Handling Data from Multiple Sources
(The challenges of integrating data from multiple sources)

Group Members:

1. Thato Ndlovu
2. Teddy Motswiri
3. Bongane Ntombela
4. Raphaahle Concilia Mogano
5. Matome Chris Ramatsoma

Overview

Integrating data from many sources poses numerous issues, such as discrepancies in data formats, schemas, quality, governance, and security, which businesses must address to establish a cohesive and functional data ecosystem.

Challenges of Integrating Data from Multiple Sources

- Integrating data from multiple sources involves various challenges:

- **Different Formats**: A variety of formats, including CSV, JSON, XML, and databases, may be used to store data. For processing, each format might need a different set of tools or libraries.
- **Schema Variations**: Schemas (such as column names, data types, or structures) may differ between sources. Data field alignment may become challenging as a result.
- **Data Quality Issues**: Integration and analysis may be impacted by missing, inconsistent, or redundant data.
- **Conflicting Data**: Techniques for reconciliation are necessary because different sources may provide contradictory values for the same data.
- **Encoding and Localization**: Merging can be made more difficult by differences in localization (such as date formats or currency symbols) or encoding (such as UTF-8 vs. ISO-8859-1).
- **Volume and Performance**: Efficient processing methods may be necessary for large datasets in order to prevent performance bottlenecks.

Python Example: Merging a CSV File with a JSON File

The script uses sample data in both the **CSV** and **JSON** files to simulate a dataset for merging. Here's a detailed breakdown of the sample data:

CSV File Data

The CSV file (`data.csv`) contains information about devices, organized into four columns:

- **id**: A unique identifier for each row (1 to 5).

- **computer_id**: A numerical identifier for each device.
- **device_name**: The name of the device, following the pattern "CAPACITI-JHB" with varying numbers.
- **session_id**: A string representing a session ID associated with the device.

Example Data:

id	computer_id	device_name	session_id
1	175437	CAPACITI-JHB 2293	abc123xyz
2	2238623	CAPACITI-JHB 2263	ghi589xyz
3	3468732	CAPACITI-JHB 2223	lmn987xyz
4	4432489	CAPACITI-JHB 2278	def435xyz
5	5983724	CAPACITI-JHB 2209	opq461xyz

JSON File Data

The JSON file (`data.json`) contains department-related data with two fields:

1. **department_id**: A unique identifier for each department (101 to 105).
2. **department_name**: The name of the department, which is consistently "Data Engineering" in this example.

Example Data:

```
[
 {"department_id": 101, "department_name": "Data Engineering"},
 {"department_id": 102, "department_name": "Data Engineering"},
 {"department_id": 103, "department_name": "Data Engineering"},
 {"department_id": 104, "department_name": "Data Engineering"},
 {"department_id": 105, "department_name": "Data Engineering"}
]
```

Purpose of the Sample Data

- **CSV Data**: Represents a collection of device records that could be found in a more extensive IT inventory system.
- **JSON Data**: Serves as an organization's departmental information representation.

These datasets were chosen to demonstrate merging or concatenating data with mismatched attributes but the same number of rows.

Python Code:

```
import pandas as pd

# Create the main CSV file
data = pd.DataFrame({
  'id': [1, 2, 3, 4, 5],
  'computer_id': [175437, 2238623, 3468732, 4432489, 5983724],
  'device_name': [
        'CAPACITI-JHB 2293',
        'CAPACITI-JHB 2263',
        'CAPACITI-JHB 2223',
        'CAPACITI-JHB 2278',
        'CAPACITI-JHB 2209'
        ],
  'session_id': [
        'abc123xyz', 'ghi589xyz', 'lmn987xyz', 'def435xyz', 'opq461xyz'
        ]
})
data.to_csv('data.csv', index=False)

# Create the JSON file
json_data = [
        {"department_id": 101, "department_name": "Data Engineering"},
        {"department_id": 102, "department_name": "Data Engineering"},
        {"department_id": 103, "department_name": "Data Engineering"},
        {"department_id": 104, "department_name": "Data Engineering"},
        {"department_id": 105, "department_name": "Data Engineering"}
]
```

```
pd.DataFrame(json_data).to_json('data.json', orient='records', lines=True)

# Read the main files
csv_data = pd.read_csv('data.csv')
json_data = pd.read_json('data.json', lines=True)

# Merge all data
merged_data = pd.concat([csv_data, json_data], axis=1)
# Resolve conflicts and ensure data consistency
merged_data.fillna({
        'device_name': 'Unknown',
        'session_id': 'Unknown',
        'department_name': 'Unknown'

}, inplace=True)



# Drop redundant columns

if 'name_y' in merged_data.columns:

merged_data.drop(columns=['name_y'], inplace=True)



# Save the merged data to a new CSV file

merged_data.to_csv('merged_data.csv', index=False)



# Display the merged data on the terminal

print("Merged Data:")

print(merged_data)
```

Key Observations

- Despite their comparable sequential ordering, the department_id in the JSON and the id in the CSV are not specifically used to merge the data.
- There is no logical connection between the two datasets, but concatenation makes use of this sequential alignment.

Output of the code:
The code combines the sample data from the CSV and JSON files into a single dataset. The merged data is presented row-wise, combining columns from both files. Here's what the output would look like based on the provided script:

Merged Output

id	computer_id	device_name	session_id	department_id	department_name
1	175437	CAPACITI-JHB 2293	abc123xyz	101	Data Engineering
2	2238623	CAPACITI-JHB 2263	ghi589xyz	102	Data Engineering
3	3468732	CAPACITI-JHB 2223	lmn987xyz	103	Data Engineering
4	4432489	CAPACITI-JHB 2278	def435xyz	104	Data Engineering
5	5983724	CAPACITI-JHB 2209	opq461xyz	105	Data Engineering

Key Features of the Output:

- **Column Alignment**: Each row combines data
  - CSV columns: id, computer_id, device_name, and session_id.
  - JSON columns: department_id and department_name.
- **Conflict Resolution**:
  - If any values in specific columns (device_name, session_id, or department_name) were missing, they would be filled with "Unknown". However, in the provided sample data, no values are missing.
- **No Explicit Keys Used**:
  - Merging relies purely on the order of rows in the two files, not on a common key.

- **Saved Output**:
    - The merged dataset is saved as merged_data.csv for further use.
- **Display**:
    - The script prints the merged dataset to the terminal as shown above.

Techniques for resolving conflicts and ensuring data consistency

- **Handling Missing Values** - Methods for addressing dataset gaps that guarantee completeness and avoid problems brought on by incomplete datasets during processing or analysis include using statistical imputation, flagging records for additional review, or filling missing data with default values.
- **Resolving Conflicting Data** - The procedure that ensures the most accurate and trustworthy representation of the data when the same data attribute varies between sources.
- **Deduplication** - Finding and eliminating duplicate records from datasets preserves data integrity, cuts down on redundancy, and enhances data clarity and storage efficiency.
- **Data Transformation** - As part of Extract, Transform, Load (ETL) procedures, which guarantee data compatibility across systems and facilitate smooth integration and analysis, data is frequently converted into a consistent structure or format.
- **Conflict Resolutions via Weighting** - To resolve conflicting information, different data sources are given weights or priorities based on their reliability, recentness, or relevance. This creates a methodical approach to conflict resolution and guarantees that data from more dependable sources is preferred.
- **Validation Rules** - Establishing and implementing guidelines to guarantee data consistency and accuracy, such as cross-field dependencies, format checks, and range validations, which keep inaccurate data out of the system and guarantee alignment with business needs.
- **Consistent Key Naming** - Naming keys, fields, or attributes consistently across datasets improves data integration, prevents misunderstandings, and lowers errors brought on by unclear or inconsistent naming.
- **Auditing and Logging** - Monitoring and documenting data changes, such as updates, merges, and deletions, as well as metadata, such as timestamps and user actions, which promotes accountability in data handling procedures, facilitates error tracing, and offers transparency.