**📚 Advanced Concepts:**

* Data Warehousing basics
* Star/Snowflake Schema
* Slowly Changing Dimensions
* Performance Tuning

**✅ What is a Data Warehouse?**

📦 A **Data Warehouse** is like a **big storage room** where companies **store all their important data** — cleaned, arranged, and ready to analyze.

It’s not for daily work —  
it’s for **reporting, analysis, and business decisions**.

**🎯 Why We Need It?**

Imagine a big company like Flipkart or SBI.  
They have:

* Sales data
* Customer details
* Transaction history
* Website clicks
* Inventory info  
  👉 All coming from **different departments and tools**.

**We need a common place** to combine and study this data.  
That’s where **Data Warehouse** comes in.

**🧱 Basic Concepts**

| **Term** | **Meaning** | **Easy Example** |
| --- | --- | --- |
| **ETL** | Extract, Transform, Load | Get data ➝ Clean it ➝ Store in warehouse |
| **Fact Table** | Table that stores numbers (sales, amount, counts) | Sales data: quantity, revenue |
| **Dimension Table** | Table with description info | Products, Customers, Dates |
| **Schema** | Design or layout of tables | Like blueprints of your house |

**🏗️ Example:**

A **Retail Warehouse** might have:

🧮 Fact\_Sales  
📅 Dim\_Date  
👗 Dim\_Product  
🧍 Dim\_Customer

You can ask questions like:

🔎 “How much did we sell in Odisha in June?”  
🔎 “Which product is performing best?”

**🔢 1. Fact Table – *Stores measurable numbers***

**✅ Definition:**

A **Fact Table** contains **quantitative data (numbers)** for analysis — like sales, profits, or counts — and it’s usually linked to multiple **Dimension Tables**.

**🔍 Example:**

| **Date\_Key** | **Product\_Key** | **Customer\_Key** | **Sales\_Amount** | **Quantity\_Sold** |
| --- | --- | --- | --- | --- |
| 20250701 | 101 | C001 | ₹2,000 | 10 |

* **Sales\_Amount** and **Quantity\_Sold** → called **measures (facts)**.
* The **keys** (Date\_Key, Product\_Key, Customer\_Key) → link to Dimension Tables.

**📌 Key Features:**

* Contains **facts/measures** (numeric values).
* Often **very large** in size.
* **Foreign Keys** link to dimension tables.
* Supports **aggregations** (SUM, AVG, COUNT) in reporting.

**🔢 What is Quantitative Data?**

**Quantitative data** means **data that can be measured and expressed in numbers**.

It's the kind of data that answers questions like:

"How many?"  
"How much?"  
"What is the value?"

**✅ Examples:**

| **Type** | **Example** | **Why it's Quantitative?** |
| --- | --- | --- |
| Sales Amount | ₹5,000 | It shows how much money |
| Quantity Sold | 10 items | Count of products sold |
| Temperature | 36.5°C | Measurable unit |
| Age | 25 years | Can be measured in years |

**✴️ Characteristics of Quantitative Data:**

 Always **numerical**.

 Can be used in **calculations** (add, average, sum).

 Suitable for **graphs, charts, stats**.

**🚫 Opposite: Qualitative Data (Descriptive)**

| **Qualitative Data** | **Quantitative Data** |
| --- | --- |
| Color = "Red" | Price = ₹10,000 |
| Category = "Laptop" | Units Sold = 15 |
| Customer = "Paresh" | Age = 32 |

**🧾 2. Dimension Table – *Stores descriptive info***

**✅ Definition:**

A **Dimension Table** contains **textual or descriptive data** that gives **context** to the facts. These are things you slice, filter, or group your data by.

**🔍 Examples:**

**Customer Dimension**

| **Customer\_Key** | **Customer\_Name** | **City** | **Gender** |
| --- | --- | --- | --- |
| C001 | Paresh | Bhubaneswar | Male |

**Product Dimension**

| **Product\_Key** | **Product\_Name** | **Category** | **Brand** |
| --- | --- | --- | --- |
| 101 | Laptop | Electronics | Lenovo |

**📌 Key Features:**

* Contains **text fields** (names, categories, dates).
* Typically **smaller** in size than fact tables.
* Used for **filtering**, **grouping**, and **slicing** in reports.

**📊 Types of Dimension Tables in Data Warehousing**

**✅ 1. Conformed Dimension**

* A **shared dimension** used across **multiple fact tables**.
* Ensures consistency across reports.

**Example:**

* Customer\_Dim used in both **Sales Fact** and **Support Tickets Fact**.

**🔄 2. Slowly Changing Dimension (SCD)**

**This is the most important type — handles changes in dimension data over time.**

**📌 There are 3 main types:**

|  | |  |  |
| --- | --- | --- | --- |
| 🔄 What is Slowly Changing Dimension (SCD)?  SCD means:  How do you handle changes in your dimension table data over time?  For example, if a customer changes their address, what should we do in the database? Keep old? Overwrite? Store both?  That’s what SCD types decide.  🧠 Easy Explanation of Each SCD Type:   | SCD Type | 🔧 What It Does | 🧑‍💼 Easy Example | 🔍 Used When | | --- | --- | --- | --- | | Type 0 | ❌ Do nothing (data never changes) | Birth Date: Never changes in your life | Data is permanent | | Type 1 | 📝 Overwrite old data | Address changes → Just update the new one | You don’t care about history | | Type 2 | 🧾 Keep full history (add new row) | New address → Add a new row with version | You want to track history | | Type 3 | 🗂 Keep partial history (same row) | Add "Current City" and "Previous City" columns | Only track limited changes |   **💡 Real-Life Example: Changing Customer Address**  🧑 Customer Old Address:   * Name: Paresh * Address: Bhubaneswar   Later, Paresh moves to Bengaluru 🏙️  **🔁 What happens in each SCD type?**   | Type | What is stored? | | --- | --- | | SCD Type 0 | ❌ Nothing changes. Still shows Bhubaneswar. | | SCD Type 1 | ✅ Overwrites to Bengaluru. Bhubaneswar is gone. | | SCD Type 2 | ✅ New row is added: one for Bhubaneswar, one for Bengaluru. You can see full history. | | SCD Type 3 | ✅ Same row, with two columns: Previous\_City = Bhubaneswar, Current\_City = Bengaluru |   **🎯 When to Use Which SCD?**   | If you want to... | Use This Type | | --- | --- | | Keep things simple, no change allowed | Type 0 | | Just want latest info, no history | Type 1 | | Keep full change history | Type 2 | | Only track current and previous | Type 3 | | |  |  |
|  |
|  |
|  |

👉 **Type 2** is the most used in real-time analytics and time-series reporting.

**🔐 3. Junk Dimension**

* Combines low-cardinality attributes (like Yes/No, flags) into one small dimension to avoid cluttering the fact table.

**Example:**

| **ID** | **Is\_Returned** | **Is\_Promo** | **Payment\_Mode** |
| --- | --- | --- | --- |
| 1 | Yes | No | Credit Card |

**🧩 4. Degenerate Dimension**

* A dimension that **exists in the fact table** but has **no separate dimension table**.

**Example:**

* Invoice Number or Order ID in the fact table — useful for reporting, but not linked to another dimension.

**📅 5. Role-Playing Dimension**

* A single dimension table used for **different roles** in different contexts.

**Example:**

* Date\_Dim used as:
  + Order\_Date
  + Ship\_Date
  + Return\_Date

Same table, different role in different reports.

**🧾 6. Outrigger Dimension**

* A dimension that references **another dimension** (like a hierarchy).

**Example:**

* Product → links to Category → links to Department

Not common in star schema (used in **snowflake schemas**).

**🏗️ 3. Schema – *The overall design/layout of the tables***

**✅ Definition:**

A **Schema** defines **how your tables are structured and related** — like a blueprint of a house.

There are 3 common types of schemas:

**⭐ A. Star Schema – Simple and fast for querying**

* Fact table in center.
* Dimension tables around it like a star 🌟.

A white background with yellow text

AI-generated content may be incorrect.

 **Good for:** Fast reads, easy reporting.

 **Joins:** Fact table joins directly to each dimension.

**✅ Advantages of Star Schema:**

| **Advantage** | **Explanation** |
| --- | --- |
| **1. Simple and Intuitive** | Easy to understand for business users and report developers. |
| **2. Fast Query Performance** | Since dimensions are not normalized (no multiple joins), queries run faster. |
| **3. Easy for BI Tools** | Tools like Power BI, Tableau, Synapse, etc., work very well with this layout. |
| **4. Efficient Aggregation** | Perfect for SUM, AVG, COUNT operations using GROUP BY. |
| **5. One Join per Dimension** | Fact table joins directly to each dimension — no complex navigation. |

**❌ Disadvantages of Star Schema:**

| **Disadvantage** | **Explanation** |
| --- | --- |
| **1. Data Redundancy** | Because dimension tables are denormalized (flattened), the same info may repeat (e.g., category name in multiple rows). |
| **2. Larger Storage Size** | Due to repetition in dimension tables, it consumes more storage. |
| **3. Harder to Maintain** | If category or city names change, it has to be updated in many places. |
| **4. Not Ideal for Complex Relationships** | Doesn’t handle many-to-many relationships or hierarchical data well. |
| **5. Less Flexibility in Updates** | Updating one value in many rows can lead to inconsistencies if not handled carefully. |

**⭐ B. Snowflake Schema – *(More Structured, but More Complex)***

**🔧 What it is:**

A **snowflake schema** is just like a **star schema**, but the **dimension tables are broken down into smaller related tables** (normalized).

Instead of putting everything in one table, we split it to avoid repeating the same data.

**🧊 Real-Life Example:**

Let’s say we have a **Product Dimension** in a **Star Schema**:

| **Product\_ID** | **Product\_Name** | **Category** | **Department** |
| --- | --- | --- | --- |
| 101 | Laptop | Electronics | Technology |
| 102 | Mouse | Electronics | Technology |

You can see: **"Electronics" and "Technology"** are repeating.

**🧊 In Snowflake Schema, we split it:**

1. **Product Table**  
   | Product\_ID | Product\_Name | Category\_ID |  
   |------------|--------------|-------------|  
   | 101 | Laptop | 1 |
2. **Category Table**  
   | Category\_ID | Category\_Name | Department\_ID |  
   |-------------|----------------|---------------|  
   | 1 | Electronics | 10 |
3. **Department Table**  
   | Department\_ID | Department\_Name |  
   |---------------|------------------|  
   | 10 | Technology |

This reduces **repetition** = saves space. But it means **more joins** when querying.

**✅ Good for:**

* Saving storage
* Cleaner structure
* Avoids repeating text (like "Electronics")

**❌ Drawbacks:**

* Queries are slower due to **more joins**
* More complex for reporting tools
* Harder for beginners to understand

**⭐ C. Galaxy Schema (Fact Constellation) – *Multiple Fact Tables***

**🚀 What it is:**

When you have **multiple fact tables**, and they **share some dimensions**, it becomes a **galaxy schema**.

**🌌 Real Example:**

You have two fact tables:

* Sales\_Fact → for sales transactions
* Inventory\_Fact → for stock and availability

And both share common dimension tables:

* Product\_Dim
* Date\_Dim
* Store\_Dim

markdown

CopyEdit

(Sales\_Fact) (Inventory\_Fact)

\ /

\ /

\ /

Product\_Dimension

**✅ Good for:**

* **Big companies** with different types of data (sales, inventory, returns, marketing)
* Reusing dimension tables = **consistency**

**❌ Drawbacks:**

* **Very complex** structure
* Harder to maintain and build queries
* Not ideal for small datasets

**🔁 Summary Table:**

| **Schema Type** | **Easy To Use** | **Reduces Repetition** | **Good For** | **Drawback** |
| --- | --- | --- | --- | --- |
| ⭐ Star | ✅ Very Easy | ❌ Repeats data | Fast reporting | Not space efficient |
| ❄️ Snowflake | ⚠️ Medium | ✅ Yes | Storage + Clean model | Slower queries |
| 🌌 Galaxy | ❌ Complex | ✅ Shared dimensions | Large enterprises | Hard to manage |

**🚀 What You Can Do with a Data Warehouse?**

* 📊 Create reports & dashboards
* 📈 Analyze trends (weekly/monthly/yearly)
* 🧠 Help leadership take smart decisions
* 🔁 Use it in AI or ML models

**📊 In Data Warehousing:**

* Fact Table = stores **quantitative data** (e.g. Revenue, Sales, Quantity).
* Dimension Table = stores **qualitative data** (e.g. Customer Name, Product Category).

**🚀 What is Performance Tuning?**

Performance Tuning means **making your database queries, pipelines, and systems run faster and more efficiently** — using the **least time and resources**.

Think of it like:  
👉 "How can I make my SQL query or data process run **in 2 seconds instead of 20**?"

**🔧 Where is Performance Tuning Done?**

| **Area** | **Examples** |
| --- | --- |
| **SQL Queries** | Slow SELECTs, JOINs, GROUP BY |
| **Indexes** | Missing or wrong indexes |
| **Data Pipelines** | Azure Data Factory, ETL/ELT flow |
| **Data Models** | Schema design, table structures |
| **Storage** | Inefficient partitioning |

**✅ Common Performance Tuning Techniques**

**1. 🔍 Use Indexes Wisely**

* Indexes speed up SELECT queries.
* Add indexes on columns used in WHERE, JOIN, or ORDER BY.

**Example:**

CREATE INDEX idx\_customer\_id ON Sales(Customer\_ID);

⚠️ But don’t overuse indexes — they slow down INSERT/UPDATE.

**2. 📌 \*\*Avoid SELECT \*\*\***

* Instead of:

SELECT \* FROM Sales;

**Use:**

**SELECT Sale\_ID, Amount FROM Sales;**

**Reduces memory usage and improves speed.**

**3. 🔗 Optimize JOINs**

* Use proper JOIN keys with **indexed columns**.
* Avoid joining on columns with NULLs or mismatched datatypes.

**4. 📊 Use Partitioning and Clustering**

* Break big tables into **smaller chunks (partitions)** using date, region, etc.
* Improves performance for large fact tables.

**5. 🔄 Use Caching (for BI tools or ADF)**

* Cache frequently used data (like lookup tables).
* Tools like **Power BI, Azure Synapse** support caching layers.

**6. 🗃 Denormalize if needed**

* For reporting, **denormalized tables** (like Star Schema) work better than normalized ones.

**7. 📈 Analyze Query Plans**

* In SQL Server or Azure:

SET STATISTICS TIME ON;

SET STATISTICS IO ON;

Or use **Query Analyzer** to find slow steps.

**8. 🧠 Use CTAS / Temp Tables (for heavy pipelines)**

* Break big queries into steps using **temp tables** or **materialized views**.

**🧠 Real-World Example (ADF + SQL)**

* Problem: Your data pipeline is taking 30 mins to run a big join and filter.
* Tuning Actions:
  1. Index the JOIN columns.
  2. Filter early (push filters to source).
  3. Use partitioned datasets in ADF.
  4. Cache lookup data.

Result: Pipeline finishes in 8 minutes 🎉