CRUD Databases
(Create, Read, Update and Delete)
Part 3 - Traveler's Database using JSON

1   Introduction

A demonstration of **CRUD** database programming, based on using **JSON** (JavaScript Object Notation, ECMA-404) as the database format. CRUD stands for **create, read, update and delete**. These are the four basic operations every database must perform.

JSON is a language-independent text format for data interchange. JSON makes it possible to represent data in a portable format. It was derived from **JavaScript** (ECMA-262, 1999). Originally, data was transferred between web servers and web and mobile clients using the **XML** format over a protocol called **SOAP**. The **JSON** format and the **REST** (Representational State Transfer) protocol have largely replaced **XML/SOAP**.
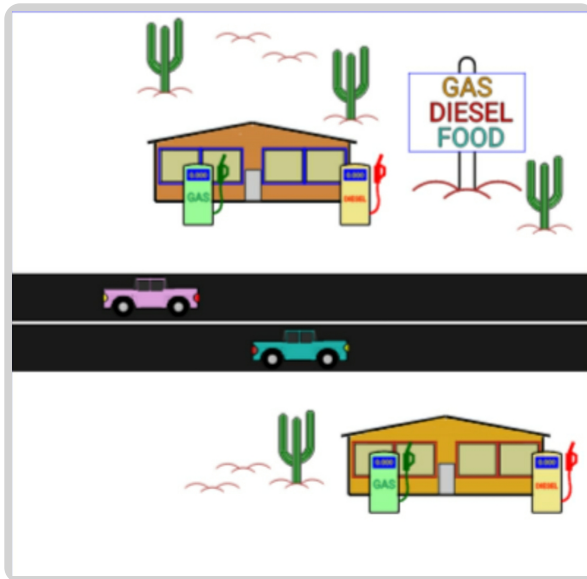

Figure 1:   Travel on a long road trip.

Scenario - You are driving through the desert on a long road trip. Gas stations are few and far between, so it's important to know where you can get fuel so you don't run out. You also need to know where you can find food and a refreshing beverage. Calculate the distance to these things relative to your location. The data is **dynamically displayed in a table**.

An algorithm was coded to simulate GPS location updates. When an update occurs, the app searches the database and calculates which database entries are the closest. The table is populated with the closest entries. **If an entry which has already been passed falls within the search window, it is prefixed with a minus sign and displayed in a different color to indicate that it is behind the current location.**

The database will store the following information:

- **name**
- **longtitude**
- **lattitude**
- **GAS** or **NO GAS**
- **DIESEL** or **NO DIESEL**
- **FOOD** or **NO FOOD**

At the beginning of the demo, and after every GPS update, the closest items in the database will be displayed. The data will be displayed in **table** format. These are actual screenshots of the data as it is displayed by the demo application.

The demo was written using **HTML, CSS, VanillaJS and SVG graphics**, and runs in your Web browser. It was tested under **Android**™ using the **Firefox** browser.

The demo is Copyright © 2020. All rights reserved.

2   Database

In order to create the database demo, we need to make some design decisions about how the database is structured.

Since this is a demo, we could make every field be a **string** (array of characters).
For example:

- ***email*** = 35 characters.
- ***first subscribed*** = 19 characters.
- ***expires*** = 19 characters.
- ***member*** or ***guest*** = 19 characters.
- ***renewal notice*** = 3 characters.

So the size of every record would be 95 characters.

There are a couple of ways to handle records based on arrays.

1) ***Fixed length fields***
The fields in every record are the same length. For example, the ***email*** field in every record would always have a length of 35 characters.

Figure 2:   Fixed length versus variable length format.

If the user enters a string that is shorter than the length of the field, we need to ***right pad*** the string (append spaces to the right side of the string) until the length of the string is equal to the length of the field.

One advantage of this is that it simplifies the software code necessary to read the records in the database. When we read the database records, ***every record will be the same length***. To access a particular record we simply calculate its location by multiplying the number of the record by the length of the record; ie, record 0 would start at character 0, record 1 would start at character 95. Record 2 would start at character 190, and so on. We know that we can read one record by reading 95 characters starting at a particular location.

Fixed length fields also simplify accessing the data within a record because we know where the field starts within the record and the length of the field.

2) ***Variable length fields***

Since we don't know how long the fields are, we don't have any idea as to the length of a record. It is necessary to ***parse*** (break up a sentence or group of words into separate components) the data to determine the field and record length. [22, 23] We can insert a character which we know the user will not type to mark the boundaries between the fields. This special character is called a ***delimiter***. We could also insert a different delimiter to mark the boundaries between records. This allows us to parse the records and the fields within the records.

3   Initial database state - Read

This is the database *table* view within the demo application. It shows the closest rest stop locations to our current location. This verifies that we are able to *read* the contents of the database.

You can see that the database is reflected in the demo application's *table* view. This verifies that we are able to *read* the contents of the database.

Figure 3:   Table view of the closest locations after GPS update.

4   Create

The database is *read-only*, so we aren't going to create any new records.

5   Delete

The database is *read-only*, so we aren't going to delete any records.

6   Update

The database is *read-only*, so we aren't going to update any records.

The demo application's *JSON* database looks like this:

[{"name" : "Petrol King #1", "longtitude" : "450", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "FOOD"}, {"name" : "Oil City #2", "longtitude" : "457", "lattitude" : "900", "gas" : "NO GAS", "diesel" : "NO DIESEL", "food" : "NO FOOD"}, {"name" : "Burger Stop #3", "longtitude" : "462", "lattitude" : "900", "gas" : "NO GAS", "diesel" : "NO DIESEL", "food" : "FOOD"}, {"name" : "Fill 'Er Up #4", "longtitude" : "462", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "NO FOOD"}, {"name" : "Taco Town #5", "longtitude" : "467", "lattitude" : "900", "gas" : "NO GAS", "diesel" : "NO DIESEL", "food" : "FOOD"}, {"name" : "Oil City #6", "longtitude" : "467", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "NO FOOD"}, {"name" : "Petrol King #7", "longtitude" : "475", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "FOOD"}, {"name" : "Fill 'Er Up #8", "longtitude" : "482", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "FOOD"}, {"name" : "Burger Stop #9", "longtitude" : "487", "lattitude" : "900", "gas" : "NO GAS", "diesel" : "NO DIESEL", "food" : "FOOD"}, {"name" : "Petrol King #10", "longtitde" : "487", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "FOOD"}, {"name" : "Oil City #11", "longtitude" : "492", "lattitude" : "900", "gas" : "NO GAS", "diesel" : "NO DIESEL", "food" : "NO FOOD"}, {"name" : "Burger Stop #12", "longtitude" : "492", "lattitude" : "900", "gas" : "NO GAS", "diesel" : "NO DIESEL", "food" : "FOOD"}, {"name" : "Fill 'Er Up #13", "longtitude" : "492", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "FOOD"}, {"name" : "Taco Town #14", "longtitude" : "502", "lattitude" : "900", "gas" : "NO GAS", "diesel" : "NO DIESEL", "food" : "FOOD"}, {"name" : "Oil City #15", "longtitude" : "502", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "FOOD"}, {"name" : "Petrol King #16", "longtitude" : "510", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "NO FOOD"}, {"name" : "Burger Stop #17", "longtitude" : "518", "lattitude" : "900", "gas" : "NO GAS", "diesel" : "NO DIESEL", "food" : "FOOD"}, {"name" : "Fill 'Er Up #18", "longtitude" : "518", "lattitude" : "900", "gas" : "GAS", "diesel" : "DIESEL", "food" : "NO FOOD"}]

7   GPS Updates

An algorithm was coded to simulate GPS location updates. When an update occurs, the app searches the database and calculates which database entries are the closest. The table is populated with the closest entries. **If an entry which has already been passed falls within the search window, it is prefixed with a minus sign and displayed in a different color to indicate that it is behind the current location.**

You can see that the update is now reflected in the demo application's *table* view.

Current Longtitude	Current Lattitude
470	900

Name	Distance	GAS	DIESEL	FOOD
Taco Town #5	-3	NO GAS	NO DIESEL	FOOD
Oil City #6	-3	GAS	DIESEL	NO FOOD
Petrol King #7	5	GAS	DIESEL	FOOD
Fill 'Er Up #8	12	GAS	DIESEL	FOOD
Burger Stop #9	17	NO GAS	NO DIESEL	FOOD
Petrol King #10	17	GAS	DIESEL	FOOD
Oil City #11	22	NO GAS	NO DIESEL	NO FOOD

Figure 4:   Table view of the closest locations after GPS update.

References

[1]   "HTML 5.2." *W3C*, 14-Dec-2017. [Online]. Available: https://www.w3.org/TR/html52/. [Accessed: 13-Jun-2021].

[2]   "Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification." *W3C*, 12-Apr-2016. [Online]. Available: https://www.w3.org/TR/CSS22/. [Accessed: 13-Jun-2021].

[3]   "Cascading Style Sheets, level 1." *W3C*, 11-Apr-2008. [Online]. Available: https://www.w3.org/TR/REC-CSS1/. [Accessed: 13-Jun-2021].

[4]   "CSS Snapshot 2020." *W3C*, 22-Dec-2020. [Online]. Available: https://www.w3.org/TR/CSS/. [Accessed: 13-Jun-2021].

[5]   "ECMA-262," *Ecma International*, Jun-2020. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/. [Accessed: 13-Jun-2021].

> ECMAScript® 2020 language specification
> 11th edition, June 2020
>
> **This is the specification for JavaScript.**

[6]   "ECMAScript Internationalization API Specification – ECMA-402 Edition 1.0." *Ecma International*, Dec-2012. [Online]. Available: https://402.ecma-international.org/1.0/. [Accessed: 13-Jun-2021].

[7]   "Scalable Vector Graphics (SVG) 2." *W3C*, 04-Oct-2018. [Online]. Available: https://www.w3.org/TR/SVG2/. [Accessed: 01-Jun-2021].

[8]   "Scalable Vector Graphics (SVG) 1.1 (Second Edition)." *W3C*, 16-Aug-2011. [Online]. Available: https://www.w3.org/TR/SVG11/. [Accessed: 01-Jun-2021].

[9]   Mozilla and individual contributors, "SVG: Scalable Vector Graphics | MDN." *MDN Web Docs*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/SVG. [Accessed: 06-Dec-2021].

[10]   J. Johnston, "What is a CRUD app and how to build one | Ultimate guide," *Budibase*, 06-Jul-2021. [Online]. Available: https://www.budibase.com/blog/crud-app/. [Accessed: 14-Sep-2021].

[11]   "What is CRUD?," *Codecademy*, 2021. [Online]. Available: https://www.codecademy.com/articles/what-is-crud. [Accessed: 14-Sep-2021].

[12]   N. Millard, "Why Are You Still Creating CRUD Apis?," *Medium*, 18-Aug-2021. [Online]. Available: https://levelup.gitconnected.com/why-are-you-still-creating-crud-apis-8790ca261bfb. [Accessed: 14-Sep-2021].

[13]   V. Kurama, "Generate a CRUD app from any database with one click! | Appsmith." *Appsmith*, 24-Aug-2021. [Online]. Available: https://www.appsmith.com/blog/generate-a-crud-app-from-any-database-with-one-click. [Accessed: 14-Sep-2021].

[14]   B. Payton, "What Is a CRUD App?," *DEV Community*, 11-Aug-2021. [Online]. Available: https://dev.to/paytondev/what-is-a-crud-app-4h2e. [Accessed: 14-Sep-2021].

References (continued)

[15]   A. Kmetiuk, "Why CRUD Applications are hard?," *The blog of Anatolii Kmetiuk*, 17-Jun-2017. [Online]. Available: http://akmetiuk.com/posts/2017-06-17-crud-apps.html. [Accessed: 14-Sep-2021].

[16]   "Introducing JSON." *json.org*. [Online]. Available: https://www.json.org/json-en.html. [Accessed: 29-Oct-2021].

[17]   Mozilla and individual contributors, "Working with JSON - Learn web development | MDN." *MDN Web Docs*, 08-Oct-2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. [Accessed: 29-Oct-2021].

[18]   J. Freeman, "What is JSON? A better format for data exchange," *InfoWorld*, 25-Oct-2019. [Online]. Available: https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html. [Accessed: 29-Oct-2021].

[19]   "JSON:API — A specification for building APIs in JSON." *JSON:API*, 29-May-2015. [Online]. Available: https://jsonapi.org/. [Accessed: 29-Oct-2021].

[20]   "ECMA-404" *Ecma International*, Dec-2017. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/. [Accessed: 29-Oct-2021].

> **The JSON data interchange syntax   2nd edition, December 2017**

[21]   L. Tagliaferri, "An Introduction to JSON," *DigitalOcean*, 08-Dec-2016. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-json. [Accessed: 29-Oct-2021].

[22]   "Definition of PARSE." *Merriam-Webster.com Dictionary*. [Online]. Available: https://www.merriam-webster.com/dictionary/parse. [Accessed: 08-Sep-2022].

> divide (a sentence) into grammatical parts and identify the parts and their relations to each other
>
> to examine in a minute way : analyze

[23]   "What is Parse? - Definition from Techopedia," *Techopedia.com*, 23-Mar-2017. [Online]. Available: http://www.techopedia.com/definition/3853/parse. [Accessed: 08-Sep-2022].

> break up a sentence or group of words into separate components

End of document