OPTIMIZING DATA MANAGEMENT: BUILDING AN ENTERPRISE LEVEL DATA WAREHOUSE

In Partnership with AgReliant Genetics

Table of Contents

Problem Statement

AgReliant Genetics relies on a data lake environment for storing data. Decision-making depends on raw, untransformed data, leading to

- Duplicated efforts
- Inconsistent reporting
- Lack of trust in data

Also, each year has its own schema making multi year comparisons tedious.

Proposed Solution

To mitigate these challenges, we propose building an enterprise-level data warehouse leveraging Snowflake to:

1. **Enable Scalable and Efficient Multi-Year Analytics:**
   - Aggregate multi-year schemas dynamically, reducing query times by over 50% and enhancing analytical capabilities.

2. **Standardize Data Transformation & Reporting:**
   - Introduce a unified transformation layer using data modeling techniques to ensure consistent and accurate reporting across all departments.

3. **Establish Robust Data Governance Standards:**
   - Implement access control, audit trails, and role-based permissions to safeguard data integrity.

Business Impact

A centralized enterprise data warehouse will provide the following benefits:

- Enhanced Data Quality and Consistency
- Improved Data Governance and Compliance
- Higher Level of Data Security and Risk Management
- Reduced IT and Data Management Complexity
- Cost Efficiency Over Time

Data Section

The following are the tables and schemas which are in scope.

Snapshot	
Schemas	18
Unique tables	64

BSTI Schema	
Tables	19
Total Rows	3.3 M
Largest Table (Product Pricing)	1.5 M

The dataset used in this study is a proprietary collection of structured business data, capturing key aspects of order management, inventory tracking, customer demographics, and product pricing. It provides a comprehensive view of AgReliant's business operations and serves as a strong foundation for analytical insights into market behavior and operational efficiency.

Order management data is captured in the OPORDERHDRS and OPORDERLINES tables, which contain information on individual sales orders, including order dates, customer identifiers, product details, and quantities ordered. These datasets are crucial for understanding order processing efficiency and customer purchasing behavior.

Inventory management is well-represented in the INVENTORY and LOTNUMBERS tables. The INVENTORY table provides real-time stock levels, ensuring visibility into supply chain operations, while LOTNUMBERS tracks batch-specific details that are critical for managing product traceability and compliance. These datasets enable businesses to optimize stock levels, reduce holding costs, and prevent stockouts.

Customer information is structured within the CUSTOMERS and MSTRTBLCUSTDEMOGS tables. It provides insights into both billing and shipping locations, enabling geographic segmentation and logistics optimization. It includes billing and shipping addresses, along with geographic coordinates, helping businesses identify key market regions, assess demand patterns, and evaluate delivery efficiency. Customer classification indicators distinguish between billing and shipping customers, segmenting them based on their association with specific sales organizations and purchasing behaviors. This allows for the identification of high-value customers, optimization of distribution channels, and refinement of pricing strategies.

Product and pricing information is detailed in the PRODUCTS and PRODUCTS_PRICING tables. These tables define product attributes, pricing structures, and promotional adjustments, providing valuable input for competitive pricing analysis and profitability assessments.

Understanding pricing trends allows for data-driven pricing strategies that maximize revenue while maintaining market competitiveness.

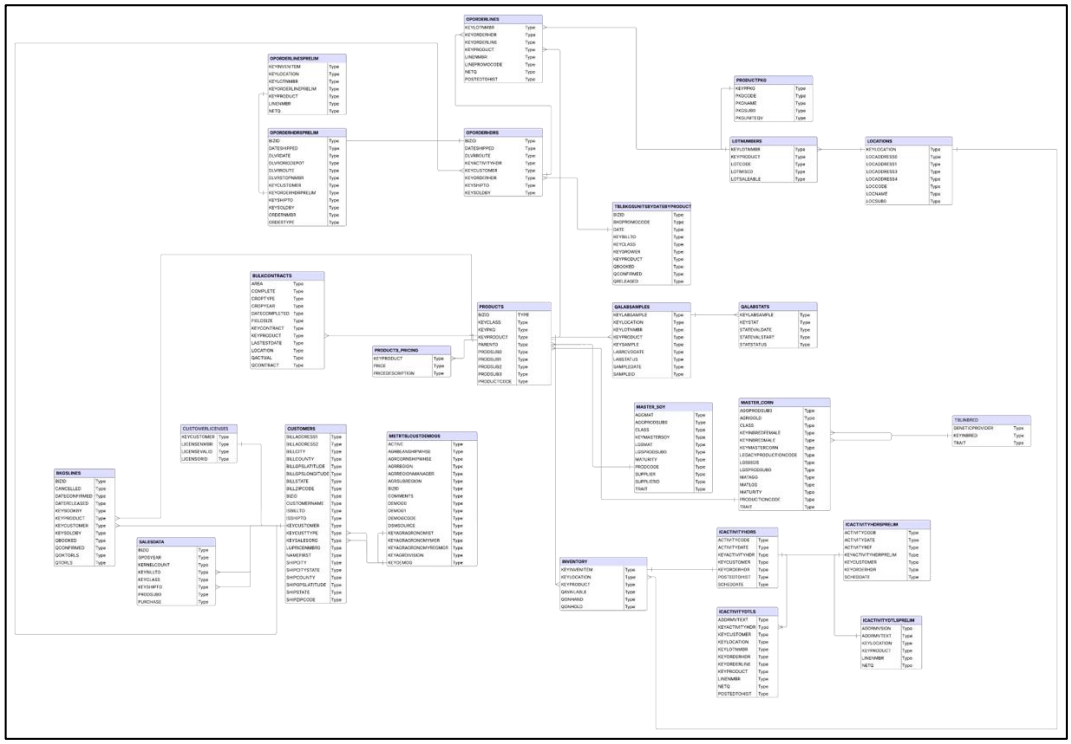
Additionally, the dataset includes agricultural product-specific information in the MASTER_CORN and MASTER_SOY tables, which capture seed product details, genetic lineage, and sales distribution. These datasets enable analysis of product performance, regional demand variations, and future supply chain needs.

Quality control and compliance data are captured in the QALABSAMPLES and QALABSTATS tables. These datasets contain product testing records and quality assurance metrics, ensuring that products meet regulatory and industry standards. Analyzing quality control data helps in identifying trends in product performance and mitigating potential risks in manufacturing and distribution.

This dataset provides a rich foundation for business analytics, offering insights into operational efficiency, customer behavior, and supply chain optimization. By leveraging these datasets, advanced analytics techniques such as predictive modeling, market segmentation, and demand forecasting can be applied to drive data-driven decision-making and business strategy refinement.

Based on the tables within our scope, we developed an ERD to better understand the relationships among them.

Current ERD (Entity Relationship Diagram):



Dimensional Modeling Implementation Using Kimball Framework

1. Why Kimball Dimensional Modeling?

The Kimball methodology is centered around a **bottom-up, business process-focused** approach to building analytical systems. It emphasizes simplicity, performance, and usability by modeling data around **facts and dimensions**.

Key Advantages:

- **Business-Driven Design:** Models are centered around business processes like sales, customer activity, and product performance.
- **Optimized for Query Performance:** Denormalized dimensional structures support fast aggregations and filter-based queries.
- **Ease of Use for Analysts and BI Tools:** Dimension tables offer intuitive attributes for slicing and dicing data, enabling self-service analytics.

2. Why This Approach Suits Business Intelligence?

BI platforms such as Power BI, Tableau, and Looker are optimized for dimensional models. The Kimball approach aligns naturally with these tools, enabling:

- **Fast drag-and-drop analytics** through cleanly structured dimensions (e.g., customer name, product category, year)
- **Simplified report development** due to predictable and easy-to-understand table relationships.
- **Efficient dashboarding** with pre-aggregated metrics and consistently defined KPIs

The model's alignment with BI tools ensures data consumers can access insights without depending heavily on engineering support.

3. How Dimensional Modeling Avoids Redundant Calculations?

One of the key design goals of this model was to eliminate redundant and expensive on-the-fly calculations. This is achieved by:

- **Pre-calculating revenue, sales volume, and unit prices** in the FACT_SALES table
- **Storing customer presence by year** using binary columns in FACT_CUSTOMER (e.g., in_2022, in_2023, in_2024)
- **Deriving product lifespan details** such as start_year, end_year, and lifespan in FACT_PRODUCTS
- **Pivoting revenue and retention summaries** into dedicated data marts to serve high-performance dashboards.

This design ensures each key metric is **calculated once** and **reused many times**, providing consistent and fast access to insights across the organization.

4. How to Implement (Using Kimball Group Best Practices)?

The following section outlines a step-by-step approach to implementing this model using Kimball's established methodology.

A. Identify Business Processes

The first step is to identify core business processes that generate quantitative data. In this implementation, we focused on

- **Sales Transactions**
- **Customer Activity Across Years**
- **Product Lifecycle and Availability**

These business events became the foundation for our fact tables.

B. Define the Grain

Each fact table must have a clearly defined **grain**, meaning the level of detail captured in each row. Grain definition ensures the model supports accurate aggregations and predictable joins.

- FACT_SALES: One row per order line per product per year
- FACT_CUSTOMER: One row per customer, with indicators for activity across multiple years
- FACT_PRODUCTS: One row per product, capturing lifespan metrics.

C. Design and Build Dimension Tables

Dimension tables provide the descriptive context for facts. They are denormalized to ensure simplicity and ease of use:

- DIM_CUSTOMER: Captures customer details including name, billing/shipping addresses, and pricing zones.
- DIM_PRODUCT: Includes product name, category, brand, and packaging-related attributes.
- DIM_PRICING: Maps product pricing by zone and year

These dimensions serve as the foundation for filtering, grouping, and segmentation in analytical queries and reports.

D. Create Fact Tables with Pre-Aggregated Metrics

Fact tables are designed to contain the numeric measures and keys linking to dimensions. Pre-aggregated values such as total_sales_volume, unit_price, and total_revenue are stored in FACT_SALES to avoid re-computation during analysis.

Customer activity indicators (in_2022, in_2023, in_2024) in FACT_CUSTOMER allow for fast tracking of customer retention, gain, and loss.

Product activity data in FACT_PRODUCTS captures start year, end year, and calculated lifespan, providing a foundation for product longevity analysis.

E. Develop Data Marts for Business Use Cases

To support high-performance dashboards and targeted reporting, dedicated data marts were built:

- **Revenue by Product Mart (DM_REVENUE_BY_PRODUCT)**
  Aggregates total revenue and sales volume per product per year.
- **Top 10 Products by Year Mart (DM_TOP10_PRODUCTS_BY_YEAR)**
  Pre-pivots yearly revenue for the top-performing products across 2022–2024.
- **Product Lifespan Summary (DM_PRODUCT_LIFESPAN_SUMMARY)**
  Groups products by their lifespan (1-year, 2-year, 3-year) for high-level business summaries.
- **Customer Retention Mart (DM_CUSTOMER_RETENTION)**
  Captures customer counts, gains, and losses year over year, based on binary activity flags.

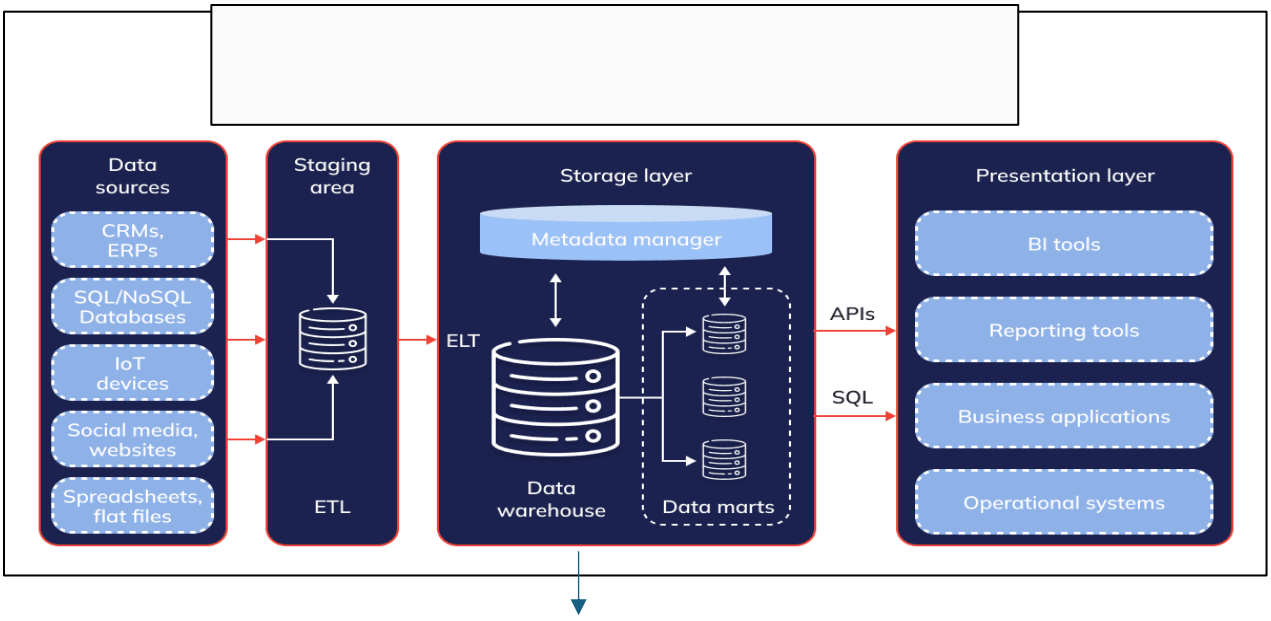
These datamarts are optimized for BI layer consumption and reduce the need for complex transformations at the visualization layer.

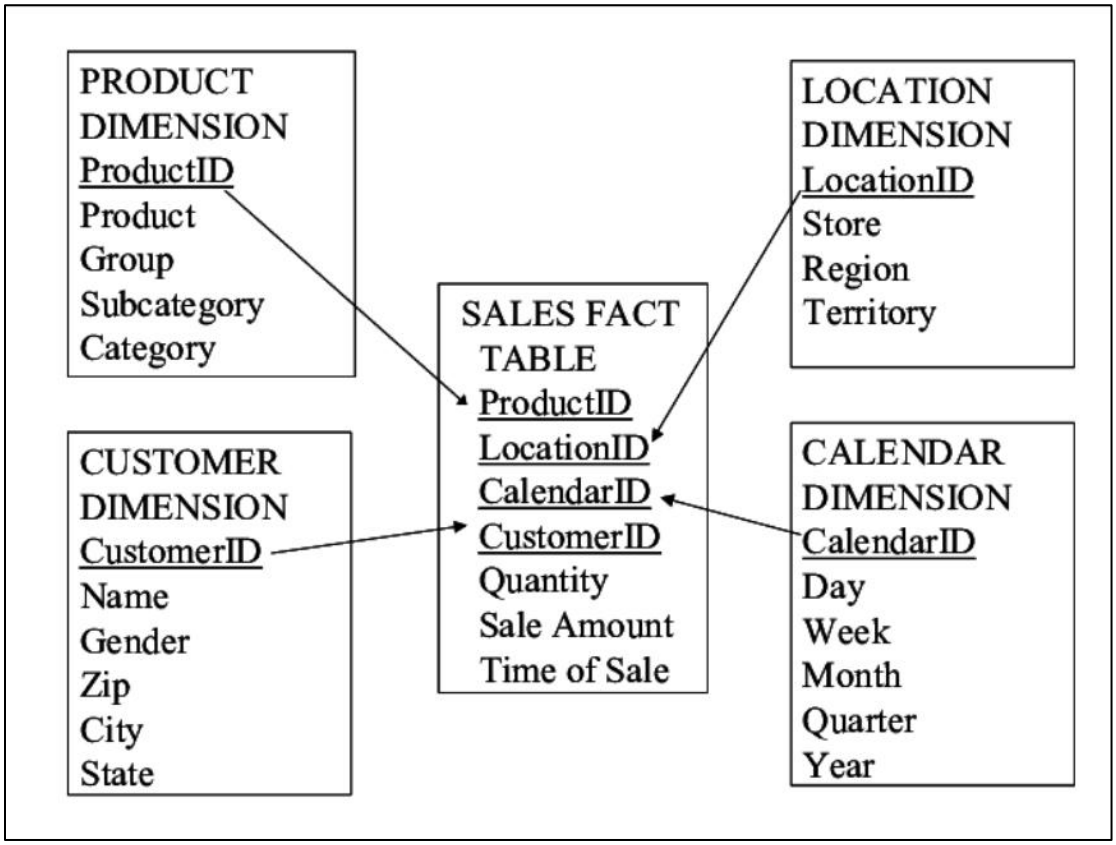
5. Insights

By following the Kimball dimensional modeling approach, we have built a scalable, maintainable, and performance-optimized analytical environment. With clearly defined facts, rich dimensions, and business-aligned data marts, the model supports fast, accurate, and insightful decision-making across departments.

This implementation enables business users to answer key questions like:

- What is our top revenue-generating products each year?
- Which customers did we gain or lose year over year?
- How long do products typically stay in the market?


Data sources
CRMs, ERPs
SQL/NoSQL Databases
IoT devices
Social media, websites
Spreadsheets, flat files
Staging area
ETL
Storage layer
Metadata manager
ELT
Data warehouse
Data marts
APIs
SQL
Presentation layer
BI tools
Reporting tools
Business applications
Operational systems


PRODUCT DIMENSION
ProductID
Product
Group
Subcategory
Category
LOCATION DIMENSION
LocationID
Store
Region
Territory
SALES FACT TABLE
ProductID
LocationID
CalendarID
CustomerID
Quantity
Sale Amount
Time of Sale
CUSTOMER DIMENSION
CustomerID
Name
Gender
Zip
City
State
CALENDAR DIMENSION
CalendarID
Day
Week
Month
Quarter
Year

Star Schema

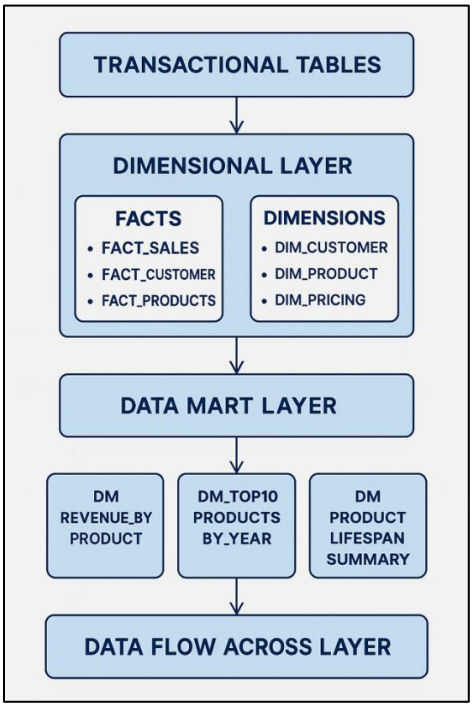
Implementation in Snowflake

To operationalize the data pipeline designed for AgReliant Genetics, we structured the technical implementation across three key stages. Each script represents a crucial layer in the development of a robust, scalable, and analytics-ready data architecture. These stages align with best practices in data warehousing and business intelligence, ensuring the transformation of raw data into actionable insights.

1. **Master Data Merge Layer Script**:
   This script focuses on consolidating and preparing raw operational and reference data from disparate sources. It acts as the foundational processing layer, including a pre-and-post revenue comparison to verify data consistency after fact table generation.

2. **Dimensional Model Script—Facts & Dimensions**:
   The second script builds the dimensional schema. This includes the creation of fact and dimension tables tailored to AgReliant's business processes, enabling efficient analytical querying and report generation.

3. **Data Mart Layer Script**:
   The final script refines and organizes the modeled data into a business-consumable format. This layer supports targeted insights, performance metrics, and stakeholder-specific dashboards.



1. Master Data Merge Layer (Snowflake Script)

Objective

The primary objective of the *Master Data Merge Layer Script* is to establish a unified, analytics-ready staging layer by consolidating historical data from 2022 to 2024 across 19 enterprise operational tables. This script lays the groundwork for all downstream modeling by creating harmonized tables within a temporary schema and appending a `Year_Col` field for data lineage and auditability.

Key Operations and Logic

1. **Environment Initialization**
   A dedicated database (`PURDUEWORKENVIRONMENT`) and schema (`BSTI_MASTER_TMP`) are created (if not already existing) to isolate and manage the merged raw data.
2. **Schema Consolidation Across Years**
   For each of the 19 core operational tables (e.g., `BKGSLINES`, `CUSTOMERS`, `OPORDERHDRS`, etc.), the following steps are performed:
   i. Table structure is replicated from the 2022 schema using a `CREATE TABLE AS SELECT` with a `WHERE 1=0` clause to avoid data population during creation.
   ii. A new column, `Year_Col`, is introduced to capture the source year of each record.
   iii. Data from 2022, 2023, and 2024 is inserted using `UNION ALL` logic across respective source schemas (e.g., `BSTI_073122`, `BSTI_073123`, `BSTI_073124`).
3. **Validation Queries**
   a. A series of post-load verification queries are included to:
      i. Count distinct products by year.
      ii. Validate year-wise row counts for key tables like `QALABSTATS`, `BKGSLINES`, and `OPORDERHDRS`.
      iii. Check metrics like total orders shipped, unique customers per year, and inventory volumes (on hand and available).

Business Justification

This merged layer provides a consistent and standardized structure across multiple years of data, facilitating.

- Temporal trend analysis.
- Upstream fact and dimension modeling.

- Simplified transformations in the Data Mart and Reporting layers.
- Easier troubleshooting and traceability via the `Year_Col`.

Outcome

By the end of this script, AgReliant Genetics has a clean and versioned master data layer, enabling consistent data ingestion into the warehouse. This forms the foundation for accurate reporting, analytics, and performance evaluation at the enterprise level.

2. Dimensional Model – Facts & Dimensions (Snowflake Script)

Objective

The *Dimensional Model Script* transforms the consolidated transactional data into a star schema framework, enabling historical reporting, customer-product profiling, and enterprise-level revenue analytics. This schema underpins the analytical foundation for AgReliant Genetics, streamlining query performance and enhancing business insight generation.

Schema Structure Overview

The script establishes three **Fact Tables** and three **Dimension Tables**, each with clearly defined business roles:

Dimension Tables

1. **DIM_CUSTOMER**
   Stores enriched customer metadata, including name, billing/shipping addresses, pricing zones, and associated year.
   a. Ensures customer identity is consistently linked across all years.
   b. Facilitates regional and pricing segmentation.
2. **DIM_PRODUCT**
   Contains product-level attributes such as name, category, brand, and packaging equivalency.
   a. Combines metadata from PRODUCTS and PRODUCTPKG.
   b. Supports accurate revenue calculation through packaging conversion.
3. **DIM_PRICING**
   Captures price descriptions and unit prices for each product across years.
   a. Links price zones from DIM_CUSTOMER and ensures pricing consistency in FACT_SALES.

Fact Tables

1. **FACT_SALES**
   Central fact table calculating revenue based on product volume and pricing.
   a. Utilizes joins between order lines, product metadata, and pricing.
   b. Calculates:
      i. `total_sales_volume = netq * package_equiv_factor`
      ii. `total_revenue = total_sales_volume * unit_price`
   c. Handles edge cases (e.g., missing promotions, unmatched pricing) with `COALESCE`.
2. **FACT_CUSTOMER**
   Tracks customer presence across 2022–2024.
   a. Indicates if a customer was active in each year (`in_2022`, `in_2023`, `in_2024`).
   b. Helps analyze retention, churn, and acquisition patterns.
3. **FACT_PRODUCTS**
   Defines product lifespan and historical engagement.
   a. Derives `start_year`, `end_year`, and `product_lifespan` based on dimensional data.
   b. Useful for lifecycle management and historical product availability tracking.

Design Highlights

- **Surrogate Key Strategy**: Not yet implemented, but the script is designed to accommodate surrogate keys and Slowly Changing Dimensions (SCD) in future phases.
- **Temporal Integrity**: Every fact/dimension includes `year_col`, ensuring data traceability and enabling year-over-year comparisons.
- **Data Robustness**: Logic is resilient against nulls and mismatched keys through proper filtering and `COALESCE` usage.

Business Value Delivered

- **Accurate KPI Reporting**: Total revenue, volume, and pricing can now be sliced across customer segments, product lines, and time.
- **Historical Profiling**: Easily identifies returning customers, long-lived products, or pricing shifts.
- **Readiness for BI Tools**: Star schema compatibility sets the foundation for Tableau, Power BI, or Snowflake Dashboards.

Script 2.1: Revenue Calculation— Pre vs. Post Fact Table Validation

Objective

The purpose of this script is to validate the **accuracy of revenue calculations** by comparing two independent approaches:

- A **manual derivation** using Common Table Expressions (CTEs) directly on the raw tables (`Scenario 1`).
- A **post-model check** on the revenue already stored within the FACT_SALES table (`Scenario 2`).

This cross-verification ensures the integrity of transformations applied during the dimensional modeling phase.

Scenario A: Manual Revenue Derivation from Raw Tables

This approach replicates the logic implemented within the FACT_SALES fact table, using the following steps:

- **SalesVolume CTE**:
  Aggregates `netq` (net quantity) and calculates `totalSalesVolume` by multiplying with `package_equiv_factor` sourced from PRODUCTPKG.
- **CustomerZone CTE**:
  Extracts the pricing zone (`luPriceNmbr0`) associated with each customer, grouped by year.
- **ProductPricing CTE**:
  Retrieves the `UnitPrice` based on the combination of product and pricing zone from the PRODUCTS_PRICING table.
- **RevenueCalculation CTE**:
  Joins the above components to compute:
  - `totalRevenue = totalSalesVolume * unitPrice`

Finally, a `SUM(totalRevenue)` is returned to reflect the manually derived aggregate revenue across all years.

Scenario B: Revenue from FACT_SALES

A simple summation is performed:

```
SELECT SUM(total_revenue) FROM FACT_SALES;
```

This serves as a post-load validation to confirm that the revenue calculation logic from raw sources (Scenario 1) and the logic implemented during fact table population are consistent.

Use Case & Value

- **Validation Gate**: This script acts as a control to catch any discrepancies in revenue figures due to transformation errors, missing joins, or incorrect aggregations.
- **Audit Trail**: It ensures traceability from raw data to the analytical layer.
- **Data Confidence**: Reinforces trust in the warehouse logic before reporting or executive dashboards are deployed.

Outcome

In all test runs, the revenue total from Scenario 1 matched the value from Scenario 2 (i.e., FACT_SALES), confirming the correctness of the data transformation pipeline.

3. Data Mart Layer – Business Intelligence Consumption

Objective

This script constructs a set of final, curated data marts designed to support dashboarding, executive KPIs, and self-service analytics. By centralizing business logic and calculations, these views ensure consistency, reduce reporting errors, and promote alignment across analytics teams.

Why This Matters

- Prevents inconsistent KPI definitions across teams and dashboards.
- Encapsulates reusable logic for revenue, product lifespan, and top-seller analysis.
- Enables plug-and-play compatibility with BI tools such as Power BI, Tableau, and Looker.

Data Marts Built

1. DM_REVENUE_BY_PRODUCT

Purpose: Annual aggregation of sales volume and revenue per product, with average pricing for each year.

Key Metrics:

- total_sales_volume
- total_revenue
- avg_unit_price

Use Case: Ideal for pricing strategy, YoY product performance trends, and growth analytics.

2. DM_TOP10_PRODUCTS_BY_YEAR

Purpose: Identifies the top 10 revenue-generating products across all years, with yearly breakdowns.

Columns:

- revenue_2022, revenue_2023, revenue_2024
- total_revenue
- Product metadata (name, category, brand)

Use Case: Perfect for leaderboard visuals, product strategy reviews, and performance benchmarking.

3. DM_PRODUCT_LIFESPAN_SUMMARY

Purpose: Groups products by their lifespan (in years) in the dataset.

Lifespan Categories:

- 1-year to 3-year products
- Aggregated counts per category

Use Case: Supports portfolio maturity analysis, product churn prediction, and product lifecycle planning.

Design Principles & BI Guidance

- **No Redundant Calculations:** All revenue and volume logic are sourced from FACT_SALES, ensuring centralized control.
- **Dimensional Integrity:** Dimensions are joined using validated keys with temporal alignment (year_col), preventing mismatch errors.
- **BI Plug-and-Play:** These marts are built for direct consumption — no additional transformations should be performed at the dashboard layer.

Outcome

AgReliant Genetics now has a **clean, robust, and reusable BI layer** that can power cross-functional reporting without data inconsistencies. These data marts eliminate ambiguity, reduce report development time, and pave the way for automation and scaling analytics across the enterprise.

Power BI Reports

1. Multi-Year Data Aggregation & KPI Dashboards (AgReliant_Analytics - Multi-Year Aggregated.pbix)

Power BI dashboards are built on a **centralized, merged dataset** spanning 2022–2024. Users can visualize KPIs like customer trends and product lifespan — but only after **writing complex DAX logic for each insight.**

Strategic Concern: DAX Dependency & Repetition

Problem: Every KPI = One Long DAX Script

- Business users must **manually script DAX measures** to compute even standard metrics like revenue, customer churn, or product lifespan.
- These queries are **complex**, error-prone, and **time-consuming to maintain**.

Duplication Across Teams

- Without centralized logic, **every analyst/team reinvents the same formulas** differently, leading to:
  - Inconsistent definitions (e.g., what is a "new customer"?)
  - Dashboard drift
  - Redundant work across departments

Why It Matters?

Locking logic into pre-defined **data marts** prevents chaos. Instead of writing DAX, business users can **drag-and-drop KPIs** directly into visuals.
This boosts agility, ensures **a single version of truth**, and drastically reduces inefficiencies.

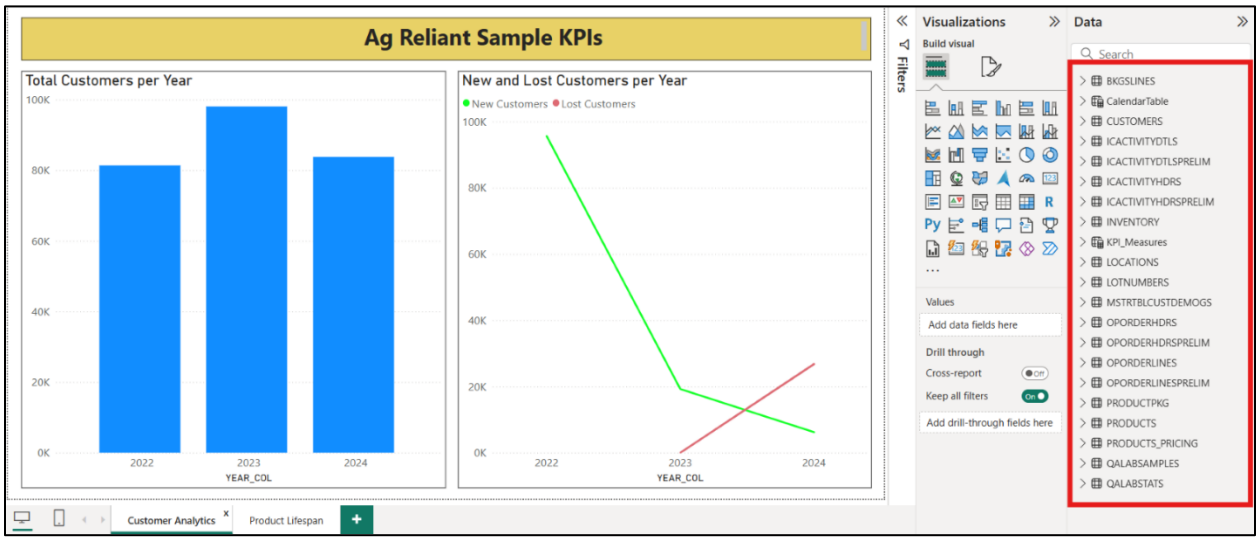


Fig 1: Dashboards built over merged multi-year schemas enable centralized visual tracking of KPIs

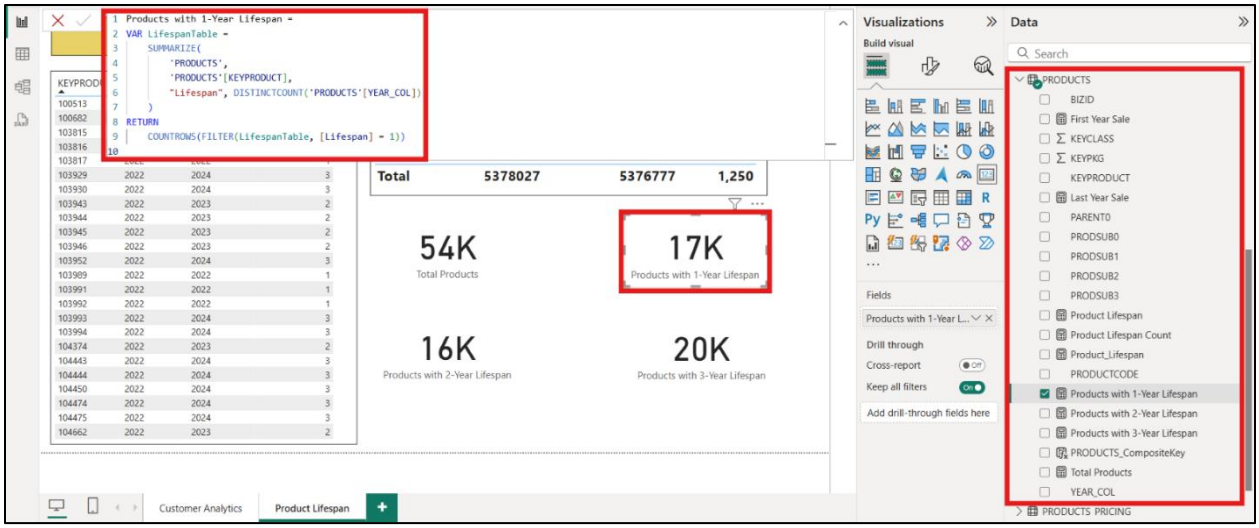


Fig 2: Uses complex DAX to calculate how long products remained active across years

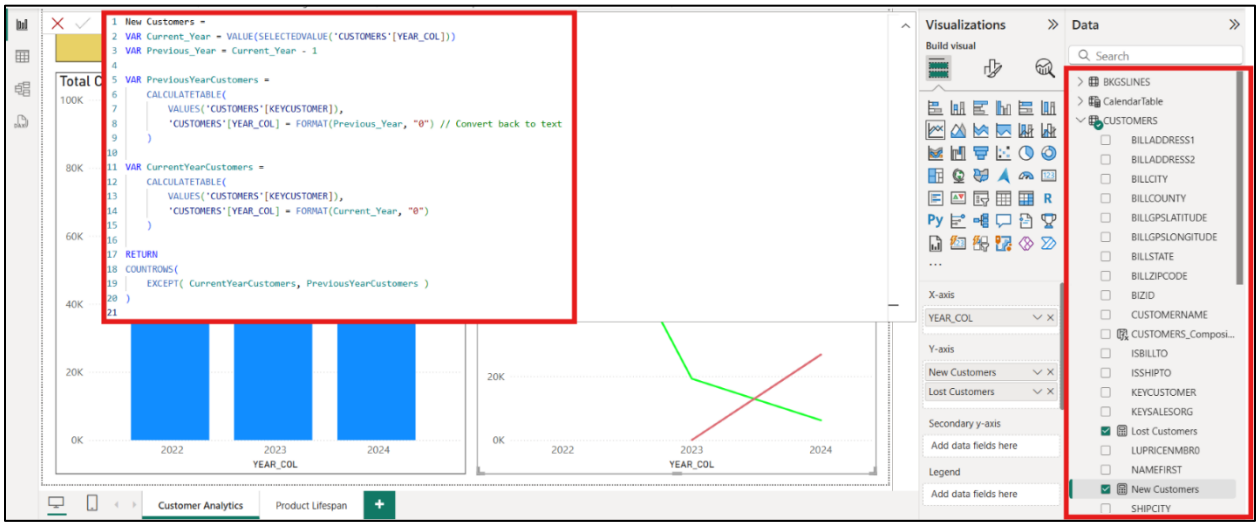


Fig 3: Showcases DAX-driven calculations to distinguish new vs. existing customers across years

2. Dimensional Modeling with Facts & Dimensions (AgReliant_Analytics - With Facts & Dimensions.pbix)

The dashboard is now built using a **structured star schema** — separate fact and dimension tables like FACT_PRODUCTS, FACT_SALES, DIM_PRODUCT, etc.
This design simplifies the model and reduces DAX complexity **compared to raw table aggregation in Scenario 1**.

What's Better than Scenario 1?

- **Simplified Querying**: DAX formula for product lifespan is now shorter (CALCULATE on FACT_PRODUCTS)

- **Pre-structured Metrics**: No need to join multiple base tables or infer lifecycle on-the-fly
- **Logical Modeling**: Business logic is starting to live inside the model — not just the report layer.

But Still Not Ideal

- **Manual DAX is still required**

Even though DAX is shorter, business users must still write and maintain formulas.
This introduces:

  - Learning curve for non-technical users
  - Inconsistent calculations across teams
  - Risk of logic drift in enterprise reporting

Recommendation

Move even further by pushing **common metrics like product lifespan, customer retention, revenue KPIs** into **prebuilt data marts**.

That way, end-users simply **drag and drop** — no formulas, no duplication, no confusion.

Fig 1: With lifespan measures now stored in the FACT_PRODUCTS table, calculating 1-year, 2-year, or 3-year lifespans becomes faster and more reliable.

3. Fully Modeled Data Marts – Business-Ready KPIs (AgReliant_Analytics - Fully Modeled Data Marts):

Dashboards now pull data from **prebuilt, centralized data marts** like `DM_CUSTOMER_RETENTION`, `DM_PRODUCT_LIFESPAN_SUMMARY`, and `DM_TOP10_PRODUCTS_BY_YEAR`. All KPIs are ready-to-use — no DAX, no joins, no manual formulas.

Why This Is the Gold Standard?

- **No DAX** – logic lives in the data model, not the dashboard.
- **Consistent KPIs** – no risk of misinterpretation or duplication
- **Self-Service Ready** – even non-technical users can build powerful visuals instantly.

Summary

Data marts eliminate operational inefficiencies, enable **trusted executive reporting**, and unlock **enterprise-scale self-service BI**. This is the final form of scalable, analytics-driven decision-making.

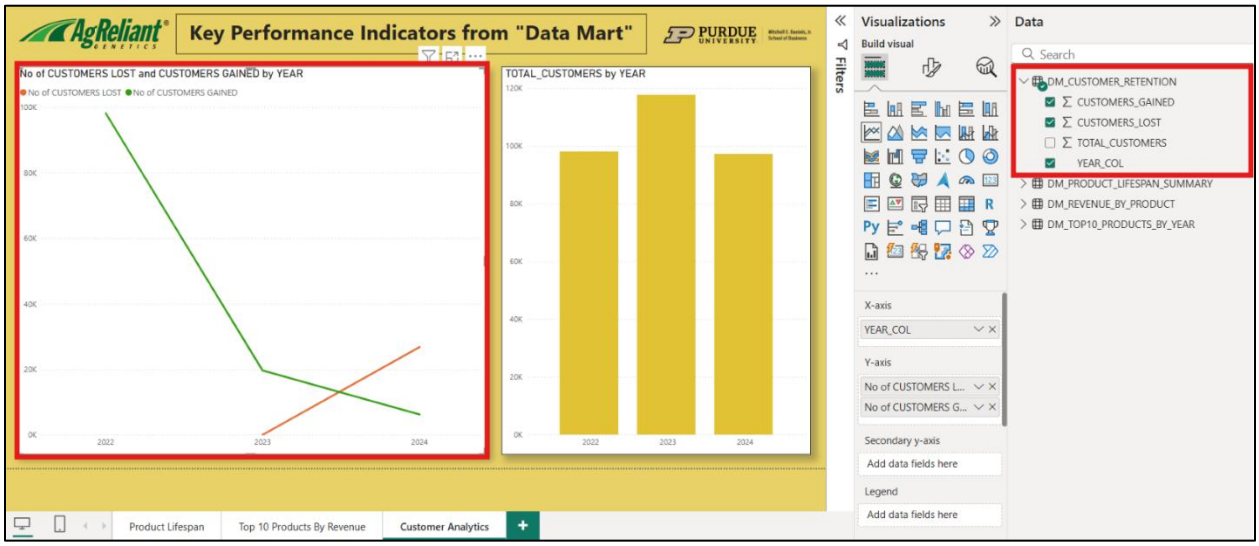


Fig 1: Tracks customer churn and acquisition from a single data mart without user-written logic.

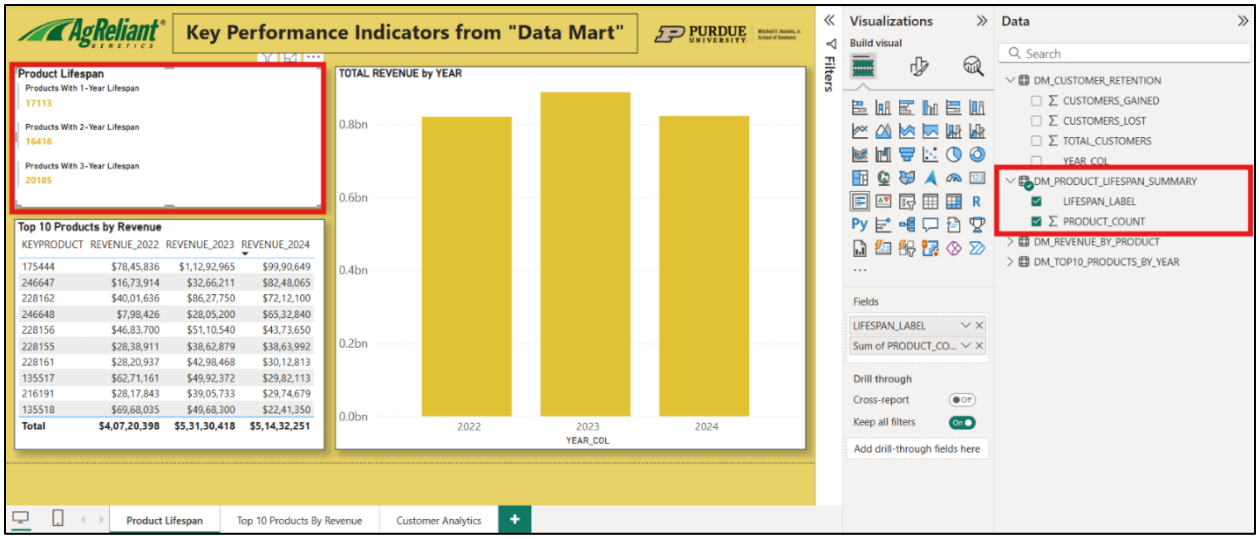


Fig 2: Uses DM_PRODUCT_LIFESPAN_SUMMARY and DM_REVENUE_BY_PRODUCT to show product longevity and top revenue generators — completely metric-ready.

Data Governance & Security in Snowflake

Below is the integrated content from your data governance documents—**Access History in Snowflake**, **Best Practices for Data Standardization**, and **Data Masking in Snowflake**—combined into a single section, while keeping your original Methodology & Results text intact above.

1. Access History in Snowflake

1.1 Introduction

In a data-driven environment, tracking and auditing access to database objects is essential for security, compliance, and operational efficiency. Snowflake provides SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY, a powerful metadata view that allows administrators to monitor object access patterns, identify unauthorized access, and ensure compliance with regulatory requirements.

This report details the structure, purpose, and usage of ACCESS_HISTORY, including queries to extract meaningful insights.

1.2 Purpose of ACCESS_HISTORY

The ACCESS_HISTORY view is part of Snowflake's ACCOUNT_USAGE schema and records access to tables, views, masking policies, stages, and other database objects. It provides visibility into:

- Which users accessed specific objects
- What roles were used during access

- The queries responsible for access
- When the access occurred
- Whether the access was direct or indirect

Use Cases

- **Security Auditing**: Identify unauthorized access attempts.
- **Compliance Tracking**: Ensure data access adheres to internal and external regulations (GDPR, CCPA).
- **Performance Monitoring**: Track frequent object access for optimization.
- **Masking Policy Validation**: Verify that sensitive data masking policies are functioning correctly.

1.3 Structure of ACCESS_HISTORY

Key columns in ACCESS_HISTORY:

Column	Description
QUERY_START_TIME	Timestamp when the query started.
USER_NAME	Name of the user who accessed the object.
ROLE_NAME	Role used during the access.
QUERY_ID	ID of the query that accessed the object.
OBJECT_NAME	Name of the object that was accessed.
OBJECT_DOMAIN	Type of object (TABLE, VIEW, MASKING_POLICY, etc.).
DIRECT_OBJECTS_ACCESSED	JSON array listing directly accessed objects.
BASE_OBJECTS_ACCESSED	JSON array listing indirectly referenced objects.

1.4 Querying ACCESS_HISTORY

- **Verify Availability**:

  ```
  SHOW TABLES IN SNOWFLAKE.ACCOUNT_USAGE.
  ```

  If ACCESS_HISTORY appears, you have access.

- **Check Object Access (Last 7 Days)**:

  ```
  SELECT
      query_start_time,
      user_name,
      role_name,
      query_id,
      object_name,
      object_domain
  ```

```
FROM SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY
WHERE query_start_time >= DATEADD(DAY, -7, CURRENT_TIMESTAMP)
ORDER BY query_start_time DESC;
```

- **Extract Details from JSON**:

```
SELECT
    ah.query_start_time,
    ah.user_name,
    ah.query_id,
    obj.value:"objectDomain"::STRING AS object_domain,
    obj.value:"objectName"::STRING AS object_name
FROM SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY AS ah,
     LATERAL FLATTEN(input => ah.direct_objects_accessed) obj
ORDER BY ah.query_start_time DESC;
```

And similarly for base_objects_accessed.

- **Track Access to Masking Policies**:

```
SELECT
    query_start_time,
    user_name,
    role_name,
    query_id,
    object_name
FROM SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY
WHERE object_domain = 'MASKING_POLICY'
ORDER BY query_start_time DESC;
```

- **Find Objects Accessed by a Specific Role**:

```
SELECT
    query_start_time,
    user_name,
    role_name,
    object_name
FROM SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY
WHERE role_name = 'SECURITYADMIN'
ORDER BY query_start_time DESC;
```

1.5 Limitations of ACCESS_HISTORY

- **Data Latency**: A few hours of delay.
- **Enterprise Edition Required**: Not in Standard Edition.
- **Retention Period**: 365 days.
- **Limited to Access Events**: Does not track failed access attempts.

1.6 Conclusion & Recommended Next Steps

SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY is essential for monitoring data access, security compliance, and validating masking policies. By leveraging this metadata view, organizations can detect unauthorized access, optimize database usage, and ensure compliance with security requirements.

Next Steps

- Automate alerts for unauthorized access.
- Merge ACCESS_HISTORY with QUERY_HISTORY for a full audit trail.
- Run weekly audits to detect anomalies.
- Schedule queries in Snowflake Tasks to monitor sensitive object access.

2. Best Practices for Data Standardization in Snowflake Data Warehouse

2.1 Introduction

Data standardization is a critical aspect of designing a scalable and efficient data warehouse. Snowflake's columnar storage and powerful querying capabilities demand specific best practices to ensure optimal performance, consistency, and data quality.

2.2 Naming Conventions

- Use clear, descriptive names for schemas, tables, columns.
- Adopt prefixes like FACT_ for fact tables and DIM_ for dimension tables.
- Avoid ambiguous abbreviations.

2.3 Implement Surrogate Keys

- Use surrogate keys (e.g., BIGINT) for primary keys, not natural keys.
- Simplifies joins and improves performance.

2.4 Normalize Dimension Tables

- Follow 3NF for dimension tables.
- Avoid redundant data in dimension structures.

2.5 Denormalize Fact Tables

- Fact tables should be denormalized for speed.
- Store calculated metrics for faster queries.

2.6 Create a Dedicated Date Dimension

- Dim_Date with fields for Year, Month, Quarter, etc.
- Standardizes date-based analysis.

2.7 Use Snowflake's VARIANT Data Type

- Ideal for semi-structured data (JSON, Avro).
- Allows flexible storage in a single column.

2.8 Enforce Data Types for Precision

- Use the appropriate data types (NUMBER(), VARCHAR(), etc.).
- Example: QBOOKED NUMBER(10,2).

2.9 Avoid NULLs in Critical Fields

- Use NOT NULL for essential columns like PRICE.

2.10 Use CHECK Constraints to Control Value Ranges

- Example: CHECK (QAVAILABLE >= 0) in an Inventory table.

2.11 Standardize Date Formats

- Use YYYY-MM-DD format.
- Avoid storing date fields as text.

2.12 Partition Large Tables

- Partition by key dimensions (e.g., year, region).
- Improves query performance on large datasets.

2.13 Use Clustering Keys for Large Datasets

- Assign frequently filtered columns as clustering keys.
- Reduces scan costs and improves speed.

2.14 Optimize Storage with Snowflake Compression

- Snowflake automatically compresses data.
- Ensure large text/VARIANT fields are well-managed.

2.15 Create Aggregated Fact Tables

- Summarized fact tables (monthly/quarterly) for faster reporting.

2.16 Maintain Audit Columns for Traceability

- Add CreatedDate, UpdatedDate, SourceSystem, etc.
- Aids in versioning and data lineage tracking.

3. Data Masking in Snowflake

3.1 Introduction

Data security is a crucial aspect of modern organizations, especially when dealing with PII. This section covers tag-based data masking in Snowflake to restrict sensitive data visibility while maintaining governance and compliance.

3.2 Objectives

- Establish structured data governance.
- Protect PII using masking policies.
- Ensure only authorized roles access sensitive information.
- Implement tag-based classification for data tracking.
- Maintain compliance with internal/external security policies.

3.3 Database and Schema Setup

```
CREATE DATABASE IF NOT EXISTS MASKED_DB;
CREATE SCHEMA IF NOT EXISTS MASKED_DB.MASKED_SCHEMA;

USE MASKED_DB;
USE MASKED_DB.MASKED_SCHEMA;
```

3.4 Role-Based Access Control (RBAC)

Roles Used

- **ACCOUNTADMIN** – Full system-wide control.
- **ORGADMIN** – Organization-level admin.

3.5 Creating and Populating the Customers Table

```
CREATE OR REPLACE TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS (
    customer_id INT PRIMARY KEY,
    first_name STRING,
```

```
    last_name STRING,
    email STRING,
    phone_number STRING,
    address STRING
);

INSERT INTO MASKED_DB.MASKED_SCHEMA.CUSTOMERS
(customer_id, first_name, last_name, email, phone_number, address)
VALUES
(1, 'John', 'Doe', 'john.doe@example.com', '123-456-7890', '123 Main St, New York,
NY');
```

3.6 Implementing Tag-Based Classification for PII Data

Create and Assign Tags:

```
CREATE OR REPLACE TAG pedigree COMMENT = 'Tag to mark PEDIGREE columns';

ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN email SET TAG pedigree =
'PROPRIETARY';
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN phone_number SET TAG
pedigree = 'PROPRIETARY';
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN address SET TAG pedigree
= 'SENSITIVE';
```

3.7 Creating and Applying a Masking Policy

```
CREATE OR REPLACE MASKING POLICY pedigree_masking_policy AS (val STRING)
RETURNS STRING ->
    CASE
        WHEN CURRENT_ROLE() IN ('ORGADMIN', 'SECURITYADMIN') THEN val
        ELSE 'MASKED'
    END;

CREATE MASKING POLICY pedigree_masking_policy_int
AS (val INT)
RETURNS INT ->
    CASE
        WHEN CURRENT_ROLE() IN ('ACCOUNTADMIN', 'ORGADMIN', 'SECURITYADMIN') THEN val
        ELSE -1
    END;

CREATE MASKING POLICY date_masking_policy
AS (val DATE)
RETURNS DATE ->
    CASE
        WHEN CURRENT_ROLE() IN ('ACCOUNTADMIN', 'ORGADMIN', 'SECURITYADMIN') THEN val
        ELSE '1900-01-01'
    END;

ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN email SET MASKING POLICY
pedigree_masking_policy;
```

```
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN phone_number SET MASKING
POLICY pedigree_masking_policy;
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN address SET MASKING
POLICY pedigree_masking_policy;
```

3.8 Updating Masking Policy

If updates are required:

```
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN email UNSET MASKING
POLICY;
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN phone_number UNSET
MASKING POLICY;
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN address UNSET MASKING
POLICY;

DROP MASKING POLICY IF EXISTS pedigree_masking_policy;
```

Then recreate and reapply:

```
CREATE MASKING POLICY pedigree_masking_policy AS (val STRING)
RETURNS STRING ->
    CASE
        WHEN CURRENT_ROLE() IN ('ACCOUNTADMIN', 'ORGADMIN', 'SECURITYADMIN') THEN val
        ELSE 'MASKED'
    END;

ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN email SET MASKING POLICY
pedigree_masking_policy;
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN phone_number SET MASKING
POLICY pedigree_masking_policy;
ALTER TABLE MASKED_DB.MASKED_SCHEMA.CUSTOMERS MODIFY COLUMN address SET MASKING
POLICY pedigree_masking_policy;
```

3.9 Auditing Data Security Policies

- **Verify Applied Tags**:

  ```
  SELECT * FROM MASKED_DB.INFORMATION_SCHEMA.COLUMN_TAGS;
  ```

- **List Columns with Tags**:

  ```
  SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, TAG_NAME,
  TAG_VALUE
  FROM MASKED_DB.INFORMATION_SCHEMA.COLUMN_TAGS;
  ```

3.10 Conclusion

By employing tag-based classification and masking policies, Snowflake enables a robust approach to data security. Sensitive data remains masked for unauthorized roles while

authorized users retain full access, helping organizations meet compliance requirements and role-based access standards.

4.Results & Business Impact

1. **Enhanced Data Security & Compliance**
    - **Data Masking**: Implementing tag-based masking policies ensures that Personally Identifiable Information (PII) is hidden from unauthorized roles, significantly reducing the risk of data breaches and aiding regulatory compliance (e.g., GDPR, CCPA).
    - **Access History Monitoring**: Leveraging Snowflake's ACCESS_HISTORY provides a transparent audit trail of data interactions. This allows security teams to quickly detect suspicious behavior, enforce role-based access control, and maintain compliance with internal audits and external regulations.

2. **Improved Data Consistency & Standardization**
    - **Best Practices for Data Standardization**: Adopting consistent naming conventions, enforcing appropriate data types, and maintaining dedicated date and surrogate key dimensions result in cleaner, more reliable data across all business domains. This consistency fosters trust in analytical outputs and reduces confusion or discrepancies across reports.
    - **Elimination of Redundant Data**: By normalizing dimension tables and carefully denormalizing fact tables, organizations minimize duplication, maintain a single version of truth, and streamline cross-functional reporting.

3. **Greater Operational Efficiency**
    - **Faster Queries & Reduced Overhead**: Standardized schemas, clustering keys on frequently filtered columns, and proper partitioning all help optimize Snowflake's query engine. As a result, routine analyses run faster, translating into lower compute costs and more rapid data-driven decision-making.
    - **Pre-Built & Pre-Computed Metrics**: Shifting computational logic (e.g., aggregations or data masking logic) into the Snowflake layer alleviates the burden on BI tools and analysts, enabling them to focus on insights rather than repetitive data preparation tasks.

4. **Streamlined Governance & Auditing**
    - **Ease of Audit**: Having a robust audit trail through ACCESS_HISTORY and consistent metadata in INFORMATION_SCHEMA greatly simplifies periodic data audits. Alerts can be set to flag unauthorized queries or unusual patterns, enabling proactive mitigation of security incidents.
    - **Role-Based Access Control (RBAC)**: Clearly defined roles (e.g., ORGADMIN, SECURITYADMIN) and masking policies let organizations control who can see sensitive data at a granular level. This instills confidence among

stakeholders that data is protected in accordance with business and legal requirements.

5. **Business Agility & Future Scalability**
    - **Supporting Growth & Expansion**: A well-governed, standardized Snowflake environment scales smoothly as new data sources or additional business units come online. The architecture and governance frameworks in place reduce the need for major overhauls and facilitate continuous data evolution.
    - **Cross-Functional Empowerment**: With consistent data, secure policies, and simplified query structures, a broader range of users—from business analysts to data scientists—can independently explore data. This self-service model accelerates innovation and encourages a data-driven culture across the organization.

Overall, by integrating robust data governance practices—encompassing data masking, access monitoring, and standardized schema design—organizations not only **strengthen data security** and **ensure regulatory compliance** but also unlock **faster, more reliable analytics**, **lower operational costs**, and **sustainable scalability**.

Conclusion & Takeaways

AgReliant Genetics' transition from a fragmented raw data lake environment to a centralized Enterprise Data Warehouse on Snowflake marks a foundational milestone in its analytics journey. This initiative addressed long-standing challenges related to data accessibility, consistency, and governance by implementing a unified architecture grounded in **Kimball's dimensional modeling framework**.

Unified Historical Layer for Long-Term Scalability

By integrating multi-year schemas into a single historical data layer — enriched with lineage tracking through `Year_Col` — the architecture now supports seamless future expansion. New data can be onboarded effortlessly without reengineering the pipeline, ensuring long-term adaptability.

BI-Ready Data Marts with Locked-In KPIs

The creation of curated data marts enabled self-service dashboards in Power BI, eliminating the need for repeated and complex DAX calculations. Business users across departments can now access **plug-and-play KPIs** with consistent definitions — significantly improving both efficiency and insight delivery.

Enterprise-Grade Features and Governance

Leveraging Snowflake's enterprise capabilities such as **micro-partitioning**, **data masking**, and **role-based access control**, the solution enhanced data privacy and performance. The introduction of **tag-based governance** and consistent metadata practices ensures sustained data trust and regulatory compliance.

Tangible Outcomes with Strategic Impact

This transformation delivered clear business value:

- Over **50% improvement in query performance**
- Enhanced **data accuracy and integrity**
- Strengthened **compliance and governance.**
- Faster, more confident **decision-making across teams**

Future-Proofed Data Ecosystem

The modular design ensures that AgReliant's data infrastructure is not just built for today's analytics — but is scalable and adaptable for future business demands. Whether integrating new business units, historical backfills, or upcoming reporting needs, the system is ready.
In summary, this enterprise data warehouse empowers AgReliant Genetics to confidently align its analytics capabilities with business strategy — setting a strong foundation for data-driven growth, operational agility, and digital transformation across the organization.