Author: Madhurima Rawat

Creating a Snowflake Schema in Data Warehouse

This experiment involves designing and implementing a snowflake schema for a retail data warehouse. The schema applies higher normalization than a star schema, reducing redundancy by breaking dimension tables into multiple related tables. This enhances data integrity but may impact query performance.

Snowflake Schema vs. Star Schema in Data Warehousing

A **data warehouse** organizes large volumes of data for analytical processing. Two widely used schema models in data warehousing are the **Star Schema** and the **Snowflake Schema**. Understanding these schemas is essential for designing efficient data warehouses that balance **query performance, storage efficiency, and data integrity**.

1. What is a Star Schema?

A **Star Schema** is a **denormalized** database structure where the **fact table** is surrounded by **dimension tables**. Each dimension table directly connects to the fact table without further normalization.

Key Characteristics of a Star Schema

1. **Denormalized Structure**: Dimension tables are not split into sub-tables, allowing direct access.
2. **Faster Query Performance**: Since there are fewer joins, queries execute quickly.
3. **Higher Redundancy**: Dimension tables store repeated data, leading to increased storage usage.
4. **Easier to Understand and Use**: The simple structure makes it ideal for business intelligence tools.

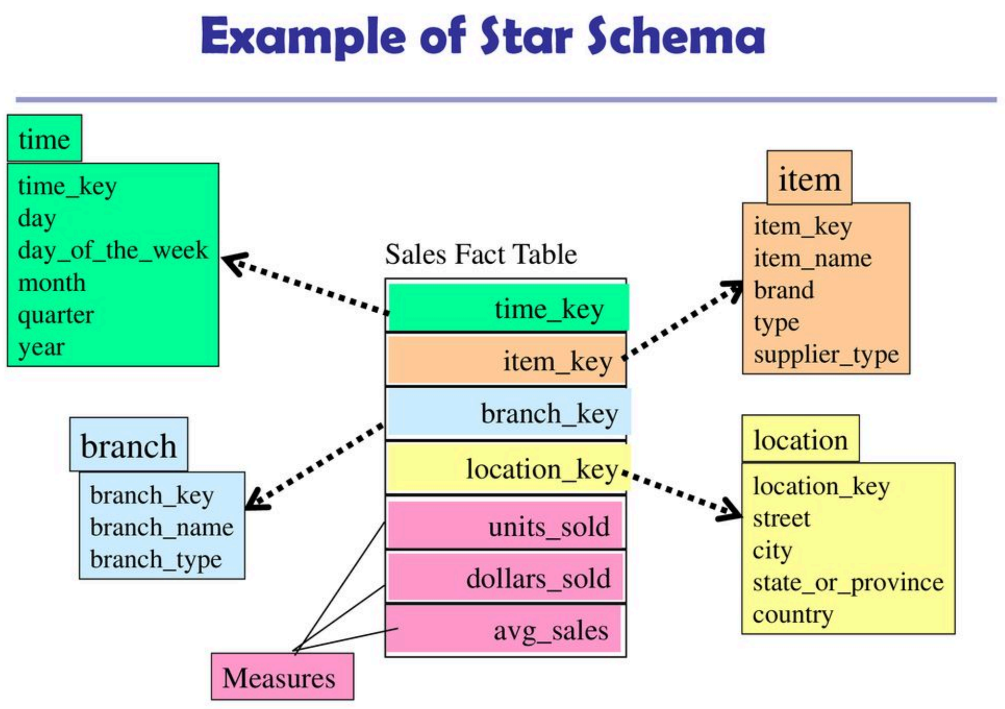
Star Schema Diagram

```
                    +----------------+
                    |   Time Dim     |
                    +----------------+
                           |
  +-------------+     +-------------+     +-------------+
  | Product Dim |     |  Fact Table |     | Customer Dim |
  +-------------+ ->  |  (Sales)    | <-  +-------------+
                           |
                    +-------------+
                    | Store Dim    |
                    +--------------+
```

Example of a Star Schema

- **Fact Table (Sales Data)**
  - sale_id
  - product_id
  - customer_id
  - store_id
  - time_id
  - amount
  - quantity
- **Dimension Tables**
  - `product_dim` : product_id, product_name, category, brand
  - `customer_dim` : customer_id, customer_name, city, region
  - `time_dim` : time_id, date, day_of_week, month, quarter, year
  - `store_dim` : store_id, store_name, location

This schema allows **fast queries**, but since attributes like `category` and `region` are stored directly in dimension tables, there is **redundancy**.



1. **Fact Table**: The central table that contains quantitative data (measures) for analysis. It is typically large and has foreign keys linking to dimension tables.
2. **Dimension Tables**: Smaller tables that store descriptive attributes (context) about the measures in the fact table. These provide insights like "who," "what," "where," and "when."

The star schema is called so because its structure resembles a star, with the fact table at the center and the dimension tables radiating outwards.

2. What is a Snowflake Schema?

A **Snowflake Schema** is a **normalized** version of the Star Schema where dimension tables are further split into multiple related tables. This reduces **data redundancy** but increases **query complexity** due to additional joins.

Key Characteristics of a Snowflake Schema

1. **Normalized Structure**: Each dimension table is broken into smaller sub-dimension tables.
2. **Reduced Data Redundancy**: Data is stored efficiently, avoiding duplicate values.
3. **More Complex Queries**: Queries require multiple joins to access information.
4. **Better Data Integrity**: Since data is normalized, inconsistencies are minimized.

Snowflake Schema Diagram

```
                    +-----------------+
                    |      Time       |
                    +-----------------+
                           |
+-------------+     +-------------+     +-------------+
| Product Cat |     | Fact Table  |     | Customer    |
| Product Dim | ->  | (Sales Data)| <-  | City        |
| Brand       |     +-------------+     | Region      |
                           |
                    +-------------+
                    | Store        |
                    +-------------+
```

Example of a Snowflake Schema

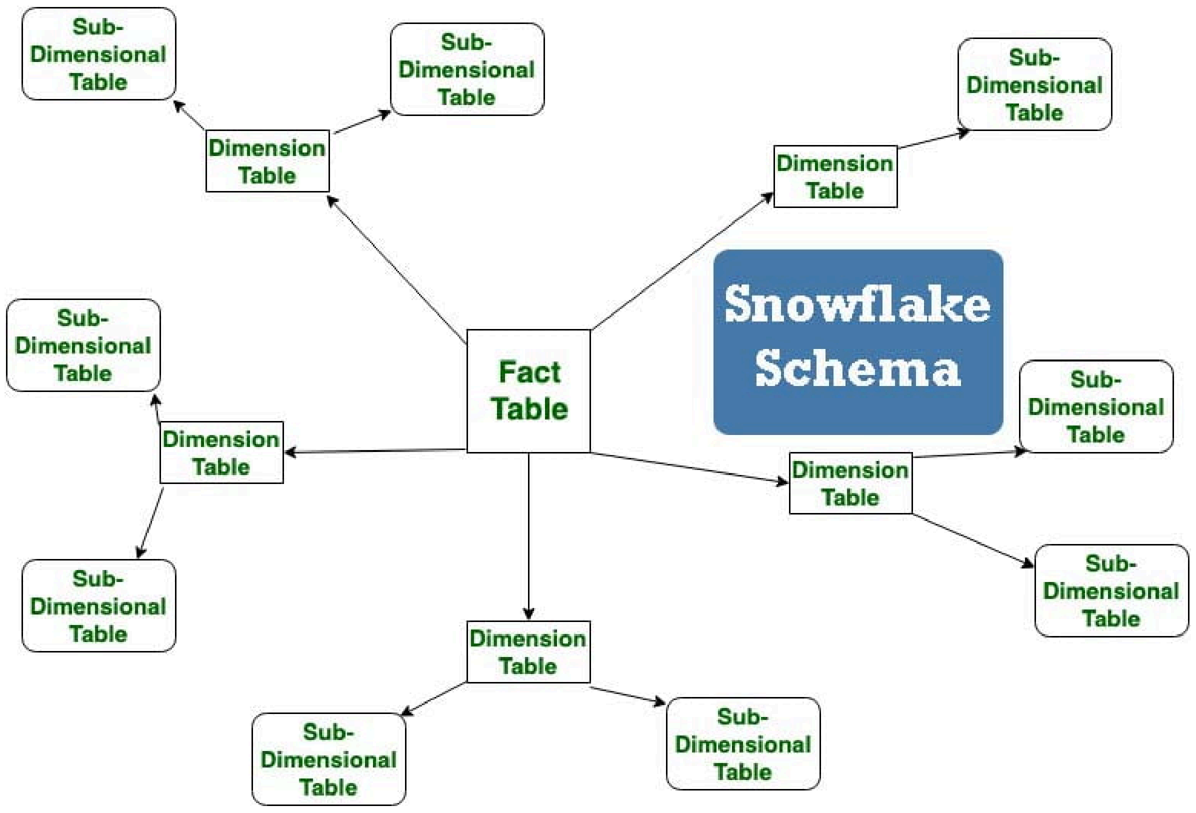
- **Fact Table (Sales Data)**
  - sale_id
  - product_id
  - customer_id
  - store_id

    - time_id
    - amount
    - quantity

- **Dimension Tables** (Normalized)

    - `product_dim` : product_id, product_name, category_id, brand_id
    - `product_category` : category_id, category_name
    - `brand` : brand_id, brand_name
    - `customer_dim` : customer_id, customer_name, city_id
    - `city` : city_id, city_name, region_id
    - `region` : region_id, region_name
    - `time_dim` : time_id, date, day_of_week, month, quarter, year
    - `store_dim` : store_id, store_name, location

By breaking dimensions into smaller tables (`product_category` , `region`), **storage efficiency increases**, but **queries require additional joins**.



3. Normalization in Database Design

Normalization is the process of structuring a relational database to **reduce redundancy** and **improve data integrity**. It involves organizing tables based on rules known as **Normal Forms (NF)**.

Key Goals of Normalization

1. **Minimize Data Redundancy**: Avoid storing the same data multiple times.
2. **Ensure Data Consistency**: Changes in one place automatically reflect across related tables.
3. **Optimize Storage Space**: Efficient use of disk space.
4. **Improve Data Integrity**: Prevent anomalies like insertion, deletion, and update inconsistencies.

4. Normal Forms in a Snowflake Schema

First Normal Form (1NF)

- **Each column contains atomic (indivisible) values**.
- **Each row has a unique identifier (primary key)**.

Example: Non-Atomic Values (Not in 1NF)

```
| product_id | product_name | categories         |
|------------|-------------|--------------------|
| 1          | Laptop      | Electronics, Tech |
```

Transformed into 1NF

```
| product_id | product_name | category_id |
|------------|-------------|------------|
| 1          | Laptop      | 101        |

| category_id | category_name  |
|------------|---------------|
| 101        | Electronics   |
```

Second Normal Form (2NF)

- **Must be in 1NF**.
- **No partial dependencies** (all non-key attributes must fully depend on the primary key).

Example: Partial Dependency (Not in 2NF)

```
| sale_id | product_id | customer_id | store_id | region |
```

- **Region depends only on** `store_id`, not the full primary key.

Transformed into 2NF

sale_id	product_id	customer_id	store_id

store_id	region

Third Normal Form (3NF)

- **Must be in 2NF**.
- **No transitive dependencies** (non-key attributes should not depend on other non-key attributes).

Example: Transitive Dependency (Not in 3NF)

customer_id	customer_name	city	region

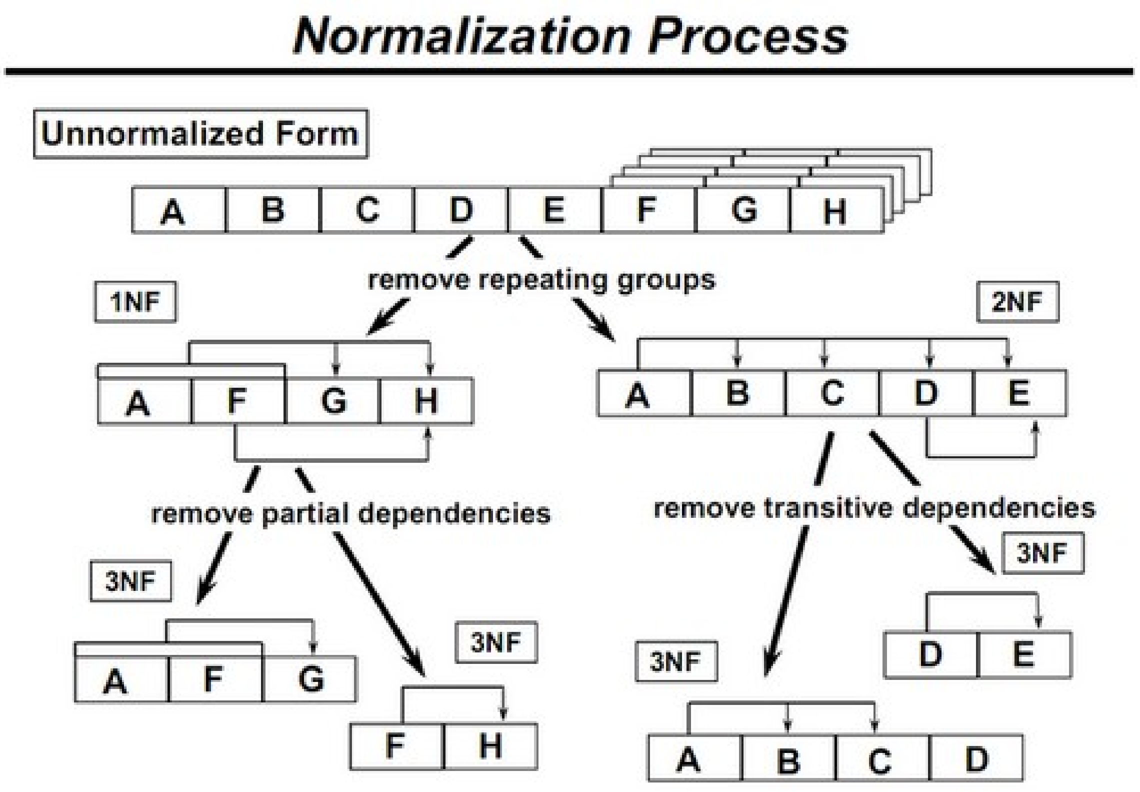
- `region` is dependent on `city`, not `customer_id`.

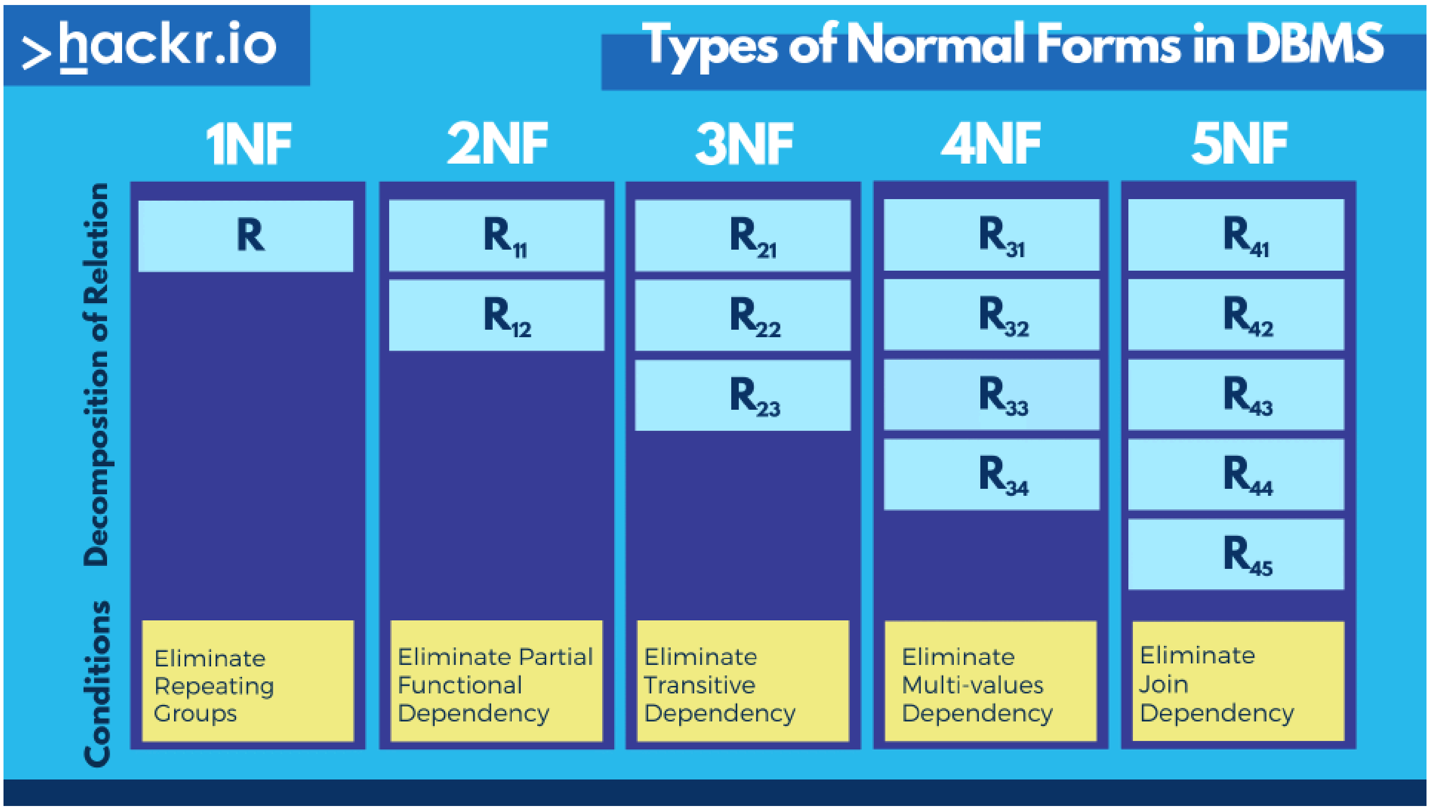
Transformed into 3NF

customer_id	customer_name	city_id

city_id	city	region_id

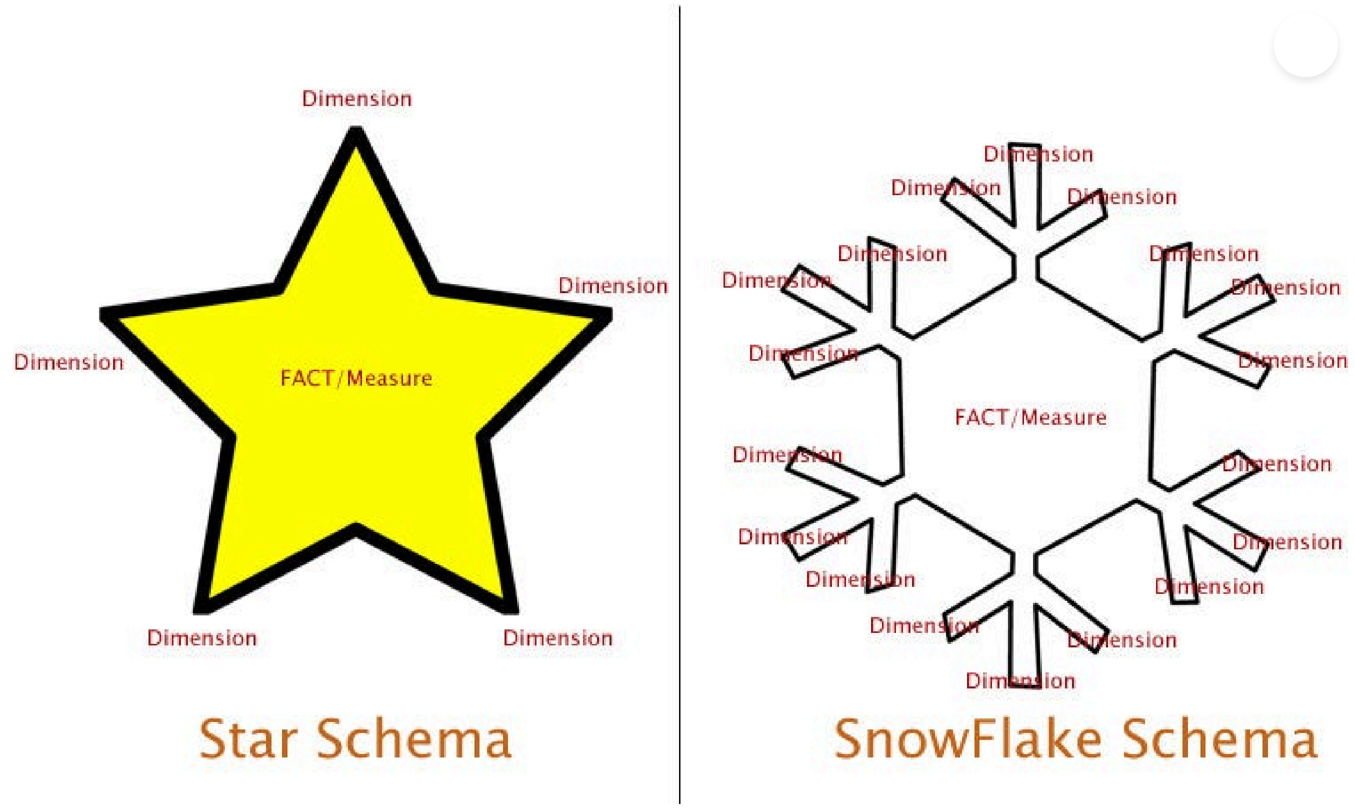
region_id	region

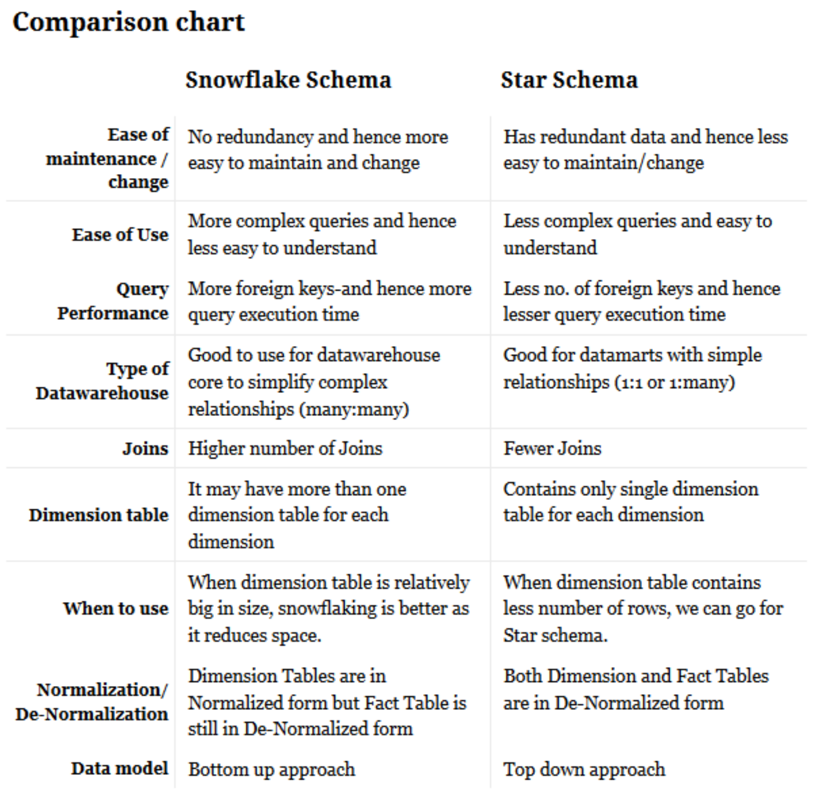



>hackr.io
Types of Normal Forms in DBMS
Decomposition of Relation
Conditions
1NF
2NF
3NF
4NF
5NF
R
R_{11}
R_{12}
R_{21}
R_{22}
R_{23}
R_{31}
R_{32}
R_{33}
R_{34}
R_{41}
R_{42}
R_{43}
R_{44}
R_{45}
Eliminate Repeating Groups
Eliminate Partial Functional Dependency
Eliminate Transitive Dependency
Eliminate Multi-values Dependency
Eliminate Join Dependency

5. Comparison: Star Schema vs. Snowflake Schema

Feature	Star Schema	Snowflake Schema
Normalization	Low (Denormalized)	High (Normalized)
Data Redundancy	More Redundant	Less Redundant
Query Complexity	Simple Queries	More Joins Required
Storage Efficiency	Uses More Space	Uses Less Space
Performance	Faster Queries	Slightly Slower Queries




Comparison chart

	Snowflake Schema	Star Schema
Ease of maintenance / change	No redundancy and hence more easy to maintain and change	Has redundant data and hence less easy to maintain/change
Ease of Use	More complex queries and hence less easy to understand	Less complex queries and easy to understand
Query Performance	More foreign keys-and hence more query execution time	Less no. of foreign keys and hence lesser query execution time
Type of Datawarehouse	Good to use for datawarehouse core to simplify complex relationships (many:many)	Good for datamarts with simple relationships (1:1 or 1:many)
Joins	Higher number of Joins	Fewer Joins
Dimension table	It may have more than one dimension table for each dimension	Contains only single dimension table for each dimension
When to use	When dimension table is relatively big in size, snowflaking is better as it reduces space.	When dimension table contains less number of rows, we can go for Star schema.
Normalization/ De-Normalization	Dimension Tables are in Normalized form but Fact Table is still in De-Normalized form	Both Dimension and Fact Tables are in De-Normalized form
Data model	Bottom up approach	Top down approach

6. Choosing Between Star and Snowflake Schema

Use Case	Best Schema
Fast Query Performance	Star Schema

Use Case	Best Schema
Storage Optimization	Snowflake Schema
Simple Reporting Needs	Star Schema
Strict Data Integrity	Snowflake Schema

In summary, **Star Schema is best for fast query performance and simpler analytics**, while **Snowflake Schema is better for maintaining data integrity and reducing redundancy**.

MySQL Implementation of Snowflake Schema

1. Creating the Database

Command

```
CREATE DATABASE RetailDataWarehouse;
USE RetailDataWarehouse;
```

Explanation

- `CREATE DATABASE RetailDataWarehouse;`
  - This command creates a new database named **RetailDataWarehouse**.
- `USE RetailDataWarehouse;`
  - This selects the created database so that all subsequent commands are executed inside it.

Expected Output

```
Query OK, 1 row affected (0.01 sec)
Database changed
```

2. Creating Normalized Dimension Tables

Product Dimension Table (Normalized)

Step 1: Creating the Product Category Table

```
CREATE TABLE product_category (
    category_id INT PRIMARY KEY AUTO_INCREMENT,
    category_name VARCHAR(100) NOT NULL
);
```

Explanation

- `category_id INT PRIMARY KEY AUTO_INCREMENT`
  - This is the primary key that uniquely identifies each product category.
  - The `AUTO_INCREMENT` ensures each new row gets a unique increasing number.
- `category_name VARCHAR(100) NOT NULL`
  - This column stores the name of the product category, such as "Electronics" or "Furniture".
- `NOT NULL` means that the column **cannot be left empty** when inserting data.

Expected Output

```
Query OK, 0 rows affected (0.02 sec)
```

Step 2: Creating the Product Table (Normalized)

```
CREATE TABLE product_dim_normalized (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(100) NOT NULL,
    category_id INT,
    brand VARCHAR(50),
    FOREIGN KEY (category_id) REFERENCES product_category(category_id)
);
```

Explanation

- `product_id INT PRIMARY KEY AUTO_INCREMENT`
  - Unique identifier for each product, automatically assigned.
- `product_name VARCHAR(100) NOT NULL`
  - Stores the product name, such as "Laptop" or "Table".
- `category_id INT`
  - This links to the `category_id` in the `product_category` table to **avoid storing repeated category names**.
- `brand VARCHAR(50)`
  - Stores the brand name of the product.
- `FOREIGN KEY (category_id) REFERENCES product_category(category_id)`
  - Establishes a **foreign key constraint**, ensuring that every product belongs to a valid category.

Expected Output

```
Query OK, 0 rows affected (0.03 sec)
```

Customer Dimension Table (Normalized)

Step 1: Creating the Region Table

```sql
CREATE TABLE region (
    region_id INT PRIMARY KEY AUTO_INCREMENT,
    region_name VARCHAR(100) NOT NULL
);
```

Explanation

- `region_id INT PRIMARY KEY AUTO_INCREMENT`
    - Unique identifier for each region.
- `region_name VARCHAR(100) NOT NULL`
    - Stores the region name, such as "North" or "South".

Expected Output

```
Query OK, 0 rows affected (0.01 sec)
```

Step 2: Creating the City Table

```sql
CREATE TABLE city (
    city_id INT PRIMARY KEY AUTO_INCREMENT,
    city_name VARCHAR(100) NOT NULL,
    region_id INT,
    FOREIGN KEY (region_id) REFERENCES region(region_id)
);
```

Explanation

- `city_id INT PRIMARY KEY AUTO_INCREMENT`
    - Unique identifier for each city.
- `city_name VARCHAR(100) NOT NULL`
    - Stores the city name, such as "Delhi" or "Mumbai".
- `region_id INT`

    - Links each city to its respective region.
- `FOREIGN KEY (region_id) REFERENCES region(region_id)`
    - Ensures that each city belongs to a valid region.

Expected Output

```
Query OK, 0 rows affected (0.02 sec)
```

Step 3: Creating the Customer Table (Normalized)

```
CREATE TABLE customer_dim_normalized (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_name VARCHAR(100) NOT NULL,
    city_id INT,
    FOREIGN KEY (city_id) REFERENCES city(city_id)
);
```

Explanation

- `customer_id INT PRIMARY KEY AUTO_INCREMENT`
    - Unique identifier for each customer.
- `customer_name VARCHAR(100) NOT NULL`
    - Stores the name of the customer.
- `city_id INT`
    - Links each customer to a city.
- `FOREIGN KEY (city_id) REFERENCES city(city_id)`
    - Ensures that every customer is linked to an existing city.

Expected Output

```
Query OK, 0 rows affected (0.02 sec)
```

Time Dimension Table (Normalized)

```
CREATE TABLE time_dim_normalized (
    time_id INT PRIMARY KEY AUTO_INCREMENT,
    date DATE NOT NULL,
    day_of_week VARCHAR(10),
    month VARCHAR(10),
    quarter VARCHAR(10),
    year INT
);
```

Explanation

- Stores information about time-related attributes such as day, month, and year.
- Helps in analyzing sales trends over different time periods.

Expected Output

```
Query OK, 0 rows affected (0.02 sec)
```

3. Creating the Fact Table

```
CREATE TABLE sales_fact_normalized (
    sale_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    customer_id INT,
    store_id INT,
    time_id INT,
    amount DECIMAL(10, 2),
    quantity INT,
    FOREIGN KEY (product_id) REFERENCES product_dim_normalized(product_id),
    FOREIGN KEY (customer_id) REFERENCES customer_dim_normalized(customer_id),
    FOREIGN KEY (time_id) REFERENCES time_dim_normalized(time_id)
);
```

Explanation

- This table records **actual sales transactions**, linking to all normalized dimension tables.
- `amount DECIMAL(10,2)` stores the total sale amount.
- `quantity INT` records the number of items sold.

Expected Output

```
Query OK, 0 rows affected (0.03 sec)
```

4. Inserting Sample Data

```
INSERT INTO product_category (category_name) VALUES ('Electronics');
INSERT INTO product_dim_normalized (product_name,
category_id, brand) VALUES ('Laptop', 1, 'Dell');
INSERT INTO region (region_name) VALUES ('North');
INSERT INTO city (city_name, region_id) VALUES ('Delhi', 1);
INSERT INTO customer_dim_normalized (customer_name, city_id) VALUES ('John Doe', 1);
INSERT INTO time_dim_normalized (date, day_of_week, month, quarter, year)
VALUES ('2025-02-02', 'Monday', 'February', 'Q1',2025);
INSERT INTO sales_fact_normalized (product_id, customer_id, store_id, time_id,
amount, quantity) VALUES (1, 1, 1, 1, 1000, 2);
```

Explanation

- Populates tables with sample data to test the schema.
- Links foreign keys correctly.

Expected Output

```
Query OK, 1 row affected (0.01 sec)
```

5. Querying the Data

```
SELECT c.customer_name, p.product_name, s.amount
FROM sales_fact_normalized s
JOIN customer_dim_normalized c ON s.customer_id = c.customer_id
JOIN product_dim_normalized p ON s.product_id = p.product_id;
```

Expected Output

customer_name	product_name	amount
John Doe	Laptop	1000

This retrieves sales records, showing **which customer purchased which product and for how much**.

Resources

- MySQL Documentation
- Snowflake Schema vs Star Schema
- Normalization Forms