

The University of Akron

IdeaExchange@UAkron

---

Williams Honors College, Honors Research  
Projects

The Dr. Gary B. and Pamela S. Williams Honors  
College

---

Spring 2024

## Managing Inventory With a Database

David Bartlett  
dmb317@uakron.edu

Follow this and additional works at: [https://ideaexchange.uakron.edu/honors\\_research\\_projects](https://ideaexchange.uakron.edu/honors_research_projects)



Part of the [Databases and Information Systems Commons](#), [Data Storage Systems Commons](#), and the [Software Engineering Commons](#)

Please take a moment to share how this work helps you [through this survey](#). Your feedback will be important as we plan further development of our repository.

---

### Recommended Citation

Bartlett, David, "Managing Inventory With a Database" (2024). *Williams Honors College, Honors Research Projects*. 1818.

[https://ideaexchange.uakron.edu/honors\\_research\\_projects/1818](https://ideaexchange.uakron.edu/honors_research_projects/1818)

This Dissertation/Thesis is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Williams Honors College, Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact [mjon@uakron.edu](mailto:mjon@uakron.edu), [uapress@uakron.edu](mailto:uapress@uakron.edu).

**Managing Inventory With a Database**

David Bartlett

College of Engineering and Polymer Science, University of Akron

November 20, 2023

**Abstract**

Large commercial companies often use warehouses to store and organize their product inventory. However, manually keeping track of inventory through physical means can be a tedious process and is at risk for a variety of potential issues. It is very easy for records to be inaccurate or duplicated, especially if large reorganizations are undertaken, as this can cause issues such as duplicate product ID numbers. Therefore, it was decided that an inventory management system utilizing a SQL database should be created. The system needed to have capabilities including allowing the entry of product information, the ability to search database records based on attributes, the ability to update product attributes, and the ability to delete records, ideally with a confirmation prompt. Additionally, the system needed to have measures in place to prevent duplicate or redundant records from being entered. Expected business benefits from this program included being able to retrieve inventory records at increased speed, being able to reorganize and update product information quickly and more efficiently, and being able to avoid potentially costly errors from conflicting or redundant records. The system was successfully created with all of these features and more, and I learned a lot about C#, the software development process, and programming projects.

## **Problem Statement**

Large commercial companies often use warehouses to store and organize their product inventory. However, manually keeping track of inventory physically can be a tedious process, and physical records can be damaged, misplaced, or lost. Additionally, should the warehouse undergo a reorganization or relabeling of products, it may result in conflicting SKU numbers or duplicate records of certain products. Therefore, an inventory management system utilizing a SQL database should be created.

## **Functional Requirements**

Prior to developing the management system, a list of requirements for how the program must function was created. The system needs to have a database that has CRUD (create, retrieve, update, and delete) functionality, preferably one that can be manipulated using SQL (structured query language). For the retrieval functionality, the user must be able to search the database by at least one attribute of a product (SKU, name, category, brand), with SKU being the primary key. Errors in the data needed to be prevented, including the entry of duplicate SKU numbers and names for multiple records. The data also needs to have permanence, in that any changes to the database would be saved after the user exits the program. The user should also be prompted to confirm critical actions such as deleting a record or exiting the application, in order to best prevent the user from losing data or accidentally taking an action that they did not intend to.

## **User Personas**

This program was developed with two user personas, the intended or expected users of the program, in mind. The first is the manager of the inventory warehouse. They have been working at the company for quite some years now. They have very little knowledge of

computers, though can use them for basic purposes, and no experience with programming. The manager is a no-nonsense kind of person who likes to make sure that everyone knows accurate and up to date information, so that there are no conflicts and so the job gets done as efficiently and effectively as possible. The other persona is a worker at the inventory warehouse. They are newer to the company but have been there for a few years. They have some knowledge with computers, but not too in-depth. As part of their job, the worker regularly needs to view information on products, as well as updating information on products when new shipments arrive or there are changes to the warehouse. The worker is frustrated when there is lost, conflicting, and/or duplicate information that causes confusion and difficulty in their work.

### **Development and Implementation**

This program was primarily developed in Microsoft Visual Studio 2022 as a C# .NET Windows Form application. Microsoft SQL Server Management Studio 18 was used for the backend SQL database. One of the primary features implemented is a SQL database that data can be written to and retrieved from, with full CRUD functionality. There is also the ability to search the database by multiple item attributes at a time (ex: category and name). The application has a data grid view that allows users to see the items in the database, and that changes based on the search parameters that the user enters in the search function. Further features include input validation, confirmation prompts for sensitive actions such as exiting the program & deleting records, data permanence, a resize function that makes sure elements correctly resize with the form, and labels displaying the total records & total quantity sums that updates in real time as the data grid view updates.

The program functions by calling a custom function, `fillTable()`, which uses a SQL data adapter from the C# `System.Data.SqlClient` namespace to fill the grid view from the database

using a SQL SELECT query, upon first loading. The program then allows the user to add a new record, or select an existing record in the data grid to modify or delete. If the user chooses to add a new record, then a new instance of the addForm object is created. Upon submission of the addForm, data that the user entered is returned to the main form and validated, then entered into the database using a SQL INSERT query, and the grid view is refreshed to reflect the change. If the user chooses to edit an existing record, a new instance of the editForm object is created, and the already existing attributes are passed as arguments to automatically fill the editForm. Upon submission of the editForm, the data is returned to the main form and validated, then entered into the database using a SQL UPDATE query, and the grid view is refreshed to reflect the change. The user can also search for existing records using the search box in the upper right-hand corner, which allows the user to enter values for the attributes of a record and then either press the enter key on their keyboard or click the search button. Doing so calls the searchFunction() function, which uses the values entered by the user to execute a SQL SELECT query to the database and attempt to retrieve record(s) matching those parameters. The grid view is refreshed to show only the records matching the query, but can be restored to default by pressing the “clear search” button. Each time the grid view is updated, a for-each loop is used to display the sum of both the total amount of records and total product quantity currently shown. The program window can also be resized by the user. Doing so calls the resize() function, which accepts a control *c* and a rectangle *r* as arguments, to resize the controls so that the UI elements all stay in constant proportion as the user resizes the application window. For a visual depiction of some of these features, see the **Figures** section of this report.

## **Backlog**

In programming, features for a project are usually prioritized on a scope from “must have” features to “will not have” features, with features that the program could or should have in between. There are often potential features in the backlog of a project that are not implemented due to issues such as time constraints or lack of resources. The backlog of features not implemented for this project includes UI improvements, search function improvements, and custom form icons. Custom form icons were a low priority feature that were not able to be implemented due to time constraints. The UI improvements were also a lower priority as the user interface had been worked upon to a point where it was functional with a simple, easy to use layout for the average user, so any further improvements would have been minor adjustments. In terms of the search function, users are required to be very specific when entering the name of a product, and I would’ve liked to have made this a little more forgiving. While this was higher priority than the UI improvements and form icons, it was still not a critical feature for the program to function, and so was cut due to time constraints.

## **Issues Encountered**

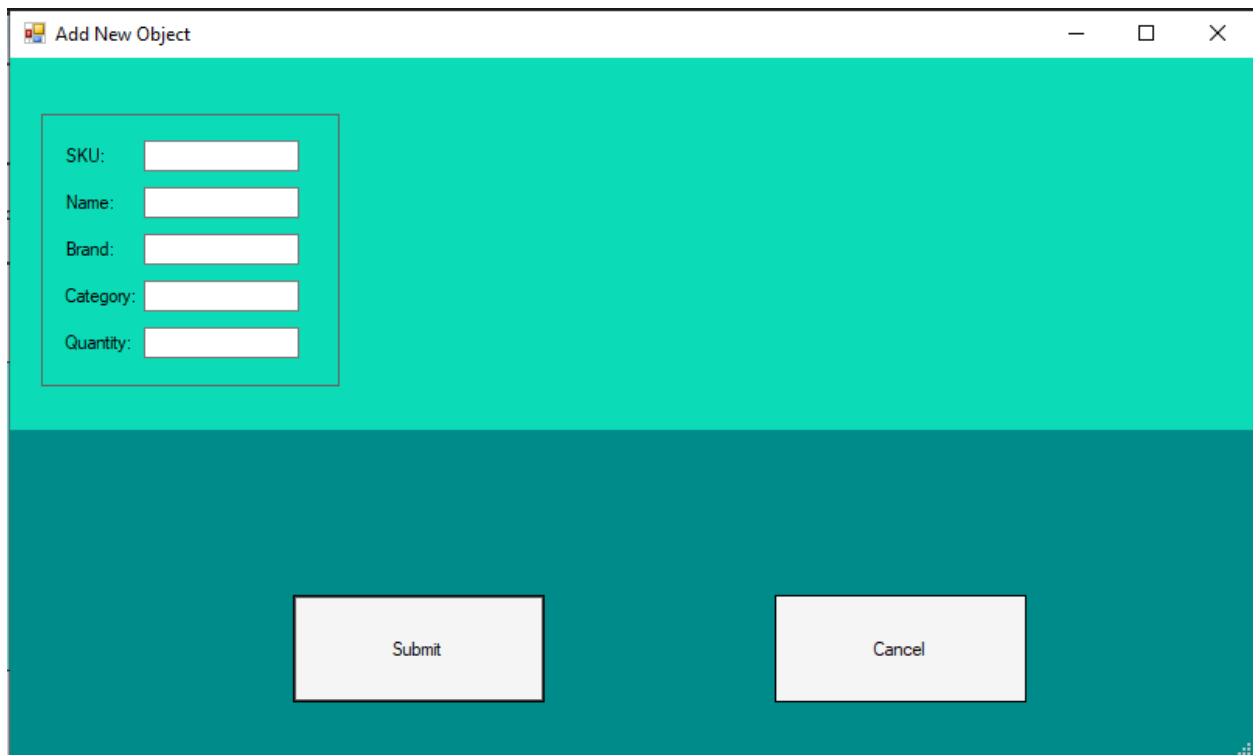
Programming projects rarely go perfectly according to plan, and this one was no exception. More time was spent on interface design and improvement than initially anticipated, as well as on connecting the database to the program and writing the various SQL queries, as the queries had to account for several different user actions and input. An issue was also encountered in the middle of development where the database was not correctly saving changes to the data after the program was exited and would revert back to a previous state. This ultimately required the database to be completely rebuilt from scratch, resulting in a loss of time that could have been spent on other features in the project backlog. Debugging the program also took more time

and a closer attention to detail than expected at certain points. Some issues were only narrowly caught, including a minor interface error that was only noticed the night before a presentation of the project.

## Figures

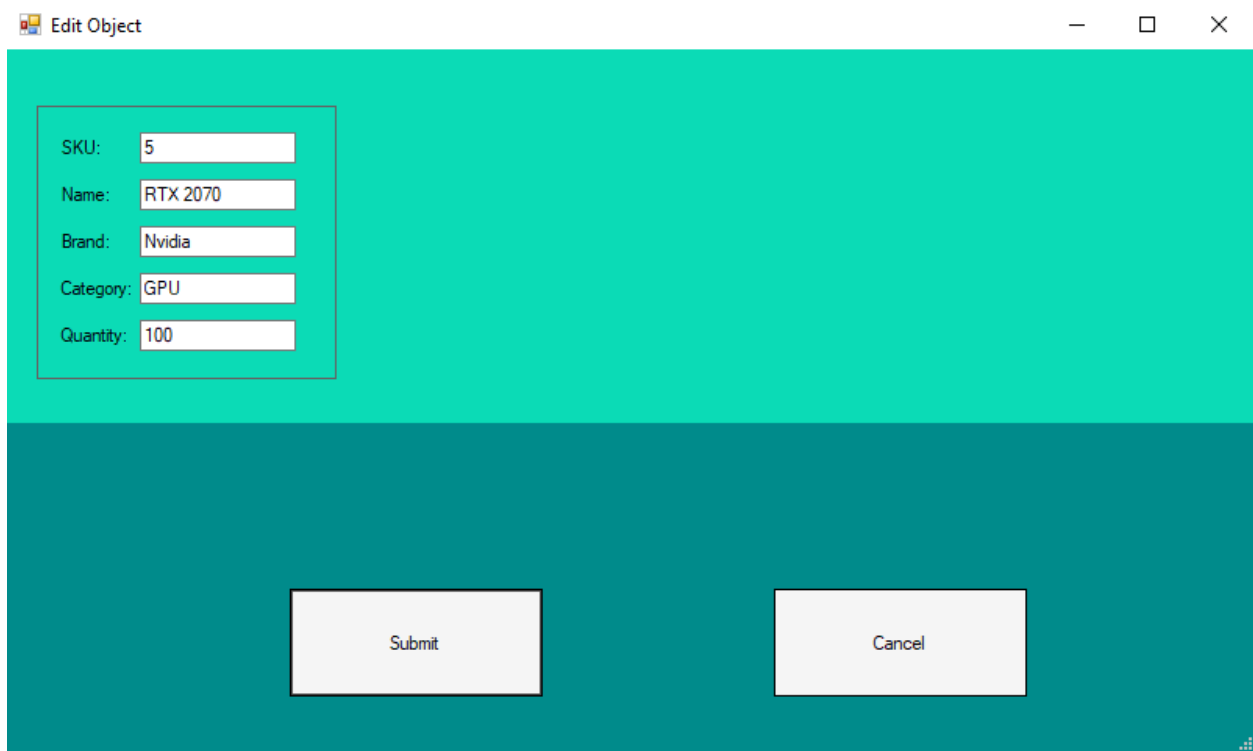
	SKU	Name	Brand	Category	Quantity
▶	1	Energizer AA - 50 Co...	Energizer	Batteries	2300
	2	Energizer AA - 20 Co...	Energizer	Batteries	3000
	3	Energizer AAA - 20 C...	Energizer	Batteries	129
	4	Energizer AAA - 50 C...	Energizer	Batteries	981
	5	RTX 2070	Nvidia	GPU	100
	6	RTX 4090	Nvidia	GPU	80
	7	RX 7900 XT	AMU	GPU	150
	8	GTX 1080	Nvidia	GPU	100
	9	RTX 3060	Nvidia	GPU	391

*Figure 1: Main interface*



A screenshot of a web application window titled "Add New Object". The window has a light blue header bar with the title and standard window controls (minimize, maximize, close). The main content area has a light blue background. On the left, there is a white rectangular form with a thin black border. Inside this form, there are five labels with corresponding input fields: "SKU:", "Name:", "Brand:", "Category:", and "Quantity:". Each input field is empty. Below the form, there are two white rectangular buttons with black borders, labeled "Submit" and "Cancel", positioned side-by-side.

*Figure 2: Adding a new record*



A screenshot of a web application window titled "Edit Object". The window has a light blue header bar with the title and standard window controls (minimize, maximize, close). The main content area has a light blue background. On the left, there is a white rectangular form with a thin black border. Inside this form, there are five labels with corresponding input fields: "SKU:", "Name:", "Brand:", "Category:", and "Quantity:". The input fields contain the following text: "5", "RTX 2070", "Nvidia", "GPU", and "100". Below the form, there are two white rectangular buttons with black borders, labeled "Submit" and "Cancel", positioned side-by-side.

*Figure 3: Editing an existing record*

The screenshot shows a Windows application titled "Inventory". On the left is a sidebar with buttons: "Add New Record", "Edit Selected Record", "Delete Selected Record", and "Exit". The main area has a teal background. At the top right, there's a search panel with input fields for "SKU:", "Name:", "Brand:" (containing "AMD"), and "Category:" (containing "GPU"). There are "Search" and "Clear Search" buttons. Below the search panel, it says "Total records: 3". A table displays the results:

	SKU	Name	Brand	Category	Quantity
▶	7	RX 7900 XT	AMD	GPU	150
	12	RX 590	AMD	GPU	77
	13	RX 6950 XT	AMD	GPU	290
*					

Below the table is a large grey rectangular area. At the bottom right of the main area, it says "Total quantity: 517".

*Figure 4: Example search results*

```
private void fillTable()
{
    int totalRec = 0;
    int totalQuant = 0;
    // Function to load database
    using (connection = new SqlConnection(connectionString))
    using (SqlDataAdapter tableAdapter = new SqlDataAdapter("SELECT * FROM Inventory", connection))
    {
        inventoryTable.Clear();
        tableAdapter.Fill(inventoryTable);
        inventoryGridView.DataSource = inventoryTable;
    }
    foreach (DataRow row in inventoryTable.Rows)
    {
        totalRec += 1;
        totalQuant += int.Parse(row[4].ToString());
        lblTotalRec.Text = totalRec.ToString();
        lblTotalQuant.Text = totalQuant.ToString();
    }
}
```

*Figure 5: Code snippet of fillTable() function*

## **Conclusion**

The inventory management system was successfully implemented, with all of the functional requirements and system capabilities being fulfilled, as well as additional features being added. Although I was unable to implement everything I ultimately wanted to in the project, the remaining items in the backlog were not critical to program functionality. I also learned much about C#, as it was a language I had not used and knew little about prior to this project. Undertaking this project also allowed me to understand some of the processes that go into, and issues that arise from, programming projects, such as how bugs manage to slip through development, as there were a few that I only narrowly caught. Ultimately, the program ensures inventory can be managed efficiently with minimal errors. An inventory management system created in C# using a SQL database makes it quicker and easier for businesses to retrieve records and ensure information is accurate and up to date, allowing for increased efficiency and more accurate business decisions.