Author: Madhurima Rawat

Slowly Changing Dimensions (SCD)

Managing Changes in Dimensional Data Over Time ⏳📈

📌 What are Slowly Changing Dimensions?

Slowly Changing Dimensions (SCD) refer to dimensions that change slowly over time, rather than on a regular schedule. In data warehousing, it's important to track these changes to preserve historical accuracy and support meaningful analysis. SCD techniques allow us to manage how historical data is stored and updated when changes occur in dimension attributes.

✅ SCD Type 0 — Fixed Dimension (No Changes Allowed) 🔒🛑

- **Behavior**:
  Once the data is loaded, it **never** changes. 🚫
  Any updates or changes in the source system are **ignored**.
  This type is used when historical accuracy is essential and no data changes should ever be reflected.

- **Example**:
  A person's **Date of Birth** 🎂 or **Country of Birth** 🌍, which should remain constant over time.

- **Outcome**:
  The data remains **exactly as it was initially loaded**, no updates are made.
  Any new data that conflicts with it is **discarded**. ❌

✅ SCD Type 1 — Overwrite the Existing Data 📝♻️

- **Behavior**:
  The dimension record is **updated in place**. 🔧
  Only the **current** data is stored; **history is lost**.

- **When to Use**:
  When historical changes are **not important**, and only the **most up-to-date information** matters. 🗓️

- **Example**:
  Correcting a **spelling mistake** 🖍️ in a customer's name or updating a **phone number** ☎️.

- **Outcome**:
  After updates, you see **only the latest values**, and **no previous information** is retained. 📄

✅ SCD Type 2 — Add a New Record (Full History Preserved) 🗃️📚

- **Behavior**:
  A new **row** is inserted for every change in data. ➕
  The old record is **closed**, and the new record becomes **current**.

- **Features**:
  Maintains a **complete history** of all changes. 🕰️
  Usually uses fields like **start date**, **end date**, and **current flag**. ⏳✅

- **When to Use**:
  When it is important to **track historical data** over time. 📈

- **Example**:
  Tracking a customer's **address changes** 🏡 over the years.

- **Outcome**:
  You can **query** both the **current** and **historical versions** of a record. 🔍

✅ SCD Type 3 — Store Previous Data in Additional Columns 🗂️🔄

- **Behavior**:
  Keeps **limited history** by adding extra columns for **previous values**. 📑
  Only a predefined number of changes are stored (typically just **current** and **previous**).

- **When to Use**:
  When only the **previous** value is needed for comparison or reporting. 🧐

- **Example**:
  Tracking a **previous address** 🏠 alongside the **current address** 🏡.

- **Outcome**:
  You have **two columns**: the **current** and **previous** values.
  Historical tracking is **limited**, and you **lose older history** beyond what's stored. 🚪

✅ SCD Type 4 — Current and History Tables (Separate Tables) 📄📜

- **Behavior**:
  Separates **current** data and **historical** data into two **distinct tables**. 📂
  The **current table** holds the **latest data**, and the **history table** stores **all previous versions**.

- **When to Use**:
  When you need **fast access** ⚡ to current data but also need to **retain a full history**.

- **Example**:
  A **Customer Current Table** 👤 for operational use and a **Customer History Table** 🕰️ for audit/reporting.

- **Outcome**:
  You can **quickly** get the **current records**, and the **history table** gives you the **change log**. 📓

✅ SCD Type 6 — Hybrid Approach (Combination of 1, 2, and 3) 🧩🛠️

- **Behavior**:
  Combines features of **SCD Type 1, Type 2, and Type 3**. 🔗
  It stores **current data**, **previous data**, and **history records** (like Type 2).
  Includes **versioning**, **start/end dates**, and **previous values** in the same table. 📋

- **When to Use**:
  When you need **comprehensive history**, **previous values**, and **current data**, all in one place. 🗃️🕵️‍♀️

- **Example**:
  Tracking a customer's **address changes**, keeping both **current/previous** values, and maintaining **full history** for audits. 🏠📜

- **Outcome**:
  You get a **rich dimension** table that supports **point-in-time analysis**, **current views**, and **auditability**. ✅📊

🟦 Summary Table 🗂️✅

Properties 📋	SCD Type 0 🚫	SCD Type 1 🔄	SCD Type 2 ⚡	SCD Type 3 📊	SCD Type 4 🗃️	SCD Type 6 🔀
Definition 📖	No changes tracked	Overwrites old data	Creates a new row for each change	Keeps limited history in columns	Stores historical data in a separate table	Combination of Type 1, Type 2, and Type 3
Example Use Case 🛍️	Immutable data (e.g., product SKUs)	Customer phone number updates	Employee salary changes over time	Product price changes with a "previous price" column	Historical customer address changes stored separately	Tracking customer loyalty level with history and latest status
Advantages ✅	Fastest and simplest	Easy to implement	Retains full history	Efficient for small history needs	Combines Type 1 & Type 2 benefits	Provides both full history and latest value in the same table
Disadvantages ❌	No history retained	Historical data is lost	More storage required	Limited historical tracking	Complexity in managing separate tables	Increased complexity due to combined methods
Tracking Method 🔍	Not applicable	Overwrite existing record	New row with start/end date or version	New column for previous value	Separate historical table linked via keys	New row for history + current values updated
Real-Life Example 🌟	Product barcodes never change	Updating a customer's email address	Tracking patient's medical history	Tracking latest two addresses of a customer	Storing detailed job history for employees	Tracking customer membership tier with history and latest tier
Real-World Example: Employee Department Change 📊						
Scenario	John Doe moves from **Sales → Marketing** on **2025-03-01**					
Surrogate Key 🆔	Not needed (Key remains same)	Same primary key, old value overwritten	New surrogate key per change	Same primary key, added "Prev_Dept" column	New surrogate key stored in history table	New surrogate key per change, plus "Prev_Dept" column
Stored Data Format 🗂️	Emp_ID = 101, Dept = Sales (Never changes)	Emp_ID = 101, Dept = Marketing (Old value lost)	**Row 1:** Emp_SK = 5001, Emp_ID = 101, Dept = Sales, Valid From: 2024-01-01, Valid To: 2025-02-28, Active: ❌ **Row 2:** Emp_SK = 6001, Emp_ID = 101, Dept = Marketing, Valid From: 2025-03-01, Valid To: NULL, Active: ✅	Emp_ID = 101, Dept = Marketing, Prev_Dept = Sales	**Main Table:** Emp_ID = 101, Dept = Marketing **History Table:** Emp_SK = 5001, Emp_ID = 101, Dept = Sales, Valid From: 2024-01-01, Valid To: 2025-02-28	**Row 1:** Emp_SK = 5001, Emp_ID = 101, Dept = Sales, Prev_Dept = NULL, Valid From: 2024-01-01, Valid To: 2025-02-28, Active: ❌ **Row 2:** Emp_SK = 6001, Emp_ID = 101, Dept = Marketing, Prev_Dept = Sales, Valid From: 2025-03-01, Valid To: NULL, Active: ✅
Valid From - Valid To 🗓️	Not applicable (No history tracked)	Not applicable (Overwrites directly)	Stored in separate rows with start and end dates	Only the latest and previous values stored	Stored in a separate history table	Stored in separate rows with start and end dates, plus previous value column
Active Record Flag 🚩	Not applicable (Single record remains)	Not applicable (Only latest record kept)	Old record set to ❌ (inactive), new record set to ✅ (active)	Not applicable (Only latest and previous stored)	Managed in the history table with inactive records	Old record set to ❌ (inactive), new record set to ✅ (active)

SCD Type	Description	History Tracking
SCD 0	No changes allowed; data stays the same 🔒	None 🚫
SCD 1	Overwrites the existing data 📝	No historical data retained ❌
SCD 2	Adds a new record and tracks history with dates/flags 🗃️	Full history preserved 📚
SCD 3	Stores previous value in additional columns 📑	Limited history (previous only) 🕰️
SCD 4	Current and history tables are separated 📂	Full history in separate table 🗄️
SCD 6	Combines SCD 1, 2, and 3 with current, previous, and full history 🧩	Comprehensive history 🗃️ ✅

SCD Implementation in MySQL

✅ Setup

We will:

- Have an `original_table` (it never changes)
- Create separate tables for **each SCD type** (`scd_0` , `scd_1` , `scd_2` , `scd_3` , `scd_4` , `scd_6`)
- Do updates **only** in the **SCD tables** (not in `original_table`)
- After each update, **SELECT** and **PRINT** the `original_table` and the updated SCD table to compare.

🟦 STEP 1: Create `original_table`

Command

```
DROP TABLE IF EXISTS original_table;

CREATE TABLE original_table (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50),
    version INT DEFAULT 1 -- version for scd6
);
```

```
INSERT INTO original_table (id, name, city)
VALUES
(1, 'John Doe', 'New York'),
(2, 'Jane Smith', 'Los Angeles'),
(3, 'Alice Brown', 'Chicago');
```

Explanation

1. **DROP TABLE IF EXISTS** makes sure any existing `original_table` is deleted, preventing duplication or errors.
2. **CREATE TABLE** defines a new `original_table` with `id`, `name`, `city`, and a `version` column (used in SCD6).
3. **INSERT INTO** adds three initial records into the table.

Output

```
id  | name        | city         | version
----|-------------|--------------|---------
1   | John Doe    | New York     | 1
2   | Jane Smith  | Los Angeles  | 1
3   | Alice Brown | Chicago      | 1
```

What we understand

- This is the **baseline** dataset, which **never changes** throughout our scenario.

🟦 STEP 2: Create Separate SCD Tables (0, 1, 2, 3, 4, 6)

🔸 SCD Type 0 Table (No changes allowed)

Command

```
CREATE TABLE scd_0 AS SELECT * FROM original_table;
```

Explanation

- We create `scd_0` as a **copy** of `original_table`.
- **Type 0** means **no changes ever** allowed.

Output after creation

```
id  | name        | city         | version
----|-------------|--------------|---------
1   | John Doe    | New York     | 1
2   | Jane Smith  | Los Angeles  | 1
3   | Alice Brown | Chicago      | 1
```

🔸 SCD Type 1 Table (Overwrite current row)

Command

```
CREATE TABLE scd_1 AS SELECT * FROM original_table;
```

Explanation

- `scd_1` is a **copy** of the original.
- **Type 1** overwrites records; no history is kept.

Output after creation

Same as `original_table`.

🔸 SCD Type 2 Table (Track history with start and end dates)

Command

```
CREATE TABLE scd_2 (
    surrogate_key INT AUTO_INCREMENT PRIMARY KEY,
    id INT,
    name VARCHAR(50),
    city VARCHAR(50),
    start_date DATE,
    end_date DATE,
    current_flag CHAR(1)
);

INSERT INTO scd_2 (id, name, city, start_date, end_date, current_flag)
SELECT id, name, city, CURDATE(), NULL, 'Y' FROM original_table;
```

Explanation

- `scd_2` stores **history**, tracks **start** and **end** dates, and shows **current records** via `current_flag`.
- Initial load sets `start_date` to **today's date**, `end_date` as NULL (still active), and `current_flag = 'Y'`.

Output after creation

```
surrogate_key | id | name        | city         | start_date | end_date | current_flag
--------------|----|-------------|--------------|------------|----------|-------------
1             | 1  | John Doe    | New York     | 2025-03-10 | NULL     | Y
2             | 2  | Jane Smith  | Los Angeles  | 2025-03-10 | NULL     | Y
3             | 3  | Alice Brown | Chicago      | 2025-03-10 | NULL     | Y
```

🔹 SCD Type 3 Table (Keep previous values in additional columns)

Command

```
CREATE TABLE scd_3 (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50),
    prev_city VARCHAR(50)
);

INSERT INTO scd_3 (id, name, city, prev_city)
SELECT id, name, city, NULL FROM original_table;
```

Explanation

- We add an extra column `prev_city` to store the **last city** before it gets updated.

Output after creation

```
id  | name        | city         | prev_city
----|-------------|--------------|-----------
1   | John Doe    | New York     | NULL
2   | Jane Smith  | Los Angeles  | NULL
3   | Alice Brown | Chicago      | NULL
```

🔹 SCD Type 4 Table (Current + Historical separate tables)

Command (Current Table)

```
CREATE TABLE scd_4_current AS SELECT * FROM original_table;
```

Command (History Table)

```sql
CREATE TABLE scd_4_history (
    history_id INT AUTO_INCREMENT PRIMARY KEY,
    id INT,
    name VARCHAR(50),
    city VARCHAR(50),
    change_date DATE
);
```

Explanation

- **Current table** stores active records.
- **History table** stores **old versions** of changed records.

Output

`scd_4_current` starts identical to `original_table`.
`scd_4_history` is empty initially.

🔸 SCD Type 6 Table (Combines 1, 2, and 3)

Command

```sql
CREATE TABLE scd_6 (
    surrogate_key INT AUTO_INCREMENT PRIMARY KEY,
    id INT,
    name VARCHAR(50),
    city VARCHAR(50),
    prev_city VARCHAR(50),
    start_date DATE,
    end_date DATE,
    current_flag CHAR(1),
    version INT
);

INSERT INTO scd_6 (id, name, city, prev_city, start_date,
end_date, current_flag, version)
SELECT id, name, city, NULL, CURDATE(), NULL, 'Y', 1
FROM original_table;
```

Explanation

- Tracks **previous values**, **start/end dates**, and **version numbers** in one table.

Output after creation

```
surrogate_key | id | name        | city         | prev_city | start_date | end_date | current_
--------------|----|-------------|--------------|-----------|------------|----------|---------
1             | 1  | John Doe    | New York     | NULL      | 2025-03-10 | NULL     | Y
2             | 2  | Jane Smith  | Los Angeles  | NULL      | 2025-03-10 | NULL     | Y
3             | 3  | Alice Brown | Chicago      | NULL      | 2025-03-10 | NULL     | Y
```

🟦 STEP 3: Update Scenarios

🔸 SCD Type 0 (No change allowed)

Command

```
-- No updates performed on scd_0

SELECT 'ORIGINAL TABLE' AS table_name, id, name, city
FROM original_table;

SELECT 'SCD_0 TABLE' AS table_name, id, name, city FROM scd_0;
```

Explanation

- **No changes** are allowed. Both tables should be identical.

Output

```
ORIGINAL TABLE
id  | name        | city
----|-------------|-------------
1   | John Doe    | New York
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago

SCD_0 TABLE
id  | name        | city
----|-------------|-------------
1   | John Doe    | New York
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago
```

What we understood:

- **SCD_0** data remains **frozen**—no updates, no history tracking.

🔸 SCD Type 1 (Overwrite current row)

Command

```
UPDATE scd_1
SET city = 'San Francisco'
WHERE id = 1;

SELECT 'ORIGINAL TABLE' AS table_name, id, name, city FROM original_table;

SELECT 'SCD_1 TABLE' AS table_name, id, name, city FROM scd_1;
```

Explanation

- We **overwrite** John Doe's city in `scd_1` .
- No history is maintained.

Output

```
ORIGINAL TABLE
id  | name        | city
----|-------------|--------------
1   | John Doe    | New York
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago

SCD_1 TABLE
id  | name        | city
----|-------------|---------------
1   | John Doe    | San Francisco
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago
```

What we understood:

- Changes in **SCD_1** are immediate and **overwrite existing data**, no previous record is saved.

🔸 SCD Type 2 (Add a new row for history)

Command

```
UPDATE scd_2
SET end_date = CURDATE(), current_flag = 'N'
WHERE id = 2 AND current_flag = 'Y';

INSERT INTO scd_2 (id, name, city, start_date, end_date,
current_flag)
```

```
VALUES (2, 'Jane Smith', 'Houston', CURDATE(), NULL, 'Y');

SELECT 'ORIGINAL TABLE' AS table_name, id, name, city
FROM original_table;

SELECT 'SCD_2 TABLE' AS table_name, id, name, city,
start_date, end_date, current_flag FROM scd_2;
```

Explanation

- **Expire** the current active record (`end_date` set, flag 'N').
- **Insert** a new record for `Jane Smith` with city `Houston` .

Output

```
ORIGINAL TABLE
id  | name        | city
----|-------------|--------------
1   | John Doe    | New York
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago

SCD_2 TABLE
surrogate_key | id | name        | city         | start_date | end_date   | current_flag
--------------|----|-------------|--------------|------------|------------|--------------
1             | 1  | John Doe    | New York     | 2025-03-10 | NULL       | Y
2             | 2  | Jane Smith  | Los Angeles  | 2025-03-10 | 2025-03-10 | N
4             | 2  | Jane Smith  | Houston      | 2025-03-10 | NULL       | Y
3             | 3  | Alice Brown | Chicago      | 2025-03-10 | NULL       | Y
```

What we understood:

- **SCD_2** keeps **history**; we can see both the old and new versions of Jane Smith.

🔸 SCD Type 3 (Track previous value in separate column)

Command

```
UPDATE scd_3
SET prev_city = city,
    city = 'Miami'
WHERE id = 3;

SELECT 'ORIGINAL TABLE' AS table_name, id, name, city FROM original_table;

SELECT 'SCD_3 TABLE' AS table_name, id, name, city, prev_city FROM scd_3;
```

Explanation

- We **move** current `city` to `prev_city`.
- Update `city` to `Miami`.

Output

```
ORIGINAL TABLE
id  | name        | city
----|-------------|-------------
1   | John Doe    | New York
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago

SCD_3 TABLE
id  | name        | city    | prev_city
----|-------------|---------|-----------
1   | John Doe    | New York | NULL
2   | Jane Smith  | Los Angeles | NULL
3   | Alice Brown | Miami   | Chicago
```

What we understood:

- **SCD_3** tracks only **one previous value** (`prev_city`).

🔸 SCD Type 4 (Insert history row and update current)

Command

```
INSERT INTO scd_4_history (id, name, city, change_date)
SELECT id, name, city, CURDATE() FROM scd_4_current WHERE id = 1;

UPDATE scd_4_current
SET city = 'Seattle'
WHERE id = 1;

SELECT 'ORIGINAL TABLE' AS table_name, id, name, city FROM original_table;

SELECT 'SCD_4_CURRENT TABLE' AS table_name, id,
name, city FROM scd_4_current;

SELECT 'SCD_4_HISTORY TABLE' AS table_name, id,
name, city, change_date FROM scd_4_history;
```

Explanation

- Move old record into **history** table.

- Update current table with the new value.

Output

```
ORIGINAL TABLE
id  | name        | city
----|-------------|--------------
1   | John Doe    | New York
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago

SCD_4_CURRENT TABLE
id  | name        | city
----|-------------|----------
1   | John Doe    | Seattle
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago

SCD_4_HISTORY TABLE
history_id | id | name      | city      | change_date
-----------|----|-----------|-----------|------------
1          | 1  | John Doe  | New York  | 2025-03-10
```

What we understood:

- **SCD_4** separates **current** and **historical** data in different tables.

🔸 SCD Type 6 (Combination of SCD 1, 2, 3)

Command

```
UPDATE scd_6
SET end_date = CURDATE(), current_flag = 'N'
WHERE id = 2 AND current_flag = 'Y';

INSERT INTO scd_6 (id, name, city, prev_city, start_date, end_date,
current_flag, version)
VALUES (2, 'Jane Smith', 'Boston', 'Houston', CURDATE(), NULL, 'Y', 2);

SELECT 'ORIGINAL TABLE' AS table_name, id,
name, city FROM original_table;

SELECT 'SCD_6 TABLE' AS table_name, id,
name, city, prev_city, start_date,
end_date, current_flag, version FROM scd_6;
```

Explanation

- Expire old row.
- Add new row with `prev_city`, `current_flag` as `Y`, increment version.

Output

```
ORIGINAL TABLE
id  | name        | city
----|-------------|-------------
1   | John Doe    | New York
2   | Jane Smith  | Los Angeles
3   | Alice Brown | Chicago

SCD_6 TABLE
surrogate_key | id | name        | city     | prev_city | start_date | end_date   | current_fl
--------------|----|-------------|----------|-----------|------------|------------|-----------
1             | 1  | John Doe    | New York | NULL      | 2025-03-10 | NULL       | Y
2             | 2  | Jane Smith  | Los Angeles | NULL   | 2025-03-10 | 2025-03-10 | N
4             | 2  | Jane Smith  | Boston   | Houston   | 2025-03-10 | NULL       | Y
3             | 3  | Alice Brown | Chicago  | NULL      | 2025-03-10 | NULL       | Y
```

What we understood:

- **SCD_6** combines **history**, **previous values**, and **versioning** into one table.

🟦 Final Structure Recap

Table	Type	Notes
original_table	Original	No changes ever made
scd_0	Type 0	No updates allowed
scd_1	Type 1	Overwrites data
scd_2	Type 2	Maintains full history with start/end dates
scd_3	Type 3	Stores previous value in another column
scd_4_current	Type 4	Current records only
scd_4_history	Type 4	Historical changes
scd_6	Type 6	Combines overwrites, history, and previous values

🟦 How It Works Recap

- The **original_table** acts as an **immutable** source of truth.
- Each **SCD** type handles data changes in its **own unique way**, letting us control how much history we retain.
- The **outputs** after each update reflect these strategies clearly.