

| Business Template  **Star vs Snowflake** |
| --- |
|  |

**Contents**

[**1 3NF Schema**](#_heading=h.e19xqlcb2hpk) **3**

[**2 4-Step Dimensional Design Process**](#_heading=h.lo4oygxeb6ul) **4**

[1. Selecting the Business Process](#_heading=h.go2ymkmn6tc0) 4

[2. Declaring the Grain](#_heading=h.vetkmz7cm7k2) 4

[3. Identifying the Dimensions](#_heading=h.dpbprqxqhc7r) 4

[4. Identifying the Facts](#_heading=h.ytmhvkg1bknu) 5

[**3 Denormalization**](#_heading=h.8w4wmx2sjpux) **5**

[1. Flattening Dimensions](#_heading=h.b18gfk4pricd) 5

[2. Handling Relationships Between Dimensions](#_heading=h.6uu7moxgocs9) 5

[3. Flattening Hierarchies](#_heading=h.yyg8cki1alsh) 6

[**3 Star Schema**](#_heading=h.1i405wujhtbh) **6**

[**3 Normalization**](#_heading=h.x4gdcqr0feq) **7**

[1. Identify hierarchies in dimension tables and flatten them](#_heading=h.8qrvoc61gv02) 7

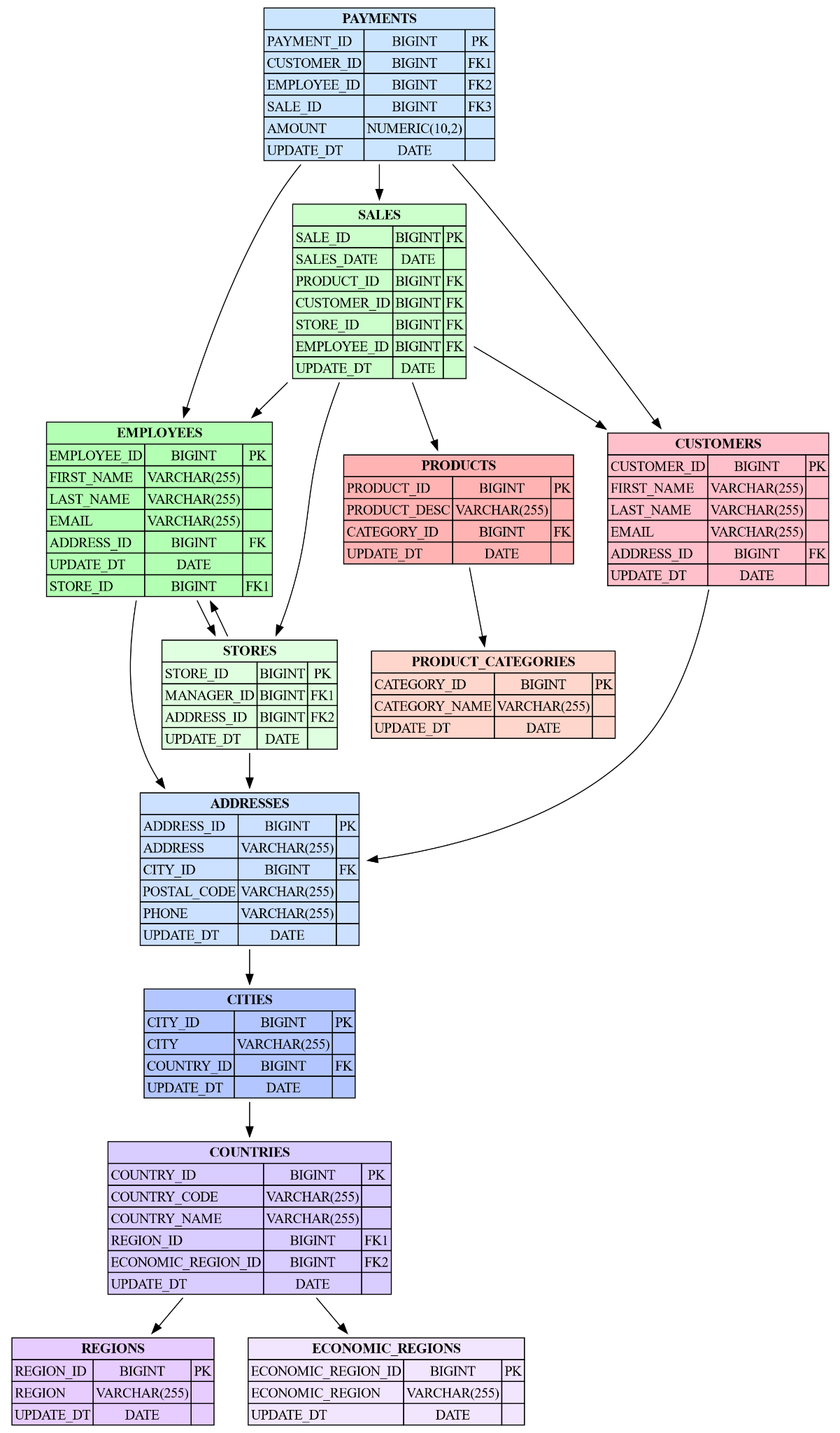
[2. Normalize dimension tables](#_heading=h.xpr6opigzzye) 7

[**4 Snowflake schema**](#_heading=h.jfkglwyd5vvg) **8**

# 

# 3NF Schema

# 



# 2 4-Step Dimensional Design Process

In deriving our star model starting from a 3NF Model, we first consider the following 4-Step Dimensional Process:

## 1. Selecting the Business Process

The business process we are analyzing is:

***"How many products have been sold in terms of units for a single payment per sale in a day?"***

In this context, we are specifically interested in analyzing the payment made during a transaction.

## 2. Declaring the Grain

We define the grain of the fact table as a single payment of a specific sale by a customer, processed by an employee for a specific product at a specific store in a day.

Since there is a one-to-many relationship between the sales table (our fact table) and the payment table, this violates the star schema principle that each fact record must correspond to a unique combination of dimension keys. To resolve this, we treat **payment\_id as a degenerate dimension and PK** within the sales fact table. A degenerate dimension is an identifier stored in the fact table without a corresponding dimension table because it carries no descriptive attributes.

By embedding payment\_id directly in the fact table and defining the composite key (sale\_id,payment\_id), the grain shifts to the payment-sales level. The grain is now uniquely identified by the foreign keys: customer\_id, product\_id, employee\_id, store\_id, plus the composite key dimension (sale\_id,payment\_id).

This denormalization procedure preserves transactional granularity and simplifies the schema, following Kimball’s dimensional modeling best practices.

## 3. Identifying the Dimensions

To determine which dimensions are required, we apply the guiding questions:

***Where? Who buys? Who sells? What? When?***

We identify the following core dimensions:

* **Customer Dimension (Who buys?)** Includes information about customers such as first name, last name, email, phone, and address\_id.
* **Employee Dimension (Who sells?)** Includes information about employees such as first name, last name, email, phone, and address\_id.
* **Store Dimension (Where?)** Includes information about stores such as store\_id, manager\_id, and address\_id.
* **Product Dimension (What?)** Includes information about products such as product\_id, product description, category\_id, and category name.

## 4. Identifying the Facts

To identify the facts, we ask:

***"What is the business process measuring?"***

Additive facts (aggregatable) include:

* **Amount:** number of units sold(i.e. Recovered from the payment table)

# 3 Denormalization

To convert our normalized schema into a star schema suitable for analytical querying, we apply several denormalization steps. These include flattening dimensions, resolving inter-dimensional relationships, and managing attribute hierarchies.

## 1. Flattening Dimensions

The first step is to **flatten sub-dimensions** into their respective core dimensions. This involves identifying related tables that can be merged, thereby simplifying the model and reducing the number of joins required.

For instance, the **product category** table, which was separate in the normalized model, is now merged into the **product dimension**. As a result, attributes like category\_id and category\_name are stored directly in the product dimension table.

## 2. Handling Relationships Between Dimensions

In dimensional modeling, we focus on the relationships between facts and dimensions. Direct relationships between dimensions—especially many-to-one or hierarchical ones—are generally avoided or simplified.

In our normalized model, there is a one-to-many relationship from stores to employees: each store has multiple employees, and each employee is assigned to one store. In the dimensional model, this relationship becomes redundant because the store associated with each employee involved in a transaction can be inferred through the fact table. Therefore, we eliminate this direct relationship and instead treat store\_id as a regular attribute in the employee dimension.

To include information about **store managers**, we assume that **each store has exactly one manager**, and that the manager is also an employee. This introduces a **one-to-one relationship between manager\_id and an existing employee\_id**.

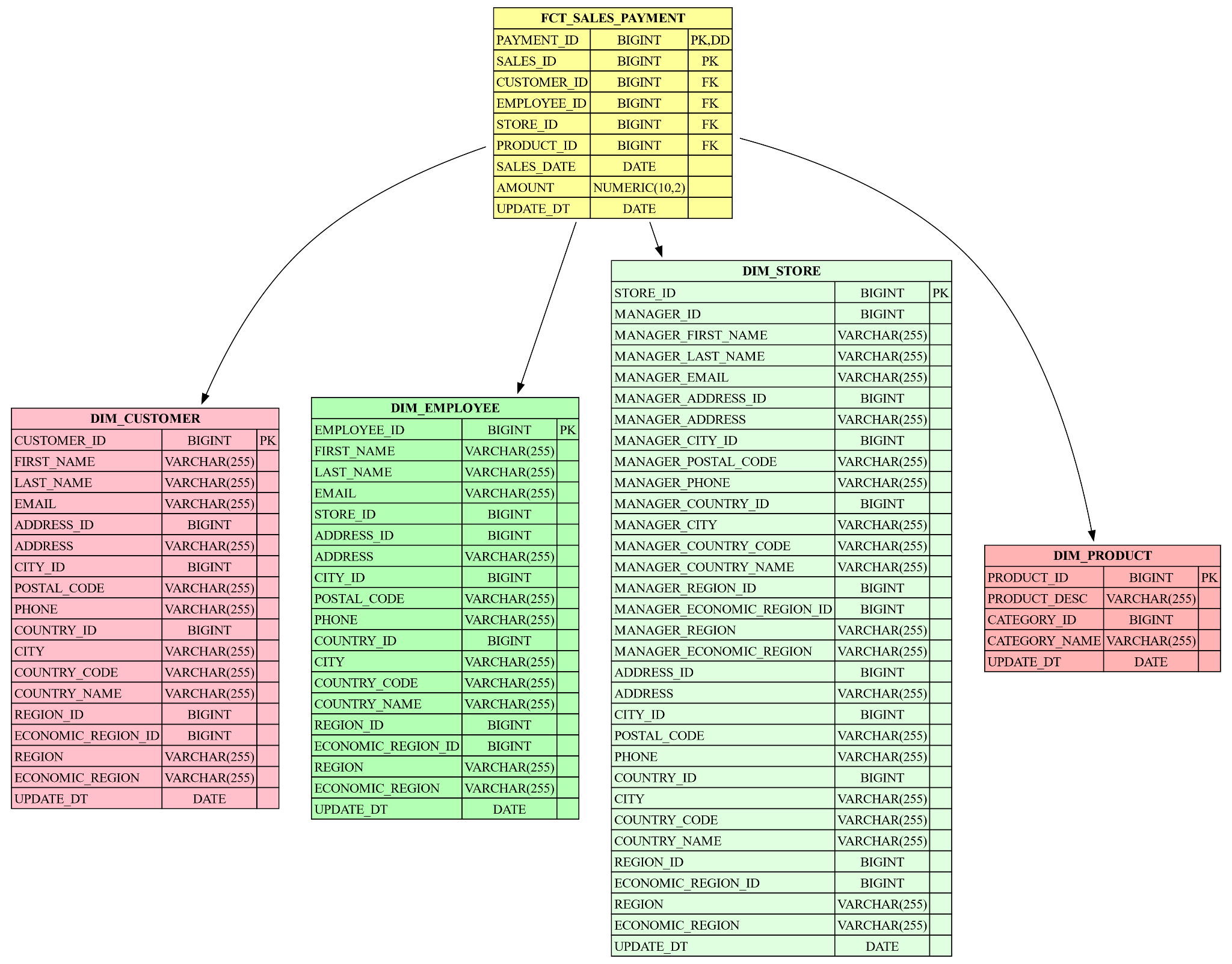
Instead of creating a separate link table or one-to-one relationship, we simply store manager\_id as a **foreign key turned attribute in the store dimension**, and retrieve the manager’s attributes by joining with the **employee dimension**. This keeps store\_id as the primary key in the store dimension and avoids unnecessary complexity.

## 3. Flattening Hierarchies

Finally, we address **attribute hierarchies**. In our case, we have a **location hierarchy** consisting of levels such as city, state, and country.

Rather than modeling this as a separate normalized location table, we **flatten the location hierarchy directly into the relevant dimensions**: **customer**, **store**, and **employee**. This allows for flexible geographic analysis (e.g., sales by city or region) without additional joins or lookups.

# 3 Star Schema



# 3 Normalization

To convert our denormalized star schema into a snowflake schema suitable for analytical querying, we apply several normalization steps. The principle of snowflaking is to normalize dimension tables by removing low cardinality attributes and forming separate related tables. The steps we follow are:

## 1. Identify hierarchies in dimension tables and flatten them

For example, one such hierarchy is the location hierarchy that we also identified during the denormalization steps. This hierarchy needs to be restored to its initial state (3NF), but this time it is related to each different table it comes from separately through a one-to-many relationship.

## 2. Normalize dimension tables

After identifying hierarchies and normalizing each dimension, we further normalize them by identifying other subdimensions or relationships that were previously denormalized.  
 For example, the Product dimension can be normalized further into a Product dimension and a Product Category subdimension. Similarly, the Store dimension can be further normalized into a Store dimension and a Manager subdimension. Within the Manager subdimension, the location hierarchy will also be further normalized. We also append foreign keys that will connect the dimension to its subdimeansions.  
 This process results in a snowflake schema as follows.

# 4 Snowflake schema

