

Customer Value & Risk Analytics Dashboard

Available Datasets (Download Below)

- **customers.csv** — demographics, income, tenure, acquisition channel, churn probability
- **accounts.csv** — account-level data linked by CustomerID
- **transactions.csv** — transaction details linked by AccountID
- **products.csv** — products held by each customer

Download Your Files

- Download customers.csv
- Download accounts.csv
- Download transactions.csv
- Download products.csv

Recommended Power BI Data Model (Star Schema)

Customers (dimension)

↓ 1-to-many via CustomerID

Accounts (fact)

↓ 1-to-many via AccountID

Transactions (fact detail)

Products (bridge/relationship table) connects many-to-many between Customers and ProductName.

This model supports:

- Segmentation
- Churn analysis
- Profitability by segment
- Product usage patterns
- Behavioral insights from transactions

Metadata Dictionary — Customer Segmentation Dataset

1. Customers Table

| Column Name | Data Type | Meaning |
|---------------------|---------------|--|
| CustomerID | String | Unique customer identifier (primary key). |
| Age | Integer | Age of the customer. |
| Gender | String | Gender (Male/Female/Other). |
| Region | String | Customer's home region (North/South/East/West/Central). |
| Occupation | String | Customer's primary occupation. |
| AnnualIncome | Number | Annual income in local currency. |
| TenureYears | Decimal | How long the customer has been with the bank (years). |
| AcquisitionChannel | String | How the customer joined the bank (Branch/Online/Referral/Call Center). |
| RelationshipManager | String / Null | Assigned RM code (RM1–RM4) or null if unassigned. |
| ChurnProbability | Decimal (0–1) | Synthetic probability of churn predicted using beta distribution. |

2. Accounts Table

| Column Name | Data Type | Meaning |
|----------------|-----------|--|
| AccountID | String | Unique account identifier (primary key). |
| CustomerID | String | Customer relationship (foreign key → Customers). |
| AccountType | String | Type of account (Checking, Savings, Credit, Investment). |
| CurrentBalance | Number | Current balance of the account. |
| CreditScore | Integer | Credit score (300–850). |

3. Transactions Table

| Column Name | Data Type | Meaning |
|-----------------|-----------|---|
| TransactionID | String | Unique transaction identifier (primary key). |
| AccountID | String | Account reference (foreign key → Accounts). |
| TransactionType | String | Deposit, Withdrawal, Payment, Transfer, Investment. |
| Amount | Number | Transaction amount. |
| Date | Date | Date of the transaction. |

4. Products Table

| Column Name | Data Type | Meaning |
|------------------|-----------|---|
| CustomerID | String | Customer reference (foreign key → Customers). |
| ProductName | String | Product held (Loan, Mortgage, Credit Card, Mutual Fund, ETF). |
| SubscriptionDate | Date | When the customer started using the product. |

✓ RelationshipManager — Clear Explanation

RelationshipManager represents the code of the **portfolio manager / relationship manager** assigned to the customer.

These managers are employees responsible for supporting and monitoring specific customers.

✓ What the RM codes mean

- **RM1** → Relationship Manager #1
- **RM2** → Relationship Manager #2
- **RM3** → Relationship Manager #3
- **RM4** → Relationship Manager #4
- **Null / Empty** → The customer has **no assigned relationship manager**

✓ What a Relationship Manager does

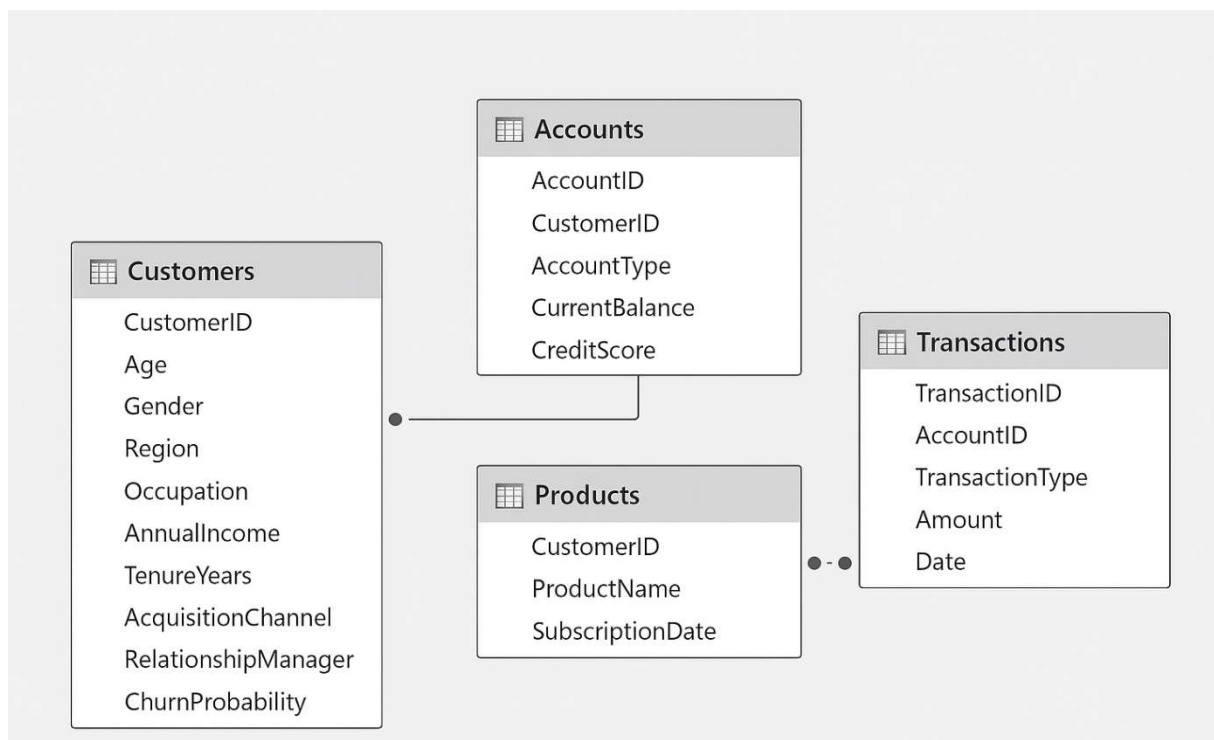
In a financial institution, a Relationship Manager typically:

- Manages high-value or strategic customers
- Provides personalized financial advice
- Helps with banking issues and product selection

- Monitors customer satisfaction and retention
- Coordinates loan requests, investment services, etc.

✓ How this field is used in analysis

- Segment customers by assigned RM
- Evaluate RM performance (portfolio revenue, churn rates, etc.)
- Compare customers with vs. without dedicated RM
- Manage customer distribution among managers



Nice — I'll turn your blueprint into a concrete, step-by-step **Power BI project plan** you can follow start-to-finish using the CSVs and metadata you already have. I'll include exact DAX measures you can paste into Power BI, modelling steps, data-prep, clustering options, recommended visuals per page, testing & deployment checklist, and useful performance/security tips.

Project plan — Power BI Dashboard for Customer Segmentation (Financial)

0. Files you already have

- `customers.csv`, `accounts.csv`, `transactions.csv`, `products.csv` (import these into Power BI).
 - `metadata_dictionary.csv` (keeps column meanings, good for report doc).
-

1. Project setup & repo

1. Create a project folder and save source CSVs and metadata there.
 2. Create a README (purpose, data sources, update frequency).
 3. Version control: keep Power BI `.pbix` and scripts under versioning (Git) or keep copies per major version.
-

2. Import & initial data prep (Power BI Desktop)

1. **Get Data → Text/CSV:** load `customers.csv`, `accounts.csv`, `transactions.csv`, `products.csv`.
2. **In Power Query Editor:**
 - Ensure correct data types (Date, Text, Decimal, Whole Number).
 - Trim whitespace, remove duplicates for keys (CustomerID, AccountID, TransactionID).
 - Standardize text fields (e.g., Region, AccountType, ProductName) using Transform → Format → Trim / Clean / Uppercase as desired.
 - Parse Date fields to Date type.
 - For RelationshipManager, replace empty strings with null (so users = blank).
3. Create a **Date table** (recommended columns: Date, Year, Quarter, Month, MonthNumber, DayOfWeek). Mark as Date table.
 - Example Power Query M snippet (simple):


```
let
    Start = #date(2023,1,1),
    End = Date.From(DateTime.LocalNow()),
    List = List.Dates(Start, Duration.Days(End-Start)+1,
        #duration(1,0,0,0)),
    Table = Table.FromList(List, Splitter.SplitByNothing(),
        {"Date"}, null, ExtraValues.Error),
    AddYear = Table.AddColumn(Table, "Year", each
        Date.Year([Date])),
    AddMonth = Table.AddColumn(AddYear, "Month", each
        Date.Month([Date])),
    AddMonthName = Table.AddColumn(AddMonth, "MonthName", each
        Date.ToText([Date], "MMMM")),
    in
        AddMonthName
```

-
4. Reduce unnecessary columns if any; keep only needed fields for analysis.
-

3. Data model & relationships

Use a star-ish model:

- **Customers** (dimension) — primary key: `CustomerID`.
- **Accounts** (bridge/fact-ish) — `AccountID` PK, `CustomerID` FK → Customers (one Customers : many Accounts).
- **Transactions** (fact table) — `AccountID` FK → Accounts (one Account : many Transactions).
- **Products** (dimension or many-to-many bridge) — `CustomerID` FK → Customers (one Customer : many Products). If one customer ↔ many product types, Products can be a fact-like table.

In Power BI Relationships pane:

- Connect `Customers[CustomerID]` → `Accounts[CustomerID]` (one-to-many, single direction from Customers to Accounts).
- Connect `Accounts[AccountID]` → `Transactions[AccountID]` (one-to-many).
- Connect `Customers[CustomerID]` → `Products[CustomerID]` (one-to-many).

Set relationship cardinalities and cross-filter direction:

- Keep cross-filter single-direction by default (`Customers` → `Accounts` → `Transactions`). Switch to both directions only when required (be cautious for performance).
-

4. Data quality & enrichment

1. **Validate keys:** ensure every Account has a Customer (or mark orphan accounts).
 2. **Create calculated columns** where helpful:
 - `FullName` if you have name fields (not present here).
 3. **Normalize categories:** create lookup tables for `AccountType`, `TransactionType`, `Region` if needed for slicers.
-

5. Clustering / Segmentation options

Two approaches:

A. Python / R (in Power BI) — external clustering

- Use **Transform data** → **Run Python script** or load pre-clustered labels CSV.

- **Typical features:** AnnualIncome, TenureYears, CurrentBalance (sum by customer), TransactionVolume, CreditScore, ChurnProbability.
- **Example Python snippet (sketch) using KMeans (you can run outside Power BI and import label):**

```
# prepare df in Python environment or Power Query Python step
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd

# df_customers: aggregated per customer with features
features =
['AnnualIncome', 'TenureYears', 'AvgBalance', 'TotalTxAmount', 'CreditScore']
X = df_customers[features].fillna(0)
X_scaled = StandardScaler().fit_transform(X)
kmeans = KMeans(n_clusters=4, random_state=42).fit(X_scaled)
df_customers['BehaviorCluster'] = kmeans.labels_.astype(str)
```

- **Save BehaviorCluster back to Customers table (as text) and use in visuals.**

B. DAX-based labelling / rule-based segments

- **Create DAX calculated column for simple RFM or rule-based segments:**

```
Segment =
SWITCH(
    TRUE(),
    Customers[AnnualIncome] > 100000 && Customers[TenureYears] > 5, "High Value",
    Customers[AnnualIncome] > 50000 && Customers[TenureYears] > 2, "Mid Value",
    Customers[ChurnProbability] > 0.6, "At Risk",
    "Other"
)
```

- **Good to prototyping; for nuanced behavioral clusters prefer Python/R.**

6. Key DAX measures (paste into Model → New measure)

Basic counts and sums

```
TotalCustomers = DISTINCTCOUNT(Customers[CustomerID])

TotalAccounts = DISTINCTCOUNT(Accounts[AccountID])

TotalTransactions = COUNTROWS(Transactions)

TotalDepositAmount = CALCULATE(SUM(Transactions[Amount]),
    Transactions[TransactionType] = "Deposit")
```

Customer-level aggregates (use relationships)

```
AvgBalancePerCustomer = AVERAGEX (VALUES (Accounts [CustomerID]),
SUM (Accounts [CurrentBalance]))
```

```
TotalBalance = SUM (Accounts [CurrentBalance])
```

Churn metrics

```
AvgChurnProbability = AVERAGE (Customers [ChurnProbability])
```

```
CustomersAtRisk = CALCULATE (DISTINCTCOUNT (Customers [CustomerID]),
Customers [ChurnProbability] > 0.6)
```

Revenue / ARPU (example proxy: fees not present — use balances/transactions as proxy)

```
AvgTransactionValue = DIVIDE (SUM (Transactions [Amount]),
COUNTROWS (Transactions))
```

```
TransactionsPerCustomer =
DIVIDE (
    COUNTROWS (Transactions),
    DISTINCTCOUNT (Transactions [AccountID]) -- or Customers if connecting
through Accounts
)
```

RM (Relationship Manager) performance

```
CustomersByRM = DISTINCTCOUNT (Customers [CustomerID])
```

```
AvgChurnByRM = AVERAGEX (VALUES (Customers [CustomerID]),
Customers [ChurnProbability])
```

```
-- better use measures with context:
```

```
AvgChurn_By_RM = CALCULATE (AVERAGE (Customers [ChurnProbability]),
ALLEXCEPT (Customers, Customers [RelationshipManager]))
```

Product Penetration

```
CustomersWithProduct = DISTINCTCOUNT (Products [CustomerID])
```

```
ProductPenetration = DIVIDE (DISTINCTCOUNT (Products [CustomerID]),
DISTINCTCOUNT (Customers [CustomerID]))
```

Lifetime Value (simple proxy)

If you have revenue per transaction, use:

```
CustomerRevenue = SUM (Transactions [Amount]) -- replace with revenue column
CLV_Simple = CALCULATE ([CustomerRevenue], DATESINPERIOD ('Date' [Date],
MAX ('Date' [Date]), -36, MONTH))
```

(Adjust to true revenue if you add fees/interest revenue fields.)

7. Suggested report pages + visuals (page-by-page)

Page 1 — Executive / Overview (Landing)

- KPI cards: `TotalCustomers`, `TotalBalance`, `TotalAccounts`, `AvgChurnProbability`, `CustomersAtRisk`.
- Line chart: Customer growth over time (use created Date table and acquisition date if available).
- Bar chart: Revenue proxy by Region.
- Donut: Product penetration (top 5).

Page 2 — Segmentation & Profiles

- Slicer: `BehaviorCluster` (if created), `Region`, `RM`.
- Cluster scatter plot: e.g., `AnnualIncome` (X) vs `AvgBalance` (Y) sized by `TotalTxAmount` and colored by `BehaviorCluster`.
- Table/matrix: Top segments with `AvgChurnProbability`, `AvgBalance`, `CountCustomers`.

Page 3 — Churn & Retention

- Bar: `ChurnProbability` distribution buckets (0–0.2, 0.2–0.4, ...).
- KPI: `CustomersAtRisk`.
- Heatmap: `TenureYears` vs `ChurnProbability` average.
- Drillthrough to Customer detail for “At Risk” customers.

Page 4 — Customer 360 / Detail (drillthrough)

- Customer profile card: `CustomerID`, `Age`, `Region`, `RM`, `Tenure`, `ChurnProbability`.
- Table: Accounts for customer (`AccountType`, `Balance`, `CreditScore`).
- Transactions table (last 12 months) with slicer to filter.

Page 5 — RM Performance

- Matrix: `RM` vs metrics (`Customers count`, `AvgBalance`, `AvgChurnProbability`, `#Products`).
- Leaderboard: top `RMs` by portfolio balance.

Page 6 — Transactions & Product Usage

- Time series: monthly transaction volume and average transaction amount.
 - Sankey (if available) or stacked bar for `TransactionType` share.
 - Product adoption over time.
-

8. Interactions, filters & UX

- Use **slicers** for Region, RM, AccountType, Cluster, Product.
- Enable **drillthrough** from segment visuals to Customer 360 page.
- Use **tooltips** with additional context (e.g., show churn probability & cluster).
- Use **bookmarks + buttons** for pre-defined views (e.g., “At Risk Customers”).

9. Row-Level Security (RLS)

- Implement RLS for sensitive data:
 - Example table `RLS_RM` with `UserPrincipalName` and `RelationshipManager`.
 - In Manage Roles: filter `Customers[RelationshipManager] = LOOKUPVALUE(RLS_RM[RelationshipManager], RLS_RM[UserPrincipalName], USERPRINCIPALNAME())`.
- Test with “View as role”.

10. Performance & best practices

1. Reduce cardinality: avoid text fields with thousands of unique values in visuals.
2. Use appropriate aggregation level for visuals (aggregated tables rather than raw transactions when possible).
3. Turn on **Query folding** in Power Query (where supported).
4. Consider **incremental refresh** for large transactions table.
5. Avoid bidirectional relationships unless necessary.
6. Use composite models / aggregations if dataset grows large.

11. Testing checklist

- Data type check: dates, numeric, text.
- Relationship integrity: no broken FK links.
- Spot-check measures vs Excel aggregates (sum of balances, distinct customer counts).
- Visual interactions: slicers and drillthrough work.
- RLS correctness: ensure users see only permitted rows.
- Performance: acceptable load time and responsive visuals.

12. Deployment

1. Publish `.pbix` to Power BI Service workspace.
2. Configure dataset refresh (Data source credentials → gateway if on-prem).
3. Schedule refresh frequency in Power BI service (daily/weekly per your needs).
4. Share / assign workspace roles (Admin/Member/Contributor/Viewer).
5. Set up App (optional) for wider distribution.

13. Documentation & handover

- Export metadata (you have `metadata_dictionary.csv`). Add:
 - Data lineage (where each CSV came from)
 - Refresh schedule
 - List of calculated columns and DAX measures (copy/paste)
 - Known caveats (synthetic churn, simplified revenue proxies)
 - Provide a short onboarding doc showing how to filter and drill to customer detail (screenshots).
-

14. Example tasks checklist (one-line actionable items)

- ☐ Import CSVs; set data types.
 - ☐ Build Date table; mark as date table.
 - ☐ Create relationships (Customers→Accounts→Transactions; Customers→Products).
 - ☐ Create behavioral cluster (Python or DAX).
 - ☐ Create DAX measures (paste from section 6).
 - ☐ Build pages & visuals (from section 7).
 - ☐ Add interactions, bookmarks, tooltips.
 - ☐ Implement RLS and test.
 - ☐ Publish and configure refresh.
-

15. Additions I can provide right away (pick any)

- I can **export a .pbix skeleton** (I cannot create the file directly here, but I can give a step-by-step script and JSON for layouts).
 - I can produce **ready-to-paste DAX for all recommended measures** (I included key ones; tell me if you want more).
 - I can provide a **Python notebook** for clustering and feature engineering using your CSVs and output a labeled CSV to import.
 - I can create a **printer-friendly project handover document** (markdown or PDF).
-

If you want, I'll now:

- A) Provide a ready-to-run **Python notebook** for clustering (with feature engineering using your CSVs), **or**
- B) Expand the DAX library with more advanced measures (e.g., rolling 12-month metrics, cohort retention), **or**
- C) Produce a direct step-by-step “Build this report” checklist with exact Visual Types & exact placement suggestions for each page.

Tell me which of A/B/C you want first and I'll produce it immediately.

Vous avez dit :

Could you tell more about this Create a README (purpose, data sources, update frequency?)

ChatGPT a dit :

Absolutely — here is a **clear, professional, and detailed explanation** of what should go into your project's **README file**, specifically focusing on:

- ✓ **Project purpose**
- ✓ **Data sources**
- ✓ **Update frequency**
- ✓ (Plus optional recommended sections)

You can use this to document your Power BI solution or include it in GitHub, SharePoint, or your BI governance folder.

README Structure for Your Power BI Customer Segmentation Project

Below is the recommended content, with explanations so you understand what each section should contain.

1. Project Purpose

This section explains **why the dashboard exists** and **what business problem it solves**.

✓ What to include:

- Business challenge: customer segmentation, retention, risk insight, product targeting.
- Who the dashboard is for: marketing, portfolio managers, executives, CRM teams.
- How the dashboard will be used.
- Expected outcomes.

✓ Example (you can copy/paste into your README):

Purpose

The goal of this project is to provide a comprehensive Power BI dashboard enabling financial institutions to analyze customer segments, understand customer behavior patterns, measure profitability and risk, and support targeted marketing strategies. The dashboard also highlights key retention metrics such as churn probability and customer lifetime value. This report supports better decision-making for portfolio managers, marketing teams, and senior leadership.

2. Data Sources

Explain **where the data comes from**, whether simulated or real.

✓ What to include:

- Source systems (CRM, Core Banking, Digital Analytics, Data Warehouse)
- Description of each table
- Whether the data is exported manually or automated
- File formats (CSV, SQL tables, API feeds)
- Any known data quality limitations

✓ Example explanation:

Data Sources

This dashboard uses four primary datasets:

1. **Customers.csv** – demographic & profile data
2. **Accounts.csv** – financial accounts, balances, credit scores
3. **Transactions.csv** – customer-level financial activities
4. **Products.csv** – financial products owned by customers
5. **Metadata Dictionary.csv** – definitions for codes and variables

The datasets are currently simulated for development purposes.

In a production environment, the data would typically be sourced from the Core Banking System, CRM, and Transaction Processing database.

3. Update Frequency

Describe **how often the data is refreshed** and **who is responsible**.

✓ **What to include:**

- Daily, weekly, monthly refresh
- Manual vs. automatic
- Data gateway usage (if applicable)
- Expected refresh time
- Dependencies (e.g., CRM data updated nightly)

✓ **Example explanation:**

Update Frequency

- **Daily refresh** at 6:00 AM via Power BI Gateway
- Transaction data is updated every night at 3:00 AM
- Customer and account data are updated at 5:00 AM
- Churn probability model is refreshed weekly
- Manual refresh can be triggered by data analysts if needed
- If any upstream system fails, the dashboard will not update

4. Data Model Description (recommended)

This explains the relationship between tables.

✓ **Example:**

The data model follows a **star schema**:

- **Fact table:** Transactions
- **Dimension tables:** Customers, Accounts, Products

Relationships:

- Customers (1) → (∞) Accounts
- Customers (1) → (∞) Transactions
- Accounts (1) → (∞) Transactions
- Products (1) → (∞) Transactions

All relationships are single direction to preserve model integrity.

5. Business Logic / Measures (recommended)

Document important logic such as:

- Churn probability
- Segment definitions
- Profitability metrics
- DAX measures
- Calculated columns

This ensures the dashboard is maintainable.

6. How to Use the Dashboard (recommended)

A short user guide for business users.

7. Version History (optional)

Document major changes over time.

8. Future Enhancements (optional)

Add planned improvements such as more advanced ML models or real-time refresh.

Fact Tables vs. Dimension Tables: A Comprehensive Guide

Introduction

In data warehousing and Business Intelligence (BI), understanding the difference between fact and dimension tables is crucial. They are the fundamental building blocks of a **star schema**, which is the simplest and most common type of data warehouse design.

The Core Concept: A Simple Analogy

Think of a **sales receipt**:

- **The Fact Table** is the **line items** on the receipt. These are the measurable actions or transactions.
 - Product: Apples, Quantity: 5, Price: \$1.00, Total: \$5.00
- **The Dimension Tables** are the **context** surrounding the receipt. They describe the "who, what, when, where, and why" of the transaction.
 - **Who?** Customer Dimension (Customer Name, Loyalty ID)
 - **When?** Date Dimension (Date, Time, Month)
 - **Where?** Store Dimension (Store Name, Address)
 - **What?** Product Dimension (Product Name, Category)

The facts (selling 5 apples) are meaningless without the dimensions that give them context.

Detailed Comparison Table

| Feature | Fact Table | Dimension Table |
|------------------------|--|--|
| Purpose | Stores measurements, metrics, or facts of a business process (e.g., sales, clicks). | Stores contextual and descriptive information about the facts. |
| Contents | Foreign Keys to dimensions and Numerical Measures (e.g., SalesAmount, Quantity). | Descriptive Attributes and a Primary Key (e.g., ProductName, CustomerAddress). |
| Record Volume | Very Large (Millions/Billions of rows). Grows quickly. | Smaller . Grows slowly as new context is added (e.g., new products). |
| Data Type | Primarily numerical and additive. | Primarily textual and descriptive. |
| Role in a Query | The " What " you are analyzing. (e.g., "What is the total revenue?") | The " How " you slice the data. (e.g., "By region?", "By product category?") |

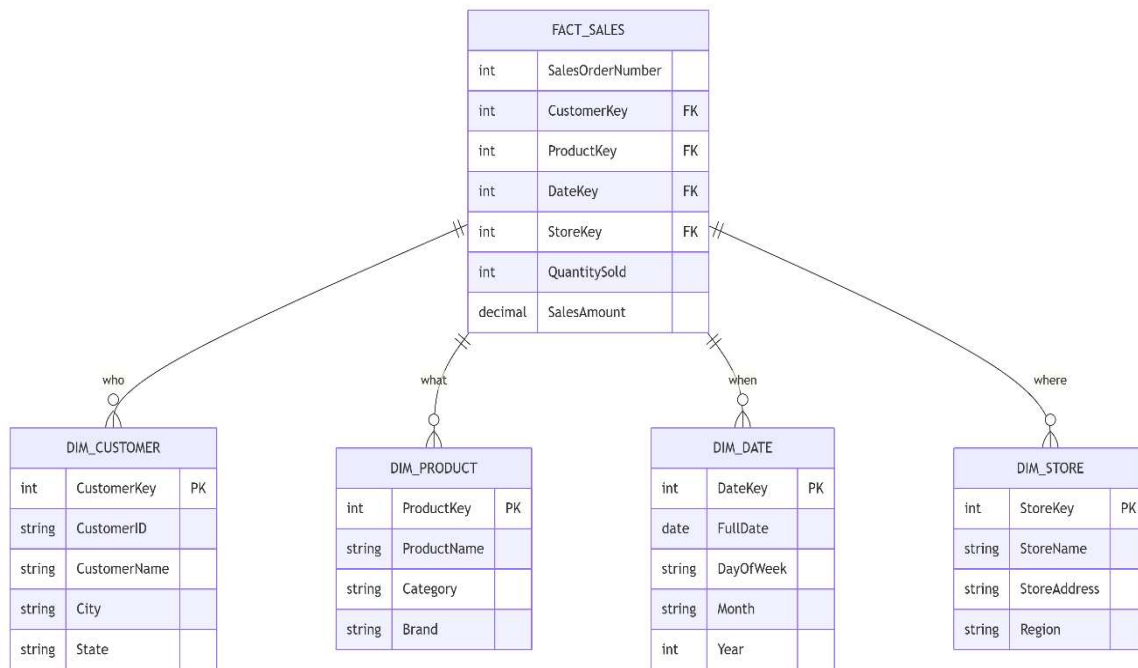
The Star Schema: How They Fit Together

In a **Star Schema**, the Fact Table sits at the center, connected to various Dimension tables, forming a star-like shape. This design is optimized for fast query performance.

Schema Diagram:

- **FACT_SALES** (Center Table)
 - Contains: SalesAmount, QuantitySold
 - Foreign Keys: CustomerKey, ProductKey, DateKey, StoreKey
- **DIM_CUSTOMER** (Surrounding Table)
 - Contains: CustomerKey (Primary Key), CustomerName, City, State
- **DIM_PRODUCT** (Surrounding Table)
 - Contains: ProductKey (Primary Key), ProductName, Category, Brand
- **DIM_DATE** (Surrounding Table)
 - Contains: DateKey (Primary Key), FullDate, Month, Year, IsHoliday
- **DIM_STORE** (Surrounding Table)
 - Contains: StoreKey (Primary Key), StoreName, Region

(You can recreate the visual diagram in Word using the SmartArt tool or simple shapes based on the description above.)



Key Takeaways

1. **Facts are the "Verbs":** They record measurable events that happen (e.g., a sale occurred).
2. **Dimensions are the "Adjectives":** They describe the qualities of those events (e.g., the product sold, the customer, the date).
3. **The Relationship:** The Fact Table answers "**How much?**" or "**How many?**". The Dimension Tables answer "**How do I want to view it?**".
4. **Benefit:** This separation makes data warehouses incredibly efficient for complex queries and reporting, allowing users to easily "slice and dice" numerical data by various descriptive criteria.

How could we deal with missing variable

- ✓ 1. If missing values are completely random (MCAR)

MCAR = Missing Completely At Random

→ The missing value has nothing to do with the variable itself *nor* with other variables.
Example: a system glitch dropped 5% of RM names.

✓ Correct approaches:

- **Impute with mean** (for numeric)
- **Impute with median** (better, less distorted)
- **Impute with mode** (for categorical)
- **Or delete rows** (safe because missingness is not biasing the dataset)

✓ **Yes — deleting rows *can be acceptable* when MCAR.**

⚠ But deletion is only safe if:

- Missing rate is low (< 5–10%)
- Data is truly random missing

✓ 2. If missing values are NOT random (MAR or MNAR)

Most real-world financial data is *not random*.

Examples:

- Only low-value customers have no RM assigned → meaningful
- High-risk customers skip income declaration → meaningful
- Customers who don't use digital banking → missing digital score

✓ **Correct approaches:**

- **Do NOT remove rows** → you remove signal
- **Do NOT impute mean/median** → you distort behavior
- Instead → create a **new category** for missing

For categorical fields:

- Replace with "Missing" or "Unknown" or "Unassigned"

For numerical fields:

Option A:

- Impute 0 or median
- Add a **binary flag: Missing_Income = 1**

Option B (better for ML):

- Impute statistically
- Add a **missing indicator variable** (0/1)

☞ This tells the model:

“Missingness itself carries information.”

⚠ Why this matters

If missingness is *not random*, it often correlates with behavior — risk, churn, profitability, or segment.

So **missingness becomes an important feature**, not noise.

✓ Summary (corrected & validated)

| Scenario | Missingness Type | Correct Approach |
|-------------------------------------|-------------------------------|---|
| Missing Completely At Random (MCAR) | Random noise | Delete rows, or impute mean/median/mode |
| Missing At Random (MAR) | Depends on other variables | Impute & add missing flag |
| Missing Not At Random (MNAR) | Missingness is meaning | Replace with special value ("Missing"), add missing indicator |

Yes — your intuition is right for MNAR:

- ▶ Use **"Missing" label**
- ▶ Use **binary variable 0/1**
- ▶ **Do NOT use mean imputation**

🔗 Applying this to YOUR project (Relationship Manager example)

Missing RM \neq random

→ Customers without RM are typically low value or low activity.

Therefore, the correct choice is:

✓ **Replace null with "Unassigned"**

✓ **Optionally create a binary field: $\text{HasRM} = 1/0$**

This preserves business meaning and improves your segmentation accuracy.

DAX

Correct Solution: Create a Calculated Column

1. **Go to the "Data" view** in Power BI
2. **Select the "Customers" table** in the right panel
3. **Click "New Column"** in the "Table tools" ribbon
4. **Copy-paste this exact formula:**

dax

Segment =

```
SWITCH(
    TRUE(),
    Customers[AnnualIncome] > 100000 && Customers[TenureYears] > 5, "High Value",
    Customers[AnnualIncome] > 50000 && Customers[TenureYears] > 2, "Mid Value",
    Customers[ChurnProbability] > 0.6, "At Risk",
    "Other"
)
```

Why It Doesn't Work as a Measure:

- **✗ Measure** = Aggregates multiple rows → requires MIN, MAX, SUM, etc.
- **✓ Calculated Column** = Works row by row → perfect for your segmentation case

What's Happening:

- As a **measure**, DAX tries to evaluate Customers[AnnualIncome] across ALL rows and doesn't know which specific value to use
- As a **calculated column**, DAX evaluates the formula for EACH individual row, which is exactly what you need for customer segmentation

Verification:

After creating the calculated column, you should see:

- A new "Segment" column in your Customers table
- Each customer classified as "High Value", "Mid Value", "At Risk" or "Other"
- No more formula errors

