before "Flight Arrival"). These discrepancies,

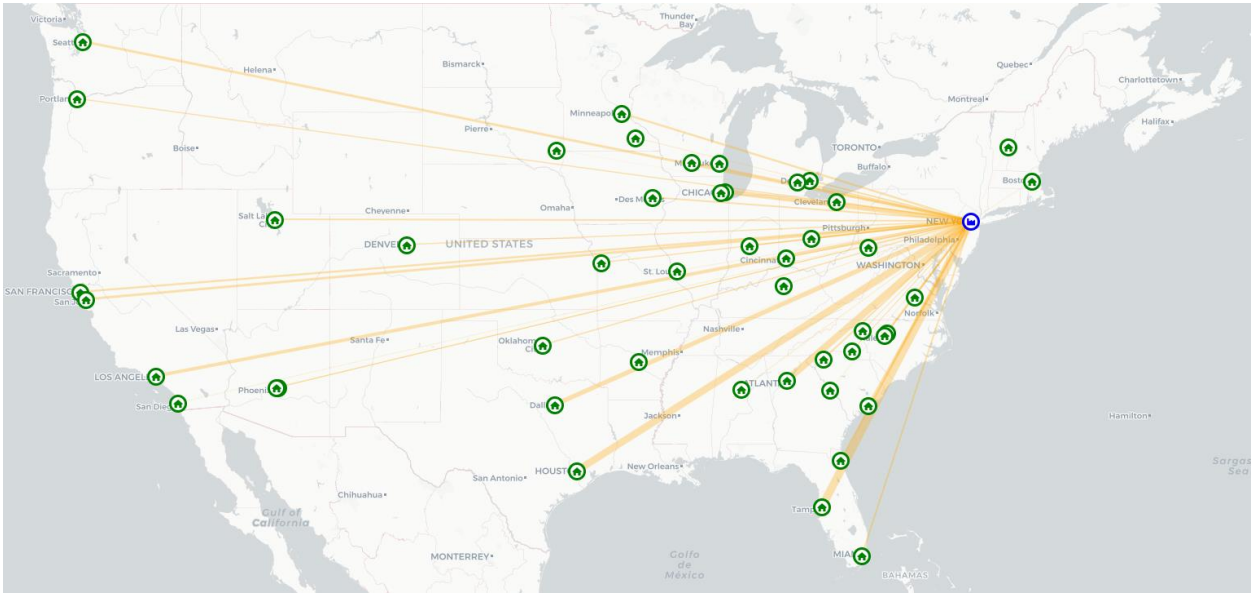
potentially due to manual or systemic errors, are under joint investigation to determine their root causes.

We created this last criterion in our query due to the fact that we encounter this discrepancy in the historical data. To allow more visibility to the capstone partner, we created a second query where we extract all the processes having negative values. With this information, the sponsor planning team is able to investigate with the couriers what has happened and review the process if necessary.

These queries are automated within the database, allowing planners easy access to this information for subsequent analysis. The final step involves clustering shipments by city of origin instead of Apheresis Center (can have several in the same city) to manage data scale without overly segmenting. Figure 15 displays a Folium Map showing the dispersion of processes across different cities (Green point) to the Manufacturing Facility "Make 1" (Blue point), with the width of arcs representing the density of shipments from each city.

Figure 15

Folium Map of the different arcs between the Apheresis Center to the Make1



Note: Represent the many-to-one flows from the Apheresis Center to the Make1. The width of each arc represents the volume of therapies over the year. Own work.

Table 1 presents the density of shipments from each location, highlighting cities with statistically significant process volumes.

Table 1

List of segregated locations and their density

Site_city	Site_supply	Site_city	Site_supply	Site_city	Site_supply	Site_city	Site_supply	Site_city	Site_supply
Location 1	10,1%	Location 11	2,4%	Location 21	1,5%	Location 31	1,1%	Location 41	0,4%
Location 2	8,8%	Location 12	2,4%	Location 22	1,5%	Location 32	0,9%	Location 42	0,4%
Location 3	8,4%	Location 13	2,4%	Location 23	1,5%	Location 33	0,9%	Location 43	0,2%
Location 4	5,9%	Location 14	2,4%	Location 24	1,5%	Location 34	0,9%	Location 44	0,2%
Location 5	5,7%	Location 15	2,2%	Location 25	1,5%	Location 35	0,7%	Location 45	0,2%
Location 6	4,2%	Location 16	2,2%	Location 26	1,3%	Location 36	0,7%	Location 46	0,2%
Location 7	3,3%	Location 17	2,2%	Location 27	1,3%	Location 37	0,7%	Location 47	0,2%
Location 8	3,1%	Location 18	2,0%	Location 28	1,3%	Location 38	0,4%		
Location 9	3,1%	Location 19	2,0%	Location 29	1,1%	Location 39	0,4%		
Location 10	2,6%	Location 20	1,8%	Location 30	1,1%	Location 40	0,4%		

Note: Represent the density of therapies per location throughout the year. The green locations are the ones with a quantity superior to 25-30 therapies per year. Own work.

This scarcity of analyzable data underscores the need for a larger dataset, given the narrow focus of this study on a single product, within one country, from one courier, in a specific year, coupled with the generally low volume of CAR-T treatments in the market.

3.3.3 Central Limit Theorem

We predicated our analysis on the assumption that our data are continuous, implying that our variables can assume any value within a specified range. This contrasts with discrete variables, which adopt distinct, separated values. Further, we assumed that infinite sampling of all continuous datasets converges to a normal distribution, a phenomenon attributed in part to the Central Limit Theorem (CLT).

The Central Limit Theorem is a cornerstone concept in statistics that describes the distribution behavior of the sum (or average) of a large number of random variables, irrespective of the original distribution of these variables. It describes that when a sufficiently large sample size is drawn from any population with a finite level of variance, the mean of the sample will approximate a normal distribution. This normalization holds true regardless of the population distribution's shape. The application of the CLT is predicated on several conditions:

- **Independence**: Each random variable (e.g., individual measurements within the sample) must be independent.
- **Sample Size**: The sample size must be sufficiently large. Although there is no strict rule defining "large," it is commonly accepted that a sample size of 30 or more is adequate for the CLT to apply.
- **Finite Variance**: The population from which samples are drawn must exhibit a finite mean and variance.

Given that these conditions were met in our dataset, we utilized the CLT to assert that:

- The sampling distribution of the sample means trends towards a normal distribution, even if the original data are not normally distributed.
- This predictability (that the sample mean is normally distributed) facilitates further statistical analyses and hypothesis testing that rely on normality.
- The standard deviation of the sampling distribution, known as the standard error, diminishes as the sample size increases.

3.3.4 Utilizing Validation Points to Predict Disruptions

With the necessary data in hand, we began to predict potential disruptions. Assuming our samples across segregated locations are normally distributed, Table 2 presents all relevant statistical data for all locations.

Table 2

Statistical Informations of the Validation points

	count	mean	std	min	25%	50%	75%	max
TenderEvent_Validation_1	457	6.638	7.461	-	3.060	4.740	7.260	60.180
FlightDepartureEvent_Validation_1	457	20.184	12.044	6.360	12.960	17.040	22.980	92.160
FlightArrivalEvent_Validation_1	457	32.906	14.709	12.060	22.680	28.140	41.340	113.940
RecoveryEvent_Validation_1	457	46.098	15.983	17.880	31.200	48.420	56.700	137.160
DeliveryEvent_Validation_1	457	51.824	15.631	20.580	39.600	53.520	61.200	138.960
DeltaFDtoTender_Validation_2	457	13.546	10.056	3.120	7.680	10.680	15.720	88.080
DeltaFAtoFD_Validation_2	457	12.722	7.528	3.360	7.200	9.900	14.760	30.360
DeltaRecoverytoFA_Validation_2	457	13.192	9.942	1.020	5.220	8.580	20.640	49.200
DeltaDeliverytoRecovery_Validation_2	457	5.726	7.267	600	1.800	3.600	6.780	89.280
PickupEvent_Validation_3	457	-	-	-	-	-	-	-
FDEvent_Validation_3	457	1.811	2.140	- 3.540	660	1.140	1.980	13.860
FAEvent_Validation_3	457	248	2.186	- 2.640	- 1.080	- 420	600	11.820
DeliveryEvent_Validation_3	457	5.250	14.849	- 88.200	- 1.500	-	12.600	82.500

Note: Represent the statistical information regarding the main validation points. Own work.

Confidence Interval (CI)

To calculate the confidence interval (CI) for a sample mean under the assumption of normal distribution, the formula used is based on the T-distribution, suitable when the population standard deviation is unknown. The formula is as follows:

$$Confidence\ Interval = \bar{x} \pm t * \left(\frac{s}{\sqrt{n}}\right)$$

where:

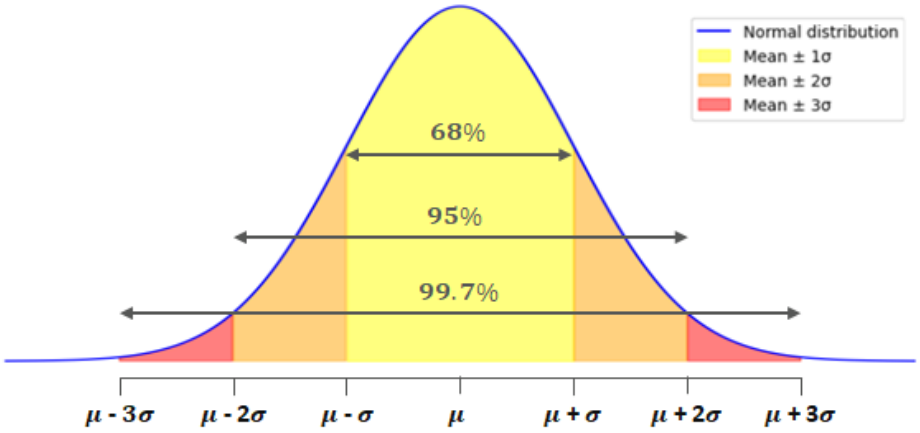
- $\bar{x}$ is the sample mean ("mean" in our table).
- t is the t-score from the t-distribution for the desired confidence level and degrees of freedom $(n-1)$.
- s is the sample standard deviation (std in our table)
- n is the sample size ("count" in our table)
- $\frac{s}{\sqrt{n}}$ is the standard error of the mean.

Choosing the Confidence Level

The selected confidence level, typically 90%, 95%, or 99%, influences the t-score and consequently the width of the confidence interval. A higher confidence level necessitates a broader interval to encompass the true population parameter with greater certainty. Figure 16 shows a representation of the confidence interval on a theoretical normal distribution.

Figure 16

Theoretical representation of the Confidence interval on a Normal Distribution



Note: Represent a randomly generated normal distribution and the confidence interval. Own work.

The calculated confidence intervals provide the Lower and Upper Bounds essential for establishing the rules for each validation level:

$$Lower\ Bound\ CI = \bar{x} - t * \left(\frac{s}{\sqrt{n}}\right)$$

$$Upper\ Bound\ CI = \bar{x} + t * \left(\frac{s}{\sqrt{n}}\right)$$

These bounds are instrumental in defining the thresholds for each validation point, thereby facilitating the detection and management of potential disruptions in the supply chain.

3.3.5 Rules for Validation Points and Alerting

We have formulated operational rules to define the application of upper and lower bounds for Validation Points 1, 2, and 3 to predict and promptly initiate alerts to the sponsor. These rules are essential for preserving the integrity of the supply chain and ensuring timely responses to potential disruptions.

Definition of which CI

Not all events carry the same level of criticality within the process. For instance, a delay in "Flight Departure" poses more significant consequences than a delay in the preceding "Tender to Airline" step. Although these events are interdependent, their impact on the process's overall completion varies. Thus, specific confidence intervals are designated for each event to appropriately size the range around the mean, which will trigger an alert in the event of significant disruptions. The following outlines the CI applications for main events:

Table 3

Definition of the Confidence Intervals

	Validation 1	Validation 2	Validation 3
Pickup	N/A	N/A	1σ Upper Bound = 68%
Tender to Airline	2σ Upper Bound = 95%	N/A	N/A
Flight Departure	1σ Upper Bound = 68%	1σ Upper Bound = 68%	1σ Upper Bound = 68%
Flight Arrival	1σ Upper Bound = 68%	1σ Upper Bound = 68%	1σ Upper Bound = 68%
Recovery	2σ Upper Bound = 95%	2σ Upper Bound = 95%	N/A
Delivery	1σ Upper Bound = 68%	1σ Upper Bound = 68%	1σ Upper Bound = 68%

Note: Represent the Confidence Interval CI defined for each Validation Points. Own work.

Below are the main comments for each main event:

1. *Pickup*: This event utilizes Validation Point 3, which compares the actual pickup time to the planned time by the courier. We employ a **1σ upper bound** to allow a margin for the information transmission from the courier to the sponsor.
2. *Tender to Airline*: This event is monitored using Validation Point 1. Given the variability of this event due to numerous exogenous factors (e.g., arrival times, desk crowding), a **2σ upper bound** is utilized as the predictor.

3. *Flight Departure*: Critical for timely follow-up, a **1σ upper bound** is employed for all three validation points to quickly alert and respond to any delays, cancellations, or reroutings.
4. *Flight Arrival*: Dependent on the prior event, disruptions during flight (e.g., rerouting or landing delays) are not visible through geolocation as no data is transmitted in-flight. Therefore, a **1σ upper bound** for all three validation points is used for enhanced visibility.
5. *Recovery*: Following the protocols set for "Tender to Airline," and with confirmation that the material was on-boarded, a **2σ upper bound** suffices for monitoring this event.
6. *Delivery*: With all three validation points available for this final event, a **1σ upper bound** is deemed most appropriate.

Definition of the treshold

When multiple validation points apply to the same event, it is crucial to define a threshold that considers variations in starting points and standard deviations, which could affect accuracy:

- **Validation Point 1** computes the CI from the pickup event onward, not recalculating for subsequent events unless there is a significant delay.
- **Validation Point 2** ecalculates the CI after each event, using the timestamp from the previous event and the historical duration data. This can lead to discrepancies if an event is confirmed much earlier than planned.
- **Validation Point 3** is more straightforward, calculating the CI around a pre-planned timestamp.

For events with multiple applicable validation points, the chosen threshold is the lowest upper bound value that exceeds the originally planned value. For events like *Flight Departure*, *Flight Arrival*, and *Delivery* this method ensures responsiveness. For *Recovery*, historical averages of the event time are used instead of planned values. The table below presents the final rules.

Table 4

List of Rules

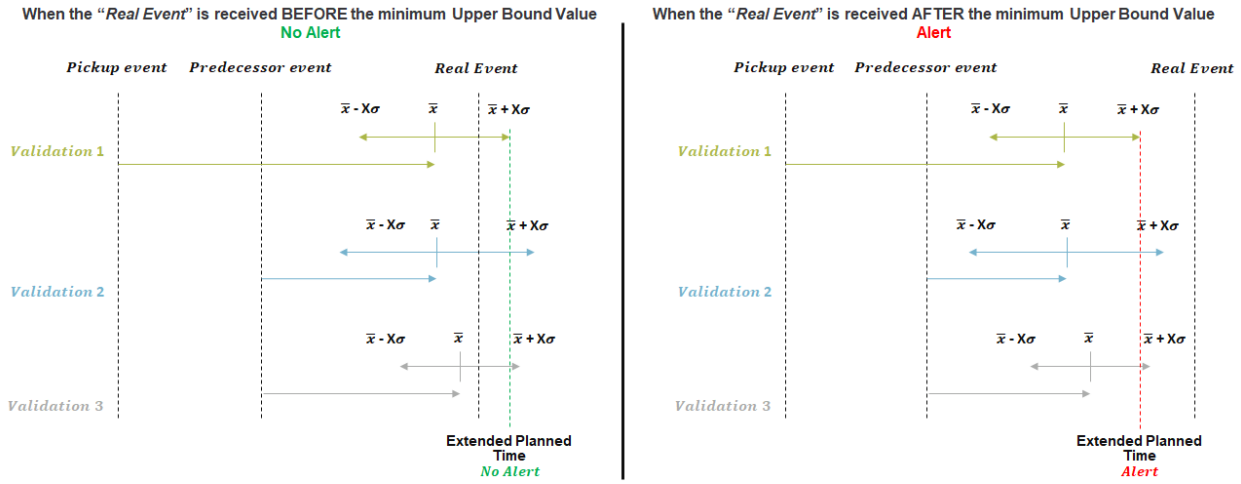
	Validation 1	Validation 2	Validation 3	Treshold
Pickup	N/A	N/A	1σ Upper Bound = 68%	Upper Bound of the only validation point calculated
Tender to Airline	2σ Upper Bound = 95%	N/A	N/A	Upper Bound of the only validation point calculated
Flight Departure	1σ Upper Bound = 68%	1σ Upper Bound = 68%	1σ Upper Bound = 68%	First Upper Bound after the last event timestamp.
Flight Arrival	1σ Upper Bound = 68%	1σ Upper Bound = 68%	1σ Upper Bound = 68%	First Upper Bound after the last event timestamp.
Recovery	2σ Upper Bound = 95%	2σ Upper Bound = 95%	N/A	First Upper Bound after the last event timestamp.
Delivery	1σ Upper Bound = 68%	1σ Upper Bound = 68%	1σ Upper Bound = 68%	First Upper Bound after the last event timestamp.

Note: Represent the list of the final Rules to be applied in the PoC for each Validation Points. Own work.

Also, Figure 17 helps visualize the range of each validation point's interval. This graph demonstrates the overlap and interactions between the intervals of Validation Points 1, 2, and 3, illustrating how these interactions impact the timing of alerts. This visualization aids in understanding the sequential dependency and priority among the validation points.

Figure 17

Visual Representation of the Rules for Validation Points 1, 2 and 3



Note: Represent the Rules used to send an alert based on the values of Validation Points 1, 2 and 3. The left side represent a "No Alert" situation, where the real Event occured before the minimum Next Upper Bound, then still in the Confidence Interval. The Right side of the figure represent an "Alert" situation. In this case, no Real Event is received until the first upper bound being reached. In this case, an Alert would have been send. Own work.

These rules facilitate a layered approach to monitoring and alerting, using the interplay between sequential validation points to optimize the accuracy and timeliness of alerts. The strategy ensures that potential disruptions are flagged as soon as deviations from expected timelines are detected, thereby enabling proactive management of the supply chain.

4 RESULTS

In this chapter, we detail the results derived from applying the Proof of Concept (PoC) to a practical scenario. This will include an illustrative example to contextualize the outcomes and enumerate the primary benefits of this project for our sponsor company.

4.1 Example

For demonstrative purposes, we selected a real case involving a shipment from "Location 1," which is the most active site in our database. The calculation of the validation points for this specific location is presented in Table 5.

Table 5

Statistical Informations for "Location 1"

	count	mean	std	min	25%	50%	75%	max	Confidence_Level	Lower_Bound	Upper_Bound
TenderEvent_Validation_1	46	5.086	2.020	1.800	3.600	4.800	5.700	11.400	0.95	5.082	5.090
FlightDepartureEvent_Validation_1	46	16.145	8.983	7.500	10.620	14.760	18.000	57.960	0.68	16.134	16.156
FlightArrivalEvent_Validation_1	46	25.106	8.901	15.300	19.965	23.490	27.360	65.940	0.68	25.095	25.117
RecoveryEvent_Validation_1	46	36.950	13.924	24.000	27.600	30.600	47.745	76.200	0.95	36.925	36.975
DeliveryEvent_Validation_1	46	39.395	13.607	25.800	30.135	33.720	50.355	78.000	0.68	39.378	39.412
DeltaFDtoTender_Validation_2	46	11.060	8.445	4.200	6.540	8.370	12.900	46.560	0.68	11.049	11.071
DeltaFAtoFD_Validation_2	46	8.961	946	7.560	8.295	8.730	9.480	11.340	0.68	8.960	8.962
DeltaRecoverytoFA_Validation_2	46	11.843	9.471	1.980	6.495	7.980	11.025	38.520	0.95	11.826	11.860
DeltaDeliverytoRecovery_Validation_2	46	2.446	1.239	600	1.665	1.890	2.985	6.000	0.68	2.444	2.448
PickupEvent_Validation_3	46	-	-	-	-	-	-	-	0.68	-	-
FDEvent_Validation_3	46	1.509	1.888	180	600	720	1.635	9.600	0.68	1.507	1.511
FAEvent_Validation_3	46	38	1.898	- 1.860	- 1.005	- 390	300	8.100	0.68	36	40
DeliveryEvent_Validation_3	46	3.865	19.764	-88.200	- 3.000	- 450	2.520	46.800	0.68	3.840	3.890

Note: Represent the list of all the newly calculated validation points for "Location 1". Own work.

Figure 18 provided compare the calculated validation points during the shipment to a dynamically recalculated baseline, which also serves as our comparison standard. The "Real" section displays the actual shipment times in minutes. The "Validation Points" section illustrates the historical calculation of these delays, and the "Recalculated Time" shows the original timestamps sent during planning alongside the new points recalculated based on the validation point values.

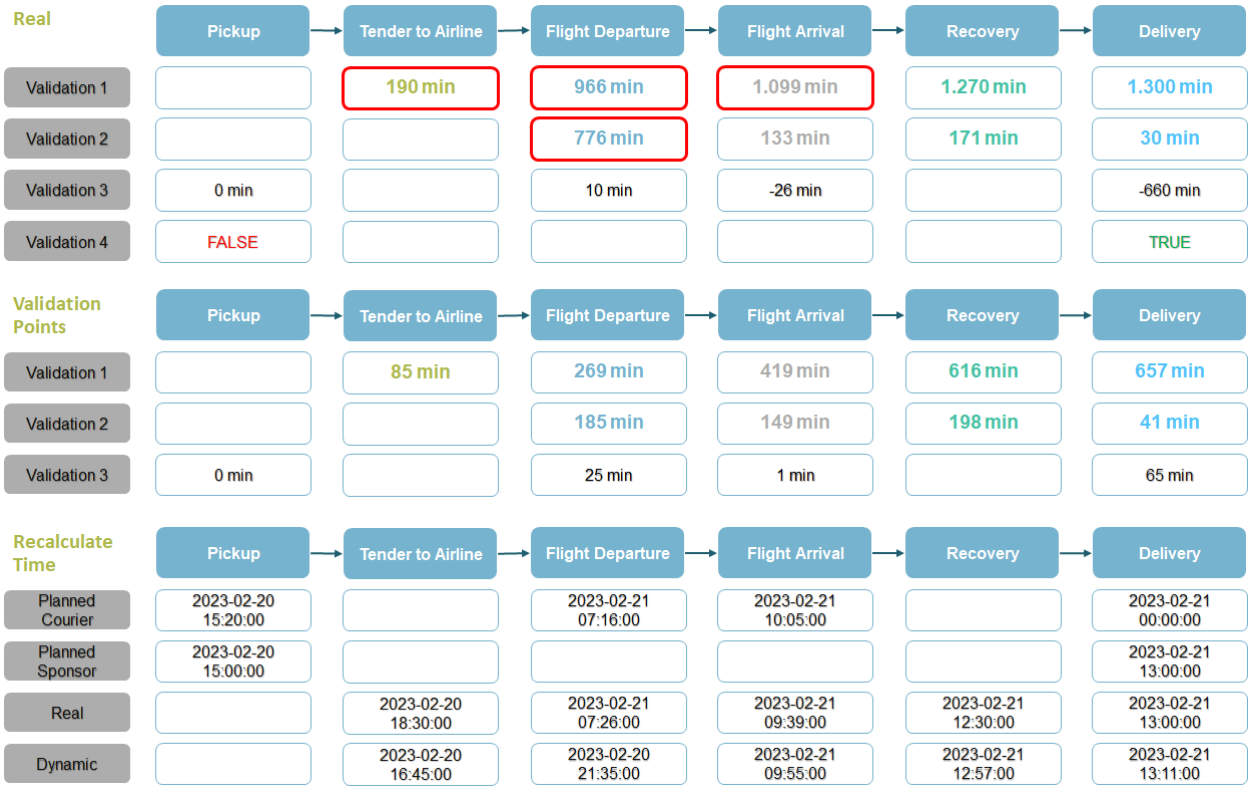
Pickup Event

Tracing the sequence of events from the company's perspective, we observe that the Validation Point 3 for the *Pickup* event indicates an on-time pickup as planned by Courier A. Historically, this event has never been delayed, aligning with sponsor records that often adjust delivery times in their systems to reflect actual timings, explaining the zero discrepancy. However, an examination of the originally planned

time by the Sponsor reveals a 20-minute delay, not triggering an alert due to the loss of the original planned time by the courier.

Figure 18

Results from a historical Shipment



Real	Pickup	Tender to Airline	Flight Departure	Flight Arrival	Recovery	Delivery
Validation 1		190 min	966 min	1.099 min	1.270 min	1.300 min
Validation 2			776 min	133 min	171 min	30 min
Validation 3	0 min		10 min	-26 min		-660 min
Validation 4	FALSE					TRUE

Validation Points	Pickup	Tender to Airline	Flight Departure	Flight Arrival	Recovery	Delivery
Validation 1		85 min	269 min	419 min	616 min	657 min
Validation 2			185 min	149 min	198 min	41 min
Validation 3	0 min		25 min	1 min		65 min

Recalculate Time	Pickup	Tender to Airline	Flight Departure	Flight Arrival	Recovery	Delivery
Planned Courier	2023-02-20 15:20:00		2023-02-21 07:16:00	2023-02-21 10:05:00		2023-02-21 00:00:00
Planned Sponsor	2023-02-20 15:00:00					2023-02-21 13:00:00
Real		2023-02-20 18:30:00	2023-02-21 07:26:00	2023-02-21 09:39:00	2023-02-21 12:30:00	2023-02-21 13:00:00
Dynamic		2023-02-20 16:45:00	2023-02-20 21:35:00	2023-02-21 09:55:00	2023-02-21 12:57:00	2023-02-21 13:11:00

Note: Represent the calculation done for a real historical process. The first quadrant “Real” show the real time calculated for each validation points. They are compared to the dynamically calculated validation points, in the quadrant of the same name. Finally, we show the timestamp considered for each calculation. Own work.

Tender to Airline Event

For the *Tender to Airline* event, historical data indicates it typically takes 85 minutes from material collection to delivery to the airline. However, in this instance, it took 190 minutes, with the event received two hours later than expected. An alert would have been triggered at the calculated alert time of "2023-02-20 16:45:00", based on the original *Pickup* Timestamp due to the absence of event confirmation.

Flight Departure Event

Continuing to the *Flight Departure*, we analyze:

- **Validation 1**: Theoretical point from *Pickup* to *Flight Departure* is 269 minutes, leading to the timestamp "2023-02-20 21:49:00".
- **Validation 2**: Normally, the duration between this and the preceding event is about 185 minutes, recalculating the timestamp to "2023-02-20 21:35:00".
- **Validation 3**: Historically, flights on this route are delayed by approximately 25 minutes. Due to the courier missing the original flight and booking the next available one, the new planned timestamp is "2023-02-21 07:16:00", with a threshold of "2023-02-21 07:41:00" based on Validation 3.

The lowest threshold rule specifies using the Validation 2 timestamp "2023-02-20 21:35:00" which is also superior to the timestamp received from the previous event, as the alert trigger. Unfortunately, in this process, the flight departed significantly later at "07:26:00" on the 21st.

Flight Arrival Event

For the *Flight Arrival* event, the delays caused by replanning are evident, with the recalculated validation points indicating significant deviations from historical norms. If no information had been received by "2023-02-21 09:55:00", an alert would have been triggered. The actual event was reported at "09:39:00", thus no alert was issued.

Recovery Event

Proceeding to the *Recovery* event, the recalculated timestamps for Validation Points 1 and 2 were set for "2023-02-21 01:36:00" and "2023-02-21 12:57:00", respectively. Since Validation Point 1's timestamp predates the latest event, Validation Point 2's timestamp is adopted as the new threshold. The actual event occurred at "2023-02-21 12:30:00", which is 27 minutes prior to our established threshold, thus no alert was warranted.

Delivery Event

Transitioning to the final *Delivery* event, the recalculated timestamps for Validation Points 1, 2, and 3 were "2023-02-21 02:17:00", "2023-02-21 13:11:00", and "2023-02-22 01:05:00". The timestamp for the last validation point was derived by adding a 65-minute historical upper bound of delays to the originally planned timestamp of "2023-02-21 00:00:00" by the courier. As the last event received was at "2023-02-21 12:30:00", the new threshold was established at "2023-02-21 13:11:00". The final event was recorded at "2023-02-21 13:00:00", 11 minutes before the threshold, hence no alert was issued, and the

product arrived before the adjusted planned time. Additionally, the shipment was delivered precisely at the time it was replanned during transit, indicating no delay according to Validation Point 4.

Conclusion

Upon reviewing this case with the sponsor company, it was noted that although the company had been informed of the delays, the notifications were only received post facto. Furthermore, the Key Performance Indicators (KPIs) used to assess the courier's performance recorded the status as "on-time", without any noted reason for delay. Despite the shipment ultimately arriving on time, it is recommended that a field be designated to report any delays during transit, irrespective of their impact on the final delivery time. This data could be instrumental in future assessments of the reliability of flights departing from this specific airport, particularly in terms of their punctuality.

In conclusion, this example demonstrates that while several timestamps were adjusted due to the initial delays, the utilization of various types of validation points—particularly Validation Point 2—enables effective monitoring and management of shipments. This approach not only accommodates discrepancies between initially planned and actual event timings but also aids in predicting the receipt of subsequent timestamps, thereby enhancing control over the shipment process.

4.2 Advantage for the company

Through several example analyses, it became apparent that many shipments, although arriving on time, experienced unreported delays. The PoC provides significant benefits to the sponsor, including:

- **Enhanced Visibility**: The sponsor gains improved oversight of ongoing processes through timely alerts.
- **Analytical Opportunities**: New data generated allows for detailed analysis of each lane, assessing the probability of event occurrences.
- **Centralized Information**: Simplifies investigations into discrepancies.
- **Increased Resilience**: Better visibility of courier-sent information enhances operational resilience.

5 DISCUSSION

This chapter aims to reflect on the prospective implementation of this solution in a production environment, acknowledging its current limitations, and to consider the broader implications of the findings presented.

5.1 Implementation in Production

As previously detailed in "Chapter 2 – State of the Art," the pharmaceutical sector is subject to stringent regulations, which also govern the integration of systems within companies. All systems integrations must comply with specific regulatory standards (e.g., CFR Part 21) and adhere to rigorous internal cybersecurity policies. Consequently, from the project's inception, it was established that implementation in production was beyond the scope of this project, necessitating collaboration across various teams and compliance with specialized requirements.

5.2 Current Limitations

Throughout the project, and as underscored in this document, we have encountered significant limitations concerning both the quantity and quality of data.

5.2.1 Quantity of Data

The primary issues identified regarding data quantity include:

- **Total Quantity of Processes/Year**: The data pertaining to the number of shipments per year is relatively sparse, primarily due to the specialized nature of the therapy involved, which is expensive and not yet broadly adopted. As the prevalence of this therapy increases, it is expected that the volume of data available for analysis will also grow, providing a more robust dataset.
- **Segregation**: The project demonstrated a need to segregate data according to various criteria such as origin, destination, and number of legs, which has significantly increased the data requirements. For instance, only 10% of origins in the analyzed dataset were considered statistically significant (with more than 25-30 shipments, as suggested by the Central Limit Theorem). While additional historical data from the company could be incorporated into this PoC, it is clear that more granular segregation exponentially increases the volume of data required.

5.2.2 Quality of Data

A critical concern highlighted is the absence of an audit trail for planning data. This deficiency hinders the PoC's ability to predict future events accurately, as original timestamps are often overwritten during the process, thereby directly impacting the sponsor's reliance on data integrity. The current practice of tracking changes through email threads to verify the authenticity of stored data is inadequate. In some instances within our project, this issue has led to the commingling of altered and authentic data,

affecting the accuracy of calculated validation points. An incorrect validation point could result in an unnecessarily broad confidence interval when a narrower one might be more appropriate.

Furthermore, the project uncovered discrepancies in the sequence of event data received in some processes. This issue is currently under investigation in collaboration with one of the couriers to ascertain whether these discrepancies are systemic or procedural.

5.3 Recommendations

To enhance the functionality of this PoC when implemented in a production setting, it is recommended to add an "ongoing" status to the current list of statuses. This addition would help track and analyze processes in real-time, providing a more dynamic and responsive system.

The discussions outlined in this chapter provide a comprehensive understanding of the challenges faced by the company in terms of data quality and quantity, as well as the quality of service provided by specialty couriers. These insights are crucial for refining the solution and ensuring its successful integration into production environments, aligning with regulatory standards and the company's operational needs.

6 CONCLUSION

In the realm of supply chain management, visibility and resilience are paramount, particularly when the stakes involve patient lives. In such high-stakes environments, 99% accuracy is insufficient; the pharmaceutical industry strives for absolute perfection. Despite couriers generally delivering shipments on time, there is a conspicuous deficiency in visibility and resilience throughout the process. This shortfall undermines confidence in the process and, by extension, the third-party logistics (3PL) providers involved.

This Proof of Concept (PoC) addresses these challenges by recalculating expected delays between events based on historical data and alerting the sponsor to potential delays as they occur. It has not only facilitated the sponsor in identifying risks in real-time but has also prompted a review of their processes, highlighting numerous potential enhancements in both procedures and data recording.

However, the project also underscored the difficulty in amassing sufficient data to accurately predict these points. Given the highly specialized and low-volume nature of the therapy involved, accumulating adequate data across all origin locations could take years. This challenge could be mitigated by a significant increase in demand or by expanding the scope to include other products and extending the analysis over a longer timeframe.

With an enriched dataset, this PoC could unlock new capabilities, such as integrating real-time external services to access flight information and status updates. This enhancement would not only improve the accuracy of future event predictions but also enable the calculation of alternative routing scenarios. By continuously monitoring and evaluating the current status of shipment routings—taking into account traffic conditions, weather forecasts, transit times, and flight statuses—the system could proactively adapt routes in response to changing conditions or unexpected events. Such adaptive routing strategies could be further refined through the application of advanced machine learning tools, which leverage historical data to optimize routing decisions dynamically.

Moreover, this situation accentuates the critical need for robust mechanisms such as audit trails, version histories, or change logs to ensure data integrity and traceability. These tools are essential for maintaining the accuracy and reliability of the data, which in turn supports the resilience and responsiveness of the supply chain.

In conclusion, while the PoC has demonstrated significant potential to enhance supply chain visibility and resilience, its effectiveness is contingent on the availability of comprehensive, high-quality data. As the project moves forward, expanding the data pool and integrating sophisticated analytical tools will be crucial in realizing the full potential of this initiative, ensuring that every shipment not only arrives on time but is also traceably managed every step of the way.

7 REFERENCES

Chen, Y.J., Abila, B., & Kamel, Y.M. (21 January 2023). CAR-T: What Is Next? *National Library of Medicine*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9913679/#:~:text=Kymriah%20(tisagenlecleucel)%20and%20Yescarta%20(,diffuse%20large%20B%2Dcell%20lymphoma

European Medicines Agency (EMA). (2018). *First two CAR-T cell medicines recommended for approval in the European Union*. https://www.ema.europa.eu/en/news/first-two-car-t-cell-medicines-recommended-approval-european-union

European Medicines Agency (EMA). (2021). *Interoperability of track and trace systems: key to public health protection*. https://www.ema.europa.eu/en/news/interoperability-track-trace-systems-key-public-health-protection

Food and Drug Administration (FDA). (2015). *Kymriah, Approved Risk Evaluation and Mitigation Strategies (REMS)*. https://www.accessdata.fda.gov/scripts/cder/rems/index.cfm?event=indvremsdetails.page&rems=368

Food and Drug Administration (FDA). (2015). *Yescarta, Approved Risk Evaluation and Mitigation Strategies (REMS)*. https://www.accessdata.fda.gov/scripts/cder/rems/index.cfm?event=IndvRemsDetails.page&REMS=375

Gerhkoff, I. (2016). *Shaping the future of airline disruption management (IROPS)*. Amadeus. https://amadeus.com/documents/en/airlines/white-paper/shaping-the-future-of-airline-disruption-management.pdf

Hanley, P. J., Bersenev, A., & Gustafson, M. P. (2022). Delivering externally manufactured cell and gene therapy products to patients: Perspectives from the academic center experience. *Cytotherapy, 24(1), 16–18*. https://doi.org/10.1016/j.jcyt.2021.09.010

Harrison, R.P., Zylberberg E., Ellison, S., & Levine, B.L. (2019). Chimeric antigen receptor – T cell therapy manufacturing: modelling the effect of offshore production on aggregate cost of goods. *Cytotherapy, 21, 224-233*. https://pubmed.ncbi.nlm.nih.gov/30770285/

Hofman, W. (2014). Control Tower Architecture for Multi – and Synchromodal Logistics with Real Time Data. *LS 2014 – 5th International Conference – Information Systems, Logistics and Supply Chain*. https://www.researchgate.net/publication/274192485_Control_Tower_Architecture_for_Multi_-_and_Synchromodal_Logistics_with_Real_Time_Data

Jørgensena, J., Hannab, E., & Kefalasa, P. (2020). Outcomes-based reimbursement for gene therapies in practice: the experience of recently launched CAR-T cell therapies in major European countries. *Journal of Market Access & Health Policy 2020, VOL. 8*. https://doi.org/10.1080/20016689.2020.1715536

Kaiser, A.D., Assenmacher, M., Schröder, B., Meyer, M., Orentas, R., Bethke, U., & Dropulic, B. (2015). Towards a commercial process for the manufacture of genetically modified T cells for therapy. *Cancer Gene Therapy 22, 72–78*. https://www.nature.com/articles/cgt201478

Kumar, B., & Gomez P. (2023). *Procurement Control Tower: Proof of Concept through Machine Learning and Natural Language Processing*. [Graduate capstone, Massachusetts Institute of Technology]. Supply Chain Management. http://dspace.mit.edu/handle/1721.1/7582

Levine, B.L., Miskin, J., Wonnacott, K., & Keir, C. (2017). Global manufacturing of CAR T cell therapy. *Molecular Therapy - Methods & Clinical Development, 4, 92–101*. https://doi.org/10.1016/j.omtm.2016.12.006

Meacle, F., Salkin, J., Rice, M., & Harris, I. (2016). Key considerations of cell and gene therapy cold chain logistics. *Cell and Gene Therapy Insights, 2(2), 223–236*. https://doi.org/10.18609/cgti.2016.025

Patil, S. (2020). *Supply chain control tower: orchestrate and drive visibility and valuable automated insights*. Deloitte consulting. Supply Chain Control Tower | Deloitte US. https://www2.deloitte.com/content/dam/Deloitte/us/Documents/process-and-operations/us-supply-chain-control-tower-orchestrate-and-drive-visibility-and-valuable-automated-insights.pdf

Papathanasiou, M.M., Stamatis, C., Lakelin, M., Farid, S., & Titchener-Hooker, N. (2020). Autologous CAR T-cell therapies supply chain: challenges and opportunities? *Cancer Gene Therapy 27, 799–809*. https://www.nature.com/articles/s41417-019-0157-z

Rutherford, C., Barry, J., Campbell, J., & Turner, M. (2017). The Importance of Understanding & Designing Cellular Therapy Supply Chains. *Cell and Gene Therapy Insights, 3(10), 873-889*. https://doi.org/10.18609/cgti.2017.087

Shah, N. (2004). Pharmaceutical supply chains: key issues and strategies for optimisation. *Computers & chemical engineering. 28(6-7) 929–941*. https://www.sciencedirect.com/science/article/-pii/S0098135403002333?via%3Dihub

Shekarian, M., & Mellat Parast, M. (2021). *An Integrative approach to supply chain disruption risk and resilience management: A literature review*. International Journal of Logistics Research and Applications, 24(5), 427–455. https://doi.org/10.1080/13675567.2020.1763935

Teng, C.W., Foley, L., O'Neill, P., & Hicks, C. (2013). An analysis of supply chain strategies in the regenerative medicine industry - Implications for future development. *International Journal of Production Economics, 149, 211-225*. https://www.sciencedirect.com/science/article/-pii/S0925527313002752

U.S. Department of Health and Human Services, Food and Drug Administration (FDA), Center for Drug Evaluation and Research (CDER), Center for Biologics Evaluation and Research (CBER) & Office of Regulatory Affairs (ORA). (2022). *Verification Systems Under the Drug Supply Chain Security Act for Certain Prescription Drugs*. https://www.fda.gov/regulatory-information/search-fda-guidance-documents/verification-systems-under-drug-supply-chain-security-act-certain-prescription-drugs

Yang, X. (2006). *Choosing Transportation Alternatives for Highly Perishable Goods: A Case Study on Nuclear Medicine*. [Graduate theses, Massachusetts Institute of Technology]. Engineering Systems Division. http://dspace.mit.edu/handle/1721.1/7582

8 APPENDIX A

8.1 Creation of the Tables

```
# Define the DIM_Site Table
class DIM_Site(Base):
    __tablename__ = "DIM_Site"
    Site_account_ID = Column("Site_account_ID", String, primary_key = True)
    Site_name = Column("Site_name", String)
    Site_account_record_type = Column("Site_account_record_type", String)
    Site_address = Column("Site_address", String)
    Site_city = Column("Site_city", String)
    Site_state = Column("Site_state", String)
    Site_country = Column("Site_country", String)
    Site_region = Column("Site_region", String)
    Site_zip = Column("Site_zip", String)
    Site_account_time_zone = Column("Site_account_time_zone", String)
    Site_category = Column("Site_category", String)
    Site_global_ID = Column("Site_global_ID", Integer)
    Site_group_ID = Column("Site_group_ID", String)
    Site_hashed_ID = Column("Site_hashed_ID", String)
    Site_latitude = Column("Site_latitude", String)
    Site_longitude = Column("Site_longitude", String)
    def __init__(self, Site_account_ID, Site_name, Site_account_record_type, Site_address,
Site_city, Site_state, Site_country, Site_region, Site_zip, Site_account_time_zone,
Site_category, Site_global_ID, Site_group_ID, Site_hashed_ID, Site_latitude, Site_longitude):
        self.Site_account_ID = Site_account_ID
        self.Site_name = Site_name
        self.Site_account_record_type = Site_account_record_type
        self.Site_address = Site_address
        self.Site_city = Site_city
        self.Site_state = Site_state
        self.Site_country = Site_country
        self.Site_region = Site_region
        self.Site_zip = Site_zip
        self.Site_account_time_zone = Site_account_time_zone
        self.Site_category = Site_category
        self.Site_global_ID = Site_global_ID
        self.Site_group_ID = Site_group_ID
        self.Site_hashed_ID = Site_hashed_ID
        self.Site_latitude = Site_latitude
        self.Site_longitude = Site_longitude
```

```
# Define the DIM_Product Table
class DIM_Product(Base):
    __tablename__ = "DIM_Product"
    Product_ID = Column("Product_ID", String, primary_key = True)
    Protocol = Column("Protocol", String)
    Product_step_type = Column("Product_step_type", String)
    Product_weight = Column("Product_weight", Integer)
    Product_weightUnits = Column("Product_weightUnits", String)
    Product_storage_temperature = Column("Product_storage_temperature", String)
    Product_storage_condition = Column("Product_storage_condition", String)
    def __init__(self, Product_ID, Protocol, Product_step_type, Product_weight,
Product_weightUnits, Product_storage_temperature, Product_storage_condition):
        self.Product_ID = Product_ID
        self.Protocol = Protocol
        self.Product_step_type = Product_step_type
        self.Product_weight = Product_weight
        self.Product_weightUnits = Product_weightUnits
        self.Product_storage_temperature = Product_storage_temperature
        self.Product_storage_condition = Product_storage_condition

# Define the DIM_Packaging Table
class DIM_Packaging(Base):
    __tablename__ = "DIM_Packaging"
    Packaging_ID = Column("Packaging_ID", String, primary_key = True)
    Packaging_name = Column("Packaging_name", String)
    Packaging_empty_weight = Column("Packaging_empty_weight", Integer)
    Packaging_weightUnits = Column("Packaging_weightUnits", String)
    Packaging_storage_temperature = Column("Packaging_storage_temperature", String)
    Packaging_temp_expiry = Column("Packaging_temp_expiry", String)
    Packaging_tempUnits = Column("Packaging_tempUnits", String)
    def __init__(self, Packaging_ID, Packaging_name, Packaging_empty_weight,
Packaging_weightUnits, Packaging_storage_temperature, Packaging_temp_expiry,
Packaging_tempUnits):
        self.Packaging_ID = Packaging_ID
        self.Packaging_name = Packaging_name
        self.Packaging_empty_weight = Packaging_empty_weight
        self.Packaging_weightUnits = Packaging_weightUnits
        self.Packaging_storage_temperature = Packaging_storage_temperature
        self.Packaging_temp_expiry = Packaging_temp_expiry
        self.Packaging_tempUnits = Packaging_tempUnits

# Define the FACT_Shipment_Baseline Table (Table created by the Sponsor Company for the Courier)
class FACT_Shipment_Baseline(Base):
    __tablename__ = "FACT_Shipment_Baseline"
```

```
    Join_ID = Column("Join_ID", String, primary_key = True)
    Product_ID = Column("Product_ID", String, ForeignKey("DIM_Product.Product_ID"))
    Protocol = Column("Protocol", String, ForeignKey("DIM_Product.Protocol"))
    Product_step_type = Column("Product_step_type", String,
ForeignKey("DIM_Product.Product_step_type"))
    Packaging_ID = Column("Packaging_ID", String, ForeignKey("DIM_Packaging.Packaging_ID"))
    Packaging_name = Column("Packaging_name", String, ForeignKey("DIM_Packaging.Packaging_name"))
    Origin_site_ID = Column("Origin_site_ID", String, ForeignKey("DIM_Site.Site_account_ID"))
    Origin_site_name = Column("Origin_site_name", String, ForeignKey("DIM_Site.Site_name"))
    Origin_site_account_record_type = Column("Origin_site_account_record_type", String,
ForeignKey("DIM_Site.Site_account_record_type"))
    Origin_site_account_time_zone = Column("Origin_site_account_time_zone", String,
ForeignKey("DIM_Site.Site_account_time_zone"))
    Destination_site_ID = Column("Destination_site_ID", String,
ForeignKey("DIM_Site.Site_account_ID"))
    Destination_site_name = Column("Destination_site_name", String,
ForeignKey("DIM_Site.Site_name"))
    Destination_site_account_record_type = Column("Destination_site_account_record_type", String,
ForeignKey("DIM_Site.Site_account_record_type"))
    Destination_site_account_time_zone = Column("o  Destination_site_account_time_zone ", String,
ForeignKey("DIM_Site.Site_account_time_zone"))
    Collection_appointment_tz = Column("Collection_appointment_tz", DateTime)
    Planned_collection_end_tz = Column("Planned_collection_end_tz", DateTime)
    Collection_expiry_tz_planned = Column("Collection_expiry_tz_planned", DateTime)
    Collection_delivery_tz_planned = Column("Collection_delivery_tz_planned", DateTime)
    Collection_pickup_tz_planned = Column("Collection_pickup_tz_planned", DateTime)
    def __init__(self, Join_ID, Product_ID, Protocol, Product_step_type, Packaging_ID,
Packaging_name, Origin_site_ID, Origin_site_name, Origin_site_account_record_type,
Origin_site_account_time_zone, Destination_site_ID, Destination_site_name,
Destination_site_account_record_type, Destination_site_account_time_zone,
Collection_appointment_tz, Planned_collection_end_tz, Collection_expiry_tz_planned,
Collection_delivery_tz_planned, Collection_pickup_tz_planned):
        self.Join_ID = Join_ID
        self.Product_ID = Product_ID
        self.Protocol = Protocol
        self.Product_step_type = Product_step_type
        self.Packaging_ID = Packaging_ID
        self.Packaging_name = Packaging_name
        self.Origin_site_ID = Origin_site_ID
        self.Origin_site_name = Origin_site_name
        self.Origin_site_account_record_type = Origin_site_account_record_type
        self.Origin_site_account_time_zone = Origin_site_account_time_zone
        self.Destination_site_ID = Destination_site_ID
        self.Destination_site_name = Destination_site_name
        self.Destination_site_account_record_type = Destination_site_account_record_type
```

```
        self.Destination_site_account_time_zone = Destination_site_account_time_zone
        self.Collection_appointment_tz = Collection_appointment_tz
        self.Planned_collection_end_tz = Planned_collection_end_tz
        self.Collection_expiry_tz_planned = Collection_expiry_tz_planned
        self.Collection_delivery_tz_planned = Collection_delivery_tz_planned
        self.Collection_pickup_tz_planned = Collection_pickup_tz_planned


# Define the FACT_Segments_Planned Table
class FACT_Segments_Planned(Base):
    __tablename__ = "FACT_Segments_Planned"
    idx = Column('idx', Integer, primary_key=True, autoincrement=True)
    Join_ID = Column("Join_ID", String, ForeignKey("FACT_Shipment_Baseline.Join_ID"))
    Ref_hawb = Column("Ref_hawb", String)
    Ref_mawb = Column("Ref_mawb", String)
    Product_step_type = Column("Product_step_type", String,
ForeignKey("DIM_Product.Product_step_type"))
    Quantity_of_Legs = Column("Quantity_of_Legs", String)
    Merge_ID = Column("Merge_ID", String)
    Status = Column("Status", String)
    Detail_weight = Column("Detail_weight", String)
    Detail_weightUnits = Column("Detail_weightUnits", String)
    Detail_quantity = Column("Detail_quantity", String)
    Detail_quantityUnits = Column("Detail_quantityUnits", String)
    Detail_packageDescription = Column("Detail_packageDescription", String)
    Origin_consigneeName_pickups = Column("Origin_consigneeName_pickups", String,
ForeignKey("DIM_Site.Site_account_ID"))
    Origin_SiteType_pickups = Column("Origin_SiteType_pickups", String,
ForeignKey("DIM_Site.Site_account_record_type"))
    Origin_zip_pickups = Column("Origin_zip_pickups", String, ForeignKey("DIM_Site.Site_zip"))
    Origin_city_pickups = Column("Origin_city_pickups", String, ForeignKey("DIM_Site.Site_city"))
    Origin_country_pickups = Column("Origin_country_pickups", String,
ForeignKey("DIM_Site.Site_country"))
    Segments_deliveryType = Column("Segments_deliveryType", String)
    Segments_flightNumber = Column("Segments_flightNumber", String)
    Segments_trailerNumber = Column("Segments_trailerNumber", String)
    Segments_localAgreedDateTime_pickups = Column("Segments_localAgreedDateTime_pickups",
DateTime)
    Segments_addressAndContacts_locationName_pickups =
Column("Segments_addressAndContacts_locationName_pickups", String)
    Segments_localAgreedDateTime_deliveries = Column("Segments_localAgreedDateTime_deliveries",
DateTime)
    Segments_addressAndContacts_locationName_deliveries =
Column("Segments_addressAndContacts_locationName_deliveries", String)
```

```
    Segments_addressAndContacts_consigneeName_deliveries =
Column("Segments_addressAndContacts_consigneeName_deliveries", String,
ForeignKey("DIM_Site.Site_account_ID"))
    Segments_addressAndContacts_SiteType_deliveries =
Column("Segments_addressAndContacts_SiteType_deliveries", String,
ForeignKey("DIM_Site.Site_account_record_type"))
    Segments_addressAndContacts_zip_deliveries =
Column("Segments_addressAndContacts_zip_deliveries", String, ForeignKey("DIM_Site.Site_zip"))
    Segments_addressAndContacts_city_deliveries =
Column("Segments_addressAndContacts_city_deliveries", String, ForeignKey("DIM_Site.Site_city"))
    Segments_addressAndContacts_country_deliveries =
Column("Segments_addressAndContacts_country_deliveries", String,
ForeignKey("DIM_Site.Site_country"))
    def __init__(self, Join_ID, Ref_hawb, Ref_mawb, Product_step_type, Quantity_of_Legs,
Merge_ID, Status, Detail_weight, Detail_weightUnits, Detail_quantity,
                 Detail_quantityUnits, Detail_packageDescription, Origin_consigneeName_pickups,
Origin_SiteType_pickups, Origin_zip_pickups, Origin_city_pickups, Origin_country_pickups,
                 Segments_deliveryType, Segments_flightNumber, Segments_trailerNumber,
Segments_localAgreedDateTime_pickups, Segments_addressAndContacts_locationName_pickups,
Segments_localAgreedDateTime_deliveries,
                 Segments_addressAndContacts_locationName_deliveries,
Segments_addressAndContacts_consigneeName_deliveries,
Segments_addressAndContacts_SiteType_deliveries, Segments_addressAndContacts_zip_deliveries,
                 Segments_addressAndContacts_city_deliveries,
Segments_addressAndContacts_country_deliveries):
        self.Join_ID = Join_ID
        self.Ref_hawb = Ref_hawb
        self.Ref_mawb = Ref_mawb
        self.Product_step_type = Product_step_type
        self.Quantity_of_Legs = Quantity_of_Legs
        self.Merge_ID = Merge_ID
        self.Status = Status
        self.Detail_weight = Detail_weight
        self.Detail_weightUnits = Detail_weightUnits
        self.Detail_quantity = Detail_quantity
        self.Detail_quantityUnits = Detail_quantityUnits
        self.Detail_packageDescription = Detail_packageDescription
        self.Origin_consigneeName_pickups = Origin_consigneeName_pickups
        self.Origin_SiteType_pickups = Origin_SiteType_pickups
        self.Origin_zip_pickups = Origin_zip_pickups
        self.Origin_city_pickups = Origin_city_pickups
        self.Origin_country_pickups = Origin_country_pickups
        self.Segments_deliveryType = Segments_deliveryType
        self.Segments_flightNumber = Segments_flightNumber
        self.Segments_trailerNumber = Segments_trailerNumber
```

```
        self.Segments_localAgreedDateTime_pickups = Segments_localAgreedDateTime_pickups
        self.Segments_addressAndContacts_locationName_pickups =
Segments_addressAndContacts_locationName_pickups
        self.Segments_localAgreedDateTime_deliveries = Segments_localAgreedDateTime_deliveries
        self.Segments_addressAndContacts_locationName_deliveries =
Segments_addressAndContacts_locationName_deliveries
        self.Segments_addressAndContacts_consigneeName_deliveries =
Segments_addressAndContacts_consigneeName_deliveries
        self.Segments_addressAndContacts_SiteType_deliveries =
Segments_addressAndContacts_SiteType_deliveries
        self.Segments_addressAndContacts_zip_deliveries =
Segments_addressAndContacts_zip_deliveries
        self.Segments_addressAndContacts_city_deliveries =
Segments_addressAndContacts_city_deliveries
        self.Segments_addressAndContacts_country_deliveries =
Segments_addressAndContacts_country_deliveries

# Define the FACT_Event Table
class FACT_Event(Base):
    __tablename__ = "FACT_Event"
    idx = Column('idx', Integer, primary_key=True, autoincrement=True)
    Join_ID = Column("Join_ID", String, ForeignKey("FACT_Shipment_Baseline.Join_ID"))
    Product_step_type = Column("Product_step_type", String,
ForeignKey("DIM_Product.Product_step_type"))
    Merge_ID = Column("Merge_ID", String, ForeignKey("FACT_Segments_Planned.Merge_ID"))
    Event_location = Column("Event_location", String)
    Event_desc = Column("Event_desc", String)
    Event_timestamp = Column("Event_timestamp", DateTime)
    def __init__(self, Join_ID, Product_step_type, Merge_ID, Event_location, Event_desc,
Event_timestamp):
        self.Join_ID = Join_ID
        self.Product_step_type = Product_step_type
        self.Merge_ID = Merge_ID
        self.Event_location = Event_location
        self.Event_desc = Event_desc
        self.Event_timestamp = Event_timestamp

# Define the FACT_Alert_calculation_validation1_2 table
class FACT_Alert_calculation_validation1_2 (Base):
    __tablename__ = 'FACT_Alert_calculation_validation1_2'
    id = Column(Integer, primary_key=True)
    Merge_ID = Column(String)
    Status = Column("Status", String, ForeignKey("FACT_Segments_Planned.Status"))
    Origin_city_pickups = Column("Origin_city_pickups", String,
ForeignKey("FACT_Segments_Planned.Origin_city_pickups"))
```

```
    TenderEvent_Validation_1 = Column(Integer)
    FlightDepartureEvent_Validation_1 = Column(Integer)
    FlightArrivalEvent_Validation_1 = Column(Integer)
    RecoveryEvent_Validation_1 = Column(Integer)
    DeliveryEvent_Validation_1 = Column(Integer)
    DeltaFDtoTender_Validation_2 = Column(Integer)
    DeltaFAtoFD_Validation_2 = Column(Integer)
    DeltaRecoverytoFA_Validation_2 = Column(Integer)
    DeltaDeliverytoRecovery_Validation_2 = Column(Integer)
    def __init__(self, Merge_ID, Status, Origin_city_pickups, TenderEvent_Validation_1,
FlightDepartureEvent_Validation_1, FlightArrivalEvent_Validation_1, RecoveryEvent_Validation_1,
DeliveryEvent_Validation_1,
                 DeltaFDtoTender_Validation_2, DeltaFAtoFD_Validation_2,
DeltaRecoverytoFA_Validation_2, DeltaDeliverytoRecovery_Validation_2):
        self.Merge_ID = Merge_ID
        self.Status = Status
        self.Origin_city_pickups = Origin_city_pickups
        self.TenderEvent_Validation_1 = TenderEvent_Validation_1
        self.FlightDepartureEvent_Validation_1 = FlightDepartureEvent_Validation_1
        self.FlightArrivalEvent_Validation_1 = FlightArrivalEvent_Validation_1
        self.RecoveryEvent_Validation_1 = RecoveryEvent_Validation_1
        self.DeliveryEvent_Validation_1 = DeliveryEvent_Validation_1
        self.DeltaFDtoTender_Validation_2 = DeltaFDtoTender_Validation_2
        self.DeltaFAtoFD_Validation_2 = DeltaFAtoFD_Validation_2
        self.DeltaRecoverytoFA_Validation_2 = DeltaRecoverytoFA_Validation_2
        self.DeltaDeliverytoRecovery_Validation_2 = DeltaDeliverytoRecovery_Validation_2


# Define the FACT_Alert_calculation_validation3 table
class FACT_Alert_calculation_validation3 (Base):
    __tablename__ = 'FACT_Alert_calculation_validation3'
    id = Column(Integer, primary_key=True)
    Merge_ID = Column(String)
    Status = Column("Status", String, ForeignKey("FACT_Segments_Planned.Status"))
    Origin_city_pickups = Column("Origin_city_pickups", String,
ForeignKey("FACT_Segments_Planned.Origin_city_pickups"))
    PickupEvent_Validation_3 = Column(Integer)
    FDEvent_Validation_3 = Column(Integer)
    FAEvent_Validation_3 = Column(Integer)
    DeliveryEvent_Validation_3 = Column(Integer)
    def __init__(self, Merge_ID, Status, Origin_city_pickups, PickupEvent_Validation_3,
FDEvent_Validation_3, FAEvent_Validation_3, DeliveryEvent_Validation_3):
        self.Merge_ID = Merge_ID
        self.Status = Status
        self.Origin_city_pickups = Origin_city_pickups
        self.PickupEvent_Validation_3 = PickupEvent_Validation_3
```

```
        self.FDEvent_Validation_3 = FDEvent_Validation_3
        self.FAEvent_Validation_3 = FAEvent_Validation_3
        self.DeliveryEvent_Validation_3 = DeliveryEvent_Validation_3

# Define the FACT_Alert_calculation_validation4 table
class FACT_Alert_calculation_validation4 (Base):
    __tablename__ = 'FACT_Alert_calculation_validation4'
    id = Column(Integer, primary_key=True)
    Join_ID = Column(String)
    Product_step_type = Column("Product_step_type", String,
ForeignKey("DIM_Product.Product_step_type"))
    PickupEvent_Validation_4 = Column(Integer)
    DeliveryEvent_Validation_4 = Column(Integer)
    def __init__(self, Join_ID, Product_step_type, PickupEvent_Validation_4,
DeliveryEvent_Validation_4):
        self.Join_ID = Join_ID
        self.Product_step_type = Product_step_type
        self.PickupEvent_Validation_4 = PickupEvent_Validation_4
        self.DeliveryEvent_Validation_4 = DeliveryEvent_Validation_4
```

8.2 Calculate Event Calculation

8.2.1 Validation 1

```
# Functions to calculate TenderEvent_Validation_1
def Tender_Validation_1(session, Merge_ID):
    try:
        # Query to retrieve event timestamp from FACT_Event table
        event_tender_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Tendered to Airline')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        event_tender_planned = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Pickup Confirmed')\
            .scalar()
        # Perform calculation if both values are not None
        if event_tender_confirmed is not None and event_tender_planned is not None:
            Tender_Validation_1 = (event_tender_confirmed - event_tender_planned).total_seconds()
        else:
            Tender_Validation_1 = None
```

```
        return Tender_Validation_1

# Functions to calculate FlightDepartureEvent_Validation_1
def FlightDeparture_Validation_1(session, Merge_ID):
    try:
        # Query to retrieve event timestamp from FACT_Event table
        event_FlightDeparture_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Flight Departed')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        event_tender_planned = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Pickup Confirmed')\
            .scalar()
        # Perform calculation if both values are not None
        if event_FlightDeparture_confirmed is not None and event_tender_planned is not None:
            FlightDeparture_Validation_1 = (event_FlightDeparture_confirmed -
event_tender_planned).total_seconds()
        else:
            FlightDeparture_Validation_1 = None
        return FlightDeparture_Validation_1

# Functions to calculate FlightArrivalEvent_validation1
def FlightArrival_Validation_1(session, Merge_ID):
    try:
        # Query to retrieve event timestamp from FACT_Event table
        event_FlightArrival_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Flight Arrived')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        event_tender_planned = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Pickup Confirmed')\
            .scalar()
        # Perform calculation if both values are not None
        if event_FlightArrival_confirmed is not None and event_tender_planned is not None:
            FlightArrival_Validation_1 = (event_FlightArrival_confirmed -
event_tender_planned).total_seconds()
        else:
            FlightArrival_Validation_1 = None
        return FlightArrival_Validation_1
```

```
# Functions to calculate RecoveryEvent_Validation_1
def Recovery_Validation_1(session, Merge_ID):
    try:
        # Query to retrieve event timestamp from FACT_Event table
        event_Recovery_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Retrieved from Airline')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        event_tender_planned = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Pickup Confirmed')\
            .scalar()
        # Perform calculation if both values are not None
        if event_Recovery_confirmed is not None and event_tender_planned is not None:
            Recovery_Validation_1 = (event_Recovery_confirmed -
event_tender_planned).total_seconds()
        else:
            Recovery_Validation_1 = None
        return Recovery_Validation_1

# Functions to calculate DeliveryEvent_Validation_1
def Delivery_Validation_1(session, Merge_ID):
    try:
        # Query to retrieve event timestamp from FACT_Event table
        event_Delivery_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Delivered')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        event_tender_planned = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Pickup Confirmed')\
            .scalar()
        # Perform calculation if both values are not None
        if event_Delivery_confirmed is not None and event_tender_planned is not None:
            Delivery_Validation_1 = (event_Delivery_confirmed -
event_tender_planned).total_seconds()
        else:
            Delivery_Validation_1 = None
        return Delivery_Validation_1
```

8.2.2 Validation 3

```
# Functions to calculate PickupEvent_Validation_3
def pickup_event_validation_3(session, Merge_ID,
Segments_addressAndContacts_locationName_pickups):
    try:
        # Ensure Merge_ID ends with '_3'
        if not Merge_ID.endswith('_3'):
            return None
        # Query to retrieve event timestamp from FACT_Event table
        event_pickup_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Pickup Confirmed')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        shipment_planned_pickup =
session.query(FACT_Segments_Planned.Segments_localAgreedDateTime_pickups)\
            .filter(FACT_Segments_Planned.Merge_ID == Merge_ID)\
            .filter(FACT_Segments_Planned.Segments_addressAndContacts_locationName_pickups.is_(No
ne))\
            .scalar()
        # Perform calculation if both values are not None
        if event_pickup_confirmed is not None and shipment_planned_pickup is not None:
            pickup_event_validation_3 = (event_pickup_confirmed -
shipment_planned_pickup).total_seconds()
        else:
            pickup_event_validation_3 = None
        return pickup_event_validation_3

# Functions to calculate FDEvent_Validation_3
def FD_event_validation_3(session, Merge_ID, Segments_addressAndContacts_locationName_pickups):
    try:
        # Ensure Merge_ID ends with '_3'
        if not Merge_ID.endswith('_3'):
            return None
        # Query to retrieve event timestamp from FACT_Event table
        event_FD_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Flight Departed')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        shipment_planned_FD =
session.query(FACT_Segments_Planned.Segments_localAgreedDateTime_pickups)\
            .filter(FACT_Segments_Planned.Merge_ID == Merge_ID)\
```

```
            .filter(FACT_Segments_Planned.Segments_deliveryType == 'air')\
            .scalar()
        # Perform calculation if both values are not None
        if event_FD_confirmed is not None and shipment_planned_FD is not None:
            FD_event_validation_3 = (event_FD_confirmed - shipment_planned_FD).total_seconds()
        else:
            FD_event_validation_3 = None
        return FD_event_validation_3

# Functions to calculate FAEvent_Validation_3
def FA_event_validation_3(session, Merge_ID,
Segments_addressAndContacts_locationName_deliveries):
    try:
        # Ensure Merge_ID ends with '_3'
        if not Merge_ID.endswith('_3'):
            return None
        # Query to retrieve event timestamp from FACT_Event table
        event_FA_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Flight Arrived')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        shipment_planned_FA =
session.query(FACT_Segments_Planned.Segments_localAgreedDateTime_deliveries)\
            .filter(FACT_Segments_Planned.Merge_ID == Merge_ID)\
            .filter(FACT_Segments_Planned.Segments_deliveryType == 'air')\
            .scalar()
        # Perform calculation if both values are not None
        if event_FA_confirmed is not None and shipment_planned_FA is not None:
            FA_event_validation_3 = (event_FA_confirmed - shipment_planned_FA).total_seconds()
        else:
            FA_event_validation_3 = None
        return FA_event_validation_3

# Functions to calculate DeliveryEvent_Validation_3
def delivery_event_validation_3(session, Merge_ID,
Segments_addressAndContacts_locationName_deliveries):
    try:
        # Ensure Merge_ID ends with '_3'
        if not Merge_ID.endswith('_3'):
            return None
        # Query to retrieve event timestamp from FACT_Event table
        event_delivery_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Merge_ID == Merge_ID)\
            .filter(FACT_Event.Event_desc == 'Delivered')\
```

```
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Segments_Planned table
        shipment_planned_delivery =
session.query(FACT_Segments_Planned.Segments_localAgreedDateTime_deliveries)\
            .filter(FACT_Segments_Planned.Merge_ID == Merge_ID)\
            .filter(FACT_Segments_Planned.Segments_addressAndContacts_locationName_deliveries.is_
(None))\
            .scalar()
        # Perform calculation if both values are not None
        if event_delivery_confirmed is not None and shipment_planned_delivery is not None:
            delivery_event_validation_3 = (event_delivery_confirmed -
shipment_planned_delivery).total_seconds()
        else:
            delivery_event_validation_3 = None
        return delivery_event_validation_3
```

8.2.3 Validation 4

```
# Functions to calculate PickupEvent_Validation_4
def pickup_event_validation_4(session, Join_ID, Product_step_type):
    try:
        # Query to retrieve event timestamp from FACT_Event table
        event_pickup_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Join_ID == Join_ID)\
            .filter(FACT_Event.Product_step_type == Product_step_type)\
            .filter(FACT_Event.Event_desc == 'Pickup Confirmed')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Shipment_Baseline table
        baseline_collection_pickup_tz_planned =
session.query(FACT_Shipment_Baseline.Collection_pickup_tz_planned)\
            .filter(FACT_Shipment_Baseline.Join_ID == Join_ID)\
            .filter(FACT_Shipment_Baseline.Product_step_type == Product_step_type)\
            .scalar()
        # Perform calculation if both values are not None
        if event_pickup_confirmed is not None and baseline_collection_pickup_tz_planned is not
None:
            pickup_event_validation_4_result = (event_pickup_confirmed -
baseline_collection_pickup_tz_planned).total_seconds()
        else:
            pickup_event_validation_4_result = None
        return pickup_event_validation_4_result
```

```
# Functions to calculate DeliveryEvent_Validation_4
def delivery_event_validation_4(session, Join_ID, Product_step_type):
    try:
        # Query to retrieve event timestamp from FACT_Event table
        event_delivery_confirmed = session.query(FACT_Event.Event_timestamp)\
            .filter(FACT_Event.Join_ID == Join_ID)\
            .filter(FACT_Event.Product_step_type == Product_step_type)\
            .filter(FACT_Event.Event_desc == 'Delivered')\
            .scalar()
        # Query to retrieve Collection_pickup_tz_planned from FACT_Shipment_Baseline table
        baseline_collection_delivery_tz_planned =
session.query(FACT_Shipment_Baseline.Collection_delivery_tz_planned)\
            .filter(FACT_Shipment_Baseline.Join_ID == Join_ID)\
            .filter(FACT_Shipment_Baseline.Product_step_type == Product_step_type)\
            .scalar()
        # Perform calculation if both values are not None
        if event_delivery_confirmed is not None and baseline_collection_delivery_tz_planned is
not None:
            delivery_event_validation_4_result = (event_delivery_confirmed -
baseline_collection_delivery_tz_planned).total_seconds()
        else:
            delivery_event_validation_4_result = None
        return delivery_event_validation_4_result
```

8.3 Populating Function for the Validation Points

8.3.1 Validation Points 1 and 2

```
# Function to populate FACT_Alert_calculation_validation1_2 table
def populate_FACT_Alert_calculation_validation1_2(Session):
    # Set to store unique Merge_ID values
    unique_merge_ids = set()

    # Query FACT_Segments_Planned to get Merge_ID and
Segments_addressAndContacts_locationName_pickups
    segments_Planned_data = session.query(
        FACT_Segments_Planned.Merge_ID,
        FACT_Segments_Planned.Status,
        FACT_Segments_Planned.Origin_city_pickups,
        FACT_Segments_Planned.Segments_addressAndContacts_locationName_pickups
    ).all()
```

```
    # Iterate over each row in FACT_Segments_Planned to populate
FACT_Alert_calculation_validation1_2
    for Merge_ID, Status, Origin_city_pickups, Segments_addressAndContacts_locationName_pickups
in segments_Planned_data:

      # Only process entries with Merge_ID ending with '_3'
        if Merge_ID.endswith('_3'):

          # Check if Merge_ID is already processed
          if Merge_ID in unique_merge_ids:

            #print(f"Merge_ID {Merge_ID} already processed, skipping...")
            continue

          #print(f"Processing Merge_ID {Merge_ID}")
          # Calculate TenderEvent_Validation_1
          Tender_Validation_1_result = Tender_Validation_1(session, Merge_ID)

          # Calculate FlightDepartureEvent_Validation_1
          FlightDeparture_Validation_1_result = FlightDeparture_Validation_1(session, Merge_ID)

          # Calculate FlightArrivalEvent_Validation_1
          FlightArrival_Validation_1_result = FlightArrival_Validation_1(session, Merge_ID)

          # Calculate RecoveryEvent_Validation_1
          Recovery_Validation_1_result = Recovery_Validation_1(session, Merge_ID)

          # Calculate DeliveryEvent_Validation_1
          Delivery_Validation_1_result = Delivery_Validation_1(session, Merge_ID)

          # Calculate DeltaFDtoTender_Validation_2
          if Tender_Validation_1_result is not None and FlightDeparture_Validation_1_result is
not None:
            DeltaFDtoTender_Validation_2_result = FlightDeparture_Validation_1_result -
Tender_Validation_1_result
          else:
            DeltaFDtoTender_Validation_2_result = None

          # Calculate DeltaFAtoFD_Validation_2
          if FlightDeparture_Validation_1_result is not None and
FlightArrival_Validation_1_result is not None:
            DeltaFAtoFD_Validation_2_result = FlightArrival_Validation_1_result -
FlightDeparture_Validation_1_result
          else:
```

```
            DeltaFAtoFD_Validation_2_result = None

        # Calculate DeltaRecoverytoFA_Validation_2
        if FlightArrival_Validation_1_result is not None and Recovery_Validation_1_result is
not None:
            DeltaRecoverytoFA_Validation_2_result = Recovery_Validation_1_result -
FlightArrival_Validation_1_result
        else:
            DeltaRecoverytoFA_Validation_2_result = None

        # Calculate DeltaDeliverytoRecovery_Validation_2
        if Recovery_Validation_1_result is not None and Delivery_Validation_1_result is not
None:
            DeltaDeliverytoRecovery_Validation_2_result = Delivery_Validation_1_result -
Recovery_Validation_1_result
        else:
            DeltaDeliverytoRecovery_Validation_2_result = None

        # Create FACT_Alert_calculation object and add to session
        entry = FACT_Alert_calculation_validation1_2(
            Merge_ID=Merge_ID, Status=Status, Origin_city_pickups=Origin_city_pickups,
            TenderEvent_Validation_1 = Tender_Validation_1_result,
            FlightDepartureEvent_Validation_1 = FlightDeparture_Validation_1_result,
            FlightArrivalEvent_Validation_1 = FlightArrival_Validation_1_result,
            RecoveryEvent_Validation_1 = Recovery_Validation_1_result,
            DeliveryEvent_Validation_1 = Delivery_Validation_1_result,
            DeltaFDtoTender_Validation_2 = DeltaFDtoTender_Validation_2_result,
            DeltaFAtoFD_Validation_2 = DeltaFAtoFD_Validation_2_result,
            DeltaRecoverytoFA_Validation_2 = DeltaRecoverytoFA_Validation_2_result,
            DeltaDeliverytoRecovery_Validation_2 = DeltaDeliverytoRecovery_Validation_2_result)
        session.add(entry)

        # Add Merge_ID to the set of seen values
        unique_merge_ids.add(Merge_ID)

    # Commit the session to save the changes to the database
    session.commit()
```

8.3.2 Validation Point 3

```
# Function to populate FACT_Alert_calculation_validation3 table
def populate_FACT_Alert_calculation_validation3(Session):
```

```
    # Set to store unique Merge_ID values
    unique_merge_ids = set()

    # Query FACT_Segments_Planned to get Merge_ID and
Segments_addressAndContacts_locationName_pickups
    segments_Planned_data = session.query(
        FACT_Segments_Planned.Merge_ID,
        FACT_Segments_Planned.Status,
        FACT_Segments_Planned.Origin_city_pickups,
        FACT_Segments_Planned.Segments_deliveryType,
        FACT_Segments_Planned.Segments_addressAndContacts_locationName_pickups,
        FACT_Segments_Planned.Segments_addressAndContacts_locationName_deliveries).all()

    # Iterate over each row in FACT_Segments_Planned to populate
FACT_Alert_calculation_validation3
    for Merge_ID, Status, Origin_city_pickups, Segments_deliveryType,
Segments_addressAndContacts_locationName_pickups,
Segments_addressAndContacts_locationName_deliveries in segments_Planned_data:
        # Only process entries with Merge_ID ending with '_3'
        if Merge_ID.endswith('_3'):

          # Check if Merge_ID is already processed
          if Merge_ID in unique_merge_ids:

            #print(f"Merge_ID {Merge_ID} already processed, skipping...")
            continue

          # Calculate PickupEvent_Validation_3
          pickup_event_validation_3_result = pickup_event_validation_3(session, Merge_ID,
Segments_addressAndContacts_locationName_pickups)

          # Calculate FDEvent_Validation_3
          FD_event_validation_3_result = FD_event_validation_3(session, Merge_ID,
Segments_deliveryType)

          # Calculate FAEvent_Validation_3
          FA_event_validation_3_result = FA_event_validation_3(session, Merge_ID,
Segments_deliveryType)

          # Calculate DeliveryEvent_Validation_3
          delivery_event_validation_3_result = delivery_event_validation_3(session, Merge_ID,
Segments_addressAndContacts_locationName_deliveries)

          # Create FACT_Alert_calculation object and add to session
          entry = FACT_Alert_calculation_validation3(
```

```
            Merge_ID=Merge_ID, Status=Status, Origin_city_pickups=Origin_city_pickups,
PickupEvent_Validation_3 = pickup_event_validation_3_result,
            FDEvent_Validation_3 = FD_event_validation_3_result, FAEvent_Validation_3 =
FA_event_validation_3_result,
            DeliveryEvent_Validation_3 = delivery_event_validation_3_result)
        session.add(entry)

        # Add Merge_ID to the set of seen values
        unique_merge_ids.add(Merge_ID)

    # Commit the session to save the changes to the database
    session.commit()
```

8.3.3 Validation Points 4

```
# Function to populate FACT_Alert_calculation_validation4 table
def populate_FACT_Alert_calculation_validation4(Session):
    # Query FACT_Shipment_Baseline to get Join_ID and Product_step_type
    shipment_baseline_data = session.query(FACT_Shipment_Baseline.Join_ID,
FACT_Shipment_Baseline.Product_step_type).all()

    # Iterate over each row in FACT_Shipment_Baseline to populate
FACT_Alert_calculation_validation4
    for Join_ID, Product_step_type in shipment_baseline_data:
        # Calculate PickupEvent_Validation_4
        pickup_event_validation_4_result = pickup_event_validation_4(session, Join_ID,
Product_step_type)

        # Calculate DeliveryEvent_Validation_4
        delivery_event_validation_4_result = delivery_event_validation_4(session, Join_ID,
Product_step_type)

        # Create FACT_Alert_calculation_validation4 object and add to session
        entry = FACT_Alert_calculation_validation4(Join_ID=Join_ID,
Product_step_type=Product_step_type,
                                                   PickupEvent_Validation_4=pickup_event_validati
on_4_result,
                                                   DeliveryEvent_Validation_4=delivery_event_vali
dation_4_result)
        session.add(entry)

    # Commit the session to save the changes to the database
    session.commit()
```