

3-15-2023

## Teaching Case: Using Python and AWS for NoSQL in a BI Course

Michel Mitri

James Madison University, mitrimx@jmu.edu

Follow this and additional works at: <https://aisel.aisnet.org/jise>

---

### Recommended Citation

Mitri, Michel (2023) "Teaching Case: Using Python and AWS for NoSQL in a BI Course," *Journal of Information Systems Education*: Vol. 34 : Iss. 1 , 41-48.

Available at: <https://aisel.aisnet.org/jise/vol34/iss1/4>

This material is brought to you by the AIS Affiliated Journals at AIS Electronic Library (AISeL). It has been accepted for inclusion in Journal of Information Systems Education by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact [elibrary@aisnet.org](mailto:elibrary@aisnet.org).

*Teaching Case*  
**Using Python and AWS for NoSQL in a BI Course**

Michel Mitri

**Recommended Citation:** Mitri, M. (2023). Teaching Case: Using Python and AWS for NoSQL in a BI Course. *Journal of Information Systems Education*, 34(1), 41-48.

**Article Link:** <https://jise.org/Volume34/n1/JISE2023v34n1pp41-48.html>

Received: July 19, 2021  
Revised: December 2, 2021  
Accepted: May 19, 2022  
Published: March 15, 2023

Find archived papers, submission instructions, terms of use, and much more at the JISE website:  
<https://jise.org>

ISSN: 2574-3872 (Online) 1055-3096 (Print)

---

# **Teaching Case**

## **Using Python and AWS for NoSQL in a BI Course**

**Michel Mitri**

Department of Computer Information Systems and Business Analytics  
James Madison University  
Harrisonburg, VA 22807, USA  
[mitrimx@jmu.edu](mailto:mitrimx@jmu.edu)

### **ABSTRACT**

This article presents a multi-stage guided technical project coding Python scripts for utilizing Amazon Web Services (AWS) to work with a document-store database called DynamoDB. Students doing this project should have taken an introductory programming class (ideally in Python) and a database class to have experience with Python coding and database manipulation/querying in a relational environment. Students learn new data formats (Python dictionaries, JSON text data, key-value storage structures) and learn how to transform data from one format to another. They also gain experience with data visualization. The project was first carried out in a business intelligence (BI) course during Spring 2020 semester in the midst of COVID and included video tutorials. Since then, it has been refined and used each semester the BI course is taught.

**Keywords:** Programming, NoSQL, Business intelligence, Cloud computing, Active learning, Flipped classroom

### **1. INTRODUCTION**

This article presents a guided technical project for a senior-level business intelligence (BI) course in an undergraduate computer information systems and business analytics (CIS&BSAN) program. The project involves the following elements:

- 1) Python programming, including the use of dictionary structures and imported libraries
- 2) JavaScript Object Notation (JSON) data containing product bill-of-material (BOM) data from Microsoft Adventure Work's database (2017)
- 3) Two Amazon Web Service (AWS) services: the DynamoDB document-store database (Amazon DynamoDB Documentation, 2021) and Identity Access Management (IAM) for accounts, security credentials, and permissions (AWS Identity and Access Management Documentation, 2021). DynamoDB gives the "not only SQL" (NoSQL) experience, and IAM gives security exposure to the students.
- 4) Working with the Data-Driven Documents (2020) visualization library for displaying BOM hierarchies in interactive tree visualizations
- 5) Guided exercises via video tutorials

The BI course requires two prerequisites: our introductory programming course (using Python) and our database course (so students have a strong grounding in querying relational databases). Other topics covered in this course include advanced database queries, data visualization, data integration, online analytical processing (OLAP), data mining, and natural language processing. Students taking the course are typically CIS majors or CIS minors, and some of them also enroll in the business analytics program.

This exercise was developed and refined over three semesters. COVID hit during the third semester in the spring of 2020, which prompted an adjustment to the exercise approach. Previously, the course was held in a computer lab, with lectures and many guided face-to-face exercises. COVID forced all courses to go entirely online. The instructor created video lectures for the exercises, which the students followed during an asynchronous class activity. The course included synchronous and asynchronous activities, and the exercise described here was done mostly asynchronously via video lectures and associated PowerPoint documents.

### **2. OVERVIEW OF THE NOSQL EXERCISE**

Figure 1 shows an overview of the exercise described in this article. The exercise involves these tasks:

- 1) Work with JSON bill-of-materials data
- 2) Set up an AWS account and get credentials
- 3) Using AWS with Python
- 4) Create an AWS DynamoDB database table
- 5) Read JSON data from a file and populate the table
- 6) Do queries on the database
- 7) Based on the query, generate JSON data to send to a D3 tree visualization
- 8) Display the visualization

These tasks give students an active-learning experience with many activities that IS and data science professionals in the current and future technological era, including cloud computing, coding, data analysis and transformation, and visualization. They are described in the following sections.

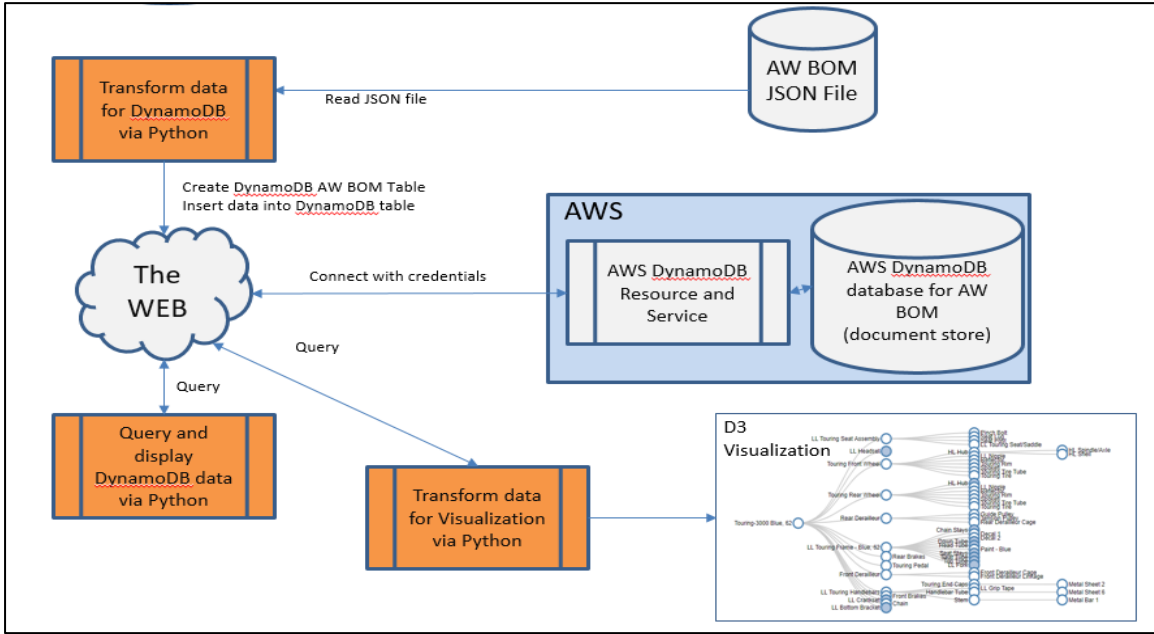


Figure 1. Overview of Python NoSQL Assignment

### 3. WORK WITH JSON BOM DATA

Students are given a text file containing JSON formatted data. This data comes from Microsoft’s AdventureWorks Sample Databases (2017) and pertains to products and their bill of materials (BOM). Adventure Works is a fictional bicycle manufacturing company and is a handy business case used throughout the BI course (Mitri, 2015).

Microsoft’s AdventureWorks Sample Databases (2017) consists of 71 tables grouped into five schemas related to the company’s business model: Sales, Purchasing, Production, Human Resources, and Person. The database contains data from almost 20,000 people (employees, customers, store contacts, vendor contacts, and general contacts). It also contains data

from over 31,000 sales transactions to customers and over 4000 purchasing transactions from suppliers. The 2016 version of the database pertains to the years 2010-2014, but this can easily be modified by updating the date fields and making the database appear more current.

This database is very comprehensive compared with the data volume in a typical textbook’s sample database. There are also several advanced data types, including bitmapped product photographs, XML documents, cryptographic-hashed passwords for security, and hierarchy id fields for representing hierarchical data relationships. The database provides a rich treasure trove of pedagogical opportunities capable of benefiting many aspects of information technology and business education (Mitri, 2015).

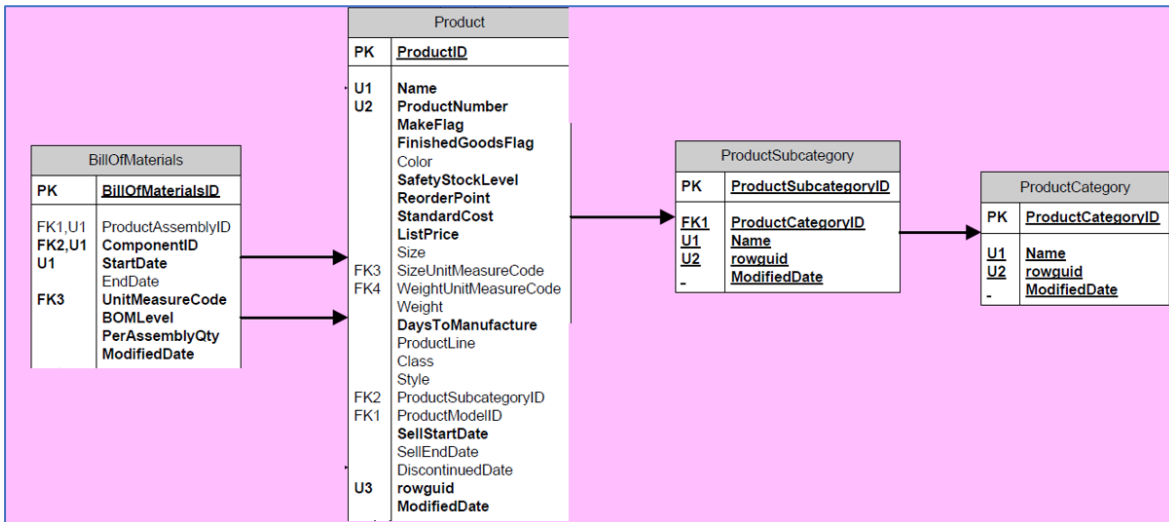


Figure 2. Adventure Works Product and Bill of Materials Tables

```

119- {
120-   "ProductID": "984",
121-   "ProductName": "Mountain-500 Silver, 40",
122-   "Model": "Mountain-500",
123-   "Description": "Suitable for any type of riding, on or off-road. Fits any budget. Smooth-shifting with a comfortable ride.",
124-   "Category": "Bikes",
125-   "Subcategory": "Mountain Bikes",
126-   "ProductPhotoID": "1",
127-   "ThumbnailPhotoFileName": "no_image_available_small.gif",
128-   "LargePhotoFileName": "no_image_available_large.gif",
129-   "BOM": [
130-     {
131-       "ComponentID": "514",
132-       "ComponentName": "LL Mountain Seat Assembly",
133-       "BOM": [
134-         {
135-           "ComponentID": "497",
136-           "ComponentName": "Pinch Bolt"
137-         },
138-         {
139-           "ComponentID": "528",
140-           "ComponentName": "Seat Lug"
141-         },
142-         {
143-           "ComponentID": "530",
144-           "ComponentName": "Seat Post"
145-         },
146-         {
147-           "ComponentID": "908",
148-           "ComponentName": "LL Mountain Seat/Saddle"
149-         }
150-       ]
151-     },
152-     {
153-       "ComponentID": "805",
154-       "ComponentName": "LL Headset"
155-     },
156-     {
157-       "ComponentID": "808",
158-       "ComponentName": "LL Mountain Handlebars"
159-     },
160-     {
161-       "ComponentID": "815",
162-       "ComponentName": "LL Mountain Front Wheel"
163-     },
164-     {
165-       "ComponentID": "823",
166-       "ComponentName": "LL Mountain Rear Wheel"
167-     }
168-   ]
169- }

```

Figure 3. JSON Format of Adventure Works Product and Bill of Materials Data

```

import json

# this function recurses through the BOM hierarchy
def recurse_BOM(id, bom, level):
    for component in bom:
        #print name
        for i in range(0,level):
            print('\t', end='')
        print(component['ComponentID'], component['ComponentName'])

        try:
            # make recursive call
            recurse_BOM(component['ComponentID'], component['BOM'], level+1)
        except KeyError:
            print(end='')

# select the product id to display
prodid = '984'

#open the json file
with open('AW BOM.json') as json_file:
    #load the json data from the file
    data = json.load(json_file)
    #loop through the data
    for p in data:
        if p['ProductID'] == prodid:
            #print information about desired product
            print(p['ProductID'], p['ProductName'], p['Category'],
                  p['Subcategory'], p['ListPrice'])
            recurse_BOM(p['ProductID'], p['BOM'], 1)

```

```

984 Mountain-500 Silver, 40 Bikes Mountain Bikes 564.99
514 LL Mountain Seat Assembly
497 Pinch Bolt
528 Seat Lug
530 Seat Post
908 LL Mountain Seat/Saddle
805 LL Headset
1 Adjustable Race
4 Headset Ball Bearings
323 Crown Race
402 Keyed Washer
459 Lock Nut 19
462 Lower Head Race
808 LL Mountain Handlebars
328 Mountain End Caps
482 Metal Sheet 2
356 LL Grip Tape
398 Handlebar Tube
487 Metal Sheet 6
529 Stem
477 Metal Bar 1
815 LL Mountain Front Wheel
400 LL Hub
523 LL Spindle/Axle
525 LL Shell
490 LL Nipple
506 Reflector
507 LL Mountain Rim
527 Spokes
921 Mountain Tire Tube
928 LL Mountain Tire
823 LL Mountain Rear Wheel
400 LL Hub
523 LL Spindle/Axle
525 LL Shell
490 LL Nipple
506 Reflector
507 LL Mountain Rim
527 Spokes
921 Mountain Tire Tube

```

Figure 4. Python Recursion Code and Output Displaying Hierarchical BOM Data

For the NoSQL project, data was extracted from the tables depicted in Figure 2. Students will be familiar with this data because previous assignments involve analyzing and querying the data and metadata in the Adventure Works relational database.

The JSON representation of data extracted from these tables is shown in Figure 3. Note the hierarchical layout of the BOM data. A bicycle comprises components, some of which have subcomponents, etc.

The hierarchical nature of this data requires recursion for processing the data. Recursion is not covered in the introductory programming class, so students gain this experience as part of the project. Recursion is a fascinating phenomenon in nature. For example, the recursive Fibonacci function can be seen in spiral galaxies and snail shell shapes. Discussion of recursion helps students appreciate the connection between natural phenomenon and mathematical/computational processing.

Students write Python code to navigate the JSON data when processing the bill of materials data for a product. This allows them to produce a hierarchical display of a product, like the bicycle data in Figure 4.

This portion of the project instills knowledge and skills pertaining to understanding the correspondence between JSON data and Python dictionary structures. The key-value nature of each data format is evident from both Figure 3 and Figure 4. Python's json package includes functionality for transforming data from JSON to Python dictionaries or lists and vice versa.

As you can see from the figure, students also learn about using recursion to process a hierarchy. Essentially, students are producing and displaying a tree data structure with this code.

#### 4. SET UP AWS ACCOUNT AND GET CREDENTIALS

Most of our CIS students will already have experience creating AWS accounts and using Identity Access Management (IAM) for creating, granting permissions, and using accounts. In other classes of the curriculum, students use AWS for S3 (file storage), EC2 (reserving and using computing instances), and Elastic Beanstalk (coordinating EC2 usage). Some students, especially CIS minors, may not have this experience. Most students would not have experience setting up DynamoDB access with IAM, which is a requirement for the project.

Using IAM, students create a permission policy to allow DynamoDB access. They create an IAM user for accessing DynamoDB users. For security reasons, the root AWS user mustn't have direct access to services, which is emphasized throughout the curriculum. The students attach the permission policy to the user and then generate an AWS and secret access keys. Then they set up the configuration files on their computer, using files provided by the instructor. When students complete this task, they are ready to start working with the DynamoDB service. By going through these steps, students deepen their understanding of setting up proper authorizations in a cloud environment.

#### 5. CREATE A DYNAMODB DATABASE TABLE

AWS's DynamoDB is a document-store database management system. It is similar to MongoDB, an open-source version of document-store technology. To use DynamoDB, or any other

AWS service, in Python, students need to familiarize themselves with the AWS-provided Python package **boto3**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "dynamodb:*",
      "Resource": "*"
    }
  ]
}
```

Figure 5. AWS Permission Policy for DynamoDB in JSON Format

```
# Create a table
table = resource.create_table(
    TableName='AWProductBOM',
    KeySchema=[
        {
            'AttributeName': 'ProductID',
            'KeyType': 'HASH'
        },
        {
            'AttributeName': 'ProductName',
            'KeyType': 'RANGE'
        }
    ],
    AttributeDefinitions=[
        {
            'AttributeName': 'ProductID',
            'AttributeType': 'S'
        },
        {
            'AttributeName': 'ProductName',
            'AttributeType': 'S'
        }
    ],
    ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
)
```

Figure 6. DynamoDB Table Definition in JSON Format

The boto3 package provides the necessary interface between a Python app and the services accessible through AWS. This package includes two important modules, **client** and **resource**. While both are useful, the resource module is more important for this project. Through the boto3.resource module, a Python coder can generate a DynamoDB resource, create a table (the DynamoDB storage structure), and then perform data manipulation on the items in that table.

The principle behind document-store databases such as DynamoDB involves the idea of key-value pairs (Hoffer et al., 2016), which also comes up in topics like Python’s dictionary structures and JSON notation. The key-value idea is revisited throughout the exercise because (a) students use Python to read JSON data and process them in Python dictionary structures, (b) students convert dictionary data into JSON for various purposes, and (c) students make extensive use of the document-store database structure for data population and querying.

Students will be very familiar with the notion of primary keys, given their experience with relational databases in the prerequisite database class. But what is new for the students is the much lower degree of explicit metadata in the design of a document-store database. In DynamoDB, the only metadata specifications are for the primary key, as shown in Figure 6. In this case, the key involves two attributes (ProductID and ProductName). DynamoDB allows composite primary keys, where one attribute serves as the “hash” and the other as the “range.” Note that data types for both attributes are specified. Both are specified to be strings (‘S’ for the AttributeType) in the Attribute Definition.

These are the only specifications of metadata in the table definition. Each item in a table can contain any other data, and there is no predefined metadata for any of this data. This is an important characteristic of the big data tendency toward “schema on read” instead of “schema on write.” The relational database architecture specifies metadata when the database is designed. But in the big data framework, data is often just thrown into a “data lake,” and metadata is specified based on the need of the moment. The document-store framework of DynamoDB is an example of this philosophy of minimized up-front metadata specification.

## 6. POPULATE DYNAMODB TABLE FROM JSON DATA

Section 3 describes the Python process of inputting from JSON data to produce a Python list of objects, each of which is a dictionary of data for a product in Adventure Works product inventory, as shown in Figure 3. Students take the data from this list and populate the DynamoDB document-store table in this project segment. Figure 7 shows Python code for generating DynamoDB items (records) from the JSON data read in the code of Figure 4.

Note the `put_item` method, which is equivalent to a SQL insert statement. Here, the `Item` argument is a dictionary depicting the attribute name and value to place in the item inserted into the table. This is one of the operations of the DynamoDB Table class, an instance of which was created from the code in Figure 6. Others are `get_item`, `update_item`, and `delete_item`.

```
#open the json file
with open('AW BOM.json') as json_file:
    #load the json data from the file
    data = json.load(json_file)
    #loop through the data
    for p in data:
        #print information about a product
        print('Product ID: ' + p['ProductID'])
        print('Product Name: ' + p['ProductName'])
        print('Category: ' + p['Category'])
        print('')

        #populate DynamoDB table
        table.put_item(
            Item={
                'ProductID': p['ProductID'],
                'ProductName': p['ProductName'],
                'Category': p['Category'],
                'Subcategory': p['Subcategory'],
                'BOM': p['BOM'],
            }
        )
    )
```

Figure 7. Populating the DynamoDB Table from JSON Data Input

## 7. PERFORM QUERIES OF DYNAMODB DATA

After creating and populating the DynamoDB table, students learn how to perform queries on the data and display only selected items and attributes of this data. This is analogous to SQL’s SELECT statement operations. It is a useful pedagogical technique to identify the analogies between data query approaches in different data structures. For example, XML data structures can be navigated using a query language called XPath, which includes operations analogous to the SELECT operations in relational databases (Mitri, 2015). The same analogy can be found with document-store databases, as described here.

DynamoDB includes two approaches for selectively querying data in its databases. The first, via a Table method called **query**, is for choosing which items to download from the cloud service. This selection can only be made with regard to the primary key. For example, you can select only those records within a range of key values to download. In the case of a very large database (e.g. several gigabytes or terabytes), a local application will need to focus on a subset of records. The data for this project (from Adventure Works product data tables) is relatively small, comprising less than one megabyte, so downloading the entire data set is easy to do.

The second Table method is called **scan**, and this allows much more filtering selectivity based on the values of whichever attributes are desired. When performing a scan, you can include a **filter expression** (analogous to the WHERE clause in a SQL SELECT statement) and a **projection expression** (analogous to the fields specified in the SELECT clause). Figure 8 shows code for establishing a filter and projection when querying data for this project.

Three separate queries are shown, both involving a filter expression that selects certain items from the table and a projection expression that chooses selected attributes from each item returned. In this portion of the project, students are asked to perform a few other queries to gain practice filtering DynamoDB data.

```
# querying dynamodb database
import boto3
from boto3.dynamodb.conditions import Attr

def printresults():
    avprice, minprice, maxprice = 0, 100000, 0
    for item in items:
        print(item)
        avprice += item['ListPrice']
        minprice = min([item['ListPrice'], minprice])
        maxprice = max([item['ListPrice'], maxprice])

    avprice /= response['Count']
    print('\nBr items: ', response['Count'], 'Avg price: ', avprice, 'Min price: ', minprice, 'Max price: ', maxprice)
    print()

resource = boto3.resource('dynamodb')
table = resource.Table('AWProductBOM')

# logical operators & | ~
# comparison functions: eq, ne, gt, gte, lte, le, contains

# query 1
print('Query 1: Clothes between $20 and $50')
fe = Attr('Category').eq('Clothing') & Attr('ListPrice').gte(20) & Attr('ListPrice').lte(50)
pe = 'ProductID, ProductName, ListPrice, Subcategory'
response = table.scan(FilterExpression=fe, ProjectionExpression=pe)
items = response['Items']
printresults()

# query 2
print('Query 2: Inexpensive Components (< $50) with a BOM')
fe = Attr('Category').eq('Components') & Attr('BOM').exists() & Attr('ListPrice').lt(50)
pe = 'ProductID, ProductName, ListPrice, BOM'
response = table.scan(FilterExpression=fe, ProjectionExpression=pe)
items = response['Items']
printresults()

# query 3
print('Query 3: Shirts, Jerseys, and Vests')
fe = Attr('Subcategory').eq('Shirts') | Attr('Subcategory').eq('Vests') | Attr('Subcategory').eq('Jerseys')
pe = 'ProductID, ProductName, ListPrice, Subcategory'
response = table.scan(FilterExpression=fe, ProjectionExpression=pe)
items = response['Items']
printresults()
```

Figure 8. Python Code for Querying the DynamoDB Data

## 8. DISPLAY TREE VISUALIZATION OF A BILL OF MATERIALS HIERARCHY

Before this exercise, students in the BI class learned several data visualization tools and skills. They built dashboards using Tableau, Google Charts and Maps (JavaScript APIs), and the Python matplotlib API. For the final segment of this project, students take the data returned from a DynamoDB query and display it in another type of visualization called Data Driven Documents (d3). The d3 API is a sophisticated JavaScript library based on Scalable Vector Graphics (SVG). It is very powerful, allowing fine-grained control over all visualization elements. But there is a high learning curve, so students are not required to code d3. Instead, the instructor provided a template consisting of an HTML file with the necessary JavaScript code. Students were only required to convert data into the format needed for the visualization. The concept of data conversion for appropriate visualization is something students would have learned from previous exercises. Every visualization has an expected data format. This part of the exercise involves writing the Python code for the data transformation and simply copying and pasting results into the visualization template. The final output for a product's BOM looks like Figure 9.

## 9. USE OF VIDEO TUTORIALS FOR THE NOSQL PROJECT

This project was crafted mostly during Spring 2020, after the COVID pandemic started. Therefore, all classes took place

entirely online, as was common across higher education at that time. Although some BI coursework was done synchronously, this project occurred entirely asynchronously. The instructor created several video tutorials and lectures to guide students throughout the project. In previous semesters, the BI class was taught in a lab setting, so in-class exercises were done face-to-face in real-time. Creating the videos had the advantage of allowing students to follow along at their own pace. Still, they prevented the real-time interaction between teacher and student, and amongst the students themselves, that comes in a face-to-face format where everyone is in the same room. So, there were advantages such as *flipping the classroom* (Mok, 2014) and students can re-watch videos as often as necessary. But this approach also had disadvantages (e.g., less student interaction), but overall, the students saw it as a positive experience, as described in the next section. The video lectures are available on YouTube via this playlist: <https://www.youtube.com/playlist?list=PLfs32wBMG5xAvWdtO88oCyiRIVGOLHMDX>

## 10. STUDENT FEEDBACK

Some of the assignments in the BI course involved online discussions. The final discussion addressed this broad question: "What did I learn in this class?" Students commented on several topics, and the NoSQL experience was a popular item of discussion



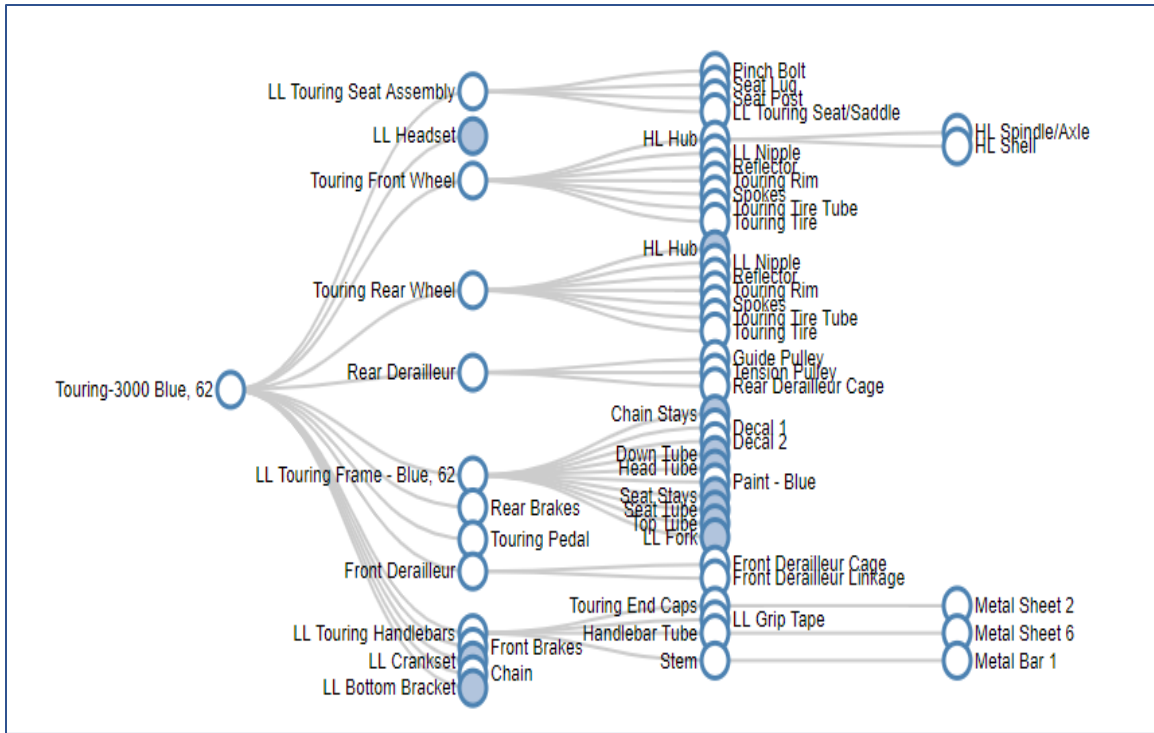



Figure 9. D3 Visualization of a Bicycle's Bill of Materials Hierarchy

This is a graded discussion: 50 points possible due May 7, 2020

 Final Discussion - What I learned about ... Apr 29, 2020 at 7:48pm

For this discussion post, I want you to write 500-600 words (which is approximately one single-spaced page in Word) describing something you learned in CIS 463 this semester. Your discussion theme can be focused on a particular topic or skill, or it can be more broad-based. If it's more focused I'll want you to dive deeply into the material of that topic. If it's broad-based I'll want you to be able to integrate and connect ideas from the different topics. Your discussion should make appropriate use of technical terminology from the lectures, slides, and exercises related to your chosen topic(s). I will judge based on quality of writing, correctness of terminology and assertions, and depth/breadth of coverage.

Figure 10. Discussion Assignment for BI Class

Below are some comments from students related to this exercise:

“One of the most insightful topics we covered in this class for me was NoSQL, both in theory and how it is applied with the exercises we did. I found NoSQL so interesting because it was a drastic change from what I had learned in the CIS major thus far.”

“I enjoyed learning about NoSQL as a growing trend both in tech overall and in big data, and how it is great for use with cloud technologies.”

“Another thing in python that I enjoyed learning was NoSQL. Again, it adds to my knowledge of python that was very surface level. Using NoSQL showed me that tables could be created in python as well as have them connected to AWS tools (DynamoDB).”

“One of the most interesting things I learned, though unfortunately through online lectures rather than in class, was using the DynamoDB on AWS. I have used several platforms on AWS in the past, such as EC2 and RDS, but I was unfamiliar with NoSQL and Dynamo. Thousands of businesses are hosting

everything in the cloud these days and learning how to utilize AWS' database resources will no doubt be a helpful skill in my future. It also helped me to polish up on JSON, which is becoming one of the most popular formats for data storage worldwide, especially in web-based resources (which I will be working with extensively for my job). Our exercise with BOM also made me realize how much easier it would be to explain unstructured data to a non-technical audience, as everyone can appreciate interactive models.”

“One topic I found interesting was our lectures on NOSQL and creating an AWS DynamoDB database in order to create multiple queries and D3 tree visualization using recursion.”

“The NoSQL lecture and exercises were personally my favorite topics in this course, and I hope I can carry the lessons learned and skills gained into my career.”

“Another valuable skill I learned is how to utilize Python to transform data. Properly formatted data improves data quality, making it easier for people to use, but also for use in programs. For example, in class we utilized clean data for visualizations and decision trees.”

“I think the information I learned about creating data visualizations and dashboards, as well as how to correctly analyze them, will be the most useful to me in my future career. As a management major, I could utilize this knowledge of data visualizations and dashboards for Human Resource Management to monitor recruiting, selection, and performance trends. I could also utilize this knowledge for supply chain management to identify any problems, improve clarity of data, and increase efficiency in the supply chain.”

## 11. CONCLUSION

The project described in this article gives students hands-on experience with several contemporary technologies. Students deepen their understanding a fluency in the Python programming language. They work with the most ubiquitous cloud services platform, AWS. They broaden their skill set with databases, adding the NoSQL approach to their existing knowledge of relational databases. And they gain some experience with using sophisticated data visualization tools and APIs.

Although the COVID pandemic presented significant challenges to teaching and learning, the exercises and tutorials of this project were a positive experience to students who took the class during that semester. By their own testimony, this experience gave them important skills highly relevant to their future careers in IT.

## 12. REFERENCES

- Amazon DynamoDB Documentation (2021). <https://docs.aws.amazon.com/dynamodb/index.html>
- AWS Identity and Access Management Documentation (2021). <https://docs.aws.amazon.com/iam/index.html>
- Data-Driven Documents (2020). <https://d3js.org/>
- Hoffer, J., Ramesh, V., & Topi, H. (2016). *Modern Database Management* (12<sup>th</sup> ed.). Pearson, Upper Saddle, NJ.
- Microsoft's AdventureWorks Sample Databases (2017), <https://github.com/microsoft/sql-server-samples/releases/tag/adventureworks>
- Mitri, M. (2015). Active Learning via a Sample Database: The Case of Microsoft's Adventure Works. *Journal of Information Systems Education*, 26(3), 177-186.
- Mok, H. N. (2014), Teaching Tip: The Flipped Classroom. *Journal of Information Systems Education*, 25(1), 7-11.

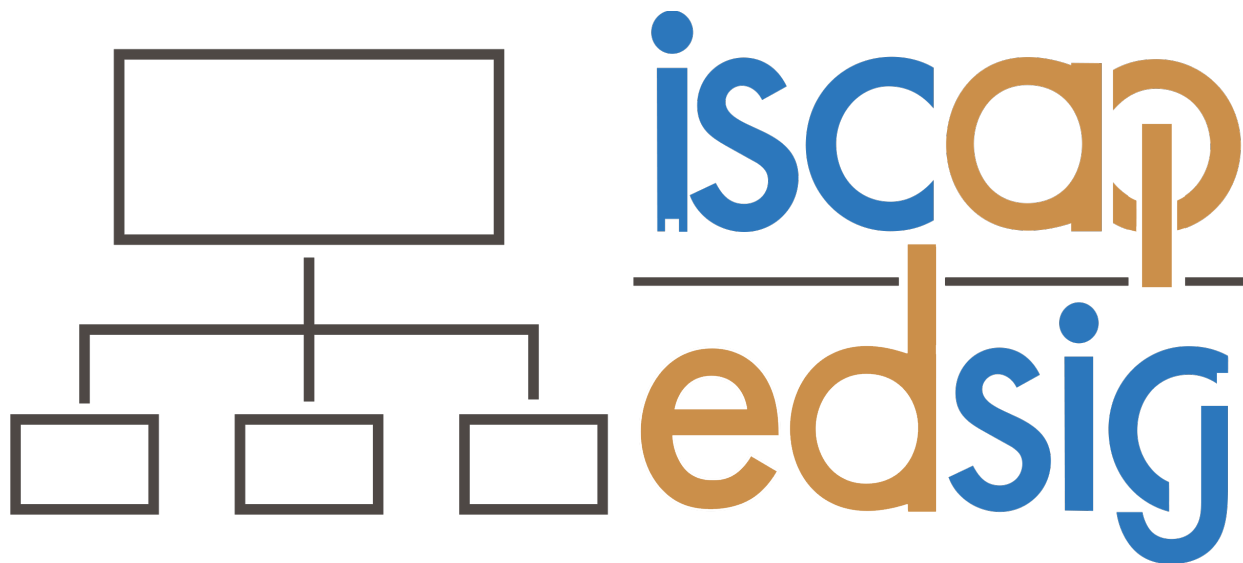
## AUTHOR BIOGRAPHY

**Michel Mitri** is a professor of computer information systems



and business analytics (CIS&BSAN) at James Madison University in Harrisonburg VA, where he has served on the faculty since 2001. His research interests include pedagogy of IS, artificial intelligence (AI), natural language processing, and data visualization.

Mitri has published in journals such as *Journal of Computer Information Systems*, *Journal of Information Systems Education*, *Communications of the AIS*, and *Expert Systems with Applications*. He is also the author of a software product called Story Analyzer, which can be seen at this site: <http://storyanalyzer.org/>.



**Information Systems & Computing Academic Professionals  
Education Special Interest Group**

**STATEMENT OF PEER REVIEW INTEGRITY**

All papers published in the *Journal of Information Systems Education* have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

Copyright ©2023 by the Information Systems & Computing Academic Professionals, Inc. (ISCAP). Permission to make digital or hard copies of all or part of this journal for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial use. All copies must bear this notice and full citation. Permission from the Editor is required to post to servers, redistribute to lists, or utilize in a for-profit or commercial use. Permission requests should be sent to the Editor-in-Chief, *Journal of Information Systems Education*, [editor@jise.org](mailto:editor@jise.org).

ISSN: 2574-3872 (Online) 1055-3096 (Print)