**🏗️ Data Warehouse Lifecycle: Step-by-Step**

| **Step #** | **Phase Name** | **What Happens (In Simple Words)** |
| --- | --- | --- |
| 1️⃣ | **Requirement Gathering** | Talk to business teams and find out: |
| What reports do they need? What questions do they want to answer? |  |  |
| 2️⃣ | **Data Modeling & Design** | Design how data will be stored: |
| Use **Star Schema**, **Snowflake**, define **fact & dimension tables**. |  |  |
| 3️⃣ | **ETL/ELT Development** | Build pipelines to **Extract**, **Transform**, and **Load** data from source systems into the warehouse. |
| 4️⃣ | **Data Warehouse Deployment** | Create actual tables in a **data warehouse platform** like Synapse, Snowflake, BigQuery, etc. |
| 5️⃣ | **Data Loading & Integration** | Use the ETL/ELT jobs to load cleaned data into the warehouse. |
| 6️⃣ | **Testing & Validation** | Test if data is **correct**, complete, and fast to query. Fix errors or mismatches. |
| 7️⃣ | **Data Access & Reporting** | Connect **BI tools** like Power BI, Tableau, Looker for dashboards, reports, analysis. |
| 8️⃣ | **Maintenance & Optimization** | Monitor performance, fix slow queries, update pipelines, and scale as data grows. |
| 9️⃣ | **Enhancement** (Ongoing) | Add new data sources, new dimensions or metrics as business grows. |

**Loading Techniques in Warehouse:**

**📦 What is Initial Load?**

💥 It’s the **first time** you move **all data** from source to warehouse/lake.

* Like filling a **new database** with **all the old data**.
* It’s usually done **once**.
* Takes **longer**, but after that, only **changes are loaded**.

**🧠 Example:**

| **Source Table** | **Initial Load to Warehouse** |
| --- | --- |
| 1,000,000 rows | Move all 1M rows at once |

**🔄 What is Incremental Load?**

🔁 After initial load, you don’t want to load everything again — so you just load the **new or changed data**.

* Faster
* Saves space
* Keeps data **updated daily/hourly**

**🔍 Simple Analogy**

| **Type** | **Analogy** |
| --- | --- |
| Initial Load | Like copying all contacts from old phone to new phone 📱 |
| Incremental Load | Like syncing only **new messages** each time 📩 |

**📂 Types of Incremental Loading**

**✅ 1. Append Load**

Only **new rows** are added (nothing is updated or deleted)

* Fast & simple

**🔁 2. Inplace Load (Update & Insert)**

Updates **existing rows** + inserts new ones

* Replaces or updates matching records
* Uses primary key or unique column

**3. Rebound Append (Delete & Reload recent portion)**

Delete recent data range (like last 7 days), then **reinsert it fresh**

* Useful when updates are frequent in recent data only
* Keeps things simple (no need to detect updates)

**Fact Table:**

**🌟 What is a Fact Table?**

A **fact table** stores **numeric/measurable data** about your business — like **sales, revenue, quantity**, etc.

It connects to **dimension tables** (like product, customer, date).

**🔢 Types of Facts in Fact Table**

There are **3 main types** of facts:

**✅ 1. Additive Facts**

These can be **summed up** across **all dimensions** (like time, product, region)

🧠 **Examples:**

* Total **Sales**
* Total **Quantity Sold**
* Total **Revenue**

💡 You can calculate:

* **Total sales per day**
* **Total sales per product**
* **Total sales per region**

✅ **Additive = Fully Summable**

**🔁 2. Semi-Additive Facts**

Can be summed up **across some dimensions**, **not all**

🧠 **Examples:**

* **Account Balance**
* **Inventory Level**
* **Bank Loan Amount**

💡 You can:

* Sum account balances **by account**
* But **NOT** by **time** (because balance changes with time)

✅ **Semi-Additive = Partially Summable**

**⛔ 3. Non-Additive Facts**

**Cannot** be summed at all

🧠 **Examples:**

* **Percentage**
* **Ratio**
* **Averages**

💡 You can:

* Use **average**, **max**, or **min** — but **not sum**

✅ **Non-Additive = Never Summable**

**📊 Summary Table**

| **Type** | **Can be Summed Across** | **Examples** |
| --- | --- | --- |
| **Additive** | All dimensions | Sales, Revenue, Quantity |
| **Semi-Additive** | Some (but not all) dimensions | Account Balance, Inventory |
| **Non-Additive** | Cannot be summed | Averages, Percentages, Ratios |

**🎯 Real-Life Example: Online Store**

Imagine a **fact table: sales\_fact**  
With columns:

| **Date** | **Product** | **Region** | **Quantity** | **Sales** | **Inventory** | **Discount %** |
| --- | --- | --- | --- | --- | --- | --- |
| 2025-07-30 | Laptop | US | 5 | 5000 | 20 | 10% |

Here:

* Quantity and Sales = ✅ **Additive**
* Inventory = 🔁 **Semi-Additive**
* Discount % = ⛔ **Non-Additive**

**📘 Types of Fact Tables**

There are **3 main types** of fact tables:

**1. ✅ Transactional Fact Table**

🔹 **Records one row per transaction or event**

🧠 **Examples:**

* Every sale in a shop
* Every item added to cart
* Every payment made

📅 Example row:

| **Date** | **Product** | **Customer** | **Quantity** | **Total Price** |
| --- | --- | --- | --- | --- |
| 2025-07-31 | Laptop | John | 1 | 1000 |

💡 Best when:

* You want **detailed** and **real-time** reporting

**2. 📆 Snapshot Fact Table**

🔹 **Takes a picture (snapshot) of data at regular time intervals**

🧠 **Examples:**

* Daily bank account balance
* Monthly inventory levels
* Weekly number of active users

📅 Example row:

| **Date** | **Account ID** | **Balance** |
| --- | --- | --- |
| 2025-07-31 | 123 | 5000 |

💡 Best when:

* You want to **track changes over time**

**3. 📊 Accumulating Snapshot Fact Table**

🔹 **Tracks the progress of a process (from start to finish)**

🧠 **Examples:**

* Order fulfillment (order placed → shipped → delivered)
* Application processing (applied → approved → completed)

📅 Example row:

| **Order ID** | **Start Date** | **Ship Date** | **Delivery Date** | **Status** |
| --- | --- | --- | --- | --- |
| ORD123 | 2025-07-01 | 2025-07-03 | 2025-07-05 | Delivered |

💡 Best when:

* You want to **monitor the life cycle of a process**

**📊 Summary Table**

| **Type** | **Description** | **Examples** |
| --- | --- | --- |
| **Transactional Fact** | Detailed records for each transaction | Sales, Payments, Orders |
| **Snapshot Fact** | Periodic snapshots of data | Daily balance, Monthly inventory |
| **Accumulating Snapshot Fact** | Tracks progress of a process | Order lifecycle, Application stages |

**🎯 Final Tip for Beginners:**

* Use **Transactional** for day-to-day detailed data
* Use **Snapshot** to track how things change over time
* Use **Accumulating Snapshot** to track process pipelines

**TYPES OF DIMENSIONS**

**🧠 1. Slowly Changing Dimension (SCD)**

**📌 Problem it solves:**

Customer or product info can **change over time** (like address, category, etc.)

**🔷 SCD Type 1: Overwrite old data (no history)**

❌ Old info is lost, only latest kept

**Example: CustomerDim\_SCD1**

| **CustomerID** | **Name** | **Country** |
| --- | --- | --- |
| C001 | John | USA |
| C002 | Sarah | UK |
| C003 | Ali | Canada |
| C004 | Ramesh | India |
| C005 | Sara | UK |
| C006 | Faizan | Pakistan |
| C007 | Priya | India |

🔁 Now John moves from USA → **Canada**

We just **overwrite**:

| **CustomerID** | **Name** | **Country** |
| --- | --- | --- |
| C001 | John | Canada |

🧠 Use when: You **don’t care about history**, just want latest.

**🔷 SCD Type 2: Keep full history (add new row)**

✅ Track every change with **new version** using surrogate key

**Example: CustomerDim\_SCD2**

| **CustomerSK** | **CustomerID** | **Name** | **Country** | **StartDate** | **EndDate** |
| --- | --- | --- | --- | --- | --- |
| 1 | C001 | John | USA | 2023-01-01 | 2024-12-31 |
| 2 | C001 | John | Canada | 2025-01-01 | NULL |
| 3 | C002 | Sarah | UK | 2023-01-01 | NULL |
| 4 | C003 | Ali | Canada | 2023-01-01 | NULL |
| 5 | C004 | Ramesh | India | 2023-01-01 | NULL |
| 6 | C005 | Sara | UK | 2023-01-01 | NULL |
| 7 | C006 | Faizan | Pakistan | 2023-01-01 | NULL |
| 8 | C007 | Priya | India | 2023-01-01 | NULL |

🧠 Use when: You want to **track changes over time** (very common in industry)

**🔷 SCD Type 3: Keep old + new in same row**

🔁 Track **just one change** — old + current value

**Example: CustomerDim\_SCD3**

| **CustomerID** | **Name** | **CurrentCountry** | **PreviousCountry** |
| --- | --- | --- | --- |
| C001 | John | Canada | USA |
| C002 | Sarah | UK | NULL |
| C003 | Ali | Canada | NULL |
| C004 | Ramesh | India | NULL |
| C005 | Sara | UK | NULL |
| C006 | Faizan | Pakistan | NULL |
| C007 | Priya | India | NULL |

🧠 Use when: You only need to track the **last change**, not full history.

**🧠 2. Role-Playing Dimension**

Same dimension table used for **different roles** in fact table

**🔷 Example: DateDim**

| **DateID** | **FullDate** | **Day** | **Month** | **Year** |
| --- | --- | --- | --- | --- |
| 1 | 2025-01-01 | Mon | Jan | 2025 |
| 2 | 2025-01-02 | Tue | Jan | 2025 |
| 3 | 2025-01-03 | Wed | Jan | 2025 |
| 4 | 2025-01-04 | Thu | Jan | 2025 |
| 5 | 2025-01-05 | Fri | Jan | 2025 |
| 6 | 2025-01-06 | Sat | Jan | 2025 |
| 7 | 2025-01-07 | Sun | Jan | 2025 |

**🔶 Used in OrderFact:**

| **OrderID** | **OrderDateID** | **ShipDateID** | **DeliveryDateID** | **Amount** |
| --- | --- | --- | --- | --- |
| O101 | 1 | 2 | 4 | 100 |
| O102 | 3 | 4 | 5 | 200 |

* OrderDateID, ShipDateID, DeliveryDateID — all link to same DateDim
* We use **different aliases/roles** during joins

🧠 Very common for **dates** in industry

**🧠 3. Conformed Dimension**

Shared across multiple fact tables — for **consistent reporting**

**🔷 Example: ProductDim**

| **ProductID** | **ProductName** | **Category** |
| --- | --- | --- |
| P001 | Laptop | Electronics |
| P002 | Headphones | Electronics |
| P003 | T-Shirt | Clothing |
| P004 | Book | Books |
| P005 | Shoes | Footwear |
| P006 | Phone | Electronics |
| P007 | Watch | Accessories |

Used in both:

* SalesFact → track product sold
* ReturnsFact → track product returned

🧠 Ensures **same product definitions** in all reports

**🧠 4. Junk Dimension**

Combines **low-cardinality fields** (flags, codes)

**🔷 Example: JunkDim**

| **JunkID** | **IsPromoUsed** | **IsReturned** | **Channel** | **DeliverySpeed** |
| --- | --- | --- | --- | --- |
| 1 | Yes | No | Online | Fast |
| 2 | No | No | MobileApp | Normal |
| 3 | Yes | Yes | InStore | Fast |
| 4 | No | Yes | Online | Slow |
| 5 | Yes | No | MobileApp | Normal |
| 6 | No | No | Online | Normal |
| 7 | Yes | Yes | Online | Fast |

* These are **not important enough** to make separate tables
* Combined into one JunkDim

🧠 Keeps your warehouse **clean and compact**

**🧠 5. Degenerate Dimension**

Appears in fact table only — no separate dimension table

**🔷 Example: SalesFact**

| **InvoiceNumber** | **ProductID** | **CustomerID** | **Quantity** | **Amount** |
| --- | --- | --- | --- | --- |
| INV1001 | P001 | C001 | 2 | 200 |
| INV1002 | P003 | C002 | 1 | 50 |
| INV1003 | P005 | C003 | 3 | 150 |
| INV1004 | P002 | C001 | 1 | 80 |

* InvoiceNumber is a **Degenerate Dimension**
* Has reporting value but **no extra info to put in a separate table**

🧠 Used for invoices, ticket numbers, order codes

**🧠 6. Outrigger Dimension**

A dimension **linked to another dimension**, not directly to the fact

**🔷 Example:**

**ProductDim**

| **ProductID** | **ProductName** | **SupplierID** |
| --- | --- | --- |
| P001 | Laptop | S001 |
| P002 | Headphones | S002 |
| P003 | T-Shirt | S003 |
| P004 | Shoes | S001 |
| P005 | Phone | S002 |
| P006 | Book | S004 |
| P007 | Watch | S003 |

**SupplierDim (outrigger)**

| **SupplierID** | **SupplierName** | **Country** |
| --- | --- | --- |
| S001 | TechWorld | USA |
| S002 | SoundCorp | Canada |
| S003 | ClothMate | India |
| S004 | BookPlanet | UK |

* SupplierDim is **not directly** linked to fact
* It's an **outrigger of ProductDim**

🧠 Used in **snowflake schema** when info is deeply related

**✅ Summary Table (Easy View)**

| **Type** | **What It Does** | **Key Use Example** |
| --- | --- | --- |
| SCD Type 1 | Overwrites old values | Current country only |
| SCD Type 2 | Keeps full history with new rows | Country changes over time |
| SCD Type 3 | Tracks current + previous in same row | Just last country change |
| Role-Playing | One dim used in different roles | DateDim for OrderDate, ShipDate |
| Conformed | Shared dim for many facts | ProductDim in Sales + Returns |
| Junk | Combines small flags or codes | PromoUsed, ReturnFlag, Channel |
| Degenerate | Key in fact table only | InvoiceNumber in SalesFact |
| Outrigger | Dimension inside a dimension | ProductDim → SupplierDim |

**APPROACHES TO WAREHOUSING:**

**🔷 1. Top-Down Approach (by Bill Inmon)**

**💡 Concept:**

Start by creating a **centralized data warehouse** first.  
Then, build **data marts** (smaller, department-specific databases) from that warehouse.

**🏗️ Step-by-Step:**

1. Collect all data from sources into a **central data warehouse**.
2. Clean and standardize the data.
3. Create **data marts** for specific departments (e.g., Sales, Marketing) if needed.

**📦 Real-Life Example:**

Imagine you're opening a **supermarket**:

* First, you build the **entire store** with all sections: fruits, bakery, clothing, electronics.
* Later, you let each section (Sales, HR, Finance) work in their specific area.

**✅ Advantages:**

* **High data quality and consistency**: One version of the truth.
* **Good for enterprise-wide solutions**.
* Scalable and structured for long-term use.

**❌ Disadvantages:**

* **Takes more time** and cost up front.
* **Slower to show results** (not ideal for urgent needs).
* Requires more planning and resources.

**🔶 2. Bottom-Up Approach (by Ralph Kimball)**

**💡 Concept:**

Start by building **data marts** first (small focused systems).  
Later, integrate them into a **central data warehouse**.

**🏗️ Step-by-Step:**

1. Build **data marts** for departments (Sales, Marketing).
2. Each mart contains only the data that team needs.
3. Later, connect all marts into a **larger warehouse**.

**📦 Real-Life Example:**

You open **individual shops** first (bakery, clothing, electronics),  
Then, eventually combine them into a big **shopping mall** (warehouse).

**✅ Advantages:**

* **Faster results** – You can build and use each part quickly.
* **Lower initial cost** and easier to implement.
* Flexible for small teams or projects.

**❌ Disadvantages:**

* **Data may not be consistent** across marts.
* Integration of marts later can be **complex**.
* Can become messy if not well-managed.

**📊 Quick Comparison Table:**

| **Feature** | **Top-Down (Inmon)** | **Bottom-Up (Kimball)** |
| --- | --- | --- |
| First Step | Central data warehouse | Department-specific marts |
| Speed | Slower | Faster |
| Data Consistency | High | Moderate |
| Cost at Start | High | Low |
| Suitable For | Large enterprises | Smaller teams or quick wins |
| Implementation | Complex, but organized | Simple, but can get messy |

**🧠 Which One Should You Use?**

* Use **Top-Down** if you want long-term consistency and have enough time/resources.
* Use **Bottom-Up** if you want **quick results** or are working on a smaller project.
*  ⭐ **Star Schema** → when you prioritize **speed, simplicity**, and **easy reporting**
*  ❄️ **Snowflake Schema** → when you care about **data accuracy, structure**, and **storage efficiency**