Quintin B. Rozelle
6/5/2025
5-2 Milestone Four
CS-499

- **Briefly describe the artifact. What is it? When was it created?**

The artifact chosen for this enhancement is the binary search tree assignment from CS-300 Data Structures and Algorithms. This was one assignment in a series that provided hands on experience with a variety of data structures. Each assignment accomplished the same task (i.e., loading, sorting, displaying, searching, and deleting records from a CSV file into memory) but did so using different data structures each time. This particular artifact was meant to showcase the use and benefits of a binary search tree.

- **Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in databases? How was the artifact improved?**

I chose this artifact since it works with and manipulates a large dataset (over 12,000 data entries). The original artifact loaded these into memory only, so this seemed like the perfect opportunity to incorporate the use of a database. Doing so comes with the benefit of data persistence between sessions and the utilization of optimized data structures and algorithms that are inherent in a standard database.

To accomplish the implementation of the database, I chose to build a CRUD API to interface with the database. This comes with the benefit of standardizing the interactions and simplifying the communications with the database. A CRUD API achieves this by eliminating the need to hardcode the query strings and calls to the database cursor in the final product which helps to keep code clear and reduces the chance of bugs. In general, all API function work the same way. First, the take in the required information to perform their task. Next they build a

query string from that data. Lastly, they pass that data along to a help function whose sole purpose is to run the query string and print a predefined message. As an example, the 'createRecords' function requires:

- The name of the table that the records will be added to
- The names of the columns that data will be added to in the form of a tuple
- The data for the records in the form a tuple of tuples. The outer tuple encapsulates all data, while each record is contained within its own tuple
- A boolean specifying whether or not to ignore duplicates

The function takes this supplied information and builds the query string. This query string is then passed to '_runQuery' (the helper function that actually runs the query) along with the success message of "Records added successfully". If '_runQuery' is successful, the message will be displayed, otherwise it will display an error message.

```
def createRecords(self,
                  tableName: str,
                  tableCols: tuple[str],
                  records: tuple[tuple[typing.Any]] | list[tuple[typing.Any]],
                  ignoreDuplicates: bool = True) -> None:
    """
    Creates multiple records

    Parameters
    ----------
    tableName: str
        The name of the table to add records to
    tableCols: tuple[str]
        The columns to add values to
    record: tuple[tuple[Any]] | list[tuples[Any]]
        The values of the records to add. Must line up with the values in tableCols
    ignoreDuplicates: bool (optional)
        Defines what to do if duplicates are found (default is True):
            True: Don't add record when the key is already in the table
            False: Update the record if the key is already in the table
    """
    # Build the query string
    queryHeader: str = f'INSERT OR {"IGNORE" if ignoreDuplicates else "REPLACE"} INTO {tableName} {tableCols} VALUES'
    queryBody: str = None
    for record in records:
        queryBody = f'{queryBody}, {record}'
    # Remove the initial 'None ,' from queryBody
    queryBody = queryBody[6:]
    queryString: str = queryHeader + queryBody
    self._runQuery(queryString, 'Records added successfully')
```

```python
def _runQuery(self, query: str, message: str | None = None) -> None:
    """
    A helper function to run query strings. Not meant to be called on it's own

    Parameters
    ----------
    query: str
        The query string to run
    message: str | None (optional)
        Custom message to display upon successfully running the query string.
        If None, will not display a message (default is None)
    """
    try:
        self.cursor.execute(query)
        self.connection.commit()
        if message:
            print(message)
    except sqlite3.Error as error:
        print(f'Error encountered: {error}')
        print(f'    Encounted while performing: {query}')
```

While my final product utilized only a few of the necessary CRUD functions (i.e., database creation; table creation; record creation, read, and delete), I chose to create the full suite of standard CRUD functions for both tables and records. Some of these were ultimately unused, but it provided me the opportunity to practice API creation while also showing my skill and ability to do so.

In addition to the CRUD API, my other major enhancement was the implementation of the database itself. I had originally thought of using a MongoDB due to my familiarity with it, but chose to use a SQLite database instead. This allowed me to gain further experience with a database I haven't used much. This also provided the benefit of a self-contained, local database which makes it easier to transfer the whole artifact to others to showcase my skills since another software (e.g., MySQL) isn't necessary to install and setup. Overall, the database structure is simple as it is just one table, but that is all that was necessary to convert the original artifact into one that used a database.

Lastly, I built the code to withstand potential SQL injection attacks. This is done through two main mechanisms. First, direct communication with the SQLite database is limited as

everything is passed through the CRUD API. Because of this, the only information that the user can provide that reaches the database is the CSV file and the bid ID numbers. Even then, this information is validated by converting inputs to integers (for the ID and monetary fields) or has quotations marks removed:

```
record: tuple[typing.Any] = (int(row[bidIdColumn]),
                             row[titleColumn].replace("'", ''),
                             row[fundColumn].replace("'", ''),
                             float((row[bidAmountColumn][1:]).replace(',','')))
```

This simple functionality will prevent SQL injection attacks like '1=1' because it will either throw an error because it cannot be converted to an int/float or will have the single quotes stripped prior to being used. Additionally, since each CRUD API function builds a query string that is surrounded in single quotes, an attack using double quotes (e.g., "1=1") will either fail the conversion to an int/float or be treated as a string.

The conversion of the ID and monetary fields to integers does run the risk of throwing an error if the values aren't numeric to begin with, but this will be caught and handled with the exception handling built into the code.

- **Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?**

  I did meet the course outcomes (i.e., 1, 2, and 4) I had originally planned for this enhancement:

- Outcome 1 – I completed this outcome though the creation of the CRUD API and implementation of comments and docstrings. The CRUD API allowed me to easily interact with the SQLite database and would allow other developers to do so as well. Further, the comments and docstrings throughout my code made it easy to understand and interact with.

- Outcome 2 – I had originally planned to accomplish this via a GUI for displaying data from this enhancement but chose not to because I felt it moved too far away from the original intent of this milestone (i.e., the focus on the implementation of a database). I realized that the code review, the narratives created for these milestones, and the final ePortfolio showcase my ability to develop and deliver professional-quality oral, written, and visual communication. With the completion of this milestone, I am considering this outcome achieved, though I still plan to showcase it with my completed ePortfolio during week 7.
- Outcome 4 – This outcome has been achieved with the completion of this milestone through my implementation of Python's CSV and sqlite3 modules. Using these shows my ability to incorporate code someone else has written to achieve the task at hand without reinventing the wheel.

The only update I have for the outcome-coverage plan is how I am completing Outcome Two. As stated above, my original plan was to show my visual communication skill though a GUI, but I chose to pivot due to concerns that this would detract from showcasing my ability to utilize a database. Instead, I will be accomplishing this outcome through the code review, narratives, and final ePortfolio. Overall, this isn't a change in when these outcomes will be completed, but the means through which they are.

- **Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?**

As stated above, my experience with SQL databases is minimal and using SQLite specifically was non-existent. Through this enhancement I learned to use them better and gained more comfort using the SQL language. Gaining the experience with implementing a self-

contained, local database (i.e., SQLite) was very beneficial since I can use this knowledge in the future if I am working on developing software that uses a database but is also meant to be shipped to an end user who doesn't have the ability or desire to setup and connect to an external third party database.

The main challenge I had with this enhancement was the design of the CRUD API. I assumed it was going to be very simple and easy to create when I started but quickly came across many necessary decisions that changed the way in which I needed to build the database. I know this is part of iterative software development, but I could have streamlined this a bit more had I spent some more time prior to coding to think about the design elements of the API. As an example, I originally thought I would need only one helper function (i.e., _runQuery) that I would pass a query string to. While coding, I discovered that the 'read' functions are significantly different enough from the 'creation', 'update', and 'delete' functions that a second helper function was needed for these, so I built '_readQuery' as well. Overall, this doesn't detract from the API and it does make the underlying functionality of the API easier to understand, but is an example of a challenge I encountered while creating this enhancement.