Reflective Essay on my Learning Journey

in Deciphering Big Data

Student ID: 12696474

Module: Deciphering Big Data October 2024

Due: 27th January 2025

Table of Contents

List of Figures

Introduction

The *Deciphering Big Data* module has been a transformative experience, combining theoretical concepts with practical applications. Through collaborative and individual tasks, I refined skills in data collection, cleaning, and database design while developing technical and interpersonal abilities. Using Rolfe et al.'s (2001) *What? So What? Now What?* framework, this reflection analyses key experiences, evaluates their impact, and outlines plans for future growth.

Data Extraction Techniques

What?

This module introduced data extraction methods like web scraping with Python’s *BeautifulSoup* and *requests* and working with formats like CSV, JSON, and XML. A web scraping activity to extract job listings and structure the data into a JSON file (Figure 1) helped me navigate HTML elements to extract details like job titles and locations.

While API-based extraction was covered theoretically, tasks such as designing security requirements for API interactions (Figure 2) improved my understanding of structured and secure data exchanges.

So What?

This experience highlighted the importance of data extraction as the foundation of the data analysis pipeline. Efficient and structured extraction facilitates smoother cleaning and modelling processes.

Web scraping presented challenges, such as interpreting HTML structures and adhering to website scraping policies, which strengthened my technical skills and underscored the importance of ethical practices and staying updated with technology.

Working with diverse file formats, like hierarchical JSON datasets, broadened my technical understanding and appreciation for data representation. Learning about APIs highlighted their critical role in secure and effective data exchanges in modern systems.

Now What?

To advance my skills, I plan to explore techniques like using Selenium for dynamic web scraping and batch processing for large datasets (Day, 2025). I also aim to deepen my knowledge of API integration, focusing on secure authentication methods and best practices.

Data Cleaning and Exploration

What?

This module focused on data cleaning and transformation to prepare datasets for analysis. Key tasks included handling missing values, addressing outliers, and converting datasets into compatible formats. I practiced cleaning UNICEF household survey data using steps from Kazil and Jarmul (2016) (Figure 3) and explored automating repetitive tasks.

I also used *FuzzyWuzzy* to address textual inconsistencies by identifying and correcting similar strings (Figure 4). I found this tool intriguing, recognising its potential to enhance dataset accuracy, especially when managing typos or naming variations.

So What?

These experiences stressed the importance of clean, well-structured data as the foundation for reliable analysis. Often overlooked, data cleaning is a vital stage in the pipeline (Big Data, Analytics & Consulting Cell NIT Warangal, 2024), ensuring accuracy and consistency before deeper analysis.

Debugging deprecated Python commands from the book's older examples highlighted the importance of staying current with Python's evolving ecosystem, strengthening my problem-solving and adaptability.

Addressing outliers and automating processes improved my ability to handle complex datasets, while resolving inconsistencies sharpened my critical thinking, preparing me for larger, more intricate projects.

Now What?

I plan to explore advanced cleaning techniques, such as machine learning for anomaly detection and pattern recognition, while building automated Python workflows to boost efficiency. Additionally, utilising open-source tools and engaging with community-driven solutions will help me adopt innovative approaches, stay updated, and contribute to resource-efficient data solutions.

Data Modelling

What?

Data modelling was a key focus of this module, emphasising logical structures for organising and managing data. Presenting an entity-relationship diagram (ERD) for a retail store database (Figure 5) deepened my understanding of logical consistency and data integrity. Normalising a dataset to third normal form (3NF), through 1NF and 2NF, refined my skills in identifying functional dependencies, organising attributes, and reducing redundancy. These structures were implemented with SQL scripts, creating tables and enforcing referential integrity with primary and foreign keys, as shown in Figure 6. Testing constraints like cascading deletes further strengthened my grasp of robust database design.

So What?

Starting with an ERD was invaluable for maintaining organisation and improving team communication. The visual representation clarified relationships and helped identify potential issues early. As IBM (no date) notes, ERDs reduce errors, ensure consistent documentation, and improve database performance. Breaking the design into conceptual, logical, and physical stages streamlined development.

Normalisation and data building translated theoretical concepts into practical database design. These processes reduced redundancy, optimised storage, and simplified querying – reflecting best practices in database management (Codd, 1970). Testing referential integrity outlined the importance of planning and strong database structures to ensure reliability.

Now What?

I plan to transition from MS Access to SQL at work to improve scalability and efficiency. Additionally, I aim to explore dimensional modelling for data warehouses and delve into NoSQL databases to manage unstructured and semi-structured data.

Teamwork and Collaboration

What?

During the Unit 6 database design project, I worked with peers to create a retail store database. We drafted a team contract, rotated roles, and implemented an ERD using MySQL Workbench. My contributions included building the ERD, explaining tables, attributes, and data types (Figure 7), and drafting key sections of the report.

Collaboration had its challenges, such as coordinating across time zones and managing differing schedules. Flexibility and consistent communication were essential. For example, we adapted plans when delays arose, ensuring steady progress despite varying availability.

So What?

This experience highlighted the importance of communication and task delegation in teamwork. As Marlow et al. (2018) emphasise, effective communication drives team success. Working with a diverse team sharpened my skills in active listening, clear expression, and adaptability, enabling productive collaboration despite differences in work styles.

Receiving feedback on my ERD and report contributions boosted my confidence and underlined the value of technical and organisational skills. Managing scheduling conflicts also reinforced the importance of proactive planning and clear communication. Ultimately, the diverse perspectives within the team led to innovative solutions and a stronger final product.

Now What?

To foster future collaborations, I plan to promote proactive communication and shared accountability. Establishing clear roles and expectations early can prevent misunderstandings and improve efficiency. Regular team reflections will help identify and resolve challenges promptly, ensuring smoother collaboration and better outcomes.

Conclusion

The individual project in Unit 11 was the culmination of the *Deciphering Big Data* module, bringing together my gained knowledge and skills. This comprehensive project provided an opportunity to apply the full range of material covered throughout the module. Notably, I refined the initial database proposal, enhanced its functionality through SQL queries, and proposed strategies to future-proof the company. I also focused on integrating compliance requirements and security measures to develop robust and reliable database solutions.

Looking ahead, I plan to explore advanced areas such as machine learning, real-time analytics, and emerging technologies like edge computing and blockchain. These fields hold great potential for innovation, and I am eager to contribute while prioritising ethical practices and compliance.

References

Big Data, Analytics & Consulting Cell NIT Warangal. (2024*) Data Cleaning 101: Taming Messy Data Like a Pro*. Available at: https://medium.com/@bdacc_club/data-cleaning-101-taming-messy-data-like-a-pro-65c9bdabf4f6 (Accessed 24 January 2025).

Codd, E.F. (1970) ‘A relational model of data for large shared data banks’, *Communications of the ACM*, 13(6), pp. 377-387. Available at: https://doi.org/10.1145/362384.362685

Day, F. (2025) *Why Every Data Scientist Should Know Selenium*. Available at: https://www.nobledesktop.com/classes-near-me/blog/why-learn-selenium-for-data-science (Accessed 23 January 2025).

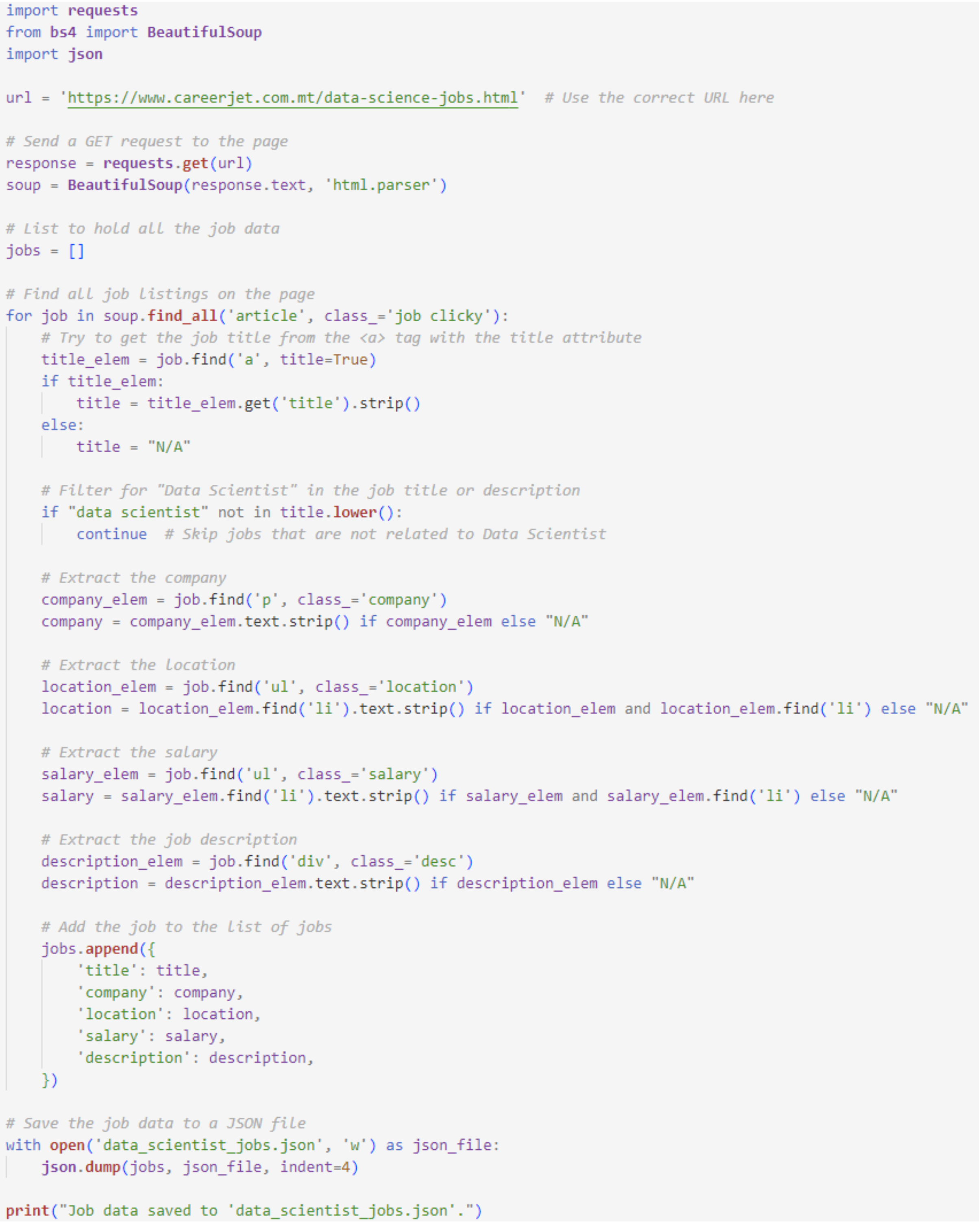
IBM. (no date) *What is data modeling?* Available at: https://www.ibm.com/think/topics/data-modeling (Accessed 24 January 2025).

Kazil, J. and Jarmul, K. (2016) *Data Wrangling with Python: Tips and Tools to Make Your Life Easier*. 1st edn. Sebastopol: O'Reilly Media Inc.

Marlow, S. L. et al. (2018) ‘Does team communication represent a one-size-fits-all approach?: A meta-analysis of team communication and performance’, *Organizational Behavior and Human Decision Processes*, 144(1), pp. 145-170. Available at: https://doi.org/10.1016/j.obhdp.2017.08.001

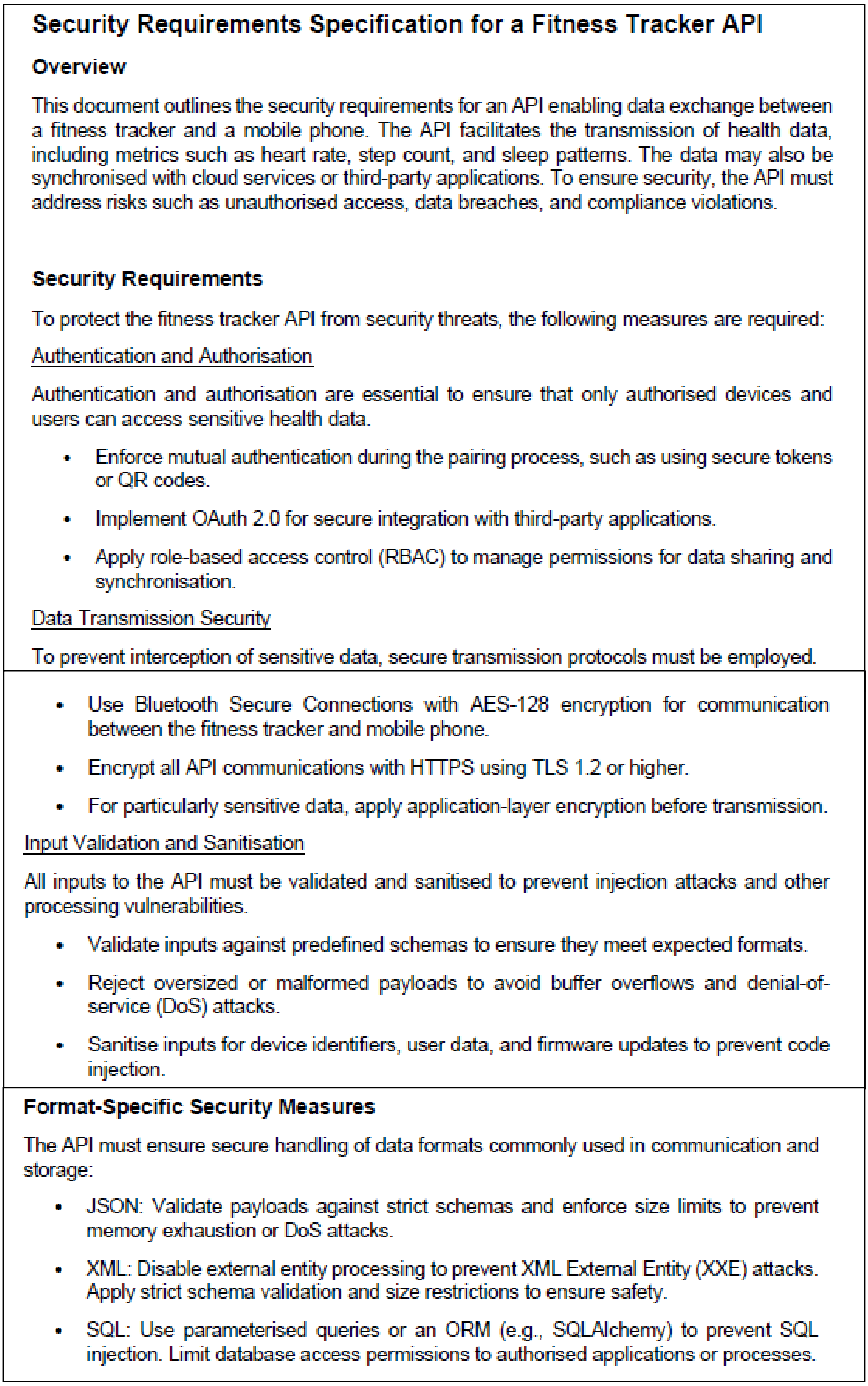
Rolfe, G., Freshwater, D. and Jasper, M. (2001) *Critical reflection in nursing and the helping professions: a user’s guide.* Basingstoke: Palgrave Macmillan.

Appendix



```python
import requests
from bs4 import BeautifulSoup
import json

url = 'https://www.careerjet.com.mt/data-science-jobs.html'  # Use the correct URL here

# Send a GET request to the page
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# List to hold all the job data
jobs = []

# Find all job listings on the page
for job in soup.find_all('article', class_='job clicky'):
    # Try to get the job title from the <a> tag with the title attribute
    title_elem = job.find('a', title=True)
    if title_elem:
        title = title_elem.get('title').strip()
    else:
        title = "N/A"

    # Filter for "Data Scientist" in the job title or description
    if "data scientist" not in title.lower():
        continue  # Skip jobs that are not related to Data Scientist

    # Extract the company
    company_elem = job.find('p', class_='company')
    company = company_elem.text.strip() if company_elem else "N/A"

    # Extract the location
    location_elem = job.find('ul', class_='location')
    location = location_elem.find('li').text.strip() if location_elem and location_elem.find('li') else "N/A"

    # Extract the salary
    salary_elem = job.find('ul', class_='salary')
    salary = salary_elem.find('li').text.strip() if salary_elem and salary_elem.find('li') else "N/A"

    # Extract the job description
    description_elem = job.find('div', class_='desc')
    description = description_elem.text.strip() if description_elem else "N/A"

    # Add the job to the list of jobs
    jobs.append({
        'title': title,
        'company': company,
        'location': location,
        'salary': salary,
        'description': description,
    })

# Save the job data to a JSON file
with open('data_scientist_jobs.json', 'w') as json_file:
    json.dump(jobs, json_file, indent=4)

print("Job data saved to 'data_scientist_jobs.json'.")
```

Figure 1 Python Code for Web Scraping Activity



Security Requirements Specification for a Fitness Tracker API

Overview

This document outlines the security requirements for an API enabling data exchange between a fitness tracker and a mobile phone. The API facilitates the transmission of health data, including metrics such as heart rate, step count, and sleep patterns. The data may also be synchronised with cloud services or third-party applications. To ensure security, the API must address risks such as unauthorised access, data breaches, and compliance violations.

Security Requirements

To protect the fitness tracker API from security threats, the following measures are required:

Authentication and Authorisation

Authentication and authorisation are essential to ensure that only authorised devices and users can access sensitive health data.

- Enforce mutual authentication during the pairing process, such as using secure tokens or QR codes.
- Implement OAuth 2.0 for secure integration with third-party applications.
- Apply role-based access control (RBAC) to manage permissions for data sharing and synchronisation.

Data Transmission Security

To prevent interception of sensitive data, secure transmission protocols must be employed.

- Use Bluetooth Secure Connections with AES-128 encryption for communication between the fitness tracker and mobile phone.
- Encrypt all API communications with HTTPS using TLS 1.2 or higher.
- For particularly sensitive data, apply application-layer encryption before transmission.

Input Validation and Sanitisation

All inputs to the API must be validated and sanitised to prevent injection attacks and other processing vulnerabilities.

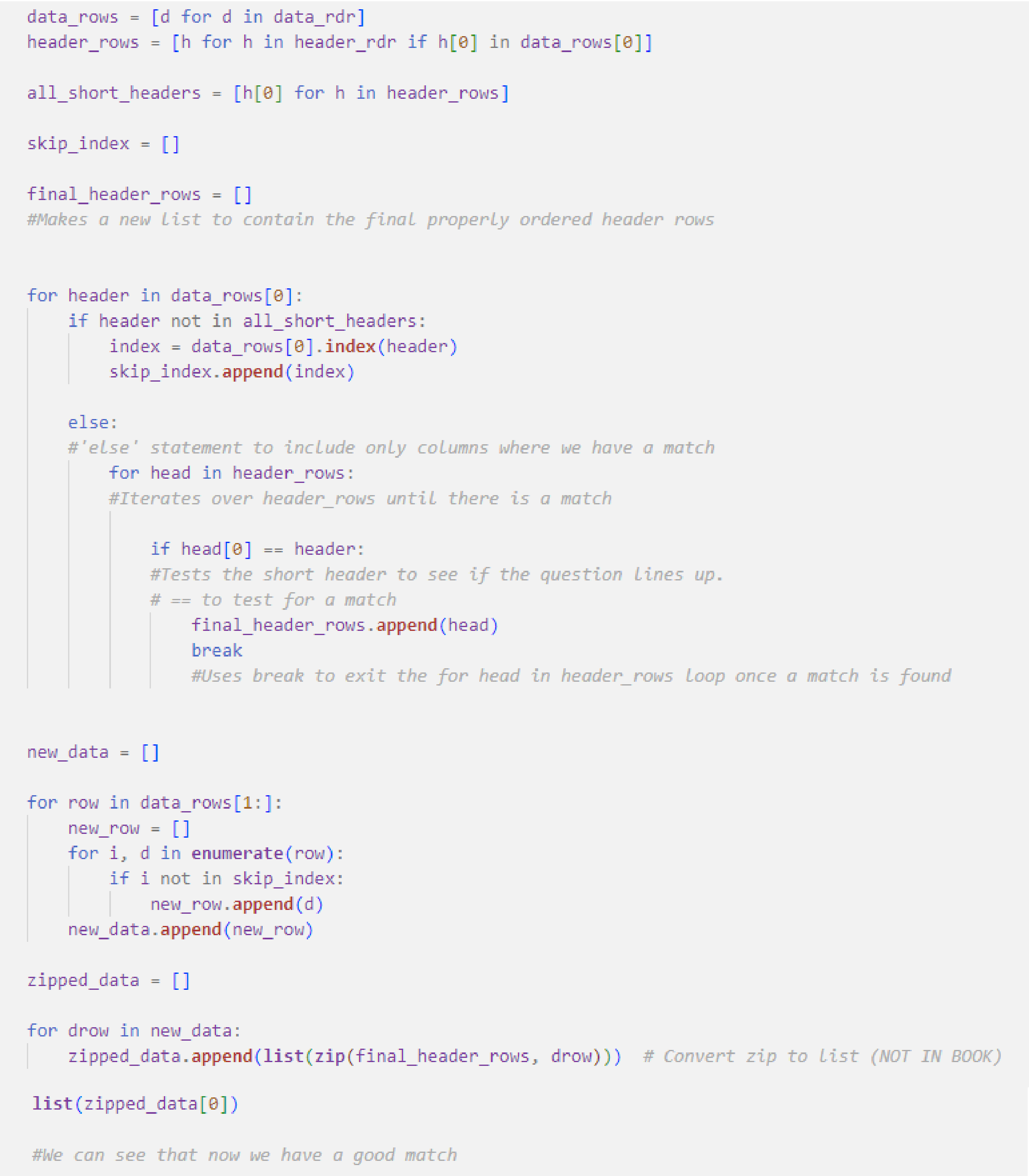
- Validate inputs against predefined schemas to ensure they meet expected formats.
- Reject oversized or malformed payloads to avoid buffer overflows and denial-of-service (DoS) attacks.
- Sanitise inputs for device identifiers, user data, and firmware updates to prevent code injection.

Format-Specific Security Measures

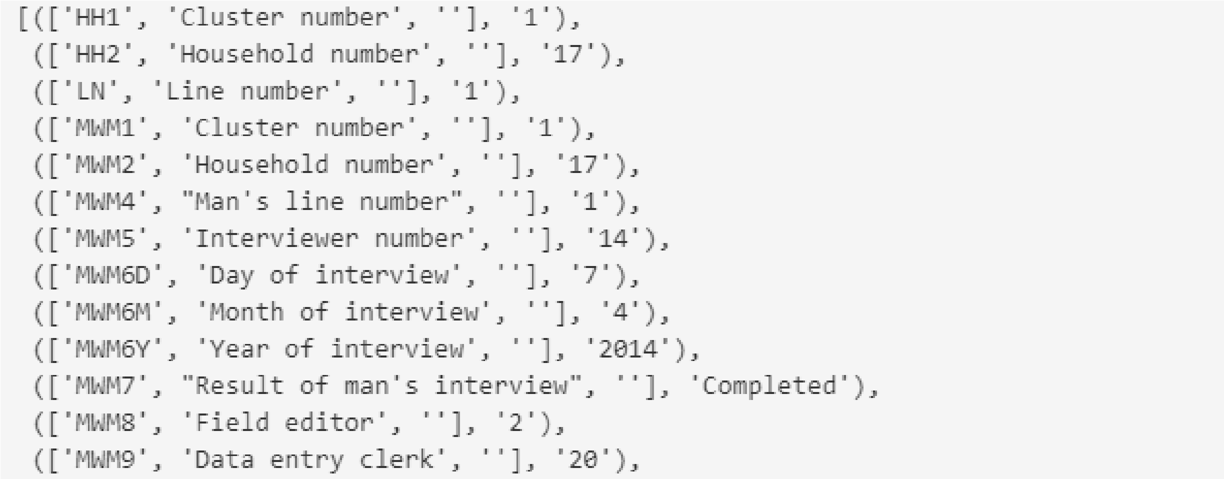
The API must ensure secure handling of data formats commonly used in communication and storage:

- JSON: Validate payloads against strict schemas and enforce size limits to prevent memory exhaustion or DoS attacks.
- XML: Disable external entity processing to prevent XML External Entity (XXE) attacks. Apply strict schema validation and size restrictions to ensure safety.
- SQL: Use parameterised queries or an ORM (e.g., SQLAlchemy) to prevent SQL injection. Limit database access permissions to authorised applications or processes.

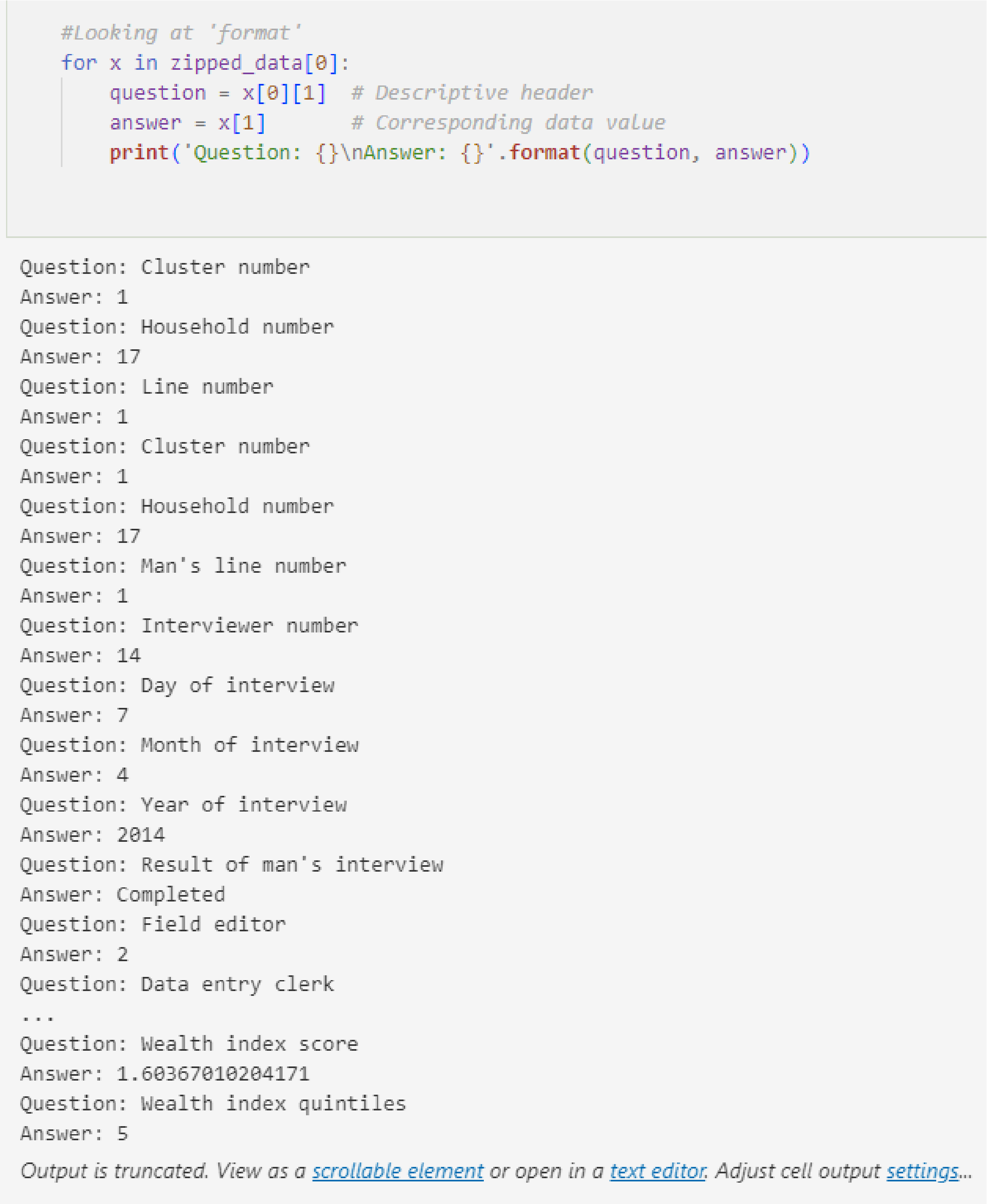
Figure 2 Snippets from API Security Requirements Task



```
data_rows = [d for d in data_rdr]
header_rows = [h for h in header_rdr if h[0] in data_rows[0]]

all_short_headers = [h[0] for h in header_rows]

skip_index = []

final_header_rows = []
#Makes a new list to contain the final properly ordered header rows


for header in data_rows[0]:
    if header not in all_short_headers:
        index = data_rows[0].index(header)
        skip_index.append(index)

    else:
    #'else' statement to include only columns where we have a match
        for head in header_rows:
        #Iterates over header_rows until there is a match

            if head[0] == header:
            #Tests the short header to see if the question lines up.
            # == to test for a match
                final_header_rows.append(head)
                break
                #Uses break to exit the for head in header_rows loop once a match is found


new_data = []

for row in data_rows[1:]:
    new_row = []
    for i, d in enumerate(row):
        if i not in skip_index:
            new_row.append(d)
    new_data.append(new_row)

zipped_data = []

for drow in new_data:
    zipped_data.append(list(zip(final_header_rows, drow)))  # Convert zip to list (NOT IN BOOK)

list(zipped_data[0])

#We can see that now we have a good match
```



```
[(['HH1', 'Cluster number', ''], '1'),
 (['HH2', 'Household number', ''], '17'),
 (['LN', 'Line number', ''], '1'),
 (['MWM1', 'Cluster number', ''], '1'),
 (['MWM2', 'Household number', ''], '17'),
 (['MWM4', "Man's line number", ''], '1'),
 (['MWM5', 'Interviewer number', ''], '14'),
 (['MWM6D', 'Day of interview', ''], '7'),
 (['MWM6M', 'Month of interview', ''], '4'),
 (['MWM6Y', 'Year of interview', ''], '2014'),
 (['MWM7', "Result of man's interview", ''], 'Completed'),
 (['MWM8', 'Field editor', ''], '2'),
 (['MWM9', 'Data entry clerk', ''], '20'),
```

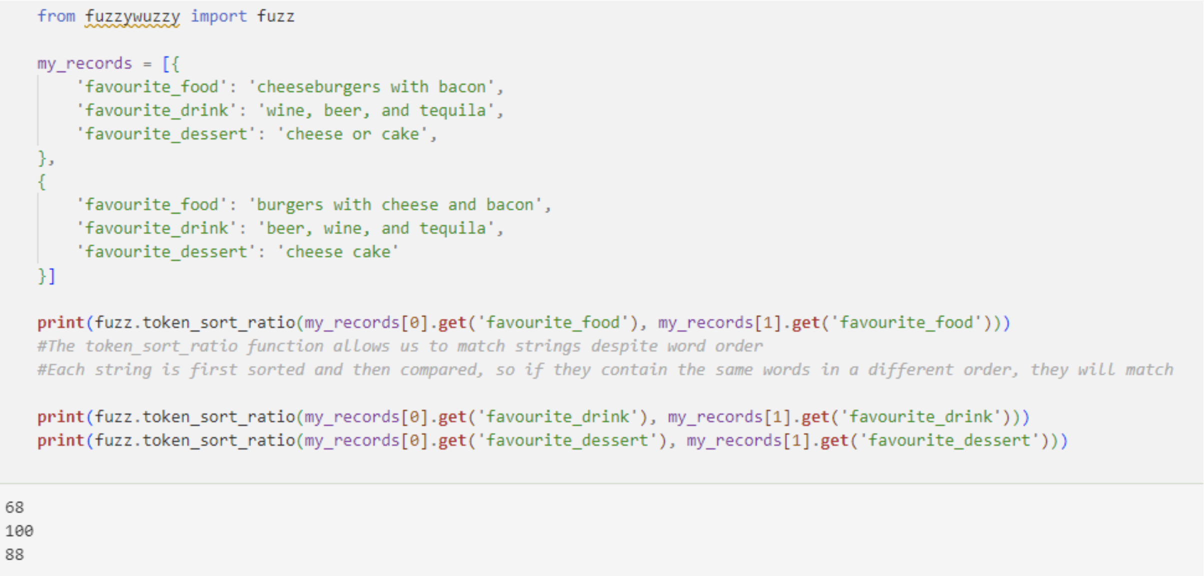


```python
#Looking at 'format'
for x in zipped_data[0]:
    question = x[0][1]  # Descriptive header
    answer = x[1]       # Corresponding data value
    print('Question: {}\nAnswer: {}'.format(question, answer))
```

```
Question: Cluster number
Answer: 1
Question: Household number
Answer: 17
Question: Line number
Answer: 1
Question: Cluster number
Answer: 1
Question: Household number
Answer: 17
Question: Man's line number
Answer: 1
Question: Interviewer number
Answer: 14
Question: Day of interview
Answer: 7
Question: Month of interview
Answer: 4
Question: Year of interview
Answer: 2014
Question: Result of man's interview
Answer: Completed
Question: Field editor
Answer: 2
Question: Data entry clerk
...
Question: Wealth index score
Answer: 1.60367010204171
Question: Wealth index quintiles
Answer: 5
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

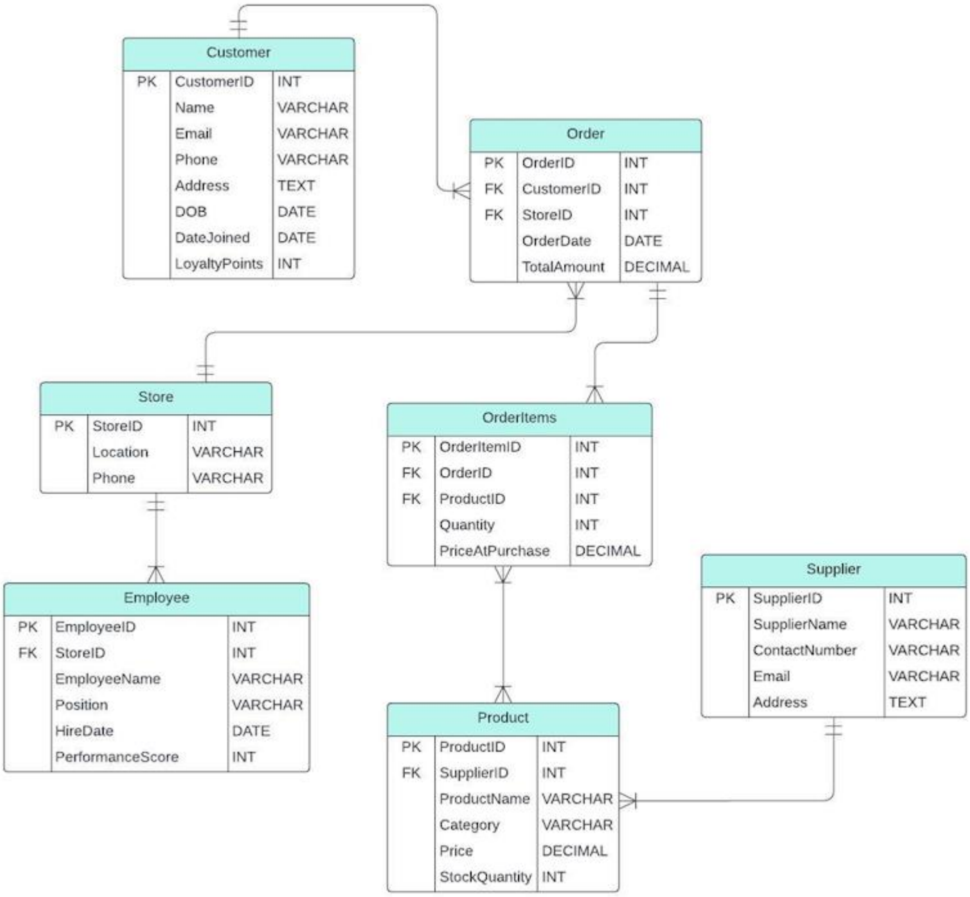
Figure 3 Snippets from Cleaning and Formatting Exercises in Jupyter Notebook, recreated from Kazil and Jarmul (2016)



```
from fuzzywuzzy import fuzz

my_records = [{
    'favourite_food': 'cheeseburgers with bacon',
    'favourite_drink': 'wine, beer, and tequila',
    'favourite_dessert': 'cheese or cake',
},
{
    'favourite_food': 'burgers with cheese and bacon',
    'favourite_drink': 'beer, wine, and tequila',
    'favourite_dessert': 'cheese cake'
}]

print(fuzz.token_sort_ratio(my_records[0].get('favourite_food'), my_records[1].get('favourite_food')))
#The token_sort_ratio function allows us to match strings despite word order
#Each string is first sorted and then compared, so if they contain the same words in a different order, they will match

print(fuzz.token_sort_ratio(my_records[0].get('favourite_drink'), my_records[1].get('favourite_drink')))
print(fuzz.token_sort_ratio(my_records[0].get('favourite_dessert'), my_records[1].get('favourite_dessert')))
```

```
68
100
88
```

Figure 4 Snippet of Python Code using FuzzyWuzzy in Jupyter Notebook, recreated from Kazil and Jarmul (2016)



Customer

PK	CustomerID	INT
	Name	VARCHAR
	Email	VARCHAR
	Phone	VARCHAR
	Address	TEXT
	DOB	DATE
	DateJoined	DATE
	LoyaltyPoints	INT

Order

PK	OrderID	INT
FK	CustomerID	INT
FK	StoreID	INT
	OrderDate	DATE
	TotalAmount	DECIMAL

Store

PK	StoreID	INT
	Location	VARCHAR
	Phone	VARCHAR

OrderItems

PK	OrderItemID	INT
FK	OrderID	INT
FK	ProductID	INT
	Quantity	INT
	PriceAtPurchase	DECIMAL

Employee

PK	EmployeeID	INT
FK	StoreID	INT
	EmployeeName	VARCHAR
	Position	VARCHAR
	HireDate	DATE
	PerformanceScore	INT

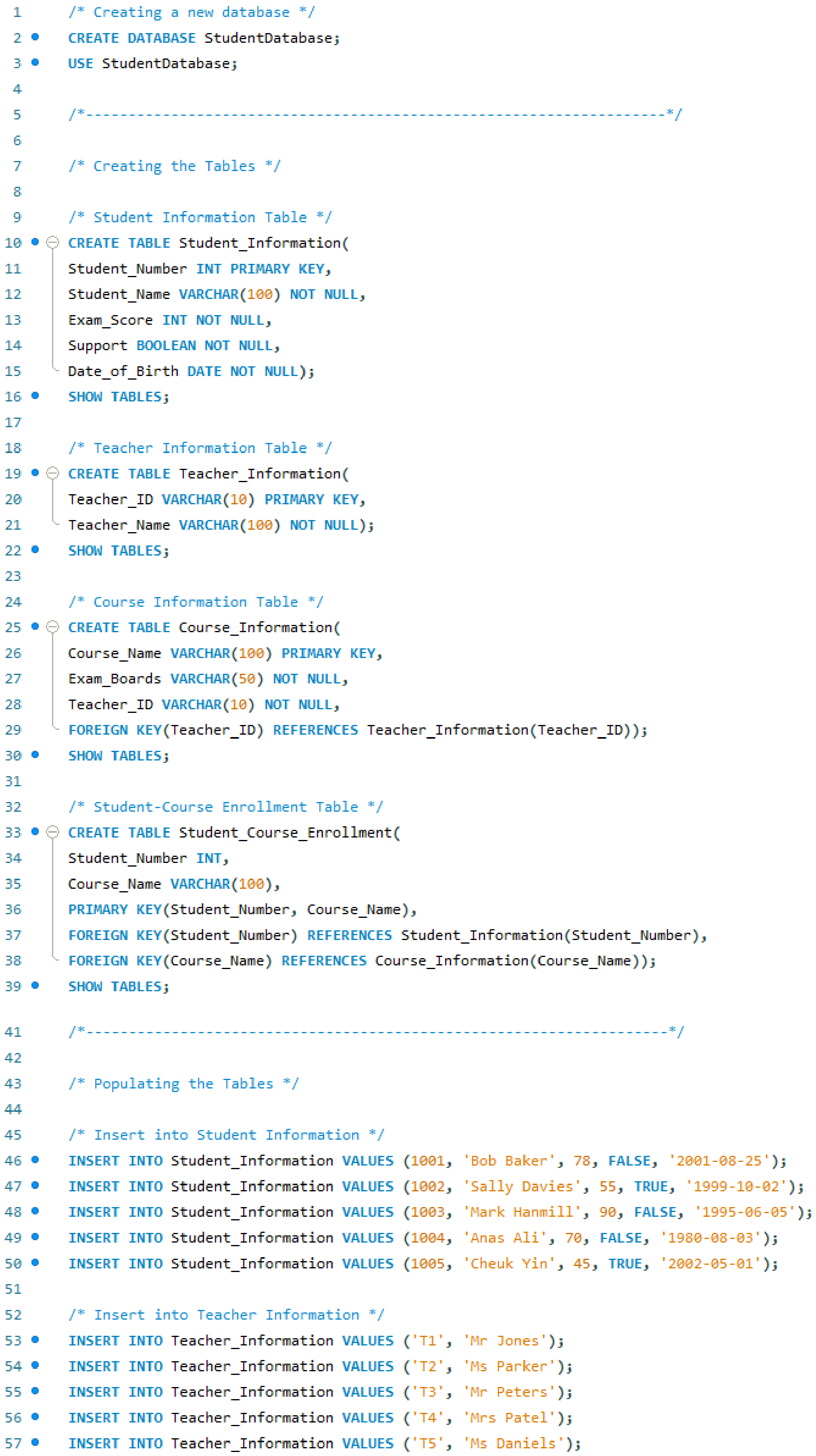
Supplier

PK	SupplierID	INT
	SupplierName	VARCHAR
	ContactNumber	VARCHAR
	Email	VARCHAR
	Address	TEXT

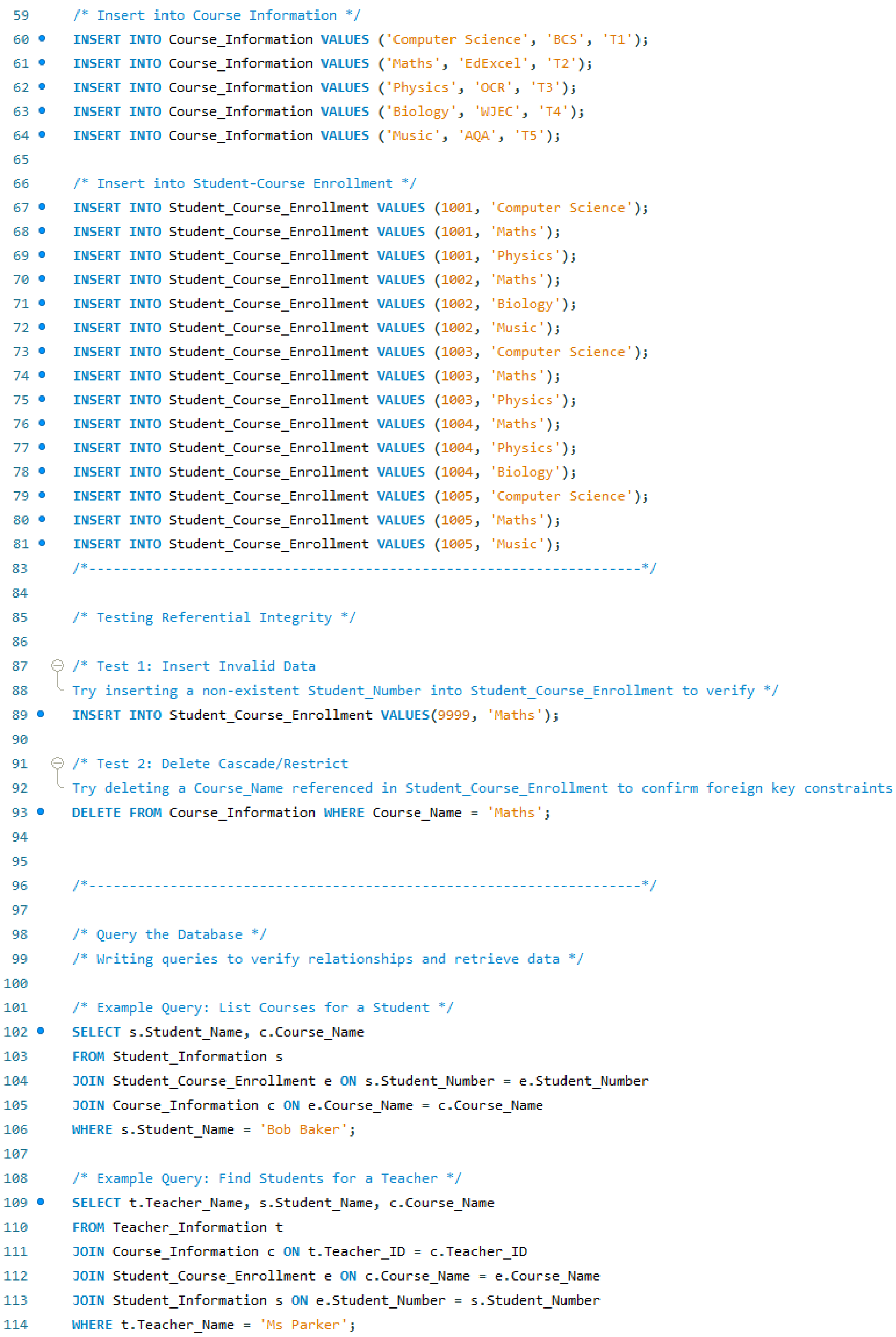
Product

PK	ProductID	INT
FK	SupplierID	INT
	ProductName	VARCHAR
	Category	VARCHAR
	Price	DECIMAL
	StockQuantity	INT

Figure 5 Entity Relationship Diagram (ERD) for Retail Store



```sql
/* Creating a new database */
CREATE DATABASE StudentDatabase;
USE StudentDatabase;

/*--------------------------------------------------------------------*/

/* Creating the Tables */

/* Student Information Table */
CREATE TABLE Student_Information(
Student_Number INT PRIMARY KEY,
Student_Name VARCHAR(100) NOT NULL,
Exam_Score INT NOT NULL,
Support BOOLEAN NOT NULL,
Date_of_Birth DATE NOT NULL);
SHOW TABLES;

/* Teacher Information Table */
CREATE TABLE Teacher_Information(
Teacher_ID VARCHAR(10) PRIMARY KEY,
Teacher_Name VARCHAR(100) NOT NULL);
SHOW TABLES;

/* Course Information Table */
CREATE TABLE Course_Information(
Course_Name VARCHAR(100) PRIMARY KEY,
Exam_Boards VARCHAR(50) NOT NULL,
Teacher_ID VARCHAR(10) NOT NULL,
FOREIGN KEY(Teacher_ID) REFERENCES Teacher_Information(Teacher_ID));
SHOW TABLES;

/* Student-Course Enrollment Table */
CREATE TABLE Student_Course_Enrollment(
Student_Number INT,
Course_Name VARCHAR(100),
PRIMARY KEY(Student_Number, Course_Name),
FOREIGN KEY(Student_Number) REFERENCES Student_Information(Student_Number),
FOREIGN KEY(Course_Name) REFERENCES Course_Information(Course_Name));
SHOW TABLES;

/*--------------------------------------------------------------------*/

/* Populating the Tables */

/* Insert into Student Information */
INSERT INTO Student_Information VALUES (1001, 'Bob Baker', 78, FALSE, '2001-08-25');
INSERT INTO Student_Information VALUES (1002, 'Sally Davies', 55, TRUE, '1999-10-02');
INSERT INTO Student_Information VALUES (1003, 'Mark Hanmill', 90, FALSE, '1995-06-05');
INSERT INTO Student_Information VALUES (1004, 'Anas Ali', 70, FALSE, '1980-08-03');
INSERT INTO Student_Information VALUES (1005, 'Cheuk Yin', 45, TRUE, '2002-05-01');

/* Insert into Teacher Information */
INSERT INTO Teacher_Information VALUES ('T1', 'Mr Jones');
INSERT INTO Teacher_Information VALUES ('T2', 'Ms Parker');
INSERT INTO Teacher_Information VALUES ('T3', 'Mr Peters');
INSERT INTO Teacher_Information VALUES ('T4', 'Mrs Patel');
INSERT INTO Teacher_Information VALUES ('T5', 'Ms Daniels');
```



```sql
/* Insert into Course Information */
INSERT INTO Course_Information VALUES ('Computer Science', 'BCS', 'T1');
INSERT INTO Course_Information VALUES ('Maths', 'EdExcel', 'T2');
INSERT INTO Course_Information VALUES ('Physics', 'OCR', 'T3');
INSERT INTO Course_Information VALUES ('Biology', 'WJEC', 'T4');
INSERT INTO Course_Information VALUES ('Music', 'AQA', 'T5');

/* Insert into Student-Course Enrollment */
INSERT INTO Student_Course_Enrollment VALUES (1001, 'Computer Science');
INSERT INTO Student_Course_Enrollment VALUES (1001, 'Maths');
INSERT INTO Student_Course_Enrollment VALUES (1001, 'Physics');
INSERT INTO Student_Course_Enrollment VALUES (1002, 'Maths');
INSERT INTO Student_Course_Enrollment VALUES (1002, 'Biology');
INSERT INTO Student_Course_Enrollment VALUES (1002, 'Music');
INSERT INTO Student_Course_Enrollment VALUES (1003, 'Computer Science');
INSERT INTO Student_Course_Enrollment VALUES (1003, 'Maths');
INSERT INTO Student_Course_Enrollment VALUES (1003, 'Physics');
INSERT INTO Student_Course_Enrollment VALUES (1004, 'Maths');
INSERT INTO Student_Course_Enrollment VALUES (1004, 'Physics');
INSERT INTO Student_Course_Enrollment VALUES (1004, 'Biology');
INSERT INTO Student_Course_Enrollment VALUES (1005, 'Computer Science');
INSERT INTO Student_Course_Enrollment VALUES (1005, 'Maths');
INSERT INTO Student_Course_Enrollment VALUES (1005, 'Music');
/*--------------------------------------------------------------------*/

/* Testing Referential Integrity */

/* Test 1: Insert Invalid Data
Try inserting a non-existent Student_Number into Student_Course_Enrollment to verify */
INSERT INTO Student_Course_Enrollment VALUES(9999, 'Maths');

/* Test 2: Delete Cascade/Restrict
Try deleting a Course_Name referenced in Student_Course_Enrollment to confirm foreign key constraints
DELETE FROM Course_Information WHERE Course_Name = 'Maths';


/*--------------------------------------------------------------------*/

/* Query the Database */
/* Writing queries to verify relationships and retrieve data */

/* Example Query: List Courses for a Student */
SELECT s.Student_Name, c.Course_Name
FROM Student_Information s
JOIN Student_Course_Enrollment e ON s.Student_Number = e.Student_Number
JOIN Course_Information c ON e.Course_Name = c.Course_Name
WHERE s.Student_Name = 'Bob Baker';

/* Example Query: Find Students for a Teacher */
SELECT t.Teacher_Name, s.Student_Name, c.Course_Name
FROM Teacher_Information t
JOIN Course_Information c ON t.Teacher_ID = c.Teacher_ID
JOIN Student_Course_Enrollment e ON c.Course_Name = e.Course_Name
JOIN Student_Information s ON e.Student_Number = s.Student_Number
WHERE t.Teacher_Name = 'Ms Parker';
```

Figure 6 Implementing and Testing a Normalised Dataset in MySQL Workbench

Table	Attribute	Data Type	Reason
Customer	CustomerID	INT	A unique integer identifier for each customer, efficient for indexing and queries.
	Name	VARCHAR	Flexible text format to store customer names of varying lengths (e.g., "John Doe").
	Email	VARCHAR	Emails are alphanumeric and variable-length; VARCHAR ensures storage efficiency.
	Phone	VARCHAR	Phone numbers often include formatting characters (e.g., +1, -), making VARCHAR suitable.
	Address	TEXT	Addresses can be lengthy and vary significantly, so TEXT provides sufficient flexibility.
	DOB	DATE	A specific date format is needed to store the customer's birth date.
	DateJoined	DATE	Captures the date the customer joined the loyalty program for tracking membership history.
	LoyaltyPoints	INT	An integer value to store points earned by customers in the loyalty program.
Order	OrderID	INT	Unique identifier for each order, optimised for indexing and relational links.
	CustomerID	INT	Foreign key linking to the Customer table. Matches the data type of CustomerID.
	StoreID	INT	Foreign key linking to the Store table. Matches the data type of StoreID.
	OrderDate	DATE	Tracks the specific date the order was placed.
	TotalAmount	DECIMAL	Stores the total cost of the order, including decimals for accuracy in monetary values.
Store	StoreID	INT	Unique identifier for each store, suitable for indexing.
	Location	VARCHAR	Text format to store store locations (e.g., city names or addresses).
	Phone	VARCHAR	Allows storage of phone numbers with varying formats.
OrderItems	OrderItemID	INT	Unique identifier for each order item, optimised for indexing.
	OrderID	INT	Foreign key linking to the Order table. Matches the data type of OrderID.
	ProductID	INT	Foreign key linking to the Product table. Matches the data type of ProductID.
	Quantity	INT	Integer to store the number of items purchased.
	PriceAtPurchase	DECIMAL	Captures the price of the product at the time of purchase, with decimal precision.
Employee	EmployeeID	INT	Unique identifier for each employee, efficient for indexing and queries.
	StoreID	INT	Foreign key linking to the Store table. Matches the data type of StoreID.
	EmployeeName	VARCHAR	Stores employee names of varying lengths.
	Position	VARCHAR	Tracks employee job titles (e.g., "Manager").
	HireDate	DATE	Tracks when the employee was hired.
	PerformanceScore	INT	Integer used for performance tracking or evaluations.
Product	ProductID	INT	Unique identifier for each product, optimised for indexing.
	SupplierID	INT	Foreign key linking to the Supplier table. Matches the data type of SupplierID.
	ProductName	VARCHAR	Stores product names, allowing flexibility for different lengths.
	Category	VARCHAR	Tracks product categories (e.g., "Electronics").
	Price	DECIMAL	Stores product prices with decimal precision for accuracy.
	StockQuantity	INT	Tracks the quantity of the product in stock.
Supplier	SupplierID	INT	Unique identifier for each supplier.
	SupplierName	VARCHAR	Stores supplier names with varying lengths.
	ContactNumber	VARCHAR	Allows flexibility for phone numbers with varying formats.
	Email	VARCHAR	Suitable for storing email addresses.
	Address	TEXT	Provides flexibility for lengthy addresses.

Figure 7 The Tables, Attributes, and Data Types for the Database Design