BWT- Data Science Task 10

Bisma Fajar

July 2024

1 Relational Database

A relational database is a type of database that organizes data into rows and columns, which collectively form a table where the data points are related to each other. Tables are fundamental structures in relational databases used to organize and store data in rows and columns.

1.1 The Shortcomings of a Single Table Approach

The limitations of using a single table for storing data:

- Data Redundancy: Duplication of data leads to inefficiency and increased storage requirements.
- Lack of Flexibility: Difficulty in accommodating changes or adding new data attributes.
- Complexity in Data Management: Challenges in maintaining data integrity and consistency.

1.2 The Concepts of Relationships

Explain the concept of relationships in relational databases:

- **Cities:** Use an example dataset related to cities to illustrate relationships.
- **Rainfall:** Extend the example to include another dataset on rainfall, showing how data can be related between tables.

1.3 Retrieving the Data

The methods for retrieving specific data from tables using SQL queries or other query languages.

1.4 Joining Data

The process of joining data from multiple tables to retrieve meaningful insights:

- **Basic Joins:** Introduce different types of joins (e.g., inner join, outer join) to combine data from related tables.
- **Advanced Techniques:** Discuss advanced techniques like subqueries and self-joins.

1.5 Summary

In summary, understanding relational databases begins with grasping the foundational role of tables in organizing data efficiently. While a single table approach may seem straightforward, its limitations in handling complex data relationships, managing data redundancy, and adapting to evolving data needs highlight the necessity of relational models. By conceptualizing relationships between datasets—illustrated through examples like cities and rainfall—users can leverage SQL queries and join operations to extract relevant insights effectively. Mastering these fundamentals equips individuals with powerful tools for data retrieval and analysis in diverse applications.

2 Non Relational Database

There are different methods to store data other than tables or relations.

2.1 Spreadsheet

Spreadsheets are a popular way to store and explore data because it requires less work to setup and get started. For example Microsoft Excel sheets. A spreadsheet is a file and will be accessible in the file system of a computer, device, or cloud based file system. The software itself may be browser based or an application that must be installed on a computer or downloaded as an app. In Excel these files are also defined as workbooks and this terminology will be used the remainder of this lesson.

A workbook contains one or more worksheets, where each worksheet are labeled by tabs. Within a worksheet are rectangles called cells, which will contain the actual data. A cell is the intersection of a row and column, where the columns are labeled with alphabetical characters and rows labeled numerically. Some spreadsheets will contain headers in the first few rows to describe the data in a cell.

2.2 Managing Spreadsheets

We can manage spreadsheets by using different formulas and built in techniques. We can easily remove or highlight duplicates and other things we want to search or replace.

3 NoSql

NoSQL is an umbrella term for the different ways to store non-relational data and can be interpreted as "non-SQL", "non-relational" or "not only SQL".

3.1 Types of NoSQL Databases

3.1.1 Document Stores

- Store data in flexible, schema-less documents (e.g., JSON or BSON format).
- Examples: MongoDB, Couchbase, Elasticsearch.

3.2 Key-Value Stores

- Simplest NoSQL model; data stored as key-value pairs.
- Examples: Redis, DynamoDB, Riak.

3.3 Column Family Stores

- Store data in columns rather than rows; designed for large-scale data storage.
- Examples: Apache Cassandra, HBase.

3.4 Graph Databases

- Represent data in terms of nodes and edges (entities and relationships).
- Examples: Neo4j, ArangoDB, OrientDB.

4 Document data stores with azure cosmos DB

Document data stores build on the concept of a key-value data store and is made up of a series of fields and objects.A Cosmos DB database fits the definition of "Not Only SQL", where Cosmos DB's document database relies on SQL to query the data. A document is a collection of fields and object values, where the fields describe what the object value represents. Below is an example of a document.

{

"firstname": "Eva",
"age": 44,
"id": "8c74a315-aebf-4a16-bb38-2430a9896ce5",
"_rid": "bHwDAPQz8s0BAAAAAAAAAA==",
"_self": "dbs/bHwDAA==/colls/bHwDAPQz8s0=/docs/bHwDAPQz8s0BAAAAAAAAAA==/",

”_etag”: ”0̈0000000-0000-0000-9f95-010a691e01d7̈”,
”_attachments”: ”attachments/”,
”_ts”: 1630544034

}

4.1 Exploring Data with the Cosmos DB Emulator

To explore data using the Cosmos DB Emulator, follow these steps:

1. **Download and Install the Emulator:**
   - Download the Cosmos DB Emulator for Windows.
   - Follow the installation instructions for macOS/Linux.
2. **Launch the Emulator:**
   - Once installed, launch the Cosmos DB Emulator on your computer.
   - It will open a browser window with the Emulator’s Explorer view.
3. **Start with Sample Data:**
   - In the Emulator’s Explorer view, click on ”Start with Sample” to create a sample database (e.g., SampleDB).
4. **Explore Containers and Documents:**
   - Expand the created database (e.g., SampleDB) to view containers (e.g., Persons).
   - Click on a container to explore its documents (items).
   - Navigate through individual documents to inspect their structure and contents.
5. **Run Queries:**
   - Click on the SQL Query button in the Emulator’s interface.
   - Write and execute SQL-like queries to retrieve specific data from containers.

5 Python for Data Processing

While databases excel at storing and querying data, custom programming in Python offers unparalleled flexibility, especially for complex data tasks. Data scientists often prefer Python due to its simplicity and extensive libraries.

5.1 Key Libraries

- **Pandas**: Manipulates DataFrames (like tables) with ease, supporting operations across rows and columns.
- **Numpy**: Handles multi-dimensional arrays efficiently, offering extensive mathematical functions.
- **Matplotlib**: Provides tools for data visualization and plotting graphs.
- **SciPy**: Includes additional scientific functions, particularly useful for statistics and optimization.

5.2 Example Setup

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import ...
```

5.3 Pandas Concepts

- **Series**: Similar to lists or arrays but with an index, supports operations considering the index.
- **DataFrame**: A collection of Series with the same index, combining multiple Series into a table-like structure.

5.4 Example Usage

```
# Example code for working with Pandas
start_date = "Jan 1, 2020"
end_date = "Mar 31, 2020"
idx = pd.date_range(start_date, end_date)
items_sold = pd.Series(np.random.randint(25, 50, size=len(idx)), index=idx)

additional_items = pd.Series(10, index=pd.date_range(start_date, end_date, freq="W"))
total_items = items_sold.add(additional_items, fill_value=0)

monthly = total_items.resample("1M").mean()
```

5.5 DataFrames Operations

- **Column Selection**: `df['column']` or `df[['column1', 'column2']]`.
- **Filtering**: `df[df['column'] > value]`.
- **Creating New Columns**: `df['new_column']=...`
- **Grouping**: `df.groupby('column').mean()` or with custom aggregations.

5.6 Getting Data

- Load data from files (e.g., CSV) using `pd.read_csv('file.csv')`.
- Data exploration with`df.head()` for previewing data.

5.7 Conclusion

Python, especially with Pandas, Numpy, Matplotlib, and SciPy, provides powerful tools for data manipulation and analysis. Its versatility and extensive community support make it a preferred choice in data science. Raw data often contains inconsistencies, making it challenging for analysis and modeling. This "dirty" data requires cleaning and transformation to handle missing, inaccurate, or incomplete data effectively. This section focuses on techniques using Python and the Pandas library, demonstrated in the accompanying notebook.

6 Importance of Data Cleaning

Clean data enhances ease of use, consistency across datasets, and improves model accuracy. Proper organization and normalization facilitate easier searching, usage, and sharing.

6.1 Common Cleaning Goals and Strategies

6.1.1 Exploring a Dataset

Data exploration helps identify and resolve issues such as missing or inconsistent data through basic querying, visualizations, and sampling.

6.1.2 Formatting

Inconsistent data presentation can hinder analysis and visualization. Standardizing formats, like dates and data types, ensures data integrity across different sources.

6.1.3 Handling Duplications

Duplicate data can skew results and should typically be removed unless they provide unique insights when merging datasets.

6.1.4 Dealing with Missing Data

Missing data leads to inaccuracies and biased results. Techniques like filling values or removing entries help mitigate these issues, depending on the context.

6.1.5 Exploring DataFrame Information

The `info()`, `head()`, and `tail()` methods in Pandas provide quick insights into DataFrame structure, size, and content.

6.1.6 Dealing with Missing Data

Detecting and handling null values using Pandas methods like `isnull()`, `notnull()`, `dropna()`, and `fillna()` improves data completeness and accuracy.

6.1.7 Removing Duplicate Data

Identifying and eliminating duplicate entries using `duplicated()` and `drop_duplicates()` ensures data integrity and improves analysis results.

6.2 Conclusion

Clean data is crucial for effective data analysis and modeling. By employing these techniques in Python with Pandas, data scientists can ensure data quality and enhance the reliability of their insights.