Prepared by Mohammad Umer

Power BI Tutorial

Content

- Database Keys

A key in a database isa attribute or a set of attributes that uniquely identifies a tuple (row) in a table. Keys play a crucial role in ensuring the integrity and reliability of a database enforcing unique constraints on the data and establishing relationships between tables.

Roll no	Name	Branch	Email
1	Nitish Singh	CSE	nitish@gmail.com
2	Ankit Sharma	EEE	ankit@gmail.com
3	Neha Verma	ME	neha@gmail.com

- Types of Keys

1. **Super Key** - A Super key is a combination of columns that uniquely identifies any row within a relational database management system (RDBMS) table
   Example: All of these combination is a Super Key.
   - Roll no
   - Email
   - Roll no + Name
   - Roll no + Branch
   - Roll no + Email
   - Roll no + Branch + Email
   - Roll no + Branch + Name
   - Roll no + Branch + Name + Email

   NOTE: 'Name' alone cannot be a Key, because in the same class multiple students can be of same Name.

2. **Candidate key** - A candidate key is a minimal Super key, meaning it has no redundant attributes. In other words, it's the smallest set of attributes that can be used to uniquely identify a tuple (row) in the table.
   Example: From the above Super Key, there will be only few Candidate key.
   - Roll no
   - Email

3. **Primary Key** - A primary key is a unique identifier for each tuple in a table. There can only be one primary key in a table, and it cannot contain null values.
   Example: From the above Candidate key, we will select only one Key and it will be called Primary Key.
   - Roll no OR Email

   - Criteria for Becoming a Primary Key / Key
     - ✓ It cannot be NULL
     - ✓ There shouldn't be Duplicates

   - Good to have Criterias for Becoming a Primary Key
     - ✓ It should be Numerical
     - ✓ It should be Small
     - ✓ It should be Constant

   Based on these criterias we will select 'Roll no' → Primary Key

4. **Alternate Key** - An alternate key is a **candidate key** that is **not used as the primary key**.

5. **Composite Key** - A composite key is a primary key that is made up of **two or more attributes**. Composite keys are used when a **single attribute is not sufficient** to uniquely identify a tuple in a table.

   Example: Let us take an example of a **Udemy-type application**. In an **Enrollment table**, a single attribute is not sufficient to act as a primary key.

Student id	Course id	Date	Payment
1	Data Science	9-1-2026	Credit
2	Data Science	9-1-2026	UPI
1	Web Dev	12-1-2026	UPI

   Here:

   - **Student ID** cannot be a primary key because a single student can enroll in multiple courses.
   - **Course ID** cannot be a primary key because multiple students can enroll in the same course.
   - **Date** cannot be a primary key because multiple enrollments can occur on the same date.
   - **Payment** cannot be a primary key because the same student can use the same payment method multiple times.

   Therefore, to form a **primary key**, we must **combine two or more attributes**, such as **(Student ID, Course ID)**, which together uniquely identify each record.

6. **Surrogate Key** – A **surrogate key** is an **artificially created key** that is added to a table when **no suitable natural primary key exists**. This key is created by the database designer and has **no business meaning**.
   Example: Consider the following dataset:

Name	Branch	CGPA

In this table, none of the attributes can uniquely identify each record. Therefore, we can add a **new column**, such as **Student_ID**, which will act as the **primary key**.

Studend_ID	Name	Branch	CGPA

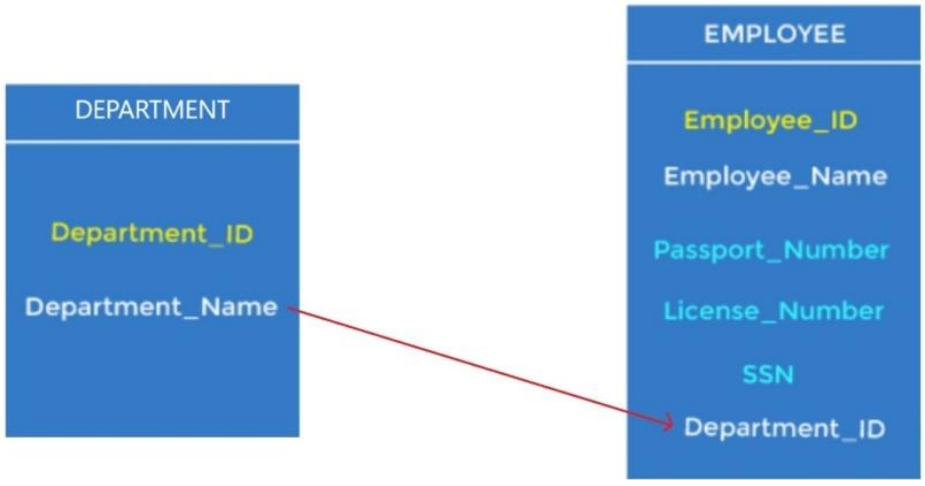
Here, **Student_ID** is a **surrogate key**.

7. **Foreign Key** - A foreign key is a primary key from one table that points to the primary key in another table, used to establish a relationship with another table.

Example: Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.

We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table. In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.

NOTE: The Foreign Key Rows should be a subset of Primary key, that means Foreign key cannot contain any row that is not present in the Primary key.



- Cardinality of Relationships

Cardinality in database relationships **refers to** the number of occurrences of an entity in a relationship with another entity. Cardinality defines the number of instances of one entity that can be associated with a single instance of the related entity.

- **One-to-One (1:1)**: Each instance of entity A is related to exactly one instance of entity B, and vice versa.

  For example, A female can marry to one male, and a male can marry to one female.

Female Table

female_id	name	age
101	Ayesha	25
102	Aafia	26
103	Sara	24

Male Table

male_id	name	age	female_id
1	Ali	28	101
2	Inam	30	102
3	Zaid	27	103

- MySQL Query

```
CREATE TABLE Female (
   female_id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT
);

CREATE TABLE Male (
   male_id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT,
   female_id INT UNIQUE,
   FOREIGN KEY (female_id) REFERENCES Female(female_id)

);
```

- Microsoft SQL Server

```
CREATE TABLE Female (
   female_id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT
);

CREATE TABLE Male (
   male_id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT,
   female_id INT UNIQUE,
   CONSTRAINT FK_Male_Female
   FOREIGN KEY (female_id) REFERENCES Female(female_id)
);
```

- **One-to-Many (1)**: One instance of entity A can be related to many instances of entity B, but each instance of B is related to only one instance of A.

  For Example, one surgeon department can accommodate many doctors.

Surgeon_Department Table

Dept_id	Dept_name	Location
1	Cardiac Surgery	Block A
2	Neuro Surgery	Block B
3	Orthopedic Surgery	Block C

Doctors Table

Doctor_id	Doctor_name	Specialization	Dept_id
101	Dr. Ali	Heart Surgeon	1
102	Dr. Umer	Bypass Specialist	1
103	Dr. Zaid	Brain Surgeon	2
104	Dr. Sara	Spine Specialist	2

- MySQL Query

```
CREATE TABLE Surgeon_Department (
   dept_id INT PRIMARY KEY,
   dept_name VARCHAR(50),
   location VARCHAR(50)
);

CREATE TABLE Doctors (
   doctor_id INT PRIMARY KEY,
   doctor_name VARCHAR(50),
   specialization VARCHAR(50),
   dept_id INT,
  FOREIGN KEY (dept_id) REFERENCES Surgeon_Department(dept_id)
);
```

- Microsoft SQL Server

```
CREATE TABLE Surgeon_Department (
   dept_id INT PRIMARY KEY,
   dept_name VARCHAR(50),
   location VARCHAR(50)
);

CREATE TABLE Doctors (
   doctor_id INT PRIMARY KEY,
   doctor_name VARCHAR(50),
   specialization VARCHAR(50),
   dept_id INT,
  CONSTRAINT FK_Doctors_Department
  FOREIGN KEY (dept_id) REFERENCES Surgeon_Department(dept_id)
);
```

- **Many-to-One:** One instance of entity B can be related to many instances of entity A, but each instance of A is related to only one instance of B.

  For Example, Multiple surgeries can be done by a single surgeon.

Surgeries Table

surgery_id	surgery_name	surgery_date	surgeon_id
101	Bypass Surgery	2025-02-01	1
102	Valve Replacement	2026-02-03	1
103	Brain Tumor Removal	2025-01-02	2
104	Spine Surgery	2025-02-06	2

Surgeon Table

surgeon_id	surgeon_name	specialization
1	Dr. Ali	Cardiac
2	Dr. Umer	Neuro

- MySQL Query

```
CREATE TABLE Surgeon (
   surgeon_id INT PRIMARY KEY,
   surgeon_name VARCHAR(50),
   specialization VARCHAR(50)
);

CREATE TABLE Surgeries (
   surgery_id INT PRIMARY KEY,
   surgery_name VARCHAR(50),
   surgery_date DATE,
   surgeon_id INT,
   FOREIGN KEY (surgeon_id) REFERENCES Surgeon(surgeon_id)
);
```

- Microsoft SQL Server

```
CREATE TABLE Surgeon (
   surgeon_id INT PRIMARY KEY,
   surgeon_name VARCHAR(50),
   specialization VARCHAR(50)
);

CREATE TABLE Surgeries (
   surgery_id INT PRIMARY KEY,
   surgery_name VARCHAR(50),
   surgery_date DATE,
   surgeon_id INT,
   CONSTRAINT FK_Surgeries_Surgeon
      FOREIGN KEY (surgeon_id) REFERENCES Surgeon(surgeon_id)
);
```

- **Many-to-Many (M)**: Many instances of entity A are related to many instances of entity B.

Employees Table

emp_id	emp_name	role
1	Ali	Developer
2	Umer	Data Scientist
3	Zaid	Tester

Project Table

project_id	project_name	deadline
101	Hospital Management System	2027-03-01
102	AI Diagnosis Tool	2025-04-15

Works_On Table (Junction Table)

emp_id	project_id	hours_per_week
1	101	20
1	102	10
2	102	25
3	101	15

- MySQL Query

```
CREATE TABLE Employees (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(50),
   role VARCHAR(50)
);

CREATE TABLE Projects (
   project_id INT PRIMARY KEY,
   project_name VARCHAR(50),
   deadline DATE
);

CREATE TABLE Works_On (
   emp_id INT,
   project_id INT,
   hours_per_week INT,
   PRIMARY KEY (emp_id, project_id),
   FOREIGN KEY (emp_id) REFERENCES Employees(emp_id),
   FOREIGN KEY (project_id) REFERENCES Projects(project_id)
);
```

- Microsoft SQL Server

```
CREATE TABLE Employees (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(50),
   role VARCHAR(50)
);

CREATE TABLE Projects (
   project_id INT PRIMARY KEY,
   project_name VARCHAR(50),
   deadline DATE
);

CREATE TABLE Works_On (
   emp_id INT,
   project_id INT,
   hours_per_week INT,
   CONSTRAINT PK_WorksOn
      PRIMARY KEY (emp_id, project_id),

   CONSTRAINT FK_WorksOn_Employee
      FOREIGN KEY (emp_id) REFERENCES Employees(emp_id),

   CONSTRAINT FK_WorksOn_Project
     FOREIGN KEY (project_id) REFERENCES Projects(project_id)
);
```

- Data Warehouse Schema

A **Data Warehouse Schema** is a **logical description** of the entire data warehouse structure. It defines how data is **organized** and **related** within the warehouse to ensure efficient **retrieval** & maintain data **integrity**.

Types

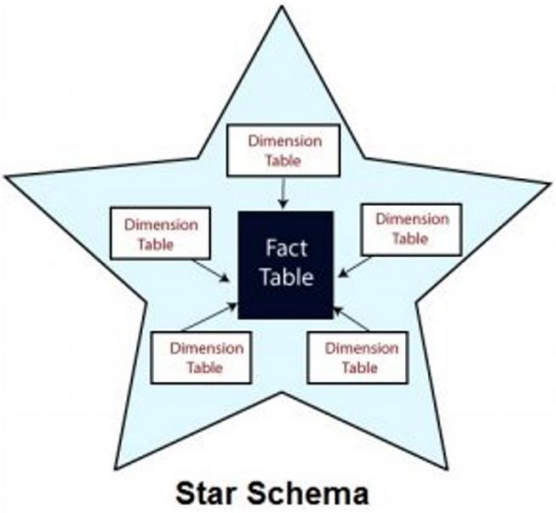
1. Start schema
2. Snowflake Schema
3. Fact constellation schema / galaxy schema

1. Start schema

It is the simplest type of schema. It consists of central **fact table** surrounded by **dimensional table**.

The fact table is a ***quantitative*** data. The dimension table is the ***descriptive attributes***.

- Structure of Start Shema:



Star Schema

Advantages

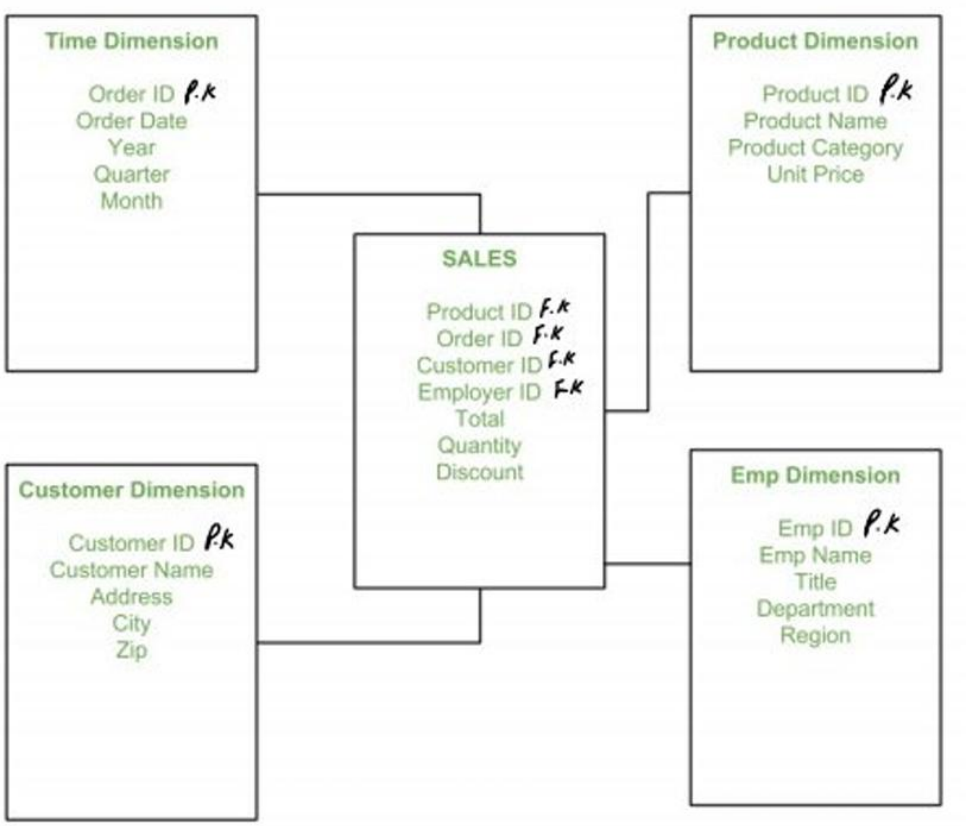
i. It is simple and easy to understand.
ii. It enhances query performance due to fever joints.
iii. It is efficient for OLAP queries.

Disadvantages

i. It can lead to data redundancy that means repeated data.
ii. It is not normalized leading to data inconsistencies.

Example:

To better understand the star schema concept, let's examine an example from a **sales scenario**. Imagine we have a data warehouse for a **retail company** that wants to analyze its **sales data** from multiple perspectives.



The **Sales fact table** is at the center of the star schema, containing numerical measures or facts related to sales transactions, such as *total sales amount*, *quantity sold*, and *discount amount*. It has foreign key columns that establish relationships with the dimension tables.

Each **dimension table** stores descriptive attributes or characteristics related to a specific dimension of the sales data. For example, the *Product Dimension* contains details about the products sold, such as product name, category, and unit price. The Time Dimension captures temporal information like order date, year, quarter, and month.

The **fact table (Sales**) is connected to each dimension table through foreign key relationships. For instance, the Product ID in the Sales fact table is a foreign key referencing the Product ID primary key in the Product Dimension table. This allows us to join the fact table with the relevant dimension tables during querying and analysis.

Q. When Star Schema is Preferred?

A **Star Schema** is preferred in scenarios where:

1. **Performance is more important than storage efficiency**
   - Since star schema has fewer joins, queries run **much faster** compared to snowflake schema.
   - Best when analysts run frequent **OLAP queries** (drill-down, roll-up, slice, dice).

2. **Data is not extremely large or redundancy is acceptable**
   - Works well when slight data duplication is manageable and does not affect system cost.

3. **Business users or non-technical analysts need to query data**
   - Its **simple, intuitive design** is easy to understand for business users without deep technical knowledge.

4. **Reporting and dashboarding require fast response times**
   - Example: Sales dashboards, Marketing campaign analytics, Inventory reports.

Example Scenarios

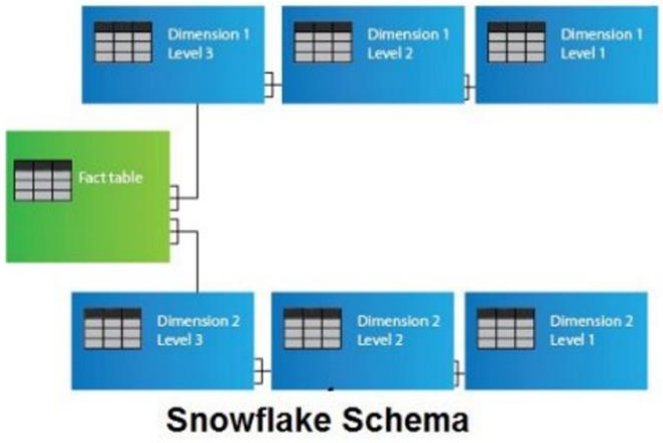
- A retail company wants to analyze **daily sales performance** quickly.
- They need dashboards that show **sales by product, region, and time** in real-time.
- Since performance and simplicity are critical → **Star Schema is chosen**.

2. Snowflake Schema

The snowflake schema is an extension or complex variation of the star schema used in data warehousing. While the star schema has a central fact table surrounded by denormalized dimension tables, the snowflake schema introduces further normalization (i.e no redundancy) by breaking down the dimension tables into additional related tables (k/as Sub-dimension tables).

This results in a hierarchical structure that resembles a snowflake, hence the name.

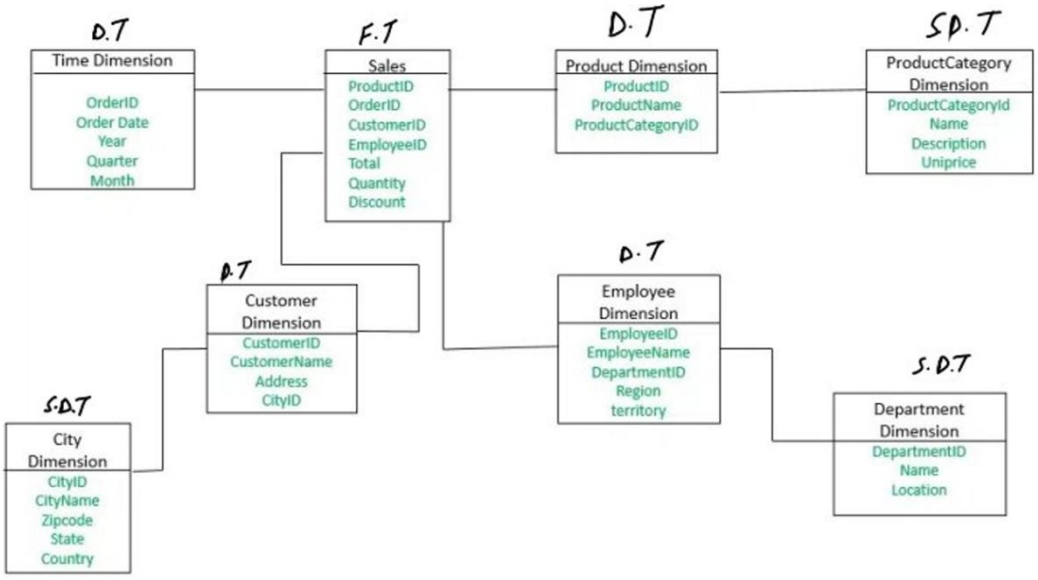
Structure of flake schema:



Example:

In a sales data warehouse, the *product dimension* table might be normalized into multiple related tables, such as *product category,* product subcategory and so on.

Each of these tables would be related to the *product dimension* table through foreign key relationships, forming a snowflake structure.



- Characteristics:

i. Compact disk space usage due to normalization.
ii. Easy implementation of new dimensions.
iii. Reduced performance due to increased number of tables and joins.
iv. Dimension tables consist of sets of attributes defining information at different levels of granularity.
v. Attributes within the same dimension table may be populated from different source systems.

- Advantages:

i. Improved query performance due to minimized disk storage and smaller lookup tables.
ii. Greater scalability in managing interrelationships between dimension levels.
iii. No redundancy, leading to easier maintenance.

- Disadvantages:

i. Increased maintenance efforts due to a higher number of lookup tables.
ii. Complex queries and reduced understandability.
iii. Longer query execution times due to increased table joins.

Q. When Snowflake Schema is Preferred (used)

A **Snowflake Schema** is preferred when:

- **Storage optimization** and **data integrity** are more critical than query speed.
- Dimensions are **very large** and contain **hierarchical relationships** (e.g., Geography → Country → State → City).
- The organization needs to minimize redundancy (saves disk space).

Example Scenarios

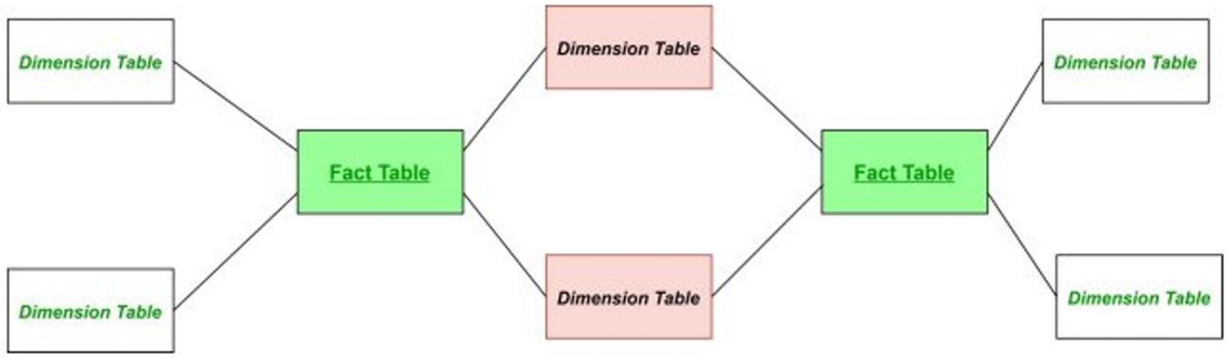
- If the same company had extremely detailed product hierarchies (brand → category → sub-category → SKU) and needed strict data consistency across multiple systems, **Snowflake Schema** would be used.
- But queries would be slower due to many joins.

3. Fact Constellation Schema

The fact constellation schema, also known as the galaxy schema, it is also an extension of the star schema used in data warehousing to represent **complex scenarios** involving *multiple fact tables* sharing *dimension tables*.

This system is mostly used in **'Data Marts'**.

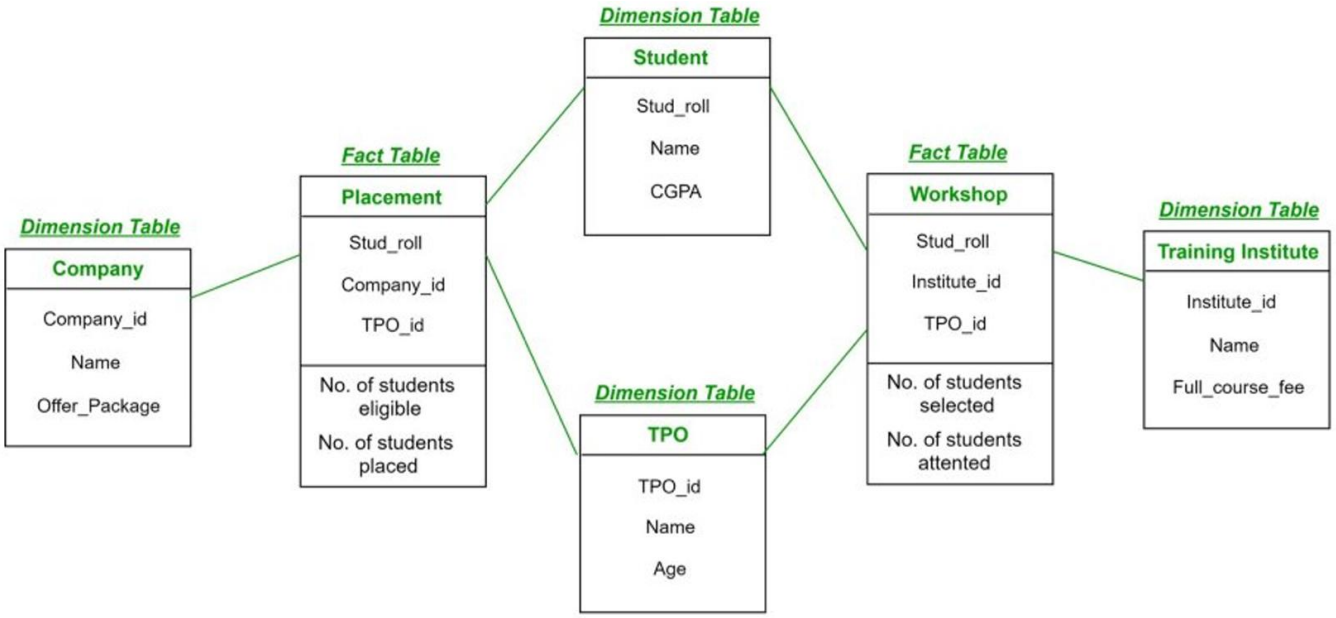
Structure of Fact Constellation Schema:



*Pink color tables are the Shared Dimension Table.

Example:

Let's consider a data warehouse for a university that needs to **analyze data related to** student placements and workshop attendance.



*where, **TPO**: Training and Placement Officer

- Advantages:

i. Provides a flexible and scalable schema for handling multiple business processes or subject areas within a single data warehouse

ii. Avoids redundant storage of shared dimension data across multiple star schemas

iii. Allows for efficient querying and analysis of data across different subject areas by leveraging shared dimensions.

- Disadvantages:

i. Increased complexity in schema design and maintenance due to the presence of multiple fact tables and shared dimensions

ii. Potential performance issues if queries involve joining multiple fact tables and shared dimensions

iii. May require additional metadata and documentation to understand the relationships between different star schemas and shared dimensions