There are many different potential data storage methods – each with different benefits and drawbacks. In this practical, I implemented CSV, JSON, and a relational database for various purposes. Here, I will go over the reasoning for these choices as well as the comparisons between the options I didn't use.

To store the raw data retrieved from online sources, I used CSV files as it's very easy to add lots of and parse the data from online to the required format. Unlike XML and JSON, tags don't need to be added to the data, which slows down the data collection process.

However, one of the draw-backs of using CSV, rather than JSON is it's lower flexibility. The ordering of the columns in the CSV file needs to be consistent for each row, which means that some errors may occur through typos. XML and JSON have the tags which allow the attributes to be placed in any order. However, as this was just the data-collection step, the main importance is speed, and the person gathering the data can just make sure to add data with the correct columns, and then as the PopulateDB class is run, those errors can be detected and skipped, with a warning displayed. This means that any errors can be easily debugged and fixed quickly.

One feature of CSV is both a benefit and draw-back compared to JSON and XML. This is in the way the data is parsed. As the name suggests, the values are separated by a delimiter (usually a comma). This means that the lines can be split by the delimiter which is quicker than parsing the other types of data. A problem arises as the delimiter cannot appear in the values themselves without causing issues, as it's hard to determine what is a delimiter and what is part of a value. This doesn't occur with parsing JSON and XML due to the libraries for parsing them which easily deal with these issues. Also, the < and > characters which may confuse XML are a lot less commonly used in data than commas. However, with CSV files, the delimiter can be changed so this issue doesn't occur as frequently.

There are fewer differences between JSON and XML than there are between either and CSV. They are both flexible, and have less redundant data than CSV as the columns which would be empty in CSV can just be omitted with JSON and XML.

Finally, there are relational databases which abstract away the data-storage to a robust, tested software layer. We don't need to worry about certain characters in our data causing as many issues, except for the key terms, such as null. Even here, the input values can be sanitised to ensure that when "null" is entered as a string, it is forced to not count as a null-value.

A key benefit of relational databases over the other types of data storage is the relations themselves. We can see how data is connected much more easily, and we can use queries to search between the relations, instead of having to go through each file individually and finding the relations by hand. Databases also can provide primary-keys which prevent duplicate results as well as automatic id generation. This prevents ambiguity with what certain pieces of data relate to which we may find with semi-structured data. This also means that adding data to the database is more robust as there are more validity checks using the schema which you don't get with other storage methods.

Databases also reduce the need for repeated data, as we can just add links to other data where needed, which is more similar to the transient data structures we use when programming and so can be more easily integrated in programs.

A draw-back of relation databases is the time it takes to design and implement a database. In this practical, a ER-Diagram needed to be created to work out what entities were needed, with soft-entities where appropriate, which adds complexity which may be harder to understand than just adding the data to a CSV, JSON or XML file. This means that for prototyping, where the requirements may change often, a database is less practical as constant changing of the structure can cause problems with transferral.