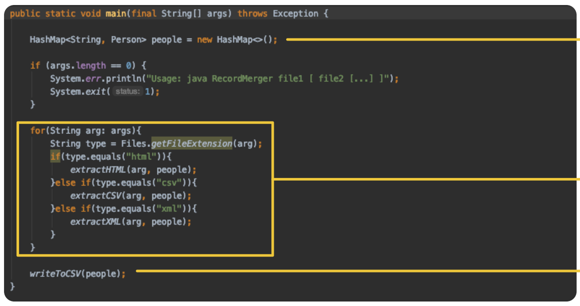
My original approach was to try to get a gist on how to use Guava and Apache Common to parse html and CSV documents and write it to CSV with given information.

In RecordMerger.java (not MergeRecord.java!)

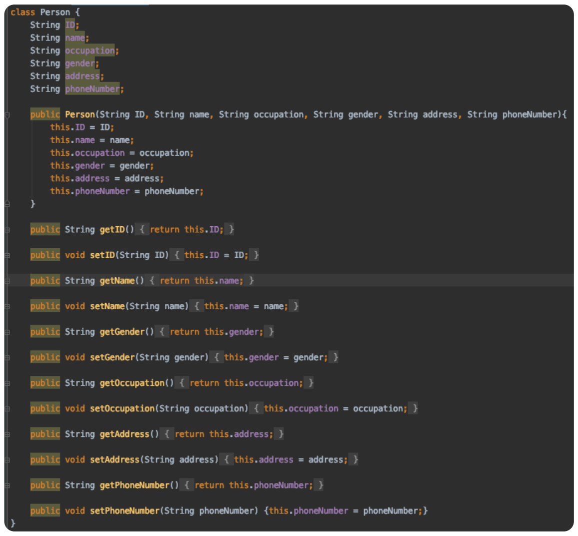


```java
public static void main(final String[] args) throws Exception {

    HashMap<String, Person> people = new HashMap<>();

    if (args.length == 0) {
        System.err.println("Usage: java RecordMerger file1 [ file2 [...] ]");
        System.exit( status: 1);
    }

    for(String arg: args){
        String type = Files.getFileExtension(arg);
        if(type.equals("html")){
            extractHTML(arg, people);
        }else if(type.equals("csv")){
            extractCSV(arg, people);
        }else if(type.equals("xml")){
            extractXML(arg, people);
        }
    }

    writeToCSV(people);
}
```

This hashmap holds all the information all the text files have, and assumes the attributes stays the same.

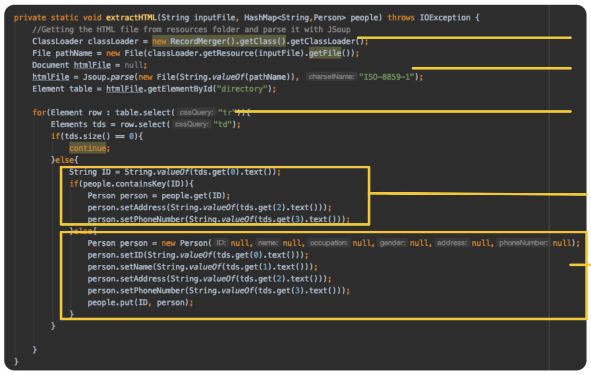
Inside the main function, this for loop detects the file type using Guava (imported in build.gradle)

And then writes it to CSV



```java
class Person {
    String ID;
    String name;
    String occupation;
    String gender;
    String address;
    String phoneNumber;

    public Person(String ID, String name, String occupation, String gender, String address, String phoneNumber){
        this.ID = ID;
        this.name = name;
        this.occupation = occupation;
        this.gender = gender;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }

    public String getID() { return this.ID; }

    public void setID(String ID) { this.ID = ID; }

    public String getName() { return this.name; }

    public void setName(String name) { this.name = name; }

    public String getGender() { return this.gender; }

    public void setGender(String gender) { this.gender = gender; }

    public String getOccupation() { return this.occupation; }

    public void setOccupation(String occupation) { this.occupation = occupation; }

    public String getAddress() { return this.address; }

    public void setAddress(String address) { this.address = address; }

    public String getPhoneNumber() { return this.phoneNumber; }

    public void setPhoneNumber(String phoneNumber) {this.phoneNumber = phoneNumber;}
}
```

Assuming the text files that needs to be parsed have the same number of attributes that needs to be parsed, creating a person object that can hold the same person's information from different text files can be helpful.



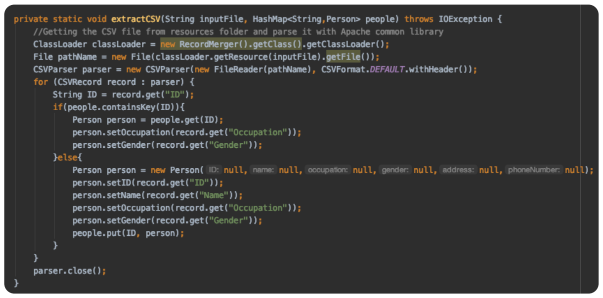
```java
private static void extractHTML(String inputFile, HashMap<String,Person> people) throws IOException {
    //Getting the HTML file from resources folder and parse it with JSoup
    ClassLoader classLoader = new RecordMerger().getClass().getClassLoader();
    File pathName = new File(classLoader.getResource(inputFile).getFile());
    Document htmlFile = null;
    htmlFile = Jsoup.parse(new File(String.valueOf(pathName)), charsetName: "ISO-8859-1");
    Element table = htmlFile.getElementById("directory");

    for(Element row : table.select( cssQuery: "tr")){
        Elements tds = row.select( cssQuery: "td");
        if(tds.size() == 0){
            continue;
        }else{
            String ID = String.valueOf(tds.get(0).text());
            if(people.containsKey(ID)){
                Person person = people.get(ID);
                person.setAddress(String.valueOf(tds.get(2).text()));
                person.setPhoneNumber(String.valueOf(tds.get(3).text()));
            }else{
                Person person = new Person( ID: null, name: null, occupation: null, gender: null, address: null, phoneNumber: null);
                person.setID(String.valueOf(tds.get(0).text()));
                person.setName(String.valueOf(tds.get(1).text()));
                person.setAddress(String.valueOf(tds.get(2).text()));
                person.setPhoneNumber(String.valueOf(tds.get(3).text()));
                people.put(ID, person);
            }
        }

    }
}
```

In extractHTML:

1. Gets the path
2. Reads the file using JSoup (imported in gradle)
3. Loop through the tags to get the information.

3.1 If the person is a new entry, a new person object is created, and the key - value pair (ID, Person) is then added to the hashmap passed from the main function

3.2 If the person is not a new entry, but has different attributes in this file file, the key-value pair (ID, Person) is extracted from the hashmap, and the new attributes are added to the person. Hashmap is being updated

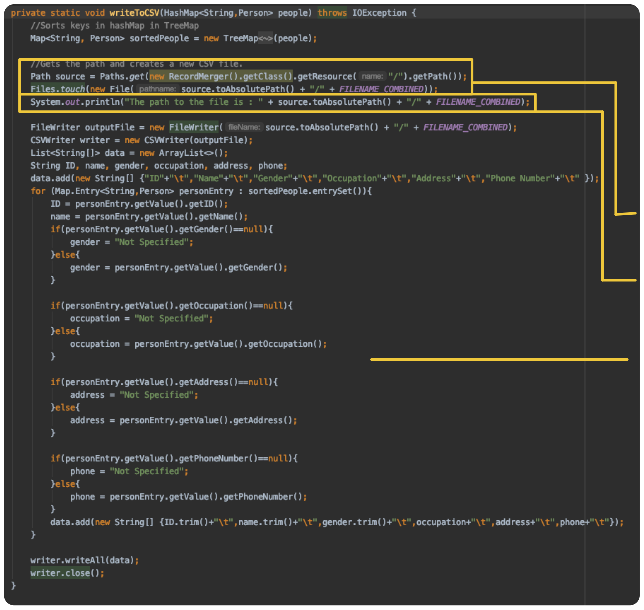


```java
private static void extractCSV(String inputFile, HashMap<String,Person> people) throws IOException {
    //Getting the CSV file from resources folder and parse it with Apache common library
    ClassLoader classLoader = new RecordMerger().getClass().getClassLoader();
    File pathName = new File(classLoader.getResource(inputFile).getFile());
    CSVParser parser = new CSVParser(new FileReader(pathName), CSVFormat.DEFAULT.withHeader());
    for (CSVRecord record : parser) {
        String ID = record.get("ID");
        if(people.containsKey(ID)){
            Person person = people.get(ID);
            person.setOccupation(record.get("Occupation"));
            person.setGender(record.get("Gender"));
        }else{
            Person person = new Person( ID: null, name: null, occupation: null, gender: null, address: null, phoneNumber: null);
            person.setID(record.get("ID"));
            person.setName(record.get("Name"));
            person.setOccupation(record.get("Occupation"));
            person.setGender(record.get("Gender"));
            people.put(ID, person);
        }
    }
    parser.close();
}
```

In CSV file:

Same logic with extractHTML(String input, HashMap<String, person> people)

Library used : Apache Common

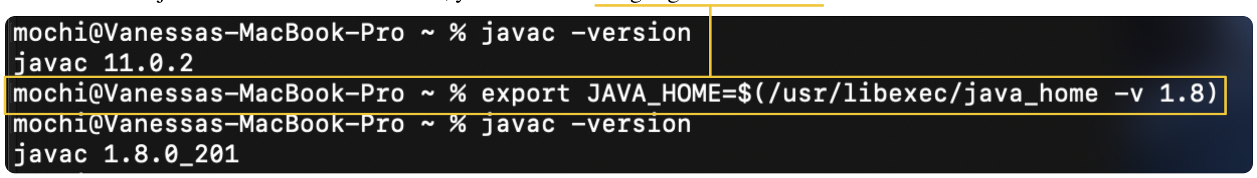


```java
private static void writeToCSV(HashMap<String,Person> people) throws IOException {
    //Sorts keys in hashMap in TreeMap
    Map<String, Person> sortedPeople = new TreeMap<~>(people);

    //Gets the path and creates a new CSV file.
    Path source = Paths.get(new RecordMerger().getClass().getResource( name: "/").getPath());
    Files.touch(new File( pathname: source.toAbsolutePath() + "/" + FILENAME_COMBINED));
    System.out.println("The path to the file is : " + source.toAbsolutePath() + "/" + FILENAME_COMBINED);

    FileWriter outputFile = new FileWriter( fileName: source.toAbsolutePath() + "/" + FILENAME_COMBINED);
    CSVWriter writer = new CSVWriter(outputFile);
    List<String[]> data = new ArrayList<>();
    String ID, name, gender, occupation, address, phone;
    data.add(new String[] {"ID"+"\t","Name"+"\t","Gender"+"\t","Occupation"+"\t","Address"+"\t","Phone Number"+"\t" });
    for (Map.Entry<String,Person> personEntry : sortedPeople.entrySet()){
        ID = personEntry.getValue().getID();
        name = personEntry.getValue().getName();
        if(personEntry.getValue().getGender()==null){
            gender = "Not Specified";
        }else{
            gender = personEntry.getValue().getGender();
        }

        if(personEntry.getValue().getOccupation()==null){
            occupation = "Not Specified";
        }else{
            occupation = personEntry.getValue().getOccupation();
        }

        if(personEntry.getValue().getAddress()==null){
            address = "Not Specified";
        }else{
            address = personEntry.getValue().getAddress();
        }

        if(personEntry.getValue().getPhoneNumber()==null){
            phone = "Not Specified";
        }else{
            phone = personEntry.getValue().getPhoneNumber();
        }
        data.add(new String[] {ID.trim()+"\t",name.trim()+"\t",gender.trim()+"\t",occupation+"\t",address+"\t",phone+"\t"});
    }

    writer.writeAll(data);
    writer.close();
}
```

In writeToCSV:

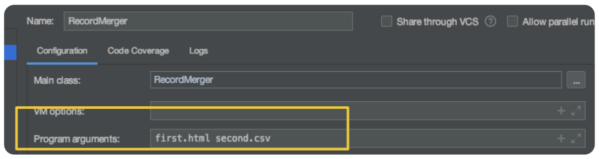
1. Gets the path of the resources folder. Assumption: code is running in IntelliJ and the resources folder exists by default.
2. Prints out the path to instruct the user to find the newly created file
3. Prints out header
4. Prints out information according to the header. Check if the information is null. If it's null, prints "Not Specified" in the cell.

To run the program, simply navigate to the src folder and run java RecordMerger.java <file> <file> in terminal. Gradle and its dependencies needs to be installed in the terminal.
Make sure the java version is 1.8. If it's 11.0, you can run the highlighted command to make it 1.8



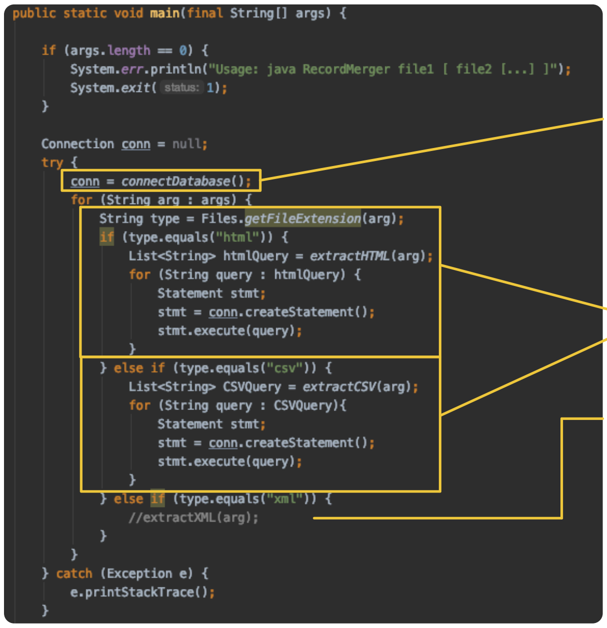
```
mochi@Vanessas-MacBook-Pro ~ % javac -version
javac 11.0.2
mochi@Vanessas-MacBook-Pro ~ % export JAVA_HOME=$(/usr/libexec/java_home -v 1.8)
mochi@Vanessas-MacBook-Pro ~ % javac -version
javac 1.8.0_201
```

Or, simply run it in intelliJ and make sure the files are passed in the configuration -> program argument



Name: RecordMerger
Main class: RecordMerger
VM options:
Program arguments: first.html second.csv

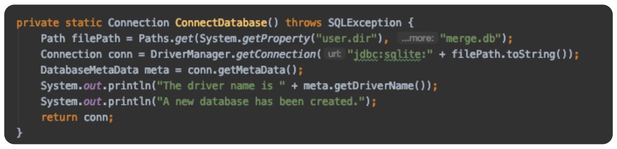
Second approach: Dynamically store and resize number of files, number of columns in the file, without knowing what are the attributes are.
In MergeRecord.java (not RecordMerge.java!)



```java
public static void main(final String[] args) {

    if (args.length == 0) {
        System.err.println("Usage: java RecordMerger file1 [ file2 [...] ]");
        System.exit( status: 1);
    }

    Connection conn = null;
    try {
        conn = connectDatabase();
        for (String arg : args) {
            String type = Files.getFileExtension(arg);
            if (type.equals("html")) {
                List<String> htmlQuery = extractHTML(arg);
                for (String query : htmlQuery) {
                    Statement stmt;
                    stmt = conn.createStatement();
                    stmt.execute(query);
                }
            } else if (type.equals("csv")) {
                List<String> CSVQuery = extractCSV(arg);
                for (String query : CSVQuery){
                    Statement stmt;
                    stmt = conn.createStatement();
                    stmt.execute(query);
                }
            } else if (type.equals("xml")) {
                //extractXML(arg);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
```

In main(final String[] args):

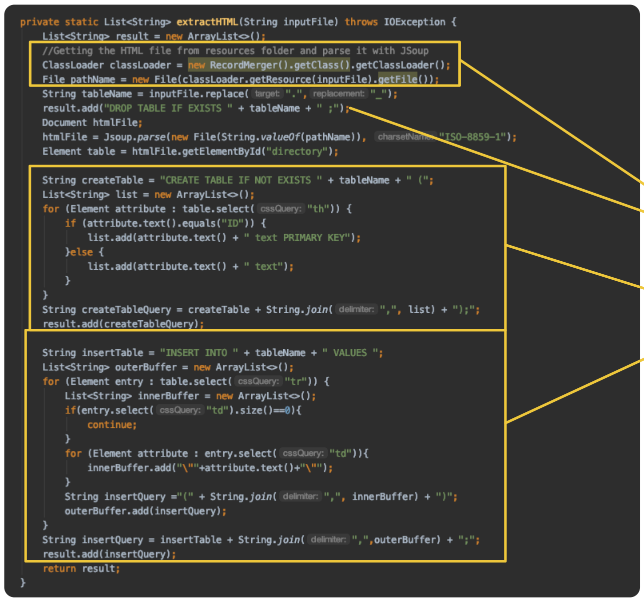
1. Create connection with the database using sqlite
2. Detect file type (with Guava)
   2.1 extractHTML(arg)/ extractCSV(arg) passes back list of SQLite queries
   2.2 Write information to database with the queries
3. New file types can be added in the future
4. Writes to CSV



```java
private static Connection ConnectDatabase() throws SQLException {
    Path filePath = Paths.get(System.getProperty("user.dir"), ...more: "merge.db");
    Connection conn = DriverManager.getConnection( url: "jdbc:sqlite:" + filePath.toString());
    DatabaseMetaData meta = conn.getMetaData();
    System.out.println("The driver name is " + meta.getDriverName());
    System.out.println("A new database has been created.");
    return conn;
}
```

In connectDatabase():

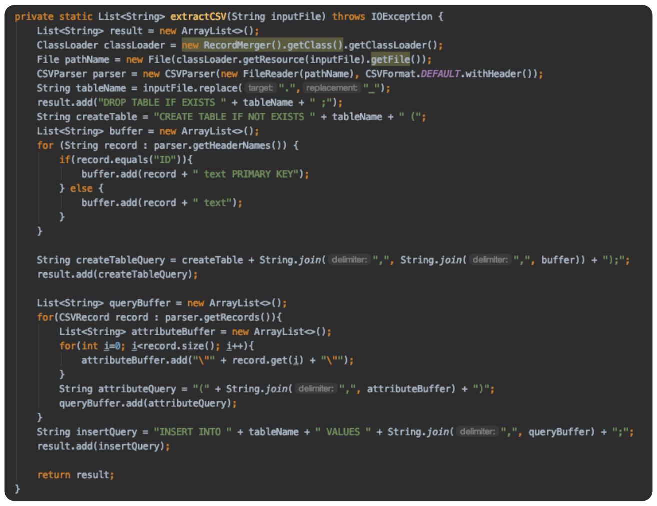
Connect to database and prints the path for the user



```java
private static List<String> extractHTML(String inputFile) throws IOException {
    List<String> result = new ArrayList<>();
    //Getting the HTML file from resources folder and parse it with JSoup
    ClassLoader classLoader = new RecordMerger().getClass().getClassLoader();
    File pathName = new File(classLoader.getResource(inputFile).getFile());
    String tableName = inputFile.replace( target: ".", replacement: "_");
    result.add("DROP TABLE IF EXISTS " + tableName + " ;");
    Document htmlFile;
    htmlFile = Jsoup.parse(new File(String.valueOf(pathName)), charsetName: "ISO-8859-1");
    Element table = htmlFile.getElementById("directory");

    String createTable = "CREATE TABLE IF NOT EXISTS " + tableName + " (";
    List<String> list = new ArrayList<>();
    for (Element attribute : table.select( cssQuery: "th")) {
        if (attribute.text().equals("ID")) {
            list.add(attribute.text() + " text PRIMARY KEY");
        }else {
            list.add(attribute.text() + " text");
        }
    }
    String createTableQuery = createTable + String.join( delimiter: ",", list) + ");";
    result.add(createTableQuery);

    String insertTable = "INSERT INTO " + tableName + " VALUES ";
    List<String> outerBuffer = new ArrayList<>();
    for (Element entry : table.select( cssQuery: "tr")) {
        List<String> innerBuffer = new ArrayList<>();
        if(entry.select( cssQuery: "td").size()==0){
            continue;
        }
        for (Element attribute : entry.select( cssQuery: "td")){
            innerBuffer.add("\""+attribute.text()+"\"");
        }
        String insertQuery ="(" + String.join( delimiter: ",", innerBuffer) + ")";
        outerBuffer.add(insertQuery);
    }
    String insertQuery = insertTable + String.join( delimiter: ",",outerBuffer) + ";";
    result.add(insertQuery);
    return result;
}
```

In extractHTML(String inputfile):

1. Gets the path
2. Make sure the table with the same name does not exist in the database, if there's a table with the same name, it will be overwritten later.
3. Create a new table for this file by adding the CREATE TABLE query into the result list.
4. Gets the header and define the structure and type of the attribute in the table by adding the INSERT INTO query into the result list
5. Return a list of queries to be executed in main.



```java
private static List<String> extractCSV(String inputFile) throws IOException {
    List<String> result = new ArrayList<>();
    ClassLoader classLoader = new RecordMerger().getClass().getClassLoader();
    File pathName = new File(classLoader.getResource(inputFile).getFile());
    CSVParser parser = new CSVParser(new FileReader(pathName), CSVFormat.DEFAULT.withHeader());
    String tableName = inputFile.replace( target: ".", replacement: "_");
    result.add("DROP TABLE IF EXISTS " + tableName + " ;");
    String createTable = "CREATE TABLE IF NOT EXISTS " + tableName + " (";
    List<String> buffer = new ArrayList<>();
    for (String record : parser.getHeaderNames()) {
        if(record.equals("ID")){
            buffer.add(record + " text PRIMARY KEY");
        } else {
            buffer.add(record + " text");
        }
    }

    String createTableQuery = createTable + String.join( delimiter: ",", String.join( delimiter: ",", buffer)) + ");";
    result.add(createTableQuery);

    List<String> queryBuffer = new ArrayList<>();
    for(CSVRecord record : parser.getRecords()){
        List<String> attributeBuffer = new ArrayList<>();
        for(int i=0; i<record.size(); i++){
            attributeBuffer.add("\"" + record.get(i) + "\"");
        }
        String attributeQuery = "(" + String.join( delimiter: ",", attributeBuffer) + ")";
        queryBuffer.add(attributeQuery);
    }
    String insertQuery = "INSERT INTO " + tableName + " VALUES " + String.join( delimiter: ",", queryBuffer) + ";";
    result.add(insertQuery);

    return result;
}
```

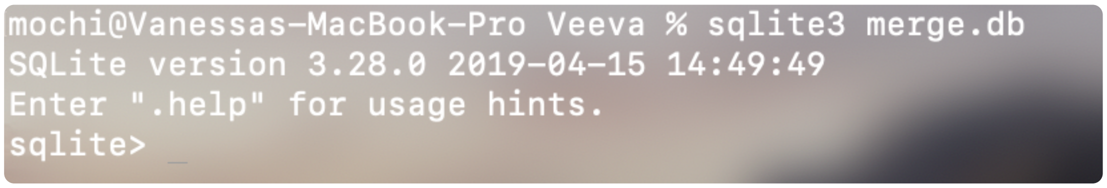
《

In extractCSV(String inputFile):

Same logic as extractHTML(String inputFile)

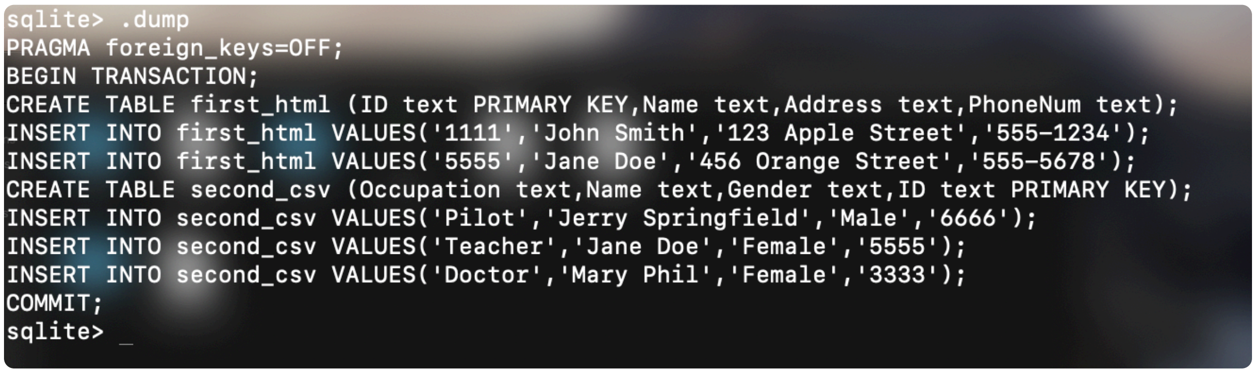
To check what's in the sqlite3 database, run the program first, and then navigate to the folder where the merge.db exists (in the main function it tells the user the path to the database) and enter the following terminal command:

sqlite3 merge.db



```
mochi@Vanessas-MacBook-Pro Veeva % sqlite3 merge.db
SQLite version 3.28.0 2019-04-15 14:49:49
Enter ".help" for usage hints.
sqlite> _
```

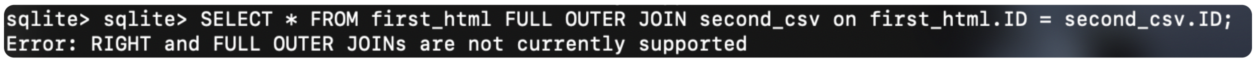
Inside sqlite3, type in ".dump" and it will display the table and its content.



```
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE first_html (ID text PRIMARY KEY,Name text,Address text,PhoneNum text);
INSERT INTO first_html VALUES('1111','John Smith','123 Apple Street','555-1234');
INSERT INTO first_html VALUES('5555','Jane Doe','456 Orange Street','555-5678');
CREATE TABLE second_csv (Occupation text,Name text,Gender text,ID text PRIMARY KEY);
INSERT INTO second_csv VALUES('Pilot','Jerry Springfield','Male','6666');
INSERT INTO second_csv VALUES('Teacher','Jane Doe','Female','5555');
INSERT INTO second_csv VALUES('Doctor','Mary Phil','Female','3333');
COMMIT;
sqlite> _
```

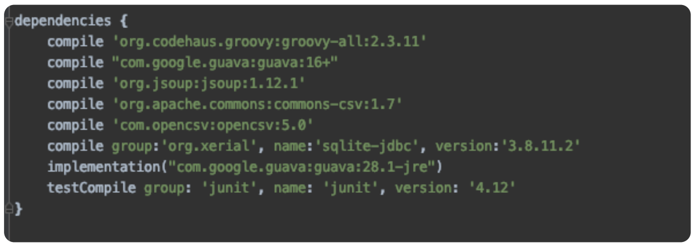
In this approach I was going to join the tables together so that I can merge information with their primary key "ID" without duplicated information using the query "SELECT * FROM first_html FULL OUTER JOIN second_csv on first_html.ID = second_csv.ID".

However, RIGHT and FULL OUTER JOINS are not currently supported in sqlite3.



```
sqlite> sqlite> SELECT * FROM first_html FULL OUTER JOIN second_csv on first_html.ID = second_csv.ID;
Error: RIGHT and FULL OUTER JOINs are not currently supported
```

Gradle dependencies (build.gradle):



```
dependencies {
    compile 'org.codehaus.groovy:groovy-all:2.3.11'
    compile "com.google.guava:guava:16+"
    compile 'org.jsoup:jsoup:1.12.1'
    compile 'org.apache.commons:commons-csv:1.7'
    compile 'com.opencsv:opencsv:5.0'
    compile group:'org.xerial', name:'sqlite-jdbc', version:'3.8.11.2'
    implementation("com.google.guava:guava:28.1-jre")
    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```