



## Datenbanksysteme I

WS 2021/22

Torsten Grust, Christian Duta, Tim Fischer

### Assignment #2

Submission Deadline: November 10, 2021 - 10:00

#### Exercise 1: JSONiq Go

(20 Points)

Consider the file `pokedex.json` provided to you in this assignment.

Implement the following queries using the query language JSONiq:

1. Return a sequence of all Pokémon names along with their number (`num`). Each item in the sequence is a JSON-object formatted as follows:

```
{ number: <num>, name: <name> }
```

2. A Pokémon  $p$  prefers opponents that regard at least one of  $p$ 's **types** as their weakness. Create a sequence that contains (the names of) all Pokémon along with the number  $n \in \{1, 2, \dots\}$  of their preferred opponents.

```
{ name: <name>, opponents: <n> }
```

3. Return the average walking distance for Pokémon that hatch from eggs. The return value is simply a number of type **double**. **Hint:** Use the built-in function `double(x)`, to cast **string** to **double**.
4. Pokémon can evolve into new forms. Return the sequence of (the names of) all Pokémon which possess the most evolutionary forms found in `next_evolution` along with the number of evolutions  $n$ . (There may be multiple such Pokémon.)

```
{ name: <name>, evolutions: <n> }
```

5. Return a (potentially deeply) nested structure that represents the evolutionary forms of Pokémon Poliwhag (`num: 060`). In this structure, (the name of) each Pokémon  $p$  is paired with an array `evolutions` of the possible evolutionary forms of  $p$ . The result of this query should be:

```
{ "pokemon" : "Poliwhag", "evolutions" : [  
  { "pokemon" : "Poliwhirl", "evolutions" : [  
    { "pokemon" : "Poliwrath", "evolutions" : [ ] },  
    { "pokemon" : "Politoed" , "evolutions" : [ ] } ]  
  },  
  { "pokemon" : "Poliwrath", "evolutions" : [ ] },  
  { "pokemon" : "Politoed" , "evolutions" : [ ] } ] }
```

**Note:** A correct solution will be way more tricky than — albeit as short as — those for the previous queries (you may have to consult the JSONiq documentation<sup>1</sup> to find a solution). If you don't see any obvious way to formulate this query, please

- (a) describe briefly why this query is particularly challenging, and
- (b) provide one possible plan of attack using your own words.

## Exercise 2: Types and Tables

(10 Points)

So far, we considered an untyped “tabular” version of the Twitter data. We now transform this untyped data into a typed relational table.

Consider the files `tweets.csv` and `users.csv` provided to you in this assignment.

1. For each of the files `tweets.csv` and `users.csv`, construct a `CREATE TABLE` statement and a `\copy` statement to load the CSV data into two tables named `tweets` and `users`, respectively. For each column, choose an *appropriate* data type (`integer`, `boolean`, `text`, `timestamp`, ...). Please hand in a `.sql` file which contains your solution. Note that both files contain unicode characters.

*Note: The PostgreSQL documentation provides additional information on the [CREATE TABLE](#)<sup>2</sup> and [\copy](#)<sup>3</sup> commands as well as the available [data types](#)<sup>4</sup>. The `\copy` meta-command may only be used from within the `psql`-shell.*

*Loading the CSV data requires UTF-8 support. This can be enabled with*

`set client_encoding to 'UTF8';`

*in the `psql`-shell. Adding this command to your `.psqlrc` enables it permanently.*

2. Now is also the perfect time to install PostgreSQL 14 on your system and use it to test your `.sql` script:
  - The given CSV files still contain header lines with column names. If these remain unchanged the import using `\copy` will fail. **Explain why!**
  - Remove the header lines to import the data only.
  - Use (and hand in) the SQL `TABLE` command to list the contents of your new tables `tweets` and `users`. See the PostgreSQL documentation [here](#)<sup>5</sup> for more information.

**Note:** PostgreSQL 14 was released fairly recently. In the unlikely case that your system does not provide an installation for it, you can also use PostgreSQL 13 instead.

**Instruction on how to download and install** PostgreSQL can be found here: <https://www.postgresql.org/download/>.

**For MacOS user:** We recommend installing PostgreSQL using **Postgres.app** which can be downloaded from here: <https://postgresapp.com/>.

---

<sup>1</sup><https://www.jsoniq.org/docs/JSONiq/webhelp/index.html>

<sup>2</sup><https://www.postgresql.org/docs/14/static/sql-createtable.html>

<sup>3</sup><https://www.postgresql.org/docs/14/app-psql.html#APP-PSQL-META-COMMANDS-COPY>

<sup>4</sup><https://www.postgresql.org/docs/14/static/datatype.html>

<sup>5</sup><https://www.postgresql.org/docs/14/sql-select.html#SQL-TABLE>