1. What is Normalization and Denormalization ?

In SQL and database design, **normalization** and **denormalization** are two important concepts used to organize data efficiently. Here's a clear breakdown:

1. Normalization

Definition:
Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity.

Goals:

- Eliminate duplicate data.
- Ensure data dependencies make sense.
- Make the database more efficient and easier to maintain.

How it works: Data is split into multiple related tables, and relationships are established using **foreign keys**.

Normal Forms (NF): There are several levels of normalization, called **normal forms**:

- **1NF (First Normal Form):** Eliminate repeating groups; ensure each column holds atomic values.
- **2NF (Second Normal Form):** Remove partial dependencies (non-key attributes depend on the whole primary key).
- **3NF (Third Normal Form):** Remove transitive dependencies (non-key attributes depend only on the primary key).

Example: Instead of storing student and course info in one table:

SNO	SNAME	COURSE1	COURSE2

You normalize it into:

- `STUDENT(SNO, SNAME)`
- `COURSE(CID, CNAME)`
- `ENROLLMENT(SNO, CID)`

2. Denormalization

Definition:
Denormalization is the process of combining normalized tables to improve read performance by reducing the number of joins.

Goals:

- Improve query performance.

- Reduce complexity in data retrieval.

How it works: Data from multiple tables is merged into one, even if it introduces redundancy.

Example: Instead of having separate STUDENT and COURSE tables, you might have:

SNO	SNAME	COURSE
1	Aarav	Math
1	Aarav	Science

This avoids joins but duplicates data.

✓ Summary

Feature	Normalization	Denormalization
Purpose	Reduce redundancy	Improve read performance
Data structure	Many related tables	Fewer, larger tables
Redundancy	Low	High
Performance	Slower reads, faster writes	Faster reads, slower writes
Use case	OLTP (transactional systems)	OLAP (reporting, analytics)

□ Reduce Redundancy

- **Meaning**: Avoid storing the same data in multiple places unnecessarily.
- **Why it's important**: It saves storage space and prevents confusion or errors when data changes.

Example:
Instead of storing a customer's address in every order record, store it once in a customer table and link it.

□ Improve Data Integrity

- **Meaning**: Ensure the data is accurate, consistent, and reliable across the database.
- **Why it's important**: It helps maintain trust in the data and prevents issues like mismatched or outdated information.

Example:
If a product price is updated, it should reflect correctly wherever it's used — this is data integrity.

2. What is Fact and Dimension Tables ?

In SQL and data warehousing, **Fact Tables** and **Dimension Tables** are key components of a **star schema** or **snowflake schema** used in **OLAP (Online Analytical Processing)** systems.

1. What is a Fact Table?

A **Fact Table** stores **quantitative data** for analysis and is typically at the center of a star schema.

✔ *Characteristics:*

- Contains **measurable facts** (e.g., sales, revenue, quantity).
- Has **foreign keys** referencing dimension tables.
- Grows **vertically** (more rows over time).

□ *Example:*

Sale_ID	Date_ID	Product_ID	Store_ID	Quantity	Revenue
1	20250701	101	10	5	2500

2. What is a Dimension Table?

A **Dimension Table** stores **descriptive attributes** related to the facts. These are used to **filter, group, and label** data in queries.

✔ *Characteristics:*

- Contains **textual or categorical data** (e.g., product name, region, date).
- Connected to fact tables via **primary key → foreign key**.
- Grows **horizontally** (more columns, fewer rows).

□ *Example:* `Product_Dimension`

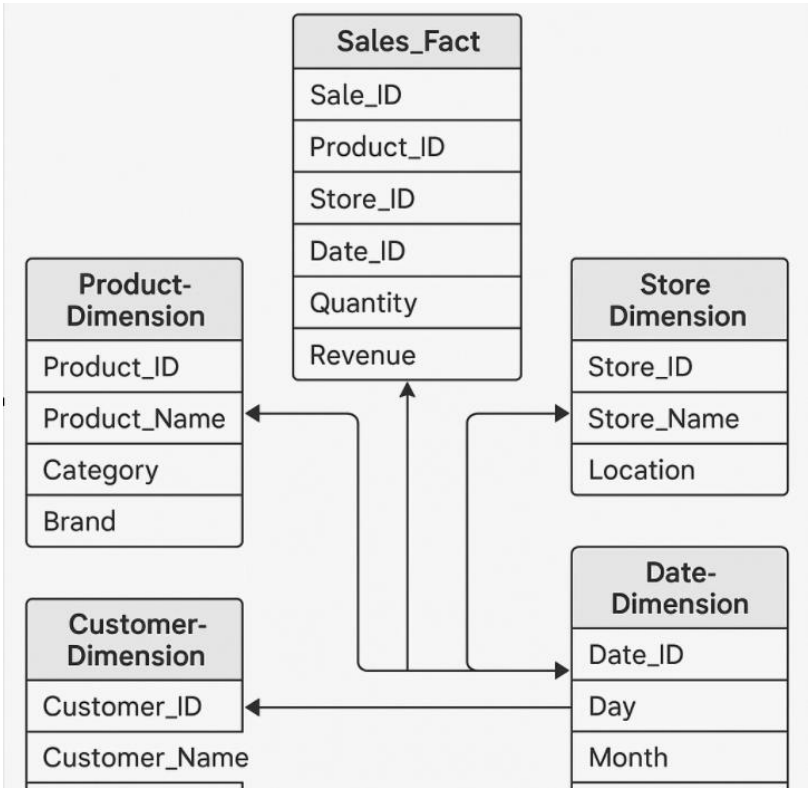
Product_ID	Product_Name	Category	Brand
101	Laptop X	Electronics	TechBrand

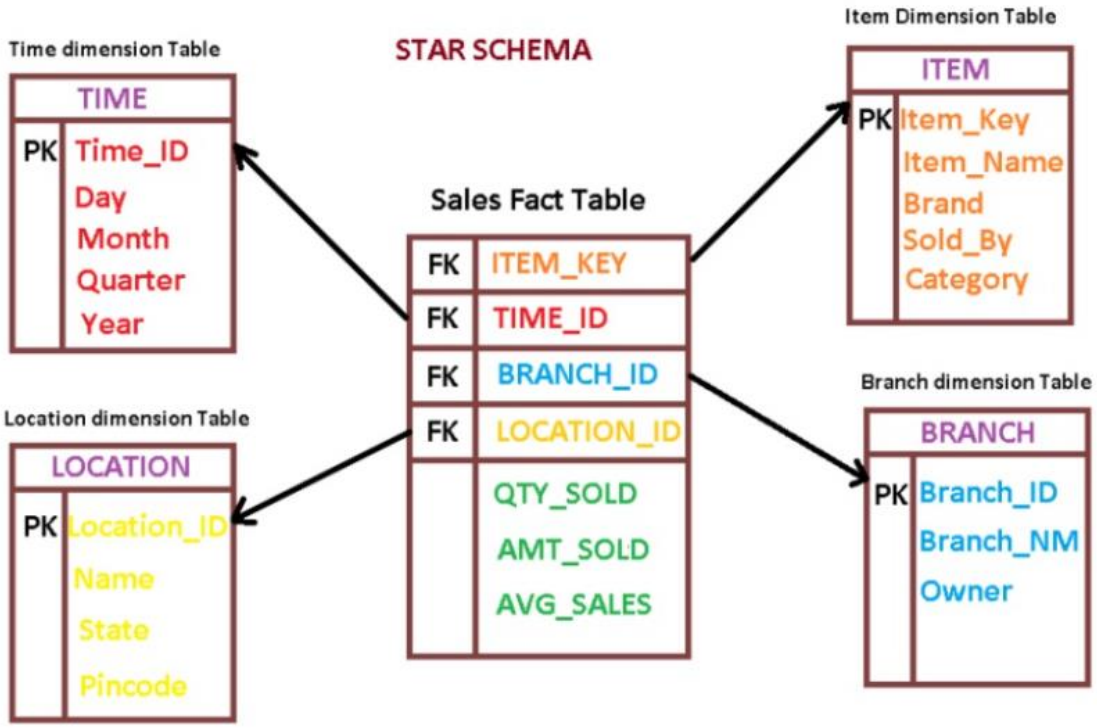
□ Analogy:

Think of a **fact table** as the **"what happened"** (e.g., 5 laptops sold), and **dimension tables** as the **"who, what, where, when"** (e.g., which laptop, which store, on what date).

□ Summary Table:

Feature	Fact Table	Dimension Table
Data Type	Numeric (measurable)	Textual (descriptive)
Purpose	Stores facts/metrics	Stores context for facts
Keys	Foreign keys to dimensions	Primary keys
Size	Large (many rows)	Smaller (fewer rows)
Example	Sales, Revenue	Product, Customer, Date





3. What is Star Schema and Snowflake Schema ?

The **Star Schema** and **Snowflake Schema** are two common ways to organize data in a **data warehouse** for analytical processing. Here's a clear comparison:

⭐ Star Schema

✅ *Structure:*

- Central **fact table** connected directly to multiple **dimension tables**.
- Dimension tables are **denormalized** (contain all descriptive data in one table).

🔹 *Example:*

```
          PRODUCT
             |
CUSTOMER — SALES_FACT — TIME
             |
          STORE
```

✅ *Pros:*

- Simple and easy to understand.
- Faster query performance due to fewer joins.
- Ideal for **OLAP** systems.

❌ *Cons:*

- Redundant data in dimension tables.
- Larger storage requirements.

❄ Snowflake Schema

✅ *Structure:*

- Central **fact table** connected to **normalized dimension tables**.
- Dimension tables are split into sub-dimensions.

🧊 *Example:*

```
          CATEGORY
             |
          PRODUCT
             |
CUSTOMER — SALES_FACT — TIME — MONTH
             |
          STORE — REGION
```

✅ *Pros:*

- Reduces data redundancy.
- Saves storage space.
- Better data integrity.

❌ *Cons:*

- More complex structure.
- Slower query performance due to more joins.

□ Summary Table

Feature	Star Schema	Snowflake Schema
Structure	Fact table + denormalized dimensions	Fact table + normalized dimensions
Joins	Fewer joins	More joins
Query Performance	Faster	Slower
Storage	More space	Less space
Complexity	Simple	Complex
Use Case	OLAP, dashboards	Data integrity, storage optimization

4. Which Scenario used What schema in Real time ?

Great question! Here's how **Star Schema** and **Snowflake Schema** were used in different project scenarios:

⋆ Example Scenario Using Star Schema

□ *Project: Sales Analytics Dashboard for a Retail Chain*

✓ *Why Star Schema?*

- The dashboard needed **fast query performance** for daily sales reports.
- Users frequently filtered data by **product, store, date, and customer**.
- Simplicity was key for business analysts writing ad hoc queries.

□ *Schema Design:*

- **Fact Table**: `Sales_Fact` (contains `Sale_ID`, `Product_ID`, `Store_ID`, `Date_ID`, `Quantity`, `Revenue`)
- **Dimension Tables**:
- `Product_Dim` (Product_ID, Name, Category, Brand)
- `Store_Dim` (Store_ID, Name, Region)
- `Date_Dim` (Date_ID, Day, Month, Year)
- `Customer_Dim` (Customer_ID, Name, Segment)

✓ *Benefits:*

- Simple joins

- Fast aggregations for KPIs like total sales, average revenue, etc.

❄ Example Scenario Using Snowflake Schema

🏦 *Project: Financial Reporting System for a Bank*

✔ *Why Snowflake Schema?*

- Required **high data integrity** and **storage efficiency**.
- Complex hierarchies like **Account → Branch → Region**.
- Reports were generated periodically, so query speed was less critical.

🧱 *Schema Design:*

- **Fact Table**: `Transaction_Fact` (Transaction_ID, Account_ID, Date_ID, Amount)
- **Dimension Tables**:
- `Account_Dim` → normalized into:
- `Account` (Account_ID, Type, Branch_ID)
- `Branch` (Branch_ID, Name, Region_ID)
- `Region` (Region_ID, Name)
- `Date_Dim` (Date_ID, Day, Month_ID)
- `Month` (Month_ID, Month_Name, Year_ID)
- `Year` (Year_ID, Year)

✔ *Benefits:*

- Reduced redundancy in location and time data.
- Easier to maintain and update hierarchical relationships.