Chapter 5: Dimensional Modeling - Dimensions

1. The Power of Context: An Introduction to Dimension Tables

Welcome back to DSI310. In our previous lecture, we laid the theoretical groundwork for analytical data modeling by distinguishing between OLTP (Online Transactional Processing) and OLAP (Online Analytical Processing) systems.[1] We learned that our source systems, the legacy Chinook and Northwind databases, are classic OLTP examples, designed for the rapid, secure processing of daily transactions.[1] We also established that our ultimate goal is to build a unified OLAP system—a data warehouse—optimized for a completely different purpose: comprehensive business analysis and strategic decision-making.[1] The centerpiece of this analytical system is the

Kimball Star Schema, a simple yet profoundly powerful design that separates transactional data from its descriptive context.[2]

This week, we begin the hands-on process of building our data warehouse by focusing on the most critical component for providing business context: the **dimension tables**. While fact tables, which we will discuss next week, contain the "what" of our business (the measures and quantitative metrics), dimension tables provide the "who, what, where, when, and how."[2] A fact table without dimensions is a collection of meaningless numbers. A sales transaction amount of

$180 has little value in isolation. However, when we link it to a dimension table that tells us it was sold to a specific customer, on a specific date, by a specific employee, and for a specific product, that number is transformed into **actionable business intelligence**.[2]

The overarching objective of this lecture is to equip you with the knowledge and skills to design robust, scalable, and intuitive dimension tables for the OmniCorp project. This involves understanding their purpose, meticulously planning their structure, and, most importantly, mastering the techniques required to handle changes over time, a concept known as **Slowly Changing Dimensions (SCD)**.

2. Anatomy of a Dimension Table: The "Who, What, Where, When, How"

A dimension table is a cornerstone of the dimensional model, providing all the descriptive attributes needed to "slice" and "dice" the data in the fact table.[2] They are typically smaller and have fewer rows than fact tables but contain a wide array of textual and categorical attributes.[2] A key design principle is that a dimension table should contain all the descriptive information related to a specific entity, even if that information is considered "junk" in a normalized OLTP system.

For instance, the DimCustomer table, which we will design for OmniCorp, will contain every descriptive attribute we might ever want to analyze about a customer: their name, address, city, country, and even their company name.[2] A business analyst can then use these attributes to answer complex questions with simple queries, such as "What was our total revenue from customers in London, UK?" or "Which companies are our top 10 buyers?"[2] This is the essence of a dimensional model: simplifying complex business questions by pre-structuring the data for analytical purposes.

3. Designing Core Dimensions for the OmniCorp Project

The OmniCorp project is a masterclass in dimension design because it requires us to unify data from two distinct legacy systems: Chinook (a music store) and Northwind (a food and beverage distributor).[3] The project assignment outlines the core dimensions we must create to enable cross-business analytics [3]:

DimCustomer, DimEmployee, DimProduct, DimTime, and DimSourceSystem.[2] Each of these dimensions presents a unique set of data engineering challenges that we must solve.

3.1 DimCustomer: Unifying a Global Customer Base

The DimCustomer table is designed to be the single source of truth for all customer information, consolidating data from the Customer table in Chinook and the Customers table

in Northwind.[2]

- **Purpose:** The primary purpose of this dimension is to provide a comprehensive view of all customers, allowing business analysts to analyze sales and revenue based on customer demographics and location.[2]
- **Attributes:** The dimension will contain attributes such as CustomerID, CustomerName, CompanyName, City, State, Country, PostalCode, Phone, and Email.[2]
- **Data Engineering Challenges:** The most significant challenge here is handling the conflicting primary keys. The CustomerID in Chinook is an integer, while the CustomerID in Northwind is an alphanumeric text string.[3] This is a classic data integration problem.
- **Proposed Solution:** A robust and scalable solution is to create a new, unique **surrogate key** for the DimCustomer table.[3] A surrogate key is a system-generated identifier that has no business meaning but is guaranteed to be unique within our data warehouse. We can achieve this by either:
    1. **Prefixing:** A simple strategy is to prepend a source system identifier to the original CustomerID. For example, a customer from Chinook might have a CustomerID of C-123, while a customer from Northwind might have N-XYZ. This ensures a unique key space for our unified dimension.
    2. **Hashing:** A more advanced approach is to use a cryptographic hash function on the original source key. This creates a unique, fixed-length identifier that is impervious to collisions and is often preferred for a truly conformed dimension that can easily be shared across multiple data marts.

Regardless of the approach, the original source keys (e.g., chinook_customer_id, northwind_customer_id) should be retained in the dimension table as **natural keys** for auditing and traceability. The project assignment requires you to justify your choice for handling these data inconsistencies, so having a clear strategy for the CustomerID unification is paramount.[3]

3.2 DimEmployee: Analyzing Performance and Hierarchy

The DimEmployee table serves to unify employee records from the Employee table in both the Chinook and Northwind databases.[2]

- **Purpose:** This dimension is crucial for understanding sales performance by individual employee and by their reporting structure.[2] It allows for analyses like "What is the total sales amount for all 'Sales Support Agent' employees from Chinook?" [3]
- **Attributes:** Key attributes include EmployeeID, EmployeeName, Title, City, Country, and a ReportsTo attribute to support hierarchical analysis.[2]
- **Data Engineering Challenges:** Similar to the DimCustomer table, the primary challenge

is to unify the EmployeeID from the two different sources. Additionally, we must handle the ReportsTo field, which establishes a hierarchical relationship. This is a common pattern in dimensional modeling, and we will need to create a parent-child relationship within the dimension table itself to allow for roll-up analysis.

3.3 DimProduct: Unifying Disparate Business Items

The DimProduct table is arguably the most complex of the primary dimensions because it must bridge the gap between two completely different types of products: music tracks from Chinook and food/beverage items from Northwind.[2]

- **Purpose:** The purpose is to provide a single, unified view of all products sold across both business units, enabling sales analysis by product, category, genre, or composer.[2]
- **Attributes:** The table will include attributes from both source systems, such as ProductID, ProductName (or Track Name), CategoryName (e.g., 'Rock', 'Beverages'), GenreName, Composer, and UnitPrice.[2]
- **Data Engineering Challenges:**
  1. **Data Type Conflicts:** The ProductID from Northwind and TrackId from Chinook will need to be unified using a surrogate key strategy, as discussed for DimCustomer.[3]
  2. **Handling Missing Attributes:** The Northwind products have a Category, but no Genre or Composer. Conversely, Chinook tracks have a Genre and Composer but no Category. The solution is to include all relevant attributes in the DimProduct table and handle missing data gracefully. For example, a Northwind product's GenreName and Composer would be set to a placeholder like 'N/A' or NULL, while a Chinook track's CategoryName would be set to NULL or 'N/A'.[3] This ensures that the dimension is complete and conformed for all products, allowing us to perform analysis across both business units.

3.4 DimTime: The Universal Dimension

The DimTime dimension is a special type of dimension. It is arguably the most powerful and is often pre-built and shared across all projects within an organization.

- **Purpose:** This dimension provides all the temporal attributes necessary for time-series analysis.[2] Instead of relying on a simple date field, a
  DimTime table allows us to easily slice data by day, week, month, quarter, or year without complex date-based functions in every query.

- **Attributes:** It contains granular attributes like DateKey (in YYYYMMDD format), FullDate, DayOfMonth, DayOfWeek, Month, Quarter, and Year.[2]
- **Implementation:** A DimTime table is typically populated once with a large range of dates (e.g., from 1900 to 2100) and then reused across all analytical projects. The DateKey serves as the primary key that links to the fact table.

3.5 DimSourceSystem: The Simplest, Most Powerful Dimension

The DimSourceSystem dimension is a simple but critical table for the OmniCorp project.

- **Purpose:** Its sole purpose is to distinguish the origin of the data.[2] It allows us to easily filter and compare data between the Chinook and Northwind businesses, a key deliverable of the project assignment.[3]
- **Attributes:** It contains just two attributes: SourceSystemID and SourceSystemName (e.g., 'Chinook', 'Northwind').[2]
- **Value:** By adding this dimension to our schema, a business analyst can perform a simple query to compare the total revenue generated from the music business versus the food and beverage business with a single GROUP BY clause on the SourceSystemName attribute.[3]

4. Managing Change: The Slowly Changing Dimension (SCD) Problem

The dimensions we design are not static; their attributes can change over time. A customer might move to a new city, an employee might get a new job title, or a product's price might be updated. This presents a critical problem for a data warehouse: how do we handle these changes in a way that preserves historical accuracy? This is the **Slowly Changing Dimension (SCD)** problem, a fundamental concept in dimensional modeling.

There are several types of SCDs, each with its own trade-offs. For this course, we will focus on the two most common and important types: Type 1 and Type 2.

4.1 SCD Type 1: The Overwrite Method

SCD Type 1, or the **overwrite method**, is the simplest approach to handling dimension

changes.

- **Logic:** When a change occurs in a dimension attribute, the old value is simply overwritten with the new one. No history is preserved.
- **Example:** Imagine a customer named Jane Doe (CustomerID 123) lives in New York. A year later, she moves to London.
    - **Initial State (DimCustomer):**

CustomerID	CustomerName	City	State	Country
123	Jane Doe	New York	NY	USA

- **After Update (Jane moves to London):**

CustomerID	CustomerName	City	State	Country
123	Jane Doe	London	NULL	UK

- **Pros:**
    - **Simplicity:** It is easy to implement and requires minimal code.
    - **Space Efficiency:** It does not increase the size of the dimension table.
- **Cons:**
    - **Loss of History:** The primary drawback is that we lose all historical data. After Jane's city is updated, we can no longer query our historical sales data to see what revenue was generated from customers in 'New York' during a period when Jane was still there. All historical facts related to Jane Doe will now appear as if they occurred while she was in London. This can lead to inaccurate historical reporting.

SCD Type 1 is only suitable for attributes where historical accuracy is not important, such as a misspelling in a name or a typo in a phone number. For any attribute that is used for historical analysis, this method is unacceptable.

4.2 SCD Type 2: The New Row Method

SCD Type 2, or the **new row method**, is the most common and powerful technique for handling dimension changes because it **preserves a complete history** of the dimension.

- **Logic:** When a change occurs in a dimension attribute, a new row is added to the dimension table to capture the new information. The original row is closed out by setting

an end_date and marking it as no longer current. The new row is given a new surrogate key, a start_date, and is marked as the current record.

- **Example:** Let's use the same scenario of Jane Doe moving from New York to London. We'll add two new columns to our dimension table: SCD_ValidFrom (the start date of the record) and SCD_ValidTo (the end date), and a flag SCD_IsCurrent to identify the most recent record.
- **Initial State (Before Change):**

SurrogateKey	CustomerID	CustomerName	City	SCD_ValidFrom	SCD_ValidTo	SCD_IsCurrent
1	C-123	Jane Doe	New York	2023-01-01	NULL	True

- **After Update (Jane moves to London on 2024-05-15):**

SurrogateKey	CustomerID	CustomerName	City	SCD_ValidFrom	SCD_ValidTo	SCD_IsCurrent
1	C-123	Jane Doe	New York	2023-01-01	2024-05-14	False
2	C-123	Jane Doe	London	2024-05-15	NULL	True

- **How it works with the Fact Table:** The fact table links to the dimension table using the **SurrogateKey**. A sales transaction that occurred on 2024-01-10 would link to SurrogateKey 1. A sales transaction that occurred on 2024-06-20 would link to SurrogateKey 2. This design ensures that all historical sales data remains linked to the correct state of the customer at the time of the transaction, preserving the historical record.
- **Pros:**
  - **Preserves History:** This is the key advantage. You have a complete, auditable history of all changes to a dimension.
  - **Supports Historical Analysis:** You can now accurately answer questions like, "What were our total sales from customers in New York, and what were they after those customers moved to London?"
- **Cons:**
  - **Increased Storage:** Each change creates a new row, which can lead to larger dimension tables.
  - **More Complex Logic:** The data loading and query logic are more complex, as you must handle date ranges and is_current flags.

4.3 Comparison Table: SCD Type 1 vs. Type 2

Feature	SCD Type 1 (Overwrite)	SCD Type 2 (New Row)
History Preservation	None. History is lost.	Full. History is preserved.
Storage Impact	Minimal. Rows are updated, not added.	High. Each change creates a new row.
Implementation Complexity	Simple. A straightforward UPDATE statement.	Complex. Requires managing multiple rows, start_date, end_date, and is_current flags.
Best Use Case	Correcting typos; attributes where history is not important.	Tracking changes in key attributes for historical analysis.

5. From Dimensions to a Unified Data Pipeline

The dimensional modeling concepts we discussed today are not just theoretical; they are an essential step in the end-to-end data pipeline we are building throughout this course. The process begins with raw, un-transformed data residing in the **Landing Zone** from sources like our relational databases or APIs.[2] Next, we perform cleansing and initial transformations in the

Staging Zone to prepare the data for modeling.[2] The final, crucial step is to build our dimensional model in the

Integration Zone, where the cleaned dimension data is combined and loaded into the Dim tables we designed today.[2] These final, curated tables are then used by our business intelligence tools to power analytics.[2]

The Dim tables we designed are the **T-Boxes** of our data model, representing the permanent, conceptual, and descriptive information about our business entities.[4] They define what a

customer, employee, or product

is in our analytical context. Next week, we will focus on the **Fact Table**, which is the **A-Box**, the collection of specific, transactional events that occurred.[4] The combination of these two components—T-Boxes and A-Boxes—will form a complete, robust, and logically sound star schema, capable of transforming OmniCorp's disparate data into a single source of truth for all business analysis.

6. Key Theories and Keywords

- **Dimension Table:** A table in a dimensional model that provides the descriptive context for the facts in a fact table. It contains the "who, what, where, when, and how" attributes of a business event.[2]
- **Surrogate Key:** A system-generated, unique identifier for each row in a dimension table. It has no business meaning but is essential for creating a unique key space in a data warehouse, especially when unifying data from multiple sources.[3]
- **Natural Key:** The original primary key from the source system (e.g., the CustomerID from the Northwind database). While not used for joins in the data warehouse, it is retained in the dimension table for traceability and auditing.
- **Slowly Changing Dimension (SCD):** A set of techniques for managing changes to dimension attributes over time, ensuring that the historical record of those changes is either preserved or overwritten, depending on the business requirements.
- **SCD Type 1:** The overwrite method. When a dimension attribute changes, the old value is overwritten with the new one. This is the simplest approach but results in a complete loss of history for that attribute.
- **SCD Type 2:** The new row method. When a dimension attribute changes, a new row is added to the dimension table to capture the new information. The old row is marked as inactive, and the new one as current, thereby preserving a complete history of the dimension.
- **T-Box (Terminological Box):** A concept from knowledge graphs and ontology that represents the permanent, conceptual schema and hierarchies of a domain.[4] Our dimension tables are the T-Boxes of our data model.[4]
- **A-Box (Assertion Box):** A concept from knowledge graphs and ontology that represents the specific, dynamic, and transactional facts or events that occur within a domain.[4] Our fact tables are the A-Boxes of our data model.[4]

Works cited

1. OLTP vs. OLAP: Differences and Applications - Snowflake, accessed September 10, 2025,

https://www.snowflake.com/en/fundamentals/olap-vs-oltp-the-differences/
2. dsi310_แบบฟอร์มเค้าโครงการบรรยาย 2025
3. dsi310: Data Lake Modeling
4. 2025 dsi310 week02
5. Abox - Wikipedia, accessed September 10, 2025, https://en.wikipedia.org/wiki/Abox