Database Commands and Troubleshooting

This document includes the code used when setting up the database (AWS RDS, Postgres) and the troubleshooting steps taken to combat the issues that have come up during the set-up. The document was made and managed solely by Hi Leung (hl19wl).

Before creating the database

Prior to the database creation, Terry (Ziyuan) created a web crawler using python to scrape the information on NOTL Museum's official database page. We did this because we did not have access to their database and we needed information to display on our application. Afterwards, she did a first clean of the data and then passed it to me.

I performed a second clean because it is always good to look at data with a fresh pair of eyes since it's easy to miss things after you become accustomed to it. During that cleaning, I made sure that all of the content was in the correct columns as there were some that weren't. As there was inconsistent information (e.g., date formats), I decided to standardise all of that into a starting year and ending year column. If there was no date, I set it to 0. And if the dates weren't specific (e.g., 1950s), I set the range from start: 1950 to end: 1959.

Setting up the database (prior to connection)

Create artifact table (now called item)

```
CREATE TABLE Artifact (
        Obj_ID varchar(255) PRIMARY KEY NOT NULL,
        Obj_name varchar(255) NOT NULL,
        Obj_desc text,
        Obj_start_year int,
        Obj_end_year int,
        Obj_URL varchar(255),
        Obj_num_images int,
        Obj_main_img varchar(255)
)
```

Remove this column because we didn't have the data for this in our import csv (see Github, Data, MuseumArtifactsImport_UTF8v2.csv for final version)

```
ALTER TABLE Artifact DROP COLUMN Obj_num_images
```

- We did this because we decided that one image would be enough to help the user identify which artifact they are looking for. If they wish to see more images, we can redirect them to the link contained in "obj_url".

Rename for clarity to match dropping of column above

```
ALTER TABLE Artifact RENAME Obj_main_img TO Obj_img
```

Importing data into table: est 4 hrs due to file and data reformatting (trial-and-error) and troubleshooting CSV file formats

- (steps taken, not in order as there was some back and forth and repetition of steps):
- Troubleshooting was all performed in excel before exporting to CSV.
- Saved CSV in UTF-8 format
- Removed newlines (=SUBSTITUTE(text,CHAR(10)," "))
- Replaced all quote marks with a quote mark symbol (’) instead of (‘)
- Replaced all commas with “,” to prevent the comma in text from delimitering
- Encapsulated all texts with double brackets to signify it is a block of text that shouldn’t be delimitered. (=CONCAT("""",text,"""))
- Fixed duplicate object IDs eg. xxx.3 and xxx.30 which are seen as duplicated. Fixed by adding an apostrophe before it (=CONCAT("'",obj_id))

Removed apostrophe from obj_id afterwards using:

UPDATE artifact SET obj_id = REPLACE(obj_id, "'","')

- This was done in order for items to be imported properly (see last point from above paragraph). Although the object ID was a varchar, postgres automatically imported it as a number meaning xxx.30 was treated the same as xxx.3 causing a duplicate key. This apostrophe made it so that the IDs were correctly treated as a string so we could import the data.
- However, we decided to remove the apostrophe after importing to keep it more consistent with the museum’s official database.

Rename artifact table to item

ALTER TABLE IF EXISTS artifact RENAME TO item

- Renamed to help programmers as they found it easier to remember item than artifact

Add sid (showcase id) column to table with default value 0

ALTER TABLE item ADD COLUMN sid integer DEFAULT 0

- We added the showcase id to the table after an hour of deliberation. Originally, we were going to have a separate table for showcases (exhibits), but after discussing with Hang, we decided that this would be more efficient in terms of querying (as we wouldn’t need joins to access this information) and reduce the database size.

Set constraint for item table to reference showcase

```
ALTER TABLE item ADD CONSTRAINT showcase_constraint
FOREIGN KEY (sid) REFERENCES showcase(sid)
ON DELETE SET NULL;
```

Create “staff” table for storing staff info and log-in

```
CREATE TABLE staff (
        staff_id SERIAL PRIMARY KEY,
        username varchar(100) UNIQUE NOT NULL,
        passcode varchar(100) UNIQUE NOT NULL,
        lastname varchar(100),
        firstname varchar(100),
        sec_level integer
)
```

Insert example login

```
INSERT INTO staff (username, passcode, lastname, firstname, sec_level)
        VALUES ('admin', 'adminpw', 'Ahd', 'Minh', 99)
```

Drop unique password constraint

```
ALTER TABLE staff DROP CONSTRAINT staff_passcode_key
```

- Removed because it doesn't make sense. I'm not sure why I added it in the first place.

Create showcase table

```
CREATE TABLE showcase (
        sid SERIAL PRIMARY KEY,
        length_m REAL,
        width_m REAL,
        x integer,
        y integer,
        floor_no integer
)
```

Create example showcase row

```
INSERT INTO showcase VALUES (0,0,0,0,0,0)
```

Connecting the database to the application

Attempting to implement a rest API - Around 3 weeks without any fruition

In order to create a more secure connection with the database and to prevent potential malicious users from causing damage to our data, we originally wanted to use a REST API to connect the database with the application. I have included the following resources I attempted to use. There were other resources used too but I excluded them due to irrelevance.

Create API based on this video: ~~https://www.youtube.com/watch?v=i5NEHwFeeuY~~
https://www.youtube.com/watch?v=tbFiVKOYcXk
~~https://www.youtube.com/watch?v=dZQbRLL7qxE~~
https://www.youtube.com/watch?v=K1OI-S0ET70

- RDS API doc: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/ProgrammingGuide.html
- RDS query API: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Using_the_Query_API.html
- https://gist.github.com/bendalby82/5bdccc8f689f5248f50be27e634c8172/revisions
  Pasted this code into AWS Cloudshell to make zip for postgres dependency for layers
  - Download file: /home/cloudshell-user/nodejs.zip
  - https://node-postgres.com/features/connecting

    - There were likely to be changes to the code but I did not record them at the time and looking back, I do not recall what those changes were.

After 3 weeks (around 25 hours or so), we decided that this was taking too much time so we were going to use a less secure method of connecting to the database - JDBC.

Connection using JDBC

We ran into some issues when trying to use JDBC. Firstly, most tutorials have the connection details hard-coded which makes it not secure. Thus we decided to use hidden files to store the database credentials. Upon start-up, the application will retrieve the credentials from the file and use it to connect to the database.

- Source: https://yfujiki.medium.com/how-to-store-use-sensitive-information-in-android-development-bc352892ece7

We followed this tutorial to set up the JDBC connection between the Postgres server and the Android app.

- Source: https://medium.com/cyber-explorer/how-to-connect-an-android-project-to-a-postgresql-database-663cb0f5ba19

However, there were some connection issues. Firstly there was the issue of missing driver components. This took 1-2 weeks to troubleshoot and the fix was to change the driver version to 42.2.5. This was able to resolve the range of difference errors with null pointers and “ManagementFactory” errors being the prominent ones.

- Null pointer may have occurred due to not being able to properly connect
- ManagementFactory was a classpath error which may have something to do with the installation but I don’t really understand it
- Also added strict mode (which may have contributed to fixing it): https://stackoverflow.com/questions/16666619/postgresql-connection-with-database-in-java-fail

There were also some errors regarding the thread as Android studio does not permit creating database connections on the main thread. Thus, we opted to create a helper class which manages the database connections using threads and the operations performed by it. Hang Li did the majority of the thread related work so I cannot provide further explanation without accuracy.

After connecting the database

Once the connection was working, I created a test code to run to understand how to use JDBC. We were able to adapt this for our other methods involving queries.

SQL statement to test later:

```
try {
        Statement statement = connection.createStatement();
        String sql = "SELECT * FROM staff";
        ResultSet resultSet = statement.executeQuery(sql);

        while (resultSet.next()) {
            System.out.println(resultSet.getString(1));
            System.out.println(resultSet.getString(2));
            System.out.println(resultSet.getString(3));
        }

        resultSet.close();
        statement.close();
    } catch (SQLException e) {
        System.out.print(e.getMessage());
        e.printStackTrace();
    }
```

The last biggest change to the database was the modification of the data types for the X and Y coordinates in the “showcase” table. As we decided to use floats for our coordinates in our application, we had to change the database’s data type for these values to real to match.