**Lab Document: Dimensional Modeling – Hands-on Exercises**

**Lab 1: Declare the Grain**

**Objective**: Understand and declare the grain of your fact table.

**Steps:**

1. **Read the Business Requirement**:
   * Example: "We want to analyze daily product sales at each store."
2. **Declare the Grain**:
   * **Grain**: One row per product per store per day.
3. **Document the Grain**:
   * Format:

sql

FACT\_SALES\_GRAIN: One row per [Product, Store, Day]

*Grain impacts all downstream modeling decisions.*

**Lab 2: Star vs. Snowflake vs. Galaxy Schema**

**Objective**: Model schemas in star, snowflake, and galaxy formats.

**Dataset:**

* Tables: Sales, Product, Customer, Location, Category

**Tasks:**

1. **Star Schema**:
   * Flatten all dimension tables and connect them directly to Sales.
2. **Snowflake Schema**:
   * Normalize Product into Product and Category.
   * Normalize Location into Store, City, Region.
3. **Galaxy Schema**:
   * Create a second fact table: Inventory.
   * Link Product, Location to both Sales and Inventory.

Use an ERD tool (e.g., dbdiagram.io or draw.io) to visualize.

**Lab 3: Degenerate & Junk Dimensions**

**Objective**: Identify and create degenerate and junk dimensions.

**Dataset:**

* SalesFact includes: InvoiceNo, PaymentType, ReturnFlag, ShippingFlag

**Steps:**

1. **Degenerate Dimension**:
   * Retain InvoiceNo in fact table (no join).
2. **Junk Dimension**:
   * Combine PaymentType, ReturnFlag, and ShippingFlag into DimJunk.

**SQL:**

sql

CREATE TABLE dim\_junk AS

SELECT DISTINCT payment\_type, return\_flag, shipping\_flag

FROM sales\_fact;

**Lab 4: Slowly Changing Dimensions (SCD) – Type 1, 2, 6**

**Objective**: Implement different SCD types in a dimension.

**Use Case:**

* DimCustomer – track customer name and address changes

**Type 1:**

sql

UPDATE dim\_customer

SET customer\_name = 'New Name'

WHERE customer\_id = 1001;

**Type 2:**

sql

UPDATE dim\_customer

SET is\_current = false, end\_date = CURRENT\_DATE

WHERE customer\_id = 1001 AND is\_current = true;

INSERT INTO dim\_customer (customer\_id, customer\_name, is\_current, start\_date)

VALUES (1001, 'New Name', true, CURRENT\_DATE);

**Type 6:**

* Add current\_flag, effective\_date, end\_date, and also previous\_address column to dimension.
* Populate previous\_address during updates.

*SCD Type 2 is most common for history tracking.*

**Lab 5: Surrogate Key vs. Natural Key**

**Objective**: Use surrogate keys in dimension modeling.

**Task:**

1. Create dim\_product with surrogate key
2. Replace all natural key joins in fact table

**SQL:**

sql

-- Step 1: Surrogate Key

CREATE SEQUENCE product\_sk\_seq START WITH 1;

-- Step 2: Dimension Table

INSERT INTO dim\_product (product\_sk, product\_code, product\_name)

VALUES (nextval('product\_sk\_seq'), 'P101', 'Laptop');

-- Step 3: Fact Table with surrogate join

INSERT INTO fact\_sales (product\_sk, date, quantity)

VALUES (1, '2023-01-01', 10);

**Lab 6: Build a Bus Matrix**

**Objective**: Design a dimensional bus matrix for an enterprise warehouse.

**Use Case:**

Company tracks Sales, Inventory, Delivery.

**Steps:**

1. List conformed dimensions: Date, Product, Customer, Location
2. List business processes: Sales, Inventory, Delivery
3. Fill the matrix:

| **Business Process** | **Date** | **Product** | **Customer** | **Location** |
| --- | --- | --- | --- | --- |
| Sales | Yes | Yes | Yes | Yes |
| Inventory | Yes | Yes | No | Yes |
| Delivery | Yes | Yes | Yes | Yes |

Use this to plan and design multiple fact tables.

**Lab 7: Surrogate Key Generator in PySpark**

**Objective**: Create surrogate keys during ETL using PySpark.

**Dataset: Load from CSV**

python

from pyspark.sql import SparkSession

from pyspark.sql.functions import monotonically\_increasing\_id

spark = SparkSession.builder.getOrCreate()

df = spark.read.csv("products.csv", header=True)

# Generate surrogate key

df\_sk = df.withColumn("product\_sk", monotonically\_increasing\_id())

df\_sk.show()

**Final Lab Challenge**

**Scenario:**

You are building a **retail warehouse**. Perform the following:

* Declare grain for SalesFact
* Create DimProduct, DimCustomer, DimDate with surrogate keys
* Build a star schema in MySQL/Postgres or Spark
* Implement SCD Type 2 on DimCustomer
* Create a bus matrix for Sales, Inventory, Returns

**Deliverables**

* SQL scripts or PySpark notebook
* Star Schema diagram (use dbdiagram.io or Lucidchart)
* Final bus\_matrix.csv