**Scalable Web Services**

A Web service is a software service used to communicate between two devices on a network. More specifically, a Web service is a software application with a standardized way of providing interoperability between disparate applications.

Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web based.

The roadmap followed for this study is as follows:

* 1. **Picking up a language.**

Python was the chosen scripting language.

* 1. **Learn the corresponding package manager.**

Package managers help you bring external dependencies in your application and to distribute your own packages.

Python has pip.

* 1. **Explore testing (writing and optimizing test cases).**
  2. **Learn and refresh relational databases.**

A relational database is a set of formally described tables from which data can be accessed or reassembled in many ways without having to reorganize the database tables. The standard user and application programming interface (API) of a relational database is the Structured Query Language (SQL).

A relational database allows you to easily find specific information. It also allows you to sort based on any field and generate reports that contain only certain fields from each record. Relational databases use tables to store information. The standard fields and records are represented as columns (fields) and rows (records) in a table.

With a relational database, you can quickly compare information because of the arrangement of data in columns. The relational database model takes advantage of this uniformity to build completely new tables out of required information from existing tables. In other words, it uses the relationship of similar data to increase the speed and versatility of the database.

The "relational" part of the name comes into play because of mathematical relations. A typical relational database has anywhere from 10 to more than 1,000 tables. Each table contains a column or columns that other tables can key on to gather information from that table.

* 1. **Build a practical application.**

For this study, we build a book manager application. Similar to a library management subsystem, it allows you to add, delete, update, and view the books available in the subsystem. It was build using Python and Flask and SQLAlchemy.

SQLAlchemy is a library that facilitates the communication between Python programs and databases. Most of the times, this library is used as an Object Relational Mapper (ORM) tool that translates Python classes to tables on relational databases and automatically converts function calls to SQL statements.

* 1. **Learn a framework.**

A framework is a collection of programs that you can use to develop your own application. It is built on top of a programming language. Framework is a set of pre-written code libraries designed to be used by developers. Frameworks eliminate the need to write a lot of repetitive code that you will find being used in many different applications. The advantage of efficiency will never be underestimated. You can expect to build a project in much less time than would be achieved writing code without a framework.

In the longer term, a framework ensures the longevity of your applications. If a development team works as they please, only that particular team will be able to maintain and upgrade the application with ease. The way that a publisher supports a proprietary solution.

On the other hand, the structure that a framework provides for the application makes it possible to avoid this pitfall altogether and it gives any developer - whether they participated in its development or not – the ability to easily “adopt” an application, to maintain it over time and to upgrade it both quickly and neatly, whenever necessary.

The framework used in this study is Flask. Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

* 1. **Learn NoSQL databases.**

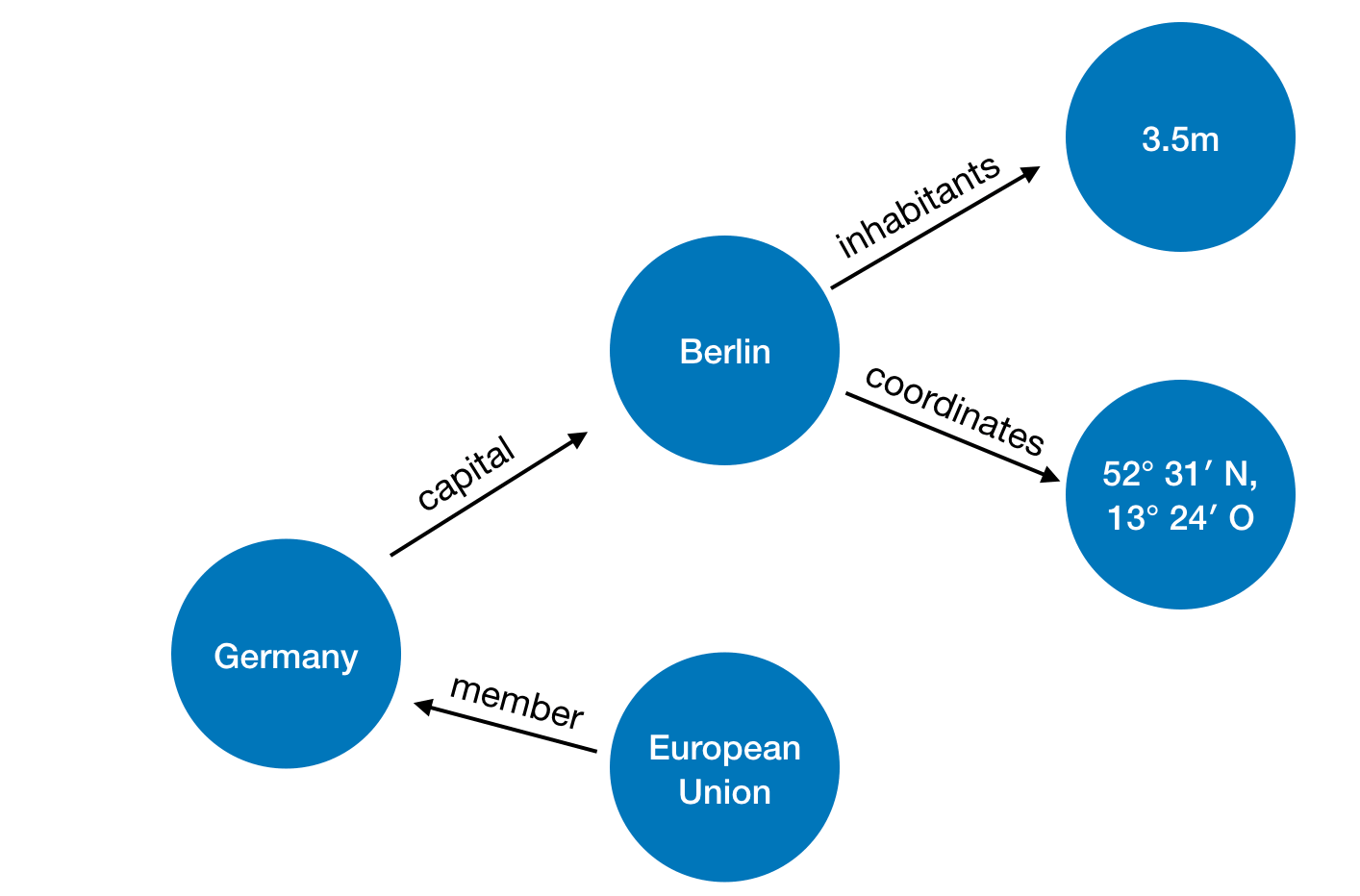
A NoSQL originally referring to non-SQL or non-relational is a database that provides a mechanism for storage and retrieval of data. This data is modeled in means other than the tabular relations used in relational databases.

Difference between SQL and NoSQL databases:

1. SQL databases are primarily called as Relational Databases (RDBMS); whereas NoSQL database are primarily called as non-relational or distributed database.
2. SQL databases are table-based databases whereas NoSQL databases are document based, key-value pairs, graph databases or wide-column stores. This means that SQL databases represent data in form of tables which consists of n number of rows of data whereas NoSQL databases are the collection of key-value pair, documents, graph databases or wide-column stores which do not have standard schema definitions which it needs to adhered to.
3. SQL databases have predefined schema whereas NoSQL databases have dynamic schema for unstructured data.
4. SQL databases are vertically scalable whereas the NoSQL databases are horizontally scalable. SQL databases are scaled by increasing the horsepower of the hardware. NoSQL databases are scaled by increasing the databases servers in the pool of resources to reduce the load.
5. SQL databases uses SQL (structured query language) for defining and manipulating the data, which is very powerful. In NoSQL database, queries are focused on collection of documents. Sometimes it is also called as UnQL (Unstructured Query Language). The syntax of using UnQL varies from database to database.
6. SQL database examples: MySql, Oracle, Sqlite, Postgres and MS-SQL. NoSQL database examples: MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb
7. For complex queries: SQL databases are good fit for the complex query intensive environment whereas NoSQL databases are not good fit for complex queries. On a high-level, NoSQL don’t have standard interfaces to perform complex queries, and the queries themselves in NoSQL are not as powerful as SQL query language.
8. For the type of data to be stored: SQL databases are not best fit for hierarchical data storage. But, NoSQL database fits better for the hierarchical data storage as it follows the key-value pair way of storing data like JSON data. NoSQL database are highly preferred for large data set (i.e. for big data). Hbase is an example for this purpose.
9. For scalability: In most typical situations, SQL databases are vertically scalable. You can manage increasing load by increasing the CPU, RAM, SSD, etc., on a single server. On the other hand, NoSQL databases are horizontally scalable. You can just add few more servers easily in your NoSQL database infrastructure to handle the large traffic.
10. For high transactional based application: SQL databases are best fit for heavy duty transactional type applications, as it is more stable and promises the atomicity as well as integrity of the data. While you can use NoSQL for transactions purpose, it is still not comparable and sable enough in high load and for complex transactional applications.
11. For support: Excellent support are available for all SQL database from their vendors. There are also lot of independent consultations who can help you with SQL database for a very large-scale deployment. For some NoSQL database you still have to rely on community support, and only limited outside experts are available for you to setup and deploy your large-scale NoSQL deployments.
12. For properties: SQL databases emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability) whereas the NoSQL database follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance)
13. For DB types: On a high-level, we can classify SQL databases as either open-source or close-sourced from commercial vendors. NoSQL databases can be classified on the basis of way of storing data as graph databases, key-value store databases, document store databases, column store database and XML databases.

For this study, we also used the Wikidata. Wikidata is a collaboratively edited knowledge base hosted by the Wikimedia Foundation. It is a common source of open data that Wikimedia projects such as Wikipedia can use, and anyone else, under a public domain license. Like Wikipedia, there are all kinds of data stored in Wikidata. As such, when you are looking for a specific dataset or if you want to answer a curious question, it can be a good start looking for that data at Wikidata first. To use Wikidata we need to know SQPARQL.

SPARQL is a query language for RDF databases. In contrast to relational databases like SQL, items are not part of any tables. Instead, items are linked with each other like a graph or network:



To describe these relations, we can use a triple. A triple is a statement containing a subject predicate and object.

Examples:

1. Germany (subject) has the capital (predicate) Berlin (object).
2. Berlin (subject) has the coordinates (predicate) 3.5million (object).

The graph above can be described by various statements. And that is a huge benefit of SparQL. It is not limited to a certain structure of relational databases and new information can be easily added.

For our application, let us say we would like to get information about all the books by a specific author.

We can write a query as follows:

**WHERE**

{

?book **wdt**:**P50** **wd**:**Q35610**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

A query that also includes the title(P1476), illustrator(P110), publisher(P123) and publication date(P577).

**SELECT** ?book ?title ?illustratorLabel ?publisherLabel ?published

**WHERE**

{

?book **wdt**:**P50** **wd**:**Q35610**.

**OPTIONAL** { ?book **wdt**:**P1476** ?title. }

**OPTIONAL** { ?book **wdt**:**P110** ?illustrator. }

**OPTIONAL** { ?book **wdt**:**P123** ?publisher. }

**OPTIONAL** { ?book **wdt**:**P577** ?published. }

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

A query for the number of pages of books by each publisher.

**SELECT** ?publisher ?publisherLabel (AVG(?pages) **AS** ?avgPages)

**WHERE**

{

?book **wdt**:**P123** ?publisher;

**wdt**:**P1104** ?pages.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

**GROUP BY** ?publisher ?publisherLabel

**ORDER BY** **DESC**(?avgPages)

A query that will return the preceding and following book titles. So, If I put in the book title 'Chamber of Secrets' I would get the books before and after that returned.

SELECT \*

WHERE {

dbr:Harry\_Potter\_and\_the\_Chamber\_of\_Secrets

dbp:precededBy ?precededBy;

dbp:followedBy ?followedBy.

}

**REFERENCES:**

* Pythonspot.com
* realpython.com
* Geeksforgeeks.com
* codementor.io
* Wikipedia
* github