

# Day 5: Data Modeling for Data Engineering - Complete Guide

## What You'll Learn Today

- **Dimensional Modeling Fundamentals** - Why it matters for analytics
  - **Star Schema Design Principles** - Building scalable data warehouses
  - **Fact vs Dimension Tables** - Understanding the core building blocks
  - **Data Modeling Best Practices** - Real-world design patterns
  - **Business-Driven Design** - Aligning models with business needs
- 

## Learning Objectives

By the end of Day 5, you will:

1. Understand why dimensional modeling is crucial for analytics
  2. Design star schema architectures that scale with business growth
  3. Distinguish between different types of facts and dimensions
  4. Apply data modeling best practices for production systems
  5. Transform business requirements into robust data models
- 

## Real Kaggle Datasets for Day 5

**Primary Dataset:** Sample Superstore Dataset

- **Kaggle Link:** [kaggle.com/datasets/bravehart101/sample-supermarket-dataset](https://kaggle.com/datasets/bravehart101/sample-supermarket-dataset)
- **Size:** 9,426 records
- **Use Case:** Perfect for learning dimensional modeling with real business data

**Supporting Dataset:** Customer Analytics Dataset

- **Kaggle Link:** [kaggle.com/datasets/imakash3011/customer-personality-analysis](https://kaggle.com/datasets/imakash3011/customer-personality-analysis)
  - **Size:** 2,240 customers
  - **Use Case:** Understanding customer dimension design with rich attributes
- 

## Why Data Modeling Matters

### The Problem with Operational Data

Imagine you're a business analyst trying to answer: *"What were our top-selling products by region last quarter, and how did they perform compared to the same quarter last year?"*

With operational databases (OLTP), this simple question becomes a nightmare:

- Data is scattered across multiple normalized tables
- Requires complex joins across 8+ tables
- Query takes 45 seconds to run
- Results are hard to interpret
- Historical comparisons are nearly impossible

## The Data Modeling Solution

With proper dimensional modeling, the same question becomes:

- Single query joining 3 tables maximum
- Query completes in under 2 seconds
- Results are intuitive and business-friendly
- Historical analysis is built-in
- Business users can self-serve

**This is why data modeling is the foundation of successful data engineering.**

---

## Understanding Star Schema

### What Makes It "Star-Shaped"?

A star schema gets its name from its visual appearance: The star schema revolves around a central fact table that contains the numerical data being analyzed. This table contains foreign keys to link to dimension tables

Think of it like a star constellation:

- **Center (Fact Table):** Contains the "what happened" - sales, clicks, transactions
- **Points (Dimension Tables):** Contain the "context" - who, what, when, where, why

### Why Star Schema Works

Star schemas denormalize the data, which means adding redundant columns to some dimension tables to make querying and working with the data faster and easier. The purpose is to trade some redundancy (duplication of data) in the data model for increased query speed

## Key Benefits:

1. **Query Performance:** Fewer joins = faster queries
  2. **Business Intuition:** Mirrors how people think about business
  3. **Flexibility:** Easy to add new dimensions or facts
  4. **Self-Service Analytics:** Business users can understand the model
  5. **Tool Compatibility:** Works with all BI tools
- 

## Fact Tables: The Heart of Analytics

### Understanding Facts

Facts are the **measurable events** in your business. They answer "what happened" and "how much."

### Examples of Facts:

- A customer purchased a product (sales fact)
- A user clicked an ad (clickstream fact)
- Inventory was adjusted (inventory fact)
- An employee worked hours (payroll fact)

### Types of Fact Tables

#### 1. Transaction Facts

**What they are:** Individual business events as they occur **Grain:** One row per transaction/event

**Example:** Each sale in a retail system

#### Business Value:

- Track individual customer behaviors
- Analyze transaction patterns
- Support detailed drill-down analysis

#### 2. Snapshot Facts

**What they are:** Regular snapshots of a situation at specific points in time **Grain:** One row per entity per time period **Example:** Daily inventory levels for each product

#### Business Value:

- Track changing balances over time

- Monitor performance trends
- Support period-over-period analysis

### 3. Accumulating Snapshot Facts

**What they are:** Track a business process from start to finish **Grain:** One row per process instance, updated as process progresses **Example:** Order fulfillment process (order → payment → shipping → delivery)

#### Business Value:

- Monitor process performance
- Identify bottlenecks
- Track SLA compliance

### Choosing the Right Grain

**Grain = the most atomic level of detail captured in the fact table**

**Rule:** Always choose the lowest practical grain because:

- You can always aggregate up, but never drill down below your grain
  - Business requirements change, and detailed data provides flexibility
  - Storage is cheap, but re-engineering is expensive
- 

## Dimension Tables: Providing Context

### Understanding Dimensions

Dimensions provide the **descriptive context** for facts. They answer "who, what, when, where, why, and how."

Dimension tables describe business entities—the things you model. Entities can include products, people, places, and concepts including time itself

### Types of Dimensions

#### 1. Conformed Dimensions

**What they are:** Dimensions used consistently across multiple fact tables **Example:** Customer dimension used in both sales and support fact tables

#### Business Value:

- Enables drill-across analysis
- Ensures consistent definitions
- Reduces development time
- Improves data governance

## 2. Role-Playing Dimensions

**What they are:** Single dimension used multiple times in same fact table **Example:** Date dimension used as order\_date, ship\_date, delivery\_date

### Design Pattern:

- Create one physical date dimension
- Use views or aliases for different roles
- Maintain referential integrity

## 3. Degenerate Dimensions

**What they are:** Dimension attributes stored in the fact table **Example:** Order number, invoice number, transaction ID

### When to Use:

- High cardinality identifiers
- No other descriptive attributes
- Primarily used for drill-through

## 4. Junk Dimensions

**What they are:** Collection of miscellaneous flags and indicators **Example:** Combining payment\_method, shipping\_priority, gift\_wrap\_flag

### Business Value:

- Reduces fact table width
- Groups related attributes
- Simplifies model maintenance

## Why Time Dimension is Special

**Every business analysis involves time.** The time dimension is your most important dimension because:

- It's used in virtually every query
- It enables trend analysis
- It supports period comparisons
- It provides fiscal calendar support

## Time Dimension Attributes

### Essential Attributes:

- Calendar attributes (year, quarter, month, week, day)
- Fiscal attributes (fiscal year, fiscal quarter)
- Business attributes (holiday flags, business day indicators)
- Relative attributes (current period flags, prior period references)

## Business Calendar vs Fiscal Calendar

**Calendar Year:** January 1 - December 31

- Good for: General reporting, external comparisons
- Used by: Public companies, government entities

**Fiscal Year:** Any 12-month period

- Good for: Internal planning, budget alignment
- Used by: Most businesses for planning purposes

**Pro Tip:** Always include both calendars in your time dimension. You never know when business requirements will change.

---

## Slowly Changing Dimensions (SCD)

### The Challenge

Dimension attributes change over time. How do you handle this while preserving analytical accuracy?

**Example:** Customer "John Smith" moves from "Gold" to "Platinum" status




- Do you update the record? (You lose history)
- Do you keep history? (How?)

## SCD Types

### Type 1: Overwrite (No History)

**When to Use:** Changes don't affect historical analysis **Example:** Correcting data entry errors, updating phone numbers





#### Business Impact:

-  Simple to implement
-  No storage overhead
-  Loses historical context

### Type 2: Add New Record (Full History)

**When to Use:** Historical context is crucial for analysis **Example:** Customer status changes, product pricing changes




#### Business Impact:

-  Preserves complete history
-  Supports "as was" analysis
-  Increases storage requirements
-  More complex to implement

### Type 3: Add New Column (Limited History)

**When to Use:** Need current + previous value only **Example:** Organizational reporting changes

#### Business Impact:

-  Simple previous value tracking
-  Minimal storage overhead
-  Limited historical depth

## Choosing SCD Strategy

#### Consider these factors:

1. **Business Requirements:** How important is history?
2. **Storage Costs:** Can you afford the storage?
3. **Query Complexity:** Can users handle complexity?

#### 4. **Change Frequency:** How often do attributes change?

---

## **Real-World Data Modeling Process**

### **Step 1: Business Process Analysis**

**Start with business questions, not data sources**

**Key Questions to Ask:**

- What business processes do we want to analyze?
- What decisions will this data support?
- Who are the primary users?
- What are the key metrics (KPIs)?
- How will success be measured?

### **Step 2: Grain Declaration**

**The most critical decision in dimensional modeling**

**For our Superstore example:**

- **Process:** Sales Analysis
- **Grain:** One row per product per order (line item level)
- **Why this grain:** Supports analysis at order level, customer level, and product level

### **Step 3: Dimension Identification**

**What provides context for your facts?**

**For Superstore Sales:**

- **Customer:** Who bought?
- **Product:** What was bought?
- **Time:** When was it bought?
- **Geography:** Where was it shipped?
- **Ship Mode:** How was it shipped?

### **Step 4: Fact Identification**

**What are you measuring?**



## For Superstore Sales:

- **Additive Facts:** Sales amount, quantity, profit (can be summed)
  - **Semi-Additive Facts:** Discount percentage (can be averaged)
  - **Non-Additive Facts:** Unit price (cannot be summed meaningfully)
- 

## Business-Driven Design Patterns

### Pattern 1: Customer Analytics Model

**Business Need:** Understand customer behavior and lifetime value

#### Design Approach:

- **Customer Dimension:** Rich demographic and behavioral attributes
- **Time Dimension:** Support for cohort analysis
- **Product Dimension:** Category hierarchies for cross-sell analysis
- **Sales Facts:** Transaction-level detail for behavioral analysis

### Pattern 2: Product Performance Model

**Business Need:** Optimize product mix and pricing

#### Design Approach:

- **Product Dimension:** Multiple hierarchies (category, brand, price tier)
- **Time Dimension:** Support for seasonality analysis
- **Geography Dimension:** Regional performance comparison
- **Sales Facts:** Include cost data for margin analysis

### Pattern 3: Operational Excellence Model

**Business Need:** Monitor and improve operational efficiency

#### Design Approach:

- **Process Dimension:** Different business processes
  - **Time Dimension:** Support for SLA monitoring
  - **Resource Dimension:** People, systems, facilities involved
  - **Performance Facts:** Cycle times, error rates, costs
-

## **Model Validation and Testing**

### **Data Quality Framework**

#### **Dimension Quality Checks:**

1. **Completeness:** Are required attributes populated?
2. **Uniqueness:** Are business keys unique?
3. **Validity:** Do values conform to business rules?
4. **Consistency:** Are related attributes consistent?

#### **Fact Quality Checks:**

1. **Referential Integrity:** Do foreign keys exist in dimensions?
2. **Measure Validity:** Are numeric values reasonable?
3. **Balance:** Do aggregated values match source systems?
4. **Completeness:** Are all expected records present?

### **Business Validation**

#### **Key Validation Questions:**

- Do query results match business user expectations?
  - Can users answer their key business questions?
  - Are response times acceptable for user workflows?
  - Does the model support future business requirements?
- 

## **Performance Optimization Strategies**

### **Physical Design Considerations**

#### **Indexing Strategy**

##### **Dimension Tables:**

- Primary key index (automatic)
- Business key index for lookups
- SCD effective date indexes
- Frequently filtered attribute indexes

##### **Fact Tables:**

- Foreign key indexes for joins
- Date range indexes for time-based queries
- Composite indexes for common filter patterns

## Partitioning Strategy

### Time-Based Partitioning:

- Partition fact tables by date (monthly or yearly)
- Improves query performance
- Simplifies data lifecycle management
- Enables parallel processing

## Aggregation Strategy

### Pre-Built Aggregates:

- Common business summaries (monthly, quarterly)
  - High-level dimensional rollups
  - Frequently requested metrics
  - Balance storage cost vs query performance
- 

## Common Modeling Mistakes

### 1. Starting with Data Instead of Business

**Mistake:** Building models based on source system structure **Solution:** Start with business requirements and user needs

### 2. Choosing Wrong Grain

**Mistake:** Picking overly aggregated grain to save space **Solution:** Choose lowest practical grain for flexibility

### 3. Over-Normalizing Dimensions

**Mistake:** Creating snowflake schemas for "data purity" **Solution:** Denormalize dimensions for query performance

### 4. Ignoring SCD Requirements

**Mistake:** Not planning for attribute changes **Solution:** Design SCD strategy during initial modeling

## 5. Poor Time Dimension Design

**Mistake:** Inadequate calendar support **Solution:** Build comprehensive time dimension with all needed attributes

---

## Hands-On Implementation with Kaggle Data

### Setting Up Your Environment

```
bash

# Download the Kaggle datasets
pip install kaggle
kaggle datasets download -d bravehart101/sample-supermarket-dataset --unzip
kaggle datasets download -d imakash3011/customer-personality-analysis --unzip

# Install required packages
pip install pandas numpy sqlalchemy psycopg2-binary matplotlib seaborn
```

### Loading and Exploring the Superstore Dataset

python

```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Superstore dataset
df = pd.read_csv('SampleSuperstore.csv')

# Basic exploration
print("Dataset Overview:")
print(f"Shape: {df.shape}")
print(f"Columns: {df.columns.tolist()}")
print(f>Date range: {df['Order Date'].min()} to {df['Order Date'].max()}")

# Check data types and missing values
print("\nData Types and Missing Values:")
print(df.dtypes)
print(df.isnull().sum())

# Basic business metrics
print(f"\nBusiness Metrics:")
print(f"Total Sales: ${df['Sales'].sum():,.2f}")
print(f"Total Profit: ${df['Profit'].sum():,.2f}")
print(f"Unique Customers: {df['Customer ID'].nunique()}")
print(f"Unique Products: {df['Product ID'].nunique()}")
```

## Implementing Star Schema with PostgreSQL

python

```

# Create database connection
engine = create_engine('postgresql://username:password@localhost:5432/datawarehouse')

# Step 1: Create Dimension Tables
def create_dimension_tables():
    """Create all dimension tables for the star schema"""

    # Customer Dimension
    customer_dim_sql = """
CREATE TABLE IF NOT EXISTS dim_customer (
    customer_key SERIAL PRIMARY KEY,
    customer_id VARCHAR(50) UNIQUE NOT NULL,
    customer_name VARCHAR(100) NOT NULL,
    segment VARCHAR(50) NOT NULL,
    -- SCD Type 2 fields
    effective_date DATE DEFAULT CURRENT_DATE,
    expiry_date DATE DEFAULT '2999-12-31',
    is_current BOOLEAN DEFAULT TRUE,
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
"""

    # Product Dimension
    product_dim_sql = """
CREATE TABLE IF NOT EXISTS dim_product (
    product_key SERIAL PRIMARY KEY,
    product_id VARCHAR(50) UNIQUE NOT NULL,
    product_name VARCHAR(200) NOT NULL,
    category VARCHAR(50) NOT NULL,
    sub_category VARCHAR(50) NOT NULL,
    -- Additional attributes
    profitability_tier VARCHAR(20),
    effective_date DATE DEFAULT CURRENT_DATE,
    expiry_date DATE DEFAULT '2999-12-31',
    is_current BOOLEAN DEFAULT TRUE,
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
"""

    # Geography Dimension
    geography_dim_sql = """
CREATE TABLE IF NOT EXISTS dim_geography (
    geography_key SERIAL PRIMARY KEY,

```

```

country VARCHAR(50) NOT NULL,
region VARCHAR(50) NOT NULL,
state VARCHAR(50) NOT NULL,
city VARCHAR(100) NOT NULL,
postal_code VARCHAR(20),
-- Business attributes
market_size VARCHAR(20),
effective_date DATE DEFAULT CURRENT_DATE,
expiry_date DATE DEFAULT '2999-12-31',
is_current BOOLEAN DEFAULT TRUE,
created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

*# Ship Mode Dimension*

```

ship_mode_dim_sql = """
CREATE TABLE IF NOT EXISTS dim_ship_mode (
    ship_mode_key SERIAL PRIMARY KEY,
    ship_mode VARCHAR(50) UNIQUE NOT NULL,
    ship_category VARCHAR(30) NOT NULL,
    priority_level INTEGER,
    effective_date DATE DEFAULT CURRENT_DATE,
    expiry_date DATE DEFAULT '2999-12-31',
    is_current BOOLEAN DEFAULT TRUE,
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

*# Date Dimension (comprehensive)*

```

date_dim_sql = """
CREATE TABLE IF NOT EXISTS dim_date (
    date_key INTEGER PRIMARY KEY,
    full_date DATE UNIQUE NOT NULL,
    day_of_week INTEGER NOT NULL,
    day_name VARCHAR(10) NOT NULL,
    month_number INTEGER NOT NULL,
    month_name VARCHAR(10) NOT NULL,
    quarter_number INTEGER NOT NULL,
    year_number INTEGER NOT NULL,
    is_weekday BOOLEAN NOT NULL,
    is_weekend BOOLEAN NOT NULL,
    fiscal_year INTEGER NOT NULL,
    fiscal_quarter INTEGER NOT NULL
);

```



```
.....
```

```
# Execute table creation
```

```
with engine.connect() as conn:
```

```
    conn.execute(customer_dim_sql)
```

```
    conn.execute(product_dim_sql)
```

```
    conn.execute(geography_dim_sql)
```

```
    conn.execute(ship_mode_dim_sql)
```

```
    conn.execute(date_dim_sql)
```

```
    conn.commit()
```

```
print("Dimension tables created successfully!")
```

```
# Step 2: Create Fact Table
```

```
def create_fact_table():
```

```
    """Create the central fact table"""
```

```
fact_sales_sql = """
```

```
CREATE TABLE IF NOT EXISTS fact_sales (
```

```
    sale_key SERIAL PRIMARY KEY,
```

```
    -- Foreign Keys to Dimensions
```

```
    customer_key INTEGER REFERENCES dim_customer(customer_key),
```

```
    product_key INTEGER REFERENCES dim_product(product_key),
```

```
    geography_key INTEGER REFERENCES dim_geography(geography_key),
```

```
    ship_mode_key INTEGER REFERENCES dim_ship_mode(ship_mode_key),
```

```
    order_date_key INTEGER REFERENCES dim_date(date_key),
```

```
    ship_date_key INTEGER REFERENCES dim_date(date_key),
```

```
    -- Business Keys
```

```
    order_id VARCHAR(50) NOT NULL,
```

```
    row_id INTEGER NOT NULL,
```

```
    -- Measures (Facts)
```

```
    sales_amount DECIMAL(10,2) NOT NULL,
```

```
    quantity INTEGER NOT NULL,
```

```
    discount_amount DECIMAL(10,2) DEFAULT 0,
```

```
    profit_amount DECIMAL(10,2) NOT NULL,
```

```
    -- Derived Measures
```

```
    unit_price DECIMAL(10,2),
```

```
    profit_margin DECIMAL(5,4),
```

```
    -- Audit Fields
```

```
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);  
''''
```

```
with engine.connect() as conn:  
    conn.execute(fact_sales_sql)  
    conn.commit()  
  
print("Fact table created successfully!")
```

```
# Execute table creation  
create_dimension_tables()  
create_fact_table()
```

## ETL Process Implementation

python

*# Step 3: Load Data into Dimensions*

```
def load_customer_dimension():
    """Load customer data with SCD Type 2 implementation"""

    # Extract unique customers from source
    customers = df[['Customer ID', 'Customer Name', 'Segment']].drop_duplicates()

    # Prepare data for insertion
    customer_data = []
    for _, row in customers.iterrows():
        customer_data.append({
            'customer_id': row['Customer ID'],
            'customer_name': row['Customer Name'],
            'segment': row['Segment']
        })

    # Insert into dimension table
    customer_df = pd.DataFrame(customer_data)
    customer_df.to_sql('dim_customer', engine, if_exists='append', index=False)

    print(f"Loaded {len(customer_data)} customers into dim_customer")

def load_product_dimension():
    """Load product data with business classifications"""

    # Extract unique products
    products = df[['Product ID', 'Product Name', 'Category', 'Sub-Category']].drop_duplicates()

    # Add profitability tier based on average profit
    product_profit = df.groupby('Product ID')['Profit'].mean().reset_index()
    product_profit['profitability_tier'] = pd.cut(
        product_profit['Profit'],
        bins=[-np.inf, 0, 50, np.inf],
        labels=['Low', 'Medium', 'High']
    )

    # Merge with product details
    products = products.merge(product_profit[['Product ID', 'profitability_tier']],
                              on='Product ID', how='left')

    # Prepare for insertion
    product_data = []
    for _, row in products.iterrows():
```

```

        product_data.append({
            'product_id': row['Product ID'],
            'product_name': row['Product Name'],
            'category': row['Category'],
            'sub_category': row['Sub-Category'],
            'profitability_tier': str(row['profitability_tier'])
        })

product_df = pd.DataFrame(product_data)
product_df.to_sql('dim_product', engine, if_exists='append', index=False)

print(f"Loaded {len(product_data)} products into dim_product")

def load_geography_dimension():
    """Load geography data with market size classification"""

    # Extract unique geographic combinations
    geography = df[['Country', 'Region', 'State', 'City', 'Postal Code']].drop_duplicates()

    # Add market size based on sales volume
    city_sales = df.groupby(['Country', 'Region', 'State', 'City'])['Sales'].sum().reset_index()
    city_sales['market_size'] = pd.cut(
        city_sales['Sales'],
        bins=[0, 10000, 50000, np.inf],
        labels=['Small', 'Medium', 'Large']
    )

    # Merge with geography
    geography = geography.merge(
        city_sales[['Country', 'Region', 'State', 'City', 'market_size']],
        on=['Country', 'Region', 'State', 'City'],
        how='left'
    )

    # Prepare for insertion
    geo_data = []
    for _, row in geography.iterrows():
        geo_data.append({
            'country': row['Country'],
            'region': row['Region'],
            'state': row['State'],
            'city': row['City'],
            'postal_code': row['Postal Code'],
            'market_size': str(row['market_size'])
        })

```

```
)
```

```
geo_df = pd.DataFrame(geo_data)
geo_df.to_sql('dim_geography', engine, if_exists='append', index=False)
```

```
print(f"Loaded {len(geo_data)} locations into dim_geography")
```

```
def load_ship_mode_dimension():
```

```
    """Load shipping mode data with business classifications"""
```

```
    ship_modes = df['Ship Mode'].unique()
```

```
    ship_mode_data = []
```

```
    for mode in ship_modes:
```

```
        # Classify shipping categories
```

```
        if 'Same Day' in mode:
```

```
            category = 'Express'
```

```
            priority = 1
```

```
        elif 'First Class' in mode:
```

```
            category = 'Express'
```

```
            priority = 2
```

```
        elif 'Second Class' in mode:
```

```
            category = 'Standard'
```

```
            priority = 3
```

```
        else:
```

```
            category = 'Standard'
```

```
            priority = 4
```

```
    ship_mode_data.append({
```

```
        'ship_mode': mode,
```

```
        'ship_category': category,
```

```
        'priority_level': priority
```

```
    })
```

```
    ship_df = pd.DataFrame(ship_mode_data)
```

```
    ship_df.to_sql('dim_ship_mode', engine, if_exists='append', index=False)
```

```
    print(f"Loaded {len(ship_mode_data)} ship modes into dim_ship_mode")
```

```
def load_date_dimension():
```

```
    """Load comprehensive date dimension"""
```

```
    # Get date range from data
```

```
    df['Order Date'] = pd.to_datetime(df['Order Date'])
```

```

min_date = df['Order Date'].min()
max_date = df['Order Date'].max()

# Generate date range
date_range = pd.date_range(start=min_date, end=max_date, freq='D')

date_data = []
for date in date_range:
    date_data.append({
        'date_key': int(date.strftime('%Y%m%d')),
        'full_date': date.date(),
        'day_of_week': date.dayofweek + 1,
        'day_name': date.strftime('%A'),
        'month_number': date.month,
        'month_name': date.strftime('%B'),
        'quarter_number': date.quarter,
        'year_number': date.year,
        'is_weekday': date.dayofweek < 5,
        'is_weekend': date.dayofweek >= 5,
        'fiscal_year': date.year if date.month >= 7 else date.year - 1,
        'fiscal_quarter': ((date.month - 1) // 3) + 1
    })

date_df = pd.DataFrame(date_data)
date_df.to_sql('dim_date', engine, if_exists='append', index=False)

print(f"Loaded {len(date_data)} dates into dim_date")

# Execute dimension loading
load_customer_dimension()
load_product_dimension()
load_geography_dimension()
load_ship_mode_dimension()
load_date_dimension()

```

## Fact Table Loading

python



```

def load_fact_table():
    """Load fact table with proper foreign key relationships"""

    # Prepare source data
    df['Order Date'] = pd.to_datetime(df['Order Date'])
    df['Ship Date'] = pd.to_datetime(df['Ship Date'])
    df['order_date_key'] = df['Order Date'].dt.strftime('%Y%m%d').astype(int)
    df['ship_date_key'] = df['Ship Date'].dt.strftime('%Y%m%d').astype(int)

    # Get dimension keys
    with engine.connect() as conn:
        # Customer keys
        customer_keys = pd.read_sql("""
            SELECT customer_key, customer_id
            FROM dim_customer
            WHERE is_current = TRUE
            """, conn)

        # Product keys
        product_keys = pd.read_sql("""
            SELECT product_key, product_id
            FROM dim_product
            WHERE is_current = TRUE
            """, conn)

        # Geography keys
        geography_keys = pd.read_sql("""
            SELECT geography_key, country, region, state, city
            FROM dim_geography
            WHERE is_current = TRUE
            """, conn)

        # Ship mode keys
        ship_mode_keys = pd.read_sql("""
            SELECT ship_mode_key, ship_mode
            FROM dim_ship_mode
            WHERE is_current = TRUE
            """, conn)

    # Merge to get foreign keys
    fact_data = df.merge(customer_keys, left_on='Customer ID', right_on='customer_id',
    fact_data = fact_data.merge(product_keys, left_on='Product ID', right_on='product_
    fact_data = fact_data.merge(geography_keys, on=['Country', 'Region', 'State', 'Cit

```

```

fact_data = fact_data.merge(ship_mode_keys, left_on='Ship Mode', right_on='ship_mo

# Prepare fact records
fact_records = []
for _, row in fact_data.iterrows():
    fact_records.append({
        'customer_key': row['customer_key'],
        'product_key': row['product_key'],
        'geography_key': row['geography_key'],
        'ship_mode_key': row['ship_mode_key'],
        'order_date_key': row['order_date_key'],
        'ship_date_key': row['ship_date_key'],
        'order_id': row['Order ID'],
        'row_id': row['Row ID'],
        'sales_amount': row['Sales'],
        'quantity': row['Quantity'],
        'discount_amount': row['Discount'],
        'profit_amount': row['Profit'],
        'unit_price': row['Sales'] / row['Quantity'],
        'profit_margin': row['Profit'] / row['Sales'] if row['Sales'] != 0 else 0
    })

# Load to fact table
fact_df = pd.DataFrame(fact_records)
fact_df.to_sql('fact_sales', engine, if_exists='append', index=False)

print(f"Loaded {len(fact_records)} fact records into fact_sales")

# Execute fact loading
load_fact_table()

```

## Business Intelligence Queries

python

```

def run_business_queries():
    """Execute common business intelligence queries"""

    with engine.connect() as conn:

        # Query 1: Top customers by revenue
        top_customers = pd.read_sql("""
            SELECT
                dc.customer_name,
                dc.segment,
                SUM(fs.sales_amount) as total_revenue,
                SUM(fs.profit_amount) as total_profit,
                COUNT(*) as transaction_count,
                AVG(fs.profit_margin) as avg_profit_margin
            FROM fact_sales fs
            JOIN dim_customer dc ON fs.customer_key = dc.customer_key
            WHERE dc.is_current = TRUE
            GROUP BY dc.customer_name, dc.segment
            ORDER BY total_revenue DESC
            LIMIT 20;
        """, conn)

        print("Top 20 Customers by Revenue:")
        print(top_customers)

        # Query 2: Product performance by category
        product_performance = pd.read_sql("""
            SELECT
                dp.category,
                dp.sub_category,
                COUNT(DISTINCT dp.product_id) as product_count,
                SUM(fs.sales_amount) as total_sales,
                SUM(fs.profit_amount) as total_profit,
                AVG(fs.profit_margin) as avg_profit_margin
            FROM fact_sales fs
            JOIN dim_product dp ON fs.product_key = dp.product_key
            WHERE dp.is_current = TRUE
            GROUP BY dp.category, dp.sub_category
            ORDER BY total_sales DESC;
        """, conn)

        print("\nProduct Performance by Category:")
        print(product_performance)

```

```

# Query 3: Sales trends by time
sales_trends = pd.read_sql("""
    SELECT
        dd.year_number,
        dd.quarter_number,
        SUM(fs.sales_amount) as quarterly_sales,
        SUM(fs.profit_amount) as quarterly_profit,
        COUNT(DISTINCT fs.customer_key) as unique_customers
    FROM fact_sales fs
    JOIN dim_date dd ON fs.order_date_key = dd.date_key
    GROUP BY dd.year_number, dd.quarter_number
    ORDER BY dd.year_number, dd.quarter_number;
""", conn)

print("\nSales Trends by Quarter:")
print(sales_trends)

# Execute business queries
run_business_queries()

```

## Slowly Changing Dimensions Implementation

python

```
def implement_scd_type2():
    """Demonstrate SCD Type 2 implementation"""

    # Simulate a customer segment change
    print("Implementing SCD Type 2 for Customer Dimension")

    with engine.connect() as conn:
        # Step 1: Mark old record as expired
        conn.execute("""
            UPDATE dim_customer
            SET expiry_date = CURRENT_DATE - 1,
                is_current = FALSE
            WHERE customer_id = 'CG-12520'
            AND is_current = TRUE;
        """)

        # Step 2: Insert new record with changed segment
        conn.execute("""
            INSERT INTO dim_customer (customer_id, customer_name, segment, effective_date)
            VALUES ('CG-12520', 'Claire Gute', 'Corporate', CURRENT_DATE);
        """)

        # Step 3: Verify the change
        result = pd.read_sql("""
            SELECT customer_id, customer_name, segment,
                effective_date, expiry_date, is_current
            FROM dim_customer
            WHERE customer_id = 'CG-12520'
            ORDER BY effective_date;
        """, conn)

        print("SCD Type 2 Implementation Result:")
        print(result)

        conn.commit()

    # Demonstrate SCD implementation
    implement_scd_type2()
```

## Data Quality Validation

python

```

def validate_data_quality():
    """Implement comprehensive data quality checks"""

    quality_results = []

    with engine.connect() as conn:
        # Check 1: Referential integrity
        orphan_facts = pd.read_sql("""
            SELECT COUNT(*) as orphan_count
            FROM fact_sales fs
            LEFT JOIN dim_customer dc ON fs.customer_key = dc.customer_key
            WHERE dc.customer_key IS NULL;
        """, conn)

        quality_results.append({
            'check': 'Referential Integrity - Customers',
            'result': orphan_facts.iloc[0]['orphan_count'],
            'status': 'PASS' if orphan_facts.iloc[0]['orphan_count'] == 0 else 'FAIL'
        })

        # Check 2: Data completeness
        null_sales = pd.read_sql("""
            SELECT COUNT(*) as null_count
            FROM fact_sales
            WHERE sales_amount IS NULL OR sales_amount <= 0;
        """, conn)

        quality_results.append({
            'check': 'Sales Amount Validity',
            'result': null_sales.iloc[0]['null_count'],
            'status': 'PASS' if null_sales.iloc[0]['null_count'] == 0 else 'FAIL'
        })

        # Check 3: Dimension completeness
        dim_completeness = pd.read_sql("""
            SELECT
                COUNT(*) as total_customers,
                COUNT(CASE WHEN customer_name IS NULL THEN 1 END) as null_names
            FROM dim_customer;
        """, conn)

        quality_results.append({
            'check': 'Customer Name Completeness',

```



```

        'result': dim_completeness.iloc[0]['null_names'],
        'status': 'PASS' if dim_completeness.iloc[0]['null_names'] == 0 else 'FAIL'
    })

# Display results
quality_df = pd.DataFrame(quality_results)
print("Data Quality Validation Results:")
print(quality_df)

return quality_df

# Run data quality validation
quality_results = validate_data_quality()

```

## Performance Optimization

python

```

def create_indexes_for_performance():
    """Create strategic indexes for query performance"""

    index_sql = [
        "CREATE INDEX IF NOT EXISTS idx_fact_sales_customer ON fact_sales(customer_key)",
        "CREATE INDEX IF NOT EXISTS idx_fact_sales_product ON fact_sales(product_key);",
        "CREATE INDEX IF NOT EXISTS idx_fact_sales_date ON fact_sales(order_date_key);",
        "CREATE INDEX IF NOT EXISTS idx_fact_sales_geography ON fact_sales(geography_key);",
        "CREATE INDEX IF NOT EXISTS idx_dim_customer_id ON dim_customer(customer_id);",
        "CREATE INDEX IF NOT EXISTS idx_dim_product_id ON dim_product(product_id);",
        "CREATE INDEX IF NOT EXISTS idx_dim_date_full_date ON dim_date(full_date);",
        "CREATE INDEX IF NOT EXISTS idx_dim_geography_region ON dim_geography(region, :)",
    ]

    with engine.connect() as conn:
        for sql in index_sql:
            conn.execute(sql)
        conn.commit()

    print("Performance indexes created successfully!")

# Create performance indexes
create_indexes_for_performance()

```

## Foundational Reading

### 1. "The Data Warehouse Toolkit" by Ralph Kimball

- Source: The definitive guide to dimensional modeling
- For more information, refer directly to widely adopted published content, like The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling (3rd edition, 2013) by Ralph Kimball

### 2. Star Schema Design Principles

- Introduced by Ralph Kimball in the 1990s, star schemas are efficient at storing data, maintaining history, and updating data by reducing the duplication of repetitive business definitions
- Source: Databricks Glossary

## Video Learning

### 1. Dimensional Modeling Fundamentals

- To help users interpret, perform analysis, and report on the information contained within a database or relational data model, the star schema data model organizes your data effectively and succinctly
- Platform: Coursera

### 2. Star Schema vs Snowflake Schema

- Understanding trade-offs between different modeling approaches
- Focus: Performance vs maintenance considerations

## Practice Datasets

### 1. Superstore Dataset (Primary)

- **Link:** [kaggle.com/datasets/bravehart101/sample-supermarket-dataset](https://www.kaggle.com/datasets/bravehart101/sample-supermarket-dataset)
- **Use:** Complete star schema implementation

### 2. Customer Analytics (Supporting)

- **Link:** [kaggle.com/datasets/imakash3011/customer-personality-analysis](https://www.kaggle.com/datasets/imakash3011/customer-personality-analysis)
- **Use:** Rich customer dimension design

## Design Tools

### 1. Data Modeling Tools

- **Lucidchart:** Web-based ERD creation

- **draw.io:** Free diagramming tool
- **DBSchema:** Professional database design

## 2. Documentation Tools

- **Data Dictionary Templates:** For dimension/fact documentation
  - **Business Process Documentation:** For requirements gathering
- 

## Day 5 Practical Tasks

### Task 1: Business Requirements Analysis (45 minutes)

- ☐ Identify 3 key business questions for Superstore data
- ☐ Define business processes to be analyzed
- ☐ Determine primary user groups and their needs
- ☐ Document success criteria for the data model

### Task 2: Dimensional Model Design (90 minutes)

- ☐ Design star schema for Superstore sales analysis
- ☐ Define grain for fact table(s)
- ☐ Identify all dimensions and their attributes
- ☐ Plan SCD strategy for each dimension
- ☐ Create logical model diagram

### Task 3: Time Dimension Deep Dive (30 minutes)

- ☐ Design comprehensive date dimension
- ☐ Include calendar and fiscal year support
- ☐ Add business-specific attributes (holidays, seasons)
- ☐ Plan for different date roles in fact table

### Task 4: Model Validation (45 minutes)

- ☐ Validate model against business questions
- ☐ Check for referential integrity
- ☐ Plan data quality validation rules
- ☐ Design performance optimization strategy

### Task 5: Documentation and Presentation (30 minutes)

- ☐ Create business-friendly model documentation
- ☐ Prepare model presentation for stakeholders

- ☐ Document design decisions and trade-offs
  - ☐ Plan implementation roadmap
- 



## Day 5 Deliverables

### 1. Conceptual Understanding

- Star schema design principles mastered
- Fact vs dimension table distinction clear
- SCD strategies and when to use each
- Business-driven modeling approach

### 2. Practical Design Skills

- Complete star schema design for Superstore
- Comprehensive time dimension specification
- SCD implementation strategy
- Performance optimization plan

### 3. Business Alignment

- Model validates against business requirements
- Supports key analytical use cases
- Enables self-service analytics
- Scales for future needs

### 4. Skills Assessment

Rate yourself after today (1-10):

- ☐ Star schema design: \_\_\_\_/10
- ☐ Business requirements analysis: \_\_\_\_/10
- ☐ Dimension design: \_\_\_\_/10
- ☐ Fact table design: \_\_\_\_/10
- ☐ SCD strategy selection: \_\_\_\_/10

### 5. Learning Journal Entry

Create `day-05/learning-notes.md`:

markdown

## # Day 5: Data Modeling – Learning Notes

### ## Key Concepts Mastered

- Star schema design principles and benefits
- Fact vs dimension table design patterns
- Slowly changing dimension strategies
- Business-driven modeling approach
- Time dimension comprehensive design

### ## Design Decisions Made

- Grain selection rationale for Superstore model
- SCD strategy for each dimension
- Performance optimization approaches
- Business requirement alignment

### ## Business Understanding Gained

- How modeling impacts query performance
- Why denormalization improves analytics
- Importance of time dimension design
- Balance between flexibility and performance

### ## Real-World Applications

- Retail analytics data warehouse design
- Customer behavior analysis models
- Product performance monitoring systems
- Operational excellence dashboards

### ## Tomorrow's Preparation

- Review cloud platform basics
- Understand data lake concepts
- Prepare for AWS fundamentals

---

## Tomorrow's Preview: Day 6 - Cloud Platforms Introduction


### What to expect:




- AWS fundamentals for data engineering
- Data lakes vs data warehouses
- Cloud storage and compute services
- Setting up your first cloud data pipeline

- Cost optimization strategies

### Preparation:

- Create AWS free tier account
  - Review cloud computing basics
  - Understand data storage options
- 

 Congratulations on completing Day 5! You now understand how to design data models that scale with business needs. Tomorrow, we'll move our models to the cloud!

**Progress:** 10% (5/50 days) | **Next:** Day 6 - Cloud Platforms | **Skills:** Python  + SQL  + Advanced SQL  + Data Modeling 