1. **Missing Database Driver**

A missing database driver is a result of not specifying and initializing the database driver in the code or simply not having the database driver downloaded. The driver is important as it enables the Java application to interact with the database. It acts as a translator that communicates API calls from the application to the database while also managing the connection.

To generate this issue, remove the following code:

```
String DRIVER = "com.mysql.cj.jdbc.Driver";
Class.forName(DRIVER);
```

From the class DbConn.java.

However, as of JDBC 4.0 (Java 6), the driver manager automatically loads the driver if it is in the classpath when Class.forName() is called.

To solve this problem, it is important to include the line

String DRIVER = “com.mysql.cj.jdbc.Driver”;

Which specifies the specific driver that communicates with the MySQL database. Once this is included, it must be initialized with the line

Class.forName(DRIVER);

Which loads the driver at runtime and enables the driver to be able to communicate with the database.

2. **Database Unreachable**

This issue is a result of, as the name implies, the database being unreachable. This is can be caused by several different errors. The most common one is not being tunneled in to the server hosting the MySQL database before starting the Spring Boot local server, which results in the following error when attempting to query something by opening the Web API link:

"dbError": "Problem getting connection:Communications link failure\n\nThe last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server."

This issue can be fixed by opening command prompt on your local machine and typing the following command:

ssh -L 3307:cis-linux2.temple.edu:3306 tua12345@cis-linux2.temple.edu

and replacing tua12345 with your own TU ID.

Another possible cause of this issue is specifying an incorrect database hostname in the DbConn.java class. This is located in the String dbAndPass variable, so be sure to check the hostname contains the correct credentials to access the database, making sure that the password is the given SQL and not the AccessNet password.

3. **Database Not Authorized**

This error is a result of having the incorrect credentials in your hostname variable String dbAndPass. Similar to the previous solution, make sure to check what your MySQL password is, located in your remote directory on cis-linux2, and do not enter your AccessNet password.

The error that generates is

“dbError": "Problem getting connection:Access denied for user 'tua12345'@'000.00.00.000' (using password: YES)",

4. **Syntax error in SQL Statement**

This error is fairly self-explanatory. In the files holding the various views (ex: WebUserView.java in the sample code), if the SQL statement has improper syntax for the SQL version in use, it will throw the following error:

"errorMsg": "Exception thrown in ReviewView.getReviewGameName(): Statement.executeQuery() cannot issue statements that do not produce result sets."

Here is an example of improper SQL syntax:

```
String sql = "SELCT review_id, review_game_name, review.web_user_id " + //
                    "FROM review " + //
                    "ORDER BY review_id;";
```

As you can see, there is a typo in the SELECT statement. It is good practice to try the query in MySQL Workbench prior to inserting it into the source code.

5. **Error Extracting Data from Result Set (bad column name)**

This error is self-explanatory. To generate this error, simply enter the wrong column in your SQL statement.

For example, the correct column name for this query is review_id, but review is entered instead

```
String sql = "SELECT review, review_game_name, review.web_user_id " +
                    "FROM review " + //
                    "ORDER BY review_id;";
```

This generates the following error:

"errorMsg": "Exception thrown in ReviewView.getReviewGameName(): Unknown column 'review' in 'SELECT'"

To fix this error, make sure all field names are valid and double check by writing a query in MySQL Workbench prior to inserting into the source code.

6. **Error Extracting Data from Result Set (wrong data type)**

This error is caused by converting the received field from the database into the wrong datatype. This error is most likely located in whatever View java file is being used to query the database server.

While the code iterates through the resultant data, one of the fields are being parsed incorrectly with the incorrect format function.

To generate this error, change one of the format functions to a function that will attempt to convert the output into the incorrect data type.

```
while(results.next()){
    sd = new StringData();

    sd.reviewId = Format.fmtString(results.getObject(columnLabel:"review_Id"));
    sd.reviewGameName = Format.fmtString(results.getObject(columnLabel:"review_game_name"));
    sd.webUserId = Format.fmtInteger(results.getObject(columnLabel:"web_user_id"));

    sdl.add(sd);
}
```

In the above example, reviewId is the auto-incrementing primary key of the table, so when fmtString is used instead of fmtInteger, it throws an error and results in the following error to be thrown:

"reviewId": "bad String in FormatUtils:1. Error:class java.lang.Integer cannot be cast to class java.lang.String (java.lang.Integer and java.lang.String are in module java.base of loader 'bootstrap')"

It is important to note that this error will not cause the entire query to break. This error will only appear in the specific field(s) that are converted to the wrong data type.