Scorch Shield

Final Project - CS4354

Cameron Williams, David Poole, Stephan DeLuna, Jacob Bruen



# Abstract

Scorch Shield is a database system built to modernize wildfire relief management by replacing outdated Excel-based methods. Wildfires move fast and require quick, coordinated action, but many agencies still rely on tools that aren’t designed for large-scale, real-time collaboration. Our system improves on this by using structured relationships, unique identifiers, and clear schema design to keep data consistent, accurate, and easy to manage. It supports essential operations like tracking victims, aid requests, resources, and shelter capacity, while also allowing for more advanced queries using joins, aggregations, and conditions. The database is scalable, flexible, and ready to integrate with user interfaces or larger disaster response platforms. Overall, Scorch Shield offers a strong foundation for smarter, faster, and more organized wildfire response.

# Table of Contents

[**Abstract 2**](#_78dvido3wqz6)

[**Table of Contents 2**](#_fm0xh51e5peh)

[**Team and Contribution 3**](#_co3bdwya9x5j)

[**Introduction 3**](#_tpn6z5twm1ah)

[**Design 4**](#_gc0xylklj4y3)

[**Implementation 11**](#_yg2tzrmkmtt4)

[**System Testing and Result Analysis 13**](#_2mgvi3ny05po)

[**Conclusion 22**](#_iuldxs6tzbvl)

# Team and Contribution

David Poole (Project Leader) - Design section, Result analysis section

Cameron Williams (Project Manager) - Testing section, Implementation section

Stephan DeLuna (Query Developer) - Introduction section

Jacob Bruen (Database Designer) - Conclusion section

# Introduction

Scorch Shield is a database solution designed specifically for wildfire relief management. Wildfires are dangerous, fast-moving disasters that require immediate, coordinated action across multiple agencies to contain. However, many aid organizations, emergency responders, and government agencies still rely on Excel to manage disaster data and logistics; this leads to significant issues in accuracy, collaboration, and scalability due to the outdated nature of the software. Modern database systems are crucial to improving operational speed, reliability, and coordination, ultimately enhancing response times and overall efficiency.

Excel presents many challenges when used in large-scale systems like wildfire management. It has data integrity issues because it cannot ensure data consistency, often resulting in errors, duplicates, and conflicts. It faces scalability problems since performance degrades as data volume grows, impacting system stability. There is no concurrent access, meaning multiple users cannot edit a file simultaneously without risking conflicts, duplicates, or data corruption. Additionally, there are security concerns, as Excel files are vulnerable to being copied, modified or lost due to inadequate protection.

Our database solution, Scorch Shield, addresses these problems by improving data integrity through unique identifications and structured relationships that prevent errors and duplicates. It increases scalability, enabling the system to handle large datasets without performance degradation. It provides real-time concurrent access, allowing multiple users to safely update and edit data simultaneously without conflicts or issues. It also enhances security by protecting sensitive information using authentication, encryption, and role-based access controls.

The key requirements of Scorch Shield include: data storage and management to store wildfire-related data such as incident reports, emergency responses, and aid distribution, while preventing duplicates through unique identifiers; user access and editing capabilities that allow multiple users to add, edit, and view data securely using login credentials; improved scalability and performance to ensure smooth operation even as the dataset grows; strong security measures to restrict access only to authorized users and safeguard sensitive data with encryption and role-based permissions; and integration of data sources to pull real-time information from official channels and generate synthetic estimates when live data is unavailable.

Together, these features position Scorch Shield as a comprehensive and reliable system to modernize wildfire relief management. Data sources would be pulled from government agencies and weather stations, which help monitor and report natural disasters. The following sections describe the system’s design, implementation, and testing, along with the challenges addressed during development.

# Design

### Description of Users and Their Functionalities

Our system supports three primary user roles, each with distinct responsibilities and access to the database. While role-based authentication is not implemented in code, the system is designed to conceptually support these roles through the separation of functionality.

1. Aid Coordinators

Aid Coordinators are the primary administrators of the disaster relief effort. They are responsible for managing core records and assigning tasks to volunteers.

Functionalities:

* View all disaster events, shelter data, victims, volunteers, aid requests, and available resources.
* Add/Edit/Delete disaster records.
* Assign volunteers to specific disaster events.
* Create, update, and fulfill aid requests.
* Manage shelters and resource allocations.

2. Volunteers

Volunteers interact with the system in a more limited capacity. Their primary role is to provide support during disaster events and update their availability and logs.

Functionalities:

* View assigned disaster events and related shelter information.
* Update their own availability and contact information.
* Log the assistance they have provided (if integrated in future versions).

3. Government Analysts

Government Analysts are granted read-only access to the system. Their focus is on understanding patterns, generating insights, and evaluating disaster response effectiveness.

Functionalities:

* View all disaster data, resource usage, aid request fulfillment, and volunteer activity logs.
* Generate reports on aid effectiveness, resource utilization, and response times.
* Analyze trends for policy recommendations and planning.

This role-based structure allows for clear separation of duties and prepares the system for potential implementation of authentication, dashboards, or user interfaces in future iterations. Each user group accesses only the functionality required for their responsibilities, maintaining both operational efficiency and data security.

### Normalization

To ensure data consistency and eliminate redundancy, our database schema has been normalized to the Third Normal Form (3NF). This guarantees that the data is well-structured, easy to maintain, and scalable. The following outlines how normalization has been applied throughout the system.

First Normal Form (1NF)

All relations in the schema satisfy 1NF:

* Each table has a primary key.
* All attributes contain atomic values (no arrays, lists, or sets).
* There are no repeating groups or multivalued fields in any relation.

Example: In the Victim\_Assistance\_Needed table, each assistance need is stored as a separate row linked to a single Victim\_ID, preventing multivalued fields like Needs = 'Food, Shelter'.

Second Normal Form (2NF)

All tables are in 2NF because:

* They are already in 1NF.
* Every non-key attribute is fully functionally dependent on the entire primary key (not just part of it).

Example: In the Resource\_Request table, the primary key is a composite of Resource\_ID and Request\_ID. Attributes like Amount and Status depend on both columns together, not just one of them.

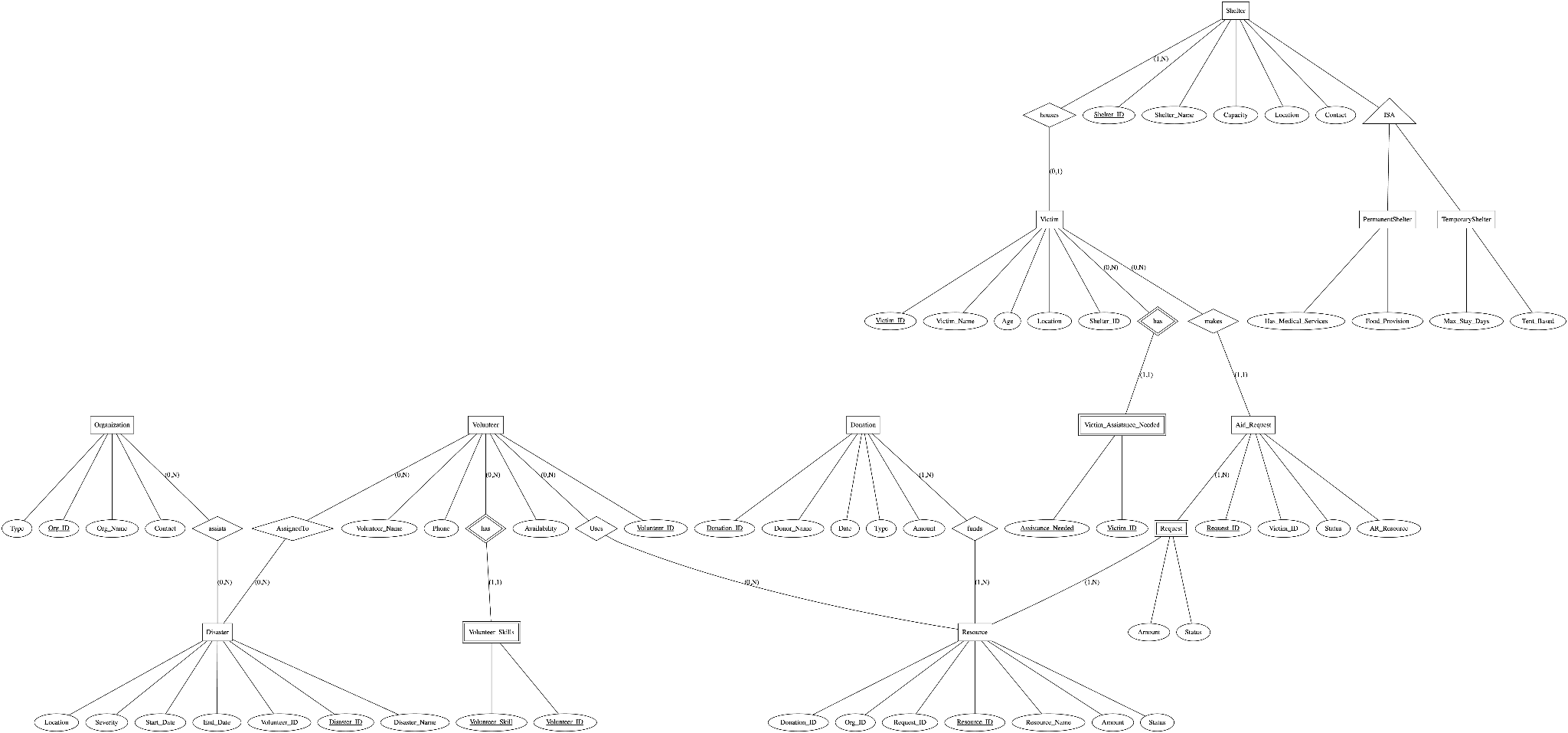
Third Normal Form (3NF)

Our tables also satisfy 3NF:

* They are in 2NF.
* There are no transitive dependencies, non-key attributes do not depend on other non-key attributes.

Example: In the Resource table, we store only the Donation\_ID as a foreign key. Details about the donation (like donor name, amount, and date) are stored in the Donation table, not duplicated in Resource.

### ER Diagram



( Higher resolution version of ER diagram: [graphviz (3).png](https://1drv.ms/i/s!Ako1Ei1GDhArkjsZfwDlC87Z8ny5?e=CY3R5G) )

### ER Diagram to Relational Schema process

Each entity and relationship in our ER diagram was translated into relational tables. Strong entities like Victim, Shelter, and Volunteer were directly mapped to tables with their respective primary keys. Weak and associative entities such as Victim\_Assistance\_Needed and Resource\_Request were implemented as separate tables with composite primary keys and foreign key constraints.

Relationships like AssignedTo and Funds that represent many-to-many associations were also implemented as separate join tables to maintain referential integrity. Attributes from relationships (e.g., Amount and Status in Resource\_Request) were preserved in the schema as columns of their respective tables. This mapping ensures that the relational schema accurately reflects the structure and constraints of the ER model while maintaining normalization and data consistency.

### Relational Schema

### User Flow Diagram:

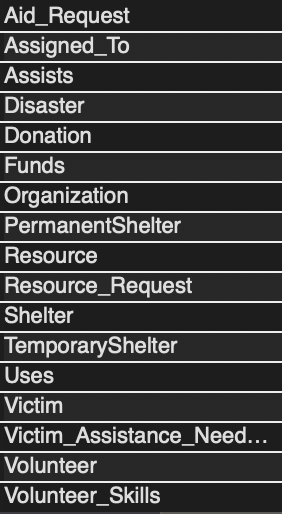
# 

# 

# Implementation

We implemented the database with MySQL using the relations seen in the ER Diagram and relational schema. Here are some queries to show the basic structure of the database and the data we gathered.

Using ***SHOW TABLES*,** we can see the structure of the database:



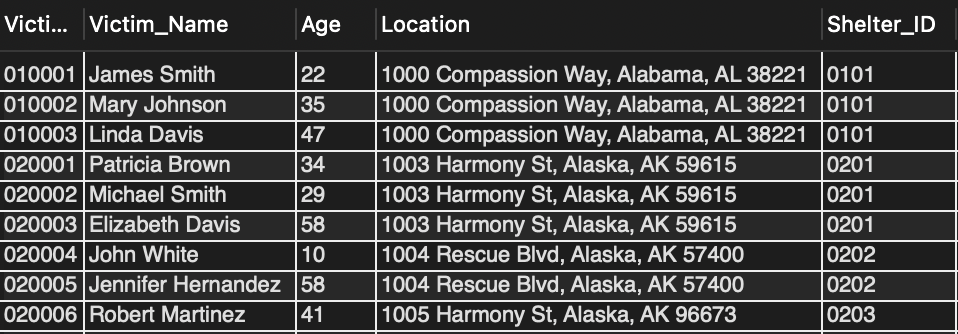
Here are some samples from a couple of our tables, demonstrating our insertions work correctly:

*SELECT \* FROM Disaster;*

*SELECT \* FROM Victim;*

*SELECT \* FROM Resource;*







To populate these tables, we used insertions like this:

*INSERT INTO Victim*

*(Victim\_ID, Victim\_Name, Age, Location, Shelter\_ID)*

*VALUES*

*('010001', 'James Smith', 22, '1000 Compassion Way, Alabama, AL 38221', '0101'),*

*('010002', 'Mary Johnson', 35, '1000 Compassion Way, Alabama, AL 38221', '0101'), …*

*INSERT INTO Disaster*

*(Disaster\_ID, Disaster\_Name, Location, Severity, Start\_Date, End\_Date)VALUES*

*('WF001', 'Alabama Wildfire 2024','Alabama',69142, '2024-07-12', '2024-09-05'),*

*('WF002', 'Alaska Wildfire 2024','Alaska',24603, '2024-10-25', '2024-10-31'), …*

*INSERT INTO Resource (Resource\_ID, Resource\_Name, Status, Amount, Donation\_ID, Org\_ID) VALUES*

*('10023','Water Bottle','In Stock',2000,'DON001','ORG001'),*

*('12340','Canned Goods','In Stock',2500,'DON002','ORG002'), …*

The system was tested and accessed through MySQL Workbench. This interface allows users to directly interact with the database using SQL queries for insertion, updates, deletion, and selection of data. It provides a clear view of table structures, relationships, and query results for system verification.

# System Testing and Result Analysis

To ensure our database was operational and useful to users, we extensively tested it with a wide range of queries that could be expected to be used by disaster control stakeholders. Here are some example queries that demonstrate the versatility of the database:

**Retrieves all victims whose names start with 'M', along with their associated aid request and resource request information. Shows NULLs if no aid was requested.**

*SELECT*

*v.Victim\_ID,*

*v.Victim\_Name,*

*v.Age,*

*v.Location,*

*v.Shelter\_ID,*

*ar.Request\_ID AS Aid\_Request\_ID,*

*ar.Status AS Aid\_Request\_Status,*

*rr.Resource\_ID,*

*rr.Amount AS Quantity\_Requested,*

*rr.Status AS Resource\_Request\_Status*

*FROM Victim AS v*

*LEFT JOIN Aid\_Request AS ar*

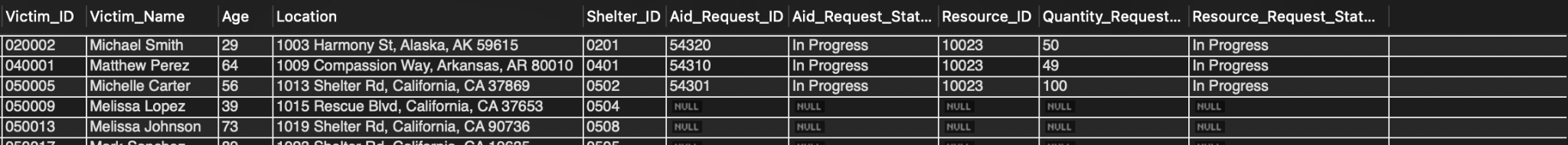
*ON v.Victim\_ID = ar.Victim\_ID*

*LEFT JOIN Resource\_Request AS rr*

*ON ar.Request\_ID = rr.Request\_ID*

*AND rr.Status IN ('In Progress','Completed')*

*WHERE v.Victim\_Name LIKE 'M%';*

**

**Calculates how much of each donated resource has been used in completed requests, showing the total amount used and its percentage of the original donation.**

*SELECT*

*r.Resource\_ID,*

*r.Resource\_Name,*

*d.Amount AS Donated\_Amount,*

*SUM(rr.Amount) AS Completed\_Amount,*

*ROUND(*

*SUM(rr.Amount) / d.Amount \* 100*

*, 2) AS Percent\_Used*

*FROM Resource\_Request rr*

*JOIN Resource r USING(Resource\_ID)*

*JOIN Donation d USING(Donation\_ID)*

*WHERE rr.Status = 'Completed'*

*GROUP BY*

*r.Resource\_ID,*

*d.Amount*

*ORDER BY*

*r.Resource\_ID;*

**

**Displays the total amount requested for each resource across all aid requests, giving a quick overview of resource demand.**

*SELECT*

*rr.Resource\_ID,*

*r.Resource\_Name,*

*SUM(rr.Amount) AS Total\_Requested*

*FROM Resource\_Request AS rr*

*JOIN Resource AS r*

*ON rr.Resource\_ID = r.Resource\_ID*

*GROUP BY*

*rr.Resource\_ID,*

*r.Resource\_Name*

*ORDER BY*

*rr.Resource\_ID;*

**

**Identifies resources with less than 50% of their original donated amount remaining and calculates their remaining percentage.**

*SELECT*

*r.Resource\_ID,*

*r.Resource\_Name,*

*r.Amount AS Current\_Amount,*

*d.Amount AS Total\_Amount,*

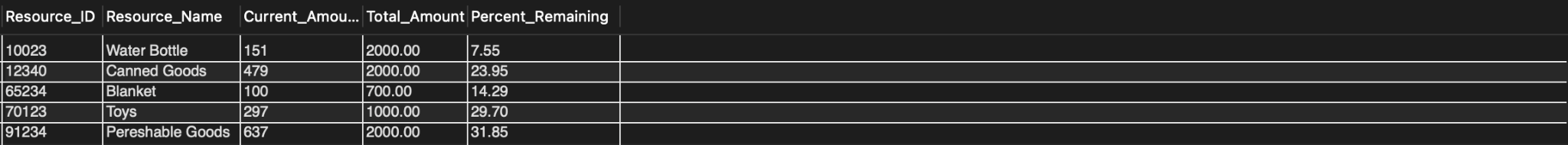
*ROUND(r.Amount / d.Amount \* 100, 2) AS Percent\_Remaining*

*FROM Resource AS r*

*JOIN Donation AS d*

*ON r.Donation\_ID = d.Donation\_ID*

*WHERE r.Amount < 0.5 \* d.Amount;*

**

*Counts how many victims are currently assigned to each shelter, helping coordinators understand shelter load.*

*SELECT*

*s.Shelter\_ID,*

*s.Shelter\_Name,*

*COUNT(v.Victim\_ID) AS Num\_Victims*

*FROM Shelter s*

*LEFT JOIN Victim v*

*ON s.Shelter\_ID = v.Shelter\_ID*

*GROUP BY s.Shelter\_ID, s.Shelter\_Name;*

**

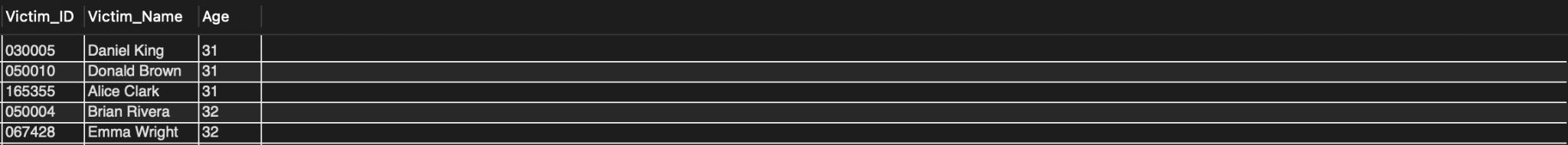
**Lists all victims older than 30, ordered by age. This could be useful for prioritizing medical or housing needs.**

*SELECT Victim\_ID, Victim\_Name, Age*

*FROM Victim*

*WHERE Age > 30*

*ORDER BY Age ASC;*

**

**Retrieves disasters with severity scores over 55,000, allowing analysts to focus on the most critical events.**

*SELECT*

*Disaster\_ID,*

*Disaster\_Name,*

*Location,*

*Severity,*

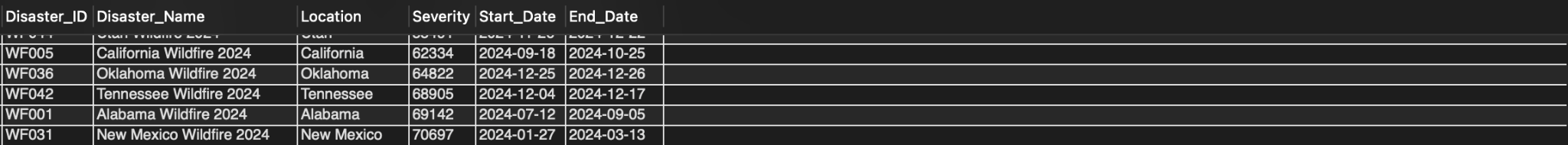
*Start\_Date,*

*End\_Date*

*FROM Disaster*

*WHERE Severity > 55000*

*ORDER BY Severity ASC;*

**

**Displays all shelters and determines their type (Permanent or Temporary) based on subclass table matches using a CASE statement.**

*SELECT \* FROM Shelter;*

*SELECT s.\*,*

*CASE*

*WHEN ps.Shelter\_ID IS NOT NULL THEN 'Permanent'*

*WHEN ts.Shelter\_ID IS NOT NULL THEN 'Temporary'*

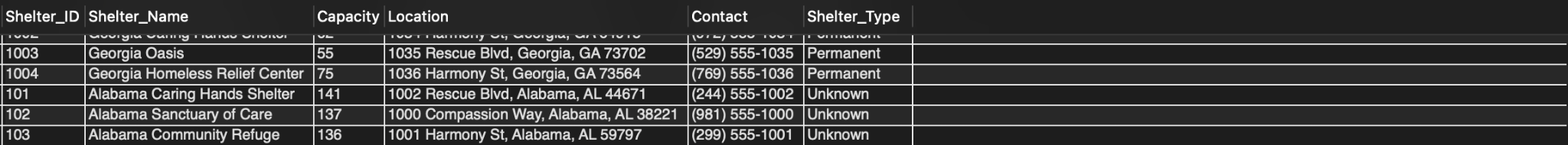
*ELSE 'Unknown'*

*END AS Shelter\_Type*

*FROM Shelter s*

*LEFT JOIN PermanentShelter ps ON s.Shelter\_ID = ps.Shelter\_ID*

*LEFT JOIN TemporaryShelter ts ON s.Shelter\_ID = ts.Shelter\_ID;*

**

**Finds temporary shelters that are tent-based and still have available space, showing current occupancy and remaining capacity.**

*SELECT*

*ts.Shelter\_ID,*

*s.Shelter\_Name,*

*s.Capacity,*

*COUNT(v.Victim\_ID) AS Current\_Occupancy,*

*s.Capacity - COUNT(v.Victim\_ID) AS Available\_Space*

*FROM TemporaryShelter AS ts*

*JOIN Shelter AS s ON ts.Shelter\_ID = s.Shelter\_ID*

*LEFT JOIN Victim AS v ON v.Shelter\_ID = s.Shelter\_ID*

*WHERE ts.Tent\_Based = TRUE*

*GROUP BY*

*ts.Shelter\_ID,*

*s.Shelter\_Name,*

*s.Capacity*

*HAVING*

*s.Capacity - COUNT(v.Victim\_ID) > 0;*

**

**This query retrieves all victims who are currently staying in shelters with a capacity below the overall average. It uses a nested subquery to calculate the average capacity and filter shelters accordingly.**

*SELECT Victim\_ID, Victim\_Name, Shelter\_ID*

*FROM Victim*

*WHERE Shelter\_ID IN (*

*SELECT Shelter\_ID*

*FROM Shelter*

*WHERE Capacity < (*

*SELECT AVG(Capacity)*

*FROM Shelter*

*)*

*);*

**

**This query finds resources that were requested in greater quantities than they were donated. A nested query first calculates total requested amounts per resource, then the outer query filters for those that exceed donation amounts.**

*SELECT Resource\_ID, Resource\_Name, Donated\_Amount, Requested\_Amount*

*FROM (*

*SELECT*

*r.Resource\_ID,*

*r.Resource\_Name,*

*d.Amount AS Donated\_Amount,*

*IFNULL(SUM(rr.Amount), 0) AS Requested\_Amount*

*FROM Resource r*

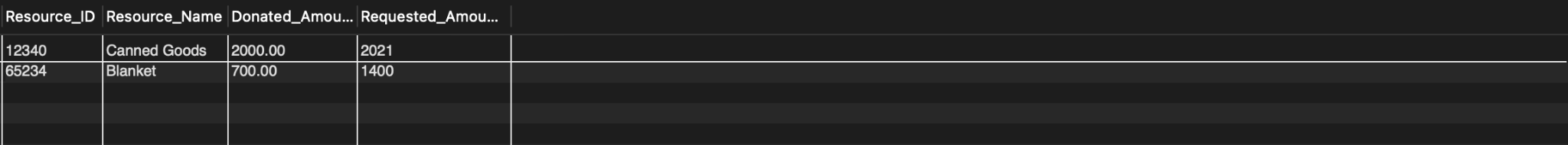
*JOIN Donation d ON r.Donation\_ID = d.Donation\_ID*

*LEFT JOIN Resource\_Request rr ON r.Resource\_ID = rr.Resource\_ID*

*GROUP BY r.Resource\_ID, r.Resource\_Name, d.Amount*

*) AS ResourceUsage*

*WHERE Requested\_Amount > Donated\_Amount;*

**

**This query returns victims who are housed in shelters with more than 3 occupants. It uses a nested query with GROUP BY and HAVING to identify crowded shelters, then retrieves all victims linked to them.**

*SELECT Victim\_ID, Victim\_Name, Shelter\_ID*

*FROM Victim*

*WHERE Shelter\_ID IN (*

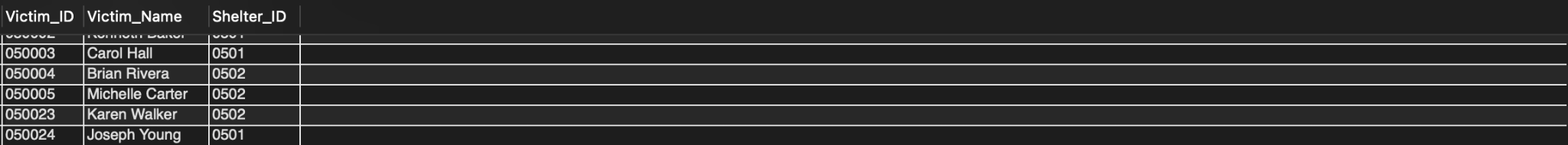
*SELECT Shelter\_ID*

*FROM Victim*

*GROUP BY Shelter\_ID*

*HAVING COUNT(\*) > 3*

*);*

**

**This query identifies all aid requests linked to the top 2 most requested resources. A nested query calculates the highest total request amounts per resource, and a derived table is used to join back with the main query to avoid MySQL’s LIMIT limitations in subqueries.**

*SELECT ar.Request\_ID, ar.Status, rr.Resource\_ID*

*FROM Aid\_Request ar*

*JOIN Resource\_Request rr ON ar.Request\_ID = rr.Request\_ID*

*WHERE rr.Resource\_ID IN (*

*SELECT Resource\_ID*

*FROM Resource\_Request*

*GROUP BY Resource\_ID*

*ORDER BY SUM(Amount) DESC*

*LIMIT 2*

*);*

**

These queries show the depth and detail of our database. With the extensive attributes and relationships, almost any data a user could need can quickly be retrieved. The queries we laid out demonstrate the relations between the entities and how those relationships can be used to join tables together and apply different SQL functions to display data efficiently in many realistic use cases. Additionally, they validate that our schema supports aggregation, conditional logic, specialization, and nested queries, making the system both practical and scalable for disaster relief operations.

# Conclusion

Scorch Shield modernizes wildfire relief operations by replacing outdated Excel workflows with a robust database system. It delivers accurate and consistent data through structured storage and supports real-time collaboration, enhancing decision-making for aid organizations, emergency responders, and government agencies. This promotes faster response times, smarter resource allocation, and better coordination during wildfire emergencies.

Building a database like Scorch Shield involves many complex components, as these systems must handle numerous aspects and data items that require constant monitoring. More importantly, the database must be designed in a way that makes sense, ensuring it is easy to adapt, modify, and maintain over time as needs evolve.

Some of the most important lessons learned during the development process included understanding the role of Entity-Relationship (ER) models and Relational Schemas, as they provide the essential framework and outline for the system, and recognizing how models and schemas often need to evolve during the construction of the database as new requirements and changes emerge throughout the project.

Overall, this project provides a strong foundation for improving wildfire relief management. The next logical step would be to develop a functional user interface, an optional requirement we did not implement in this phase, to make the system more accessible and user-friendly. Additionally, integrating Scorch Shield into the broader systems used by other organizations and agencies would greatly increase its value and impact, enabling a more unified and effective approach to disaster response.