Chapter 2: SQL Data Sources & Data Ingestion

Welcome back to DSI310. Last week, we established the strategic role of the data engineer and the foundational architecture of a modern data platform, centered on the Data Lake and its layered zones.[1] We discussed the

Landing Zone as the initial, raw, and immutable entry point for all data.[1] This week, we begin our hands-on journey by focusing on the most common and critical data source for business analytics: the relational database.

Our objective for this chapter is threefold:

1. **To understand** what a relational database is and why it's the primary source for business data.
2. **To master** the basics of using SQL (Structured Query Language) for Exploratory Data Analysis (EDA) to understand our source data.
3. **To practice** ingesting raw, un-transformed data from these databases into our data lake's Landing Zone using Python.

1. The Relational Database: The Heartbeat of Business Operations

Before we can ingest data, we must understand its source. The vast majority of transactional data—from sales orders to customer records—is stored in relational databases. These databases are the core of Online Transaction Processing (OLTP) systems that support day-to-day business operations. They are optimized for fast, reliable, and consistent data modifications, which is why they are the primary choice for applications like e-commerce, banking, and inventory management.[2]

1.1 From Flat-Files to a Relational Model

To appreciate the power of a relational database, it's helpful to compare it to a simpler, older model: the flat-file database.[2]

- **Flat-File Database:** A flat-file system stores all data in a single table, like a spreadsheet.[2] While simple, this approach leads to significant problems:
    - **Data Redundancy:** The same information, such as a customer's address, is duplicated across many records, wasting storage and creating a risk of inconsistency.[2] For example, if a customer's address changes, it must be updated in

every single record where it appears.
    - **Data Anomalies:** If a customer's address changes, it must be updated in multiple places, making the process prone to error. Deleting a record might inadvertently delete useful data that is not stored anywhere else. Imagine a flat file containing car rental data. Deleting a record for a specific car rental would also delete the customer's name and contact information, even if that customer has other rental records.[2]
    - **Lack of Scalability:** Managing a single, massive file for a large business becomes inefficient and difficult to query, as the entire file must be scanned to find information.[2]
- **Relational Database:** A relational database solves these problems by organizing data into multiple, distinct tables with predefined relationships between them.[2]
    - **Data is segmented** into logical groups (e.g., a Customers table, an Orders table, a Products table).
    - **Relationships are defined** using Key Fields to link the tables together. This is the essence of a relational model.[2] For example, a single CustomerID in a Customers table can be linked to multiple rows in an Orders table, each representing a separate transaction.
    - The key to this system is that data is stored once, eliminating redundancy and ensuring data integrity. This design is also known as a normalized schema, which is highly efficient for transactional systems.

1.2 Key Concepts in Relational Databases

Understanding the core vocabulary is essential.

- **Table:** A collection of data elements arranged in rows and columns. In our OmniCorp project, Customers, Employees, and Orders are all tables in the Northwind database.[2]
- **Field (or Column):** A single item of data about a thing. For example, CustomerID or CompanyName are fields within the Customers table.[2] The course materials also define various data types that a field can hold, such as Alphanumeric/text, Number (real or integer), Date, and Boolean values.[2] For example, Telephone numbers are considered Alphanumeric because they can contain leading zeros, which would be lost if stored as a numeric type.[2]
- **Record (or Row):** A collection of all data relating to a single item or entity. Each row in the Customers table represents a single customer.[2]
- **Primary Key (PK):** A field (or combination of fields) in a table that uniquely identifies each record. For example, CustomerID is the primary key in the Northwind Customers table.[2] It ensures that every record is unique and can be referenced by other tables.
- **Foreign Key (FK):** A field in one table that links to the primary key of another table. It establishes a relationship between the tables. For example, a CustomerID field in the

Orders table would be a foreign key that links to the CustomerID in the Customers table, showing which customer placed the order. These keys are the glue that holds the relational model together.

2. SQL: The Language of Data Exploration

SQL (Structured Query Language) is the standard language for managing and manipulating relational databases. As a data engineer, your first interaction with a new data source is often through SQL to perform Exploratory Data Analysis (EDA). EDA is the process of examining a dataset to summarize its main characteristics, often with visual methods, to better understand its structure, content, and potential quality issues before it is used for analysis.[2]

The OmniCorp project requires you to perform an EDA on the Chinook and Northwind databases as your first task.[3] We will use SQL to do this efficiently.

2.1 Step 1: Discovering the Schema and Data Types

Before you can query data, you need to know what tables and columns are available. The schemas for Chinook and Northwind are quite different.[3]

- **Finding Tables:** The first step is to list all tables in the database. A common SQL query for this is SELECT name FROM sqlite_master WHERE type='table';.
  - Running this on the Chinook database reveals tables like Album, Artist, Customer, Employee, Invoice, and Track.[2]
  - Running it on the Northwind database reveals Customers, Employees, Orders, Products, and Suppliers.[2]
  - This initial discovery is crucial for identifying the common entities we'll need to unify for the project, such as Customers, Employees, and Products.[3]
- **Understanding Data Types:** In SQLite, you can use the PRAGMA table_info() statement to get information about the columns in a table, including their data type. For example, to check the schema of the Customer table in the Chinook database, you would run:

```
#SQL
PRAGMA table_info(Customer);
```

This command returns a result set with details about each column, including its name, data type (TEXT, INTEGER, etc.), and whether it can be null.[2] This is a critical step in the OmniCorp project, as you must justify how you will handle different data types between the two source

databases, for instance, the

CustomerID in Chinook (INTEGER) versus Northwind (TEXT).[3]

2.2 Step 2: Initial Data Profiling with SQL Queries

Once you know the tables and their schemas, you can run queries to understand the data within them. This process is a form of data profiling.

- **Counting Records:** One of the simplest yet most informative queries is to count the number of records in a table to understand its size.

```
#SQL
SELECT COUNT(*) FROM Customers;
```

- **Finding Unique Values:** To understand the diversity of data in a column, you can find the unique values. This is particularly useful for categorical data. For example, to find all unique countries represented in the Customer table, you would use:

```
#SQL
SELECT DISTINCT Country FROM Customer;
```

- **Aggregating Data:** SQL is powerful for aggregation. We can find out how many customers are from each country, as shown in the course materials.[2] This query provides a summary that can reveal important business insights, such as your top customer markets.

```
#SQL
SELECT
   Country,
   COUNT(*) AS NumberOfCustomers
FROM
   Customer
GROUP BY
```

```
    Country
ORDER BY
    NumberOfCustomers DESC;
```

- **Filtering Data:** You can use a WHERE clause to filter data based on specific conditions. This is essential for identifying potential data quality issues, such as missing values.

```
#SQL

SELECT * FROM Customers WHERE CompanyName IS NULL;
```

- **Joining Tables:** While we want to extract raw, un-transformed data for the Landing Zone, a quick join can help us understand relationships between tables. This query shows how we can link InvoiceLine items to Track details.

```
#SQL

SELECT
    T.Name AS TrackName,
    I.Quantity,
    I.UnitPrice
FROM
    InvoiceLine AS I
JOIN
    Track AS T ON I.TrackId = T.TrackId
LIMIT 5;
```

This is a small example of how SQL allows you to connect data from different tables, a skill that is foundational for the entire OmniCorp project.[3]

3. Data Ingestion: Moving Data to the Landing Zone

Once you've used SQL for exploration, the next step is to programmatically extract the data and save it to the Landing Zone of our data lake. This process is known as Data Ingestion. The Landing Zone is the destination for raw, unmodified data.[2] The objective is to capture the data as-is, preserving its original structure, data types, and integrity.[4] This allows us to re-process the data later if needed, a crucial principle of a modern data platform.[4]

The course syllabus highlights that you will learn to ingest raw SQL data into the Landing Zone using Python and the SQLAlchemy library.[2]

3.1 Python and SQLAlchemy for Database Connectivity

Python is the go-to language for data engineering due to its rich ecosystem of libraries. **SQLAlchemy** is a powerful Python library that provides a consistent way to interact with many different types of databases, from SQLite to PostgreSQL and MySQL.[2] We will also use the pandas library, which is a standard for data manipulation in Python and has built-in functions to read data from a database directly into a DataFrame.

3.2 Step-by-Step Data Ingestion with Code

Here is a full, commented script that demonstrates the process of ingesting data from the Chinook and Northwind databases into our Landing Zone directory.

```
#Python

# 1. Import necessary libraries
import pandas as pd
from sqlalchemy import create_engine
import os
import requests

# 2. Define the source database URLs and the tables to ingest
# The course materials provide the direct URLs to the public SQLite databases on GitHub
CHINOOK_URL =
"https://raw.githubusercontent.com/lerocha/chinook-database/master/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite"
NORTHWIND_URL = "https://github.com/jpwhite3/northwind-SQLite3/raw/main/dist/northwind.db"

# The tables we need for the OmniCorp project
```

```
# We will focus on the main transactional and dimension tables
chinook_tables =
northwind_tables =

# 3. Create a function to ingest a single table
def ingest_table(db_url, table_name, db_name):
    """
    Ingests data from a specified SQL table into the Landing Zone.
    This function is designed to handle SQLite databases from a URL.
    Args:
        db_url (str): The URL of the database.
        table_name (str): The name of the table to ingest.
        db_name (str): A name for the database (e.g., 'chinook', 'northwind') for directory creation.
    """
    print(f"--- Ingesting table: {table_name} from {db_name} ---")

    # Define the local path where the SQLite file will be temporarily stored
    local_db_path = f"{db_name}.db"

    # Step 3.1: Download the database file from the URL
    # We use a simple HTTP request to get the file content
    try:
        response = requests.get(db_url)
        response.raise_for_status() # Raise an HTTPError for bad responses (4xx or 5xx)

        with open(local_db_path, 'wb') as f:
            f.write(response.content)

        print(f"Successfully downloaded database to {local_db_path}")

    except requests.exceptions.RequestException as e:
        print(f"Error downloading database from {db_url}: {e}")
        return

    # Step 3.2: Connect to the downloaded database
    # SQLAlchemy's create_engine function sets up the connection
    engine = create_engine(f"sqlite:///{local_db_path}")

    # Step 3.3: Formulate the extraction query
    # We select ALL columns and ALL rows to get the raw, un-transformed data
    # This adheres to the Landing Zone principle of storing data as-is
    sql_query = f"SELECT * FROM \"{table_name}\""
```

```
    try:
        # Use pandas to read the SQL query results directly into a DataFrame
        # This is a highly efficient way to pull data from a database
        df = pd.read_sql(sql_query, engine)

        # Step 3.4: Define the output directory path for the Landing Zone
        # We use a structured path: data/landing_zone/raw/<database_name>/<table_name>.parquet
        output_dir = f"data/landing_zone/raw/{db_name}"

        # Create the directory if it doesn't exist
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)
            print(f"Created directory: {output_dir}")

        # Define the output file path. We'll use the Parquet format.
        output_path = os.path.join(output_dir, f"{table_name}.parquet")

        # Step 3.5: Save the DataFrame to a Parquet file
        # The 'index=False' prevents pandas from writing the DataFrame index to the file
        df.to_parquet(output_path, index=False)

        print(f"Successfully ingested {len(df)} records into: {output_path}")
        print("--------------------------------------------------\n")

    except Exception as e:
        print(f"Error ingesting table {table_name}: {e}")
    finally:
        # Clean up the local database file after ingestion
        if os.path.exists(local_db_path):
            os.remove(local_db_path)
            print(f"Removed temporary file: {local_db_path}")

# 4. Main script to call the ingestion function for each table
if __name__ == "__main__":
    print("Starting data ingestion for Chinook database...")
    for table in chinook_tables:
        ingest_table(CHINOOK_URL, table, "chinook")

    print("\nStarting data ingestion for Northwind database...")
    for table in northwind_tables:
        ingest_table(NORTHWIND_URL, table, "northwind")
```

3.3 Explaining the Ingestion Process and File Format

The script above is a simplified representation of a data pipeline. It demonstrates the ETL process in its most basic form: **Extract** (from the SQL database) and **Load** (to the Landing Zone). The Transform step will come later in our course.

- **The Role of Python Libraries:** The requests library is used to get the database file from a web server. This simulates ingesting data from an external source. The sqlalchemy library provides the create_engine function, which acts as a powerful abstraction layer, allowing us to connect to different database types (e.g., SQLite, PostgreSQL, MySQL) using a consistent API. Finally, pandas provides a high-level, intuitive way to read the data from the database into a DataFrame and then write it to a file.[2]
- **The SELECT * Principle:** The use of SELECT * is a deliberate choice that strictly adheres to the principle of a Landing Zone—store data as-is, with no transformations.[4] This ensures that we have a permanent, raw copy of the data from the source, which can be re-processed in the future if new requirements or data quality issues are discovered.[4]
- **Why Parquet?** We chose to save the data to the Parquet format. Parquet is not just a file; it is a columnar storage format.[1] This means that instead of storing data row-by-row, it stores data by column. This is a significant advantage for analytical queries because most analytical tools only need to read a few columns at a time. By only reading the columns they need, Parquet files dramatically reduce I/O (input/output) operations and improve query performance. Parquet also preserves the data types and schema, making it a reliable format for a
  Landing Zone where data integrity is paramount.

4. The OmniCorp Project: Applying the Principles

This week's lesson is the first practical step in our semester-long project. Your task is to unify data from the Chinook and Northwind databases for OmniCorp.[3] This week, you will specifically focus on the

Data Understanding and Source-to-Target Mapping portions of Task 1, which requires you to perform an EDA on the two schemas.[3]

- **The Landing Zone is where it all begins.** You will write Python scripts to connect to each database, extract the raw data from tables like Customers and Customer, and save them to separate files in your project's Landing Zone directory.[3]
- **No transformations yet.** The data from Northwind's Customers table will be saved as one raw file, and the data from Chinook's Customer table will be saved as another raw file. At this stage, we do not attempt to merge them or change their formats. This is a critical distinction that adheres to the Landing Zone principle of immutability and schema-on-read.[4] The transformation and unification of these tables will occur later in

the course, in the Staging Zone and Integration Zone.[1]

By the end of this week, you will have successfully ingested your first set of data for the project. In the following weeks, we will learn how to read this raw data from the Landing Zone and move it to the Staging Zone to begin the cleansing and transformation process.

5. Key Theories and Keywords

To conclude, here is a summary of the core concepts we covered this week.

- **Relational Database:** A database system that organizes data into a set of tables with predefined relationships between them, using a structured approach to ensure data integrity and minimize redundancy.[2] This is the foundation of most Online Transaction Processing (OLTP) systems.
- **Flat-File Database:** A simple database that stores all data in a single, un-related table, often leading to data redundancy and inconsistencies.[2] This model is generally not scalable for complex business operations.
- **SQL (Structured Query Language):** The standard programming language used for managing, retrieving, and manipulating data in a relational database.[2] It is a declarative language, meaning you describe
  what you want to retrieve, not *how* to retrieve it.
- **Exploratory Data Analysis (EDA):** The process of using statistical and visual methods to understand the main characteristics of a dataset, identify patterns, spot anomalies, and prepare for further analysis.[2] It is a crucial first step for any data project.
- **Data Ingestion:** The process of collecting, importing, and transferring data from various sources into a storage system, such as a data lake, for subsequent processing and analysis.[2] This is the initial step of the ETL or ELT data pipeline.
- **Landing Zone:** The first layer of a data lake, which serves as the raw, immutable destination for data as it is ingested from its source systems.[4] The data here is stored in its native format and is not transformed, with the exception of stripping out sensitive personally identifiable information (PII).[6]
- **SQLAlchemy:** A popular Python library that provides a flexible and powerful way to connect to and interact with a wide range of relational databases.[2] It allows data engineers to write code that is independent of the specific database system being used.
- **Primary Key (PK):** A unique identifier for a record in a table (e.g., CustomerID).[2] Its purpose is to uniquely identify a record and ensure data integrity.
- **Foreign Key (FK):** A field in a table that links to the primary key of another table, establishing a relationship (e.g., the CustomerID in an Orders table).[2] Foreign keys are essential for relational database design.
- **Data Types:** The classification of the kind of data a field can hold, such as Alphanumeric/text, Number (real or integer), Date, and Boolean.[2] Correctly identifying

and handling these types is a fundamental data engineering task, especially when integrating data from different sources.

- **Parquet:** A columnar storage file format optimized for big data analytics. It stores data by column rather than by row, which significantly improves query performance by allowing analytical engines to read only the data they need.[1] It is also highly efficient in terms of compression and storage.

Works cited

1. dsi310_แบบฟอร์มเค้าโครงการบรรยาย 2025
2. Real-Time Analytics Use Cases and Examples - Striim, accessed September 10, 2025, https://www.striim.com/blog/real-time-analytics-use-cases-and-examples/
3. dsi310: Data Lake Modeling
4. ETL vs ELT: Key Differences, Comparisons, & Use Cases - Rivery, accessed September 10, 2025, https://rivery.io/blog/etl-vs-elt/
5. ETL vs ELT - Difference Between Data-Processing Approaches - AWS, accessed September 10, 2025, https://aws.amazon.com/compare/the-difference-between-etl-and-elt/
6. Data lake best practices | Databricks, accessed September 10, 2025, https://www.databricks.com/discover/data-lakes/best-practices