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Scorch Shield is a database system built to modernize wildfire relief management by replacing outdated Excel-based methods.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
Table of Contents 2
Team and Contribution 3
Introduction 3
Design 4
Implementation 11
System Testing and Result Analysis 13
Conclusion 22

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disaster data and logistics; this leads to significant issues in accuracy, collaboration, and scalability due to the outdated nature of the software.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.

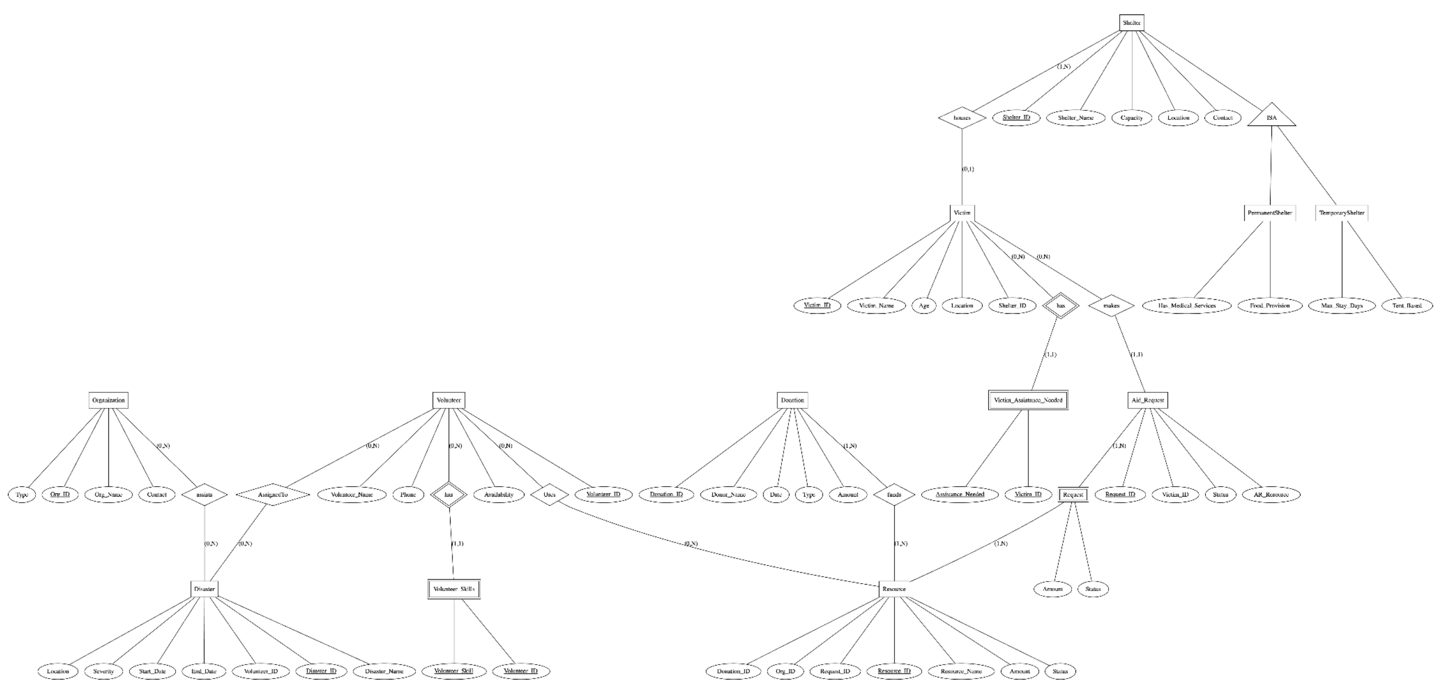
Third Normal Form (3NF)

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



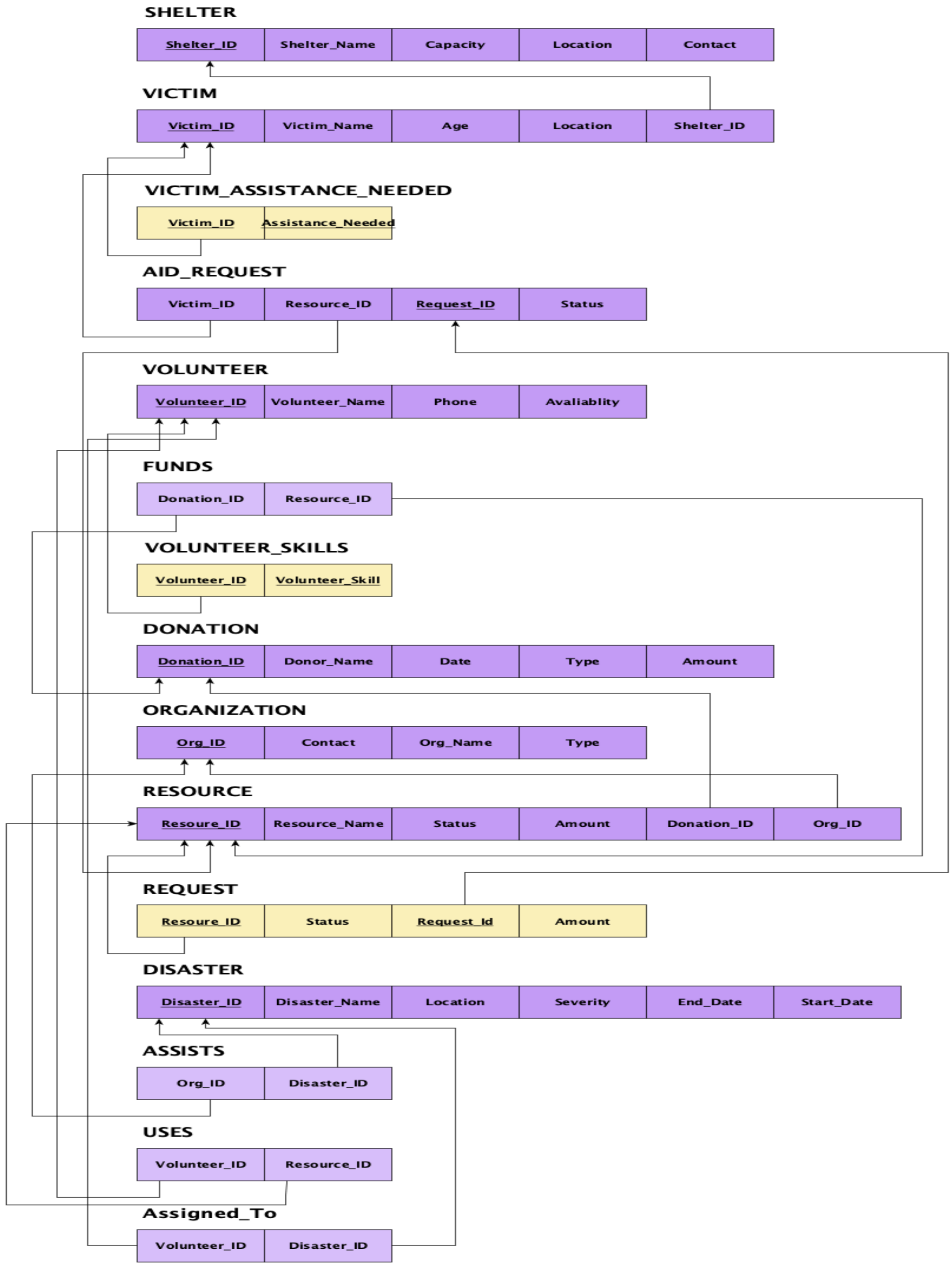
(Higher resolution version of ER diagram: [graphviz (3).png])

ER Diagram to Relational Schema process

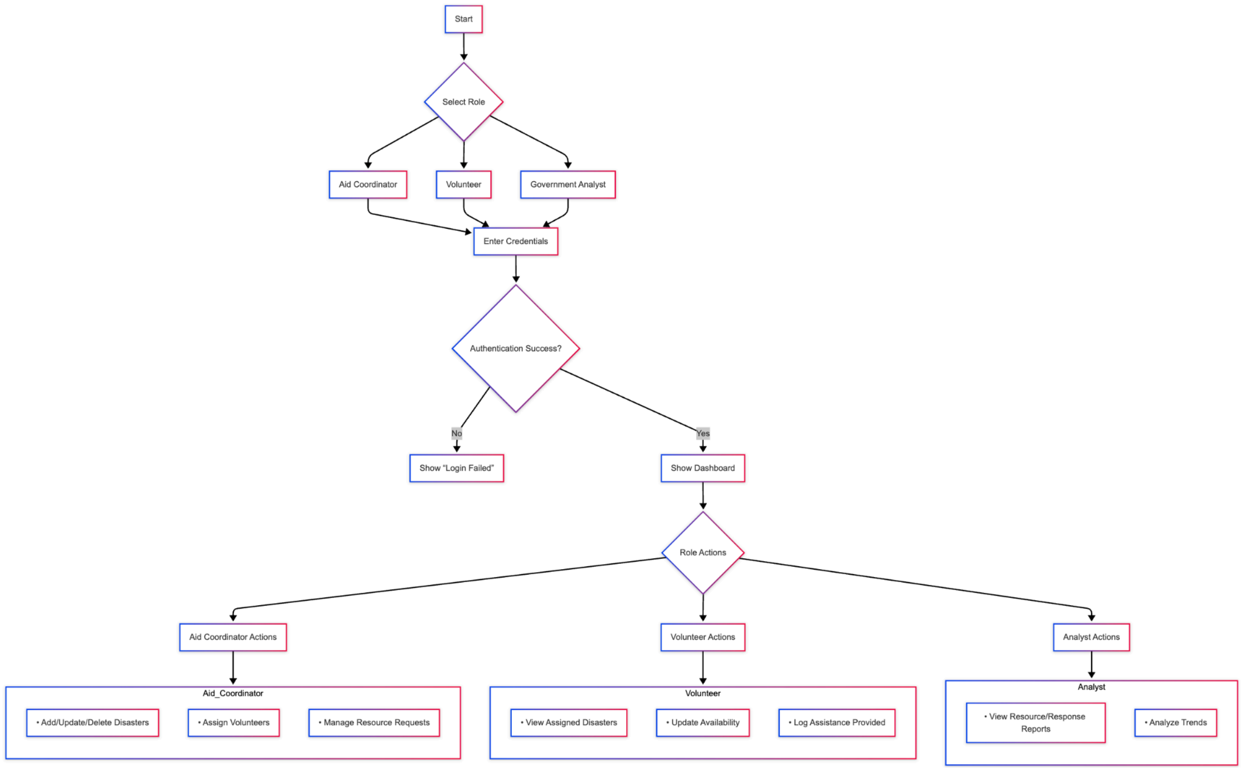
Each entity and relationship in our ER diagram was translated into relational tables. Strong entities like Victim, Shelter, and Volunteer were directly mapped to tables with their respective primary keys. Weak and associative entities such as Victim_Assistance_Needed and Resource_Request were implemented 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:

Aid_Request
Assigned_To
Assists
Disaster
Donation
Funds
Organization
PermanentShelter
Resource
Resource_Request
Shelter
TemporaryShelter
Uses
Victim
Victim_Assistance_Need…
Volunteer
Volunteer_Skills

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

*SELECT * FROM Disaster;*

*SELECT * FROM Victim;*

*SELECT * FROM Resource;*

Disaster_ID	Disaster_Name	Location	Severity	Start_Date	End_Date
WF001	Alabama Wildfire 2024	Alabama	69142	2024-07-12	2024-09-05
WF002	Alaska Wildfire 2024	Alaska	24603	2024-10-25	2024-10-31
WF003	Arizona Wildfire 2024	Arizona	38659	2024-07-29	2024-07-30
WF004	Arkansas Wildfire 2024	Arkansas	97136	2024-03-23	2024-04-16
WF005	California Wildfire 2024	California	62334	2024-09-18	2024-10-25
WF006	Colorado Wildfire 2024	Colorado	77880	2024-12-27	2024-12-31
WF007	Connecticut Wildfire 2024	Connecticut	43585	2024-11-26	2024-12-07
WF008	Delaware Wildfire 2024	Delaware	52820	2024-11-28	2024-12-26
WF009	Florida Wildfire 2024	Florida	82023	2024-09-17	2024-11-09

Victi...	Victim_Name	Age	Location	Shelter_ID
010001	James Smith	22	1000 Compassion Way, Alabama, AL 38221	0101
010002	Mary Johnson	35	1000 Compassion Way, Alabama, AL 38221	0101
010003	Linda Davis	47	1000 Compassion Way, Alabama, AL 38221	0101
020001	Patricia Brown	34	1003 Harmony St, Alaska, AK 59615	0201
020002	Michael Smith	29	1003 Harmony St, Alaska, AK 59615	0201
020003	Elizabeth Davis	58	1003 Harmony St, Alaska, AK 59615	0201
020004	John White	10	1004 Rescue Blvd, Alaska, AK 57400	0202
020005	Jennifer Hernandez	58	1004 Rescue Blvd, Alaska, AK 57400	0202
020006	Robert Martinez	41	1005 Harmony St, Alaska, AK 96673	0203

Resource_ID	Resource_Name	Status	Amount	Donation_ID	Org_ID	Request_ID
10023	Water Bottle	In Stock	1410	DON001	ORG001	NULL
12340	Canned Goods	In Stock	2059	DON002	ORG002	NULL
65234	Blanket	In Stock	1308	DON003	ORG003	NULL
70123	Toys	In Stock	873	DON006	ORG001	NULL
83456	Beds	In Stock	477	DON004	ORG004	NULL

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%';
```

Victim_ID	Victim_Name	Age	Location	Shelter_ID	Aid_Request_ID	Aid_Request_Stat...	Resource_ID	Quantity_Request...	Resource_Request_Stat...
020002	Michael Smith	29	1003 Harmony St, Alaska, AK 59615	0201	54320	In Progress	10023	50	In Progress
040001	Matthew Perez	64	1009 Compassion Way, Arkansas, AR 80010	0401	54310	In Progress	10023	49	In Progress
050005	Michelle Carter	56	1013 Shelter Rd, California, CA 37869	0502	54301	In Progress	10023	100	In Progress
050009	Melissa Lopez	39	1015 Rescue Blvd, California, CA 37653	0504	NULL	NULL	NULL	NULL	NULL
050013	Melissa Johnson	73	1019 Shelter Rd, California, CA 90736	0508	NULL	NULL	NULL	NULL	NULL

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
```

```sql
  , 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;
```

Resource_ID	Resource_Name	Donated_Amou...	Completed_Amou...	Percent_Used
10023	Water Bottle	2000.00	590	29.50
12340	Canned Goods	2000.00	441	22.05
65234	Blanket	700.00	192	27.43
70123	Toys	1000.00	127	12.70
83456	Beds	500.00	23	4.60

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

```sql
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;
```

Resource_ID	Resource_Name	Total_Request...
10023	Water Bottle	1849
12340	Canned Goods	2021
65234	Blanket	1400
70123	Toys	703
83456	Beds	123

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;
```

Resource_ID	Resource_Name	Current_Amou...	Total_Amount	Percent_Remaining
10023	Water Bottle	151	2000.00	7.55
12340	Canned Goods	479	2000.00	23.95
65234	Blanket	100	700.00	14.29
70123	Toys	297	1000.00	29.70
91234	Pereshable Goods	637	2000.00	31.85

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;
```

Shelter_ID	Shelter_Name	Num_Victims
1001	Georgia Harbor of Warmth	5
1002	Georgia Caring Hands Shelter	2
1003	Georgia Oasis	0
1004	Georgia Homeless Relief Center	0
101	Alabama Caring Hands Shelter	0

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;
```

Victim_ID	Victim_Name	Age
030005	Daniel King	31
050010	Donald Brown	31
165355	Alice Clark	31
050004	Brian Rivera	32
067428	Emma Wright	32

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;
```

Disaster_ID	Disaster_Name	Location	Severity	Start_Date	End_Date
WF005	California Wildfire 2024	California	62334	2024-09-18	2024-10-25
WF036	Oklahoma Wildfire 2024	Oklahoma	64822	2024-12-25	2024-12-26
WF042	Tennessee Wildfire 2024	Tennessee	68905	2024-12-04	2024-12-17
WF001	Alabama Wildfire 2024	Alabama	69142	2024-07-12	2024-09-05
WF031	New Mexico Wildfire 2024	New Mexico	70697	2024-01-27	2024-03-13

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;
```

Shelter_ID	Shelter_Name	Capacity	Location	Contact	Shelter_Type
[illegible]	Georgia Caring Hands Shelter	[illegible]	[illegible] Harmony St, Georgia, GA [illegible]	[illegible]	Permanent
1003	Georgia Oasis	55	1035 Rescue Blvd, Georgia, GA 73702	(529) 555-1035	Permanent
1004	Georgia Homeless Relief Center	75	1036 Harmony St, Georgia, GA 73564	(769) 555-1036	Permanent
101	Alabama Caring Hands Shelter	141	1002 Rescue Blvd, Alabama, AL 44671	(244) 555-1002	Unknown
102	Alabama Sanctuary of Care	137	1000 Compassion Way, Alabama, AL 38221	(981) 555-1000	Unknown
103	Alabama Community Refuge	136	1001 Harmony St, Alabama, AL 59797	(299) 555-1001	Unknown

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;
```

Shelter_ID	Shelter_Name	Capacity	Current_Occupan...	Available_Space
1707	Kentucky Safe Harbor	64	3	61
1902	Maryland Rescue & Relief	198	5	193
4401	Utah New Beginnings Home	187	5	182
504	California New Beginnings Home	130	0	130
511	California Haven of Hope	58	0	58

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
  )
);
```

Victim_ID	Victim_Name	Shelter_ID
[illegible]	Robert Baker	[illegible]
120992	Zoe Hill	1201
125763	Eli Cooper	1201
128501	Mila Thompson	1201
164558	Nina Vargas	1602
165733	Ethan O'Brien	1602

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;
```

Resource_ID	Resource_Name	Donated_Amou...	Requested_Amou...
12340	Canned Goods	2000.00	2021
65234	Blanket	700.00	1400

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
);
```

Victim_ID	Victim_Name	Shelter_ID
[illegible]	[illegible]	[illegible]
050003	Carol Hall	0501
050004	Brian Rivera	0502
050005	Michelle Carter	0502
050023	Karen Walker	0502
050024	Joseph Young	0501

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
);
```

Request_ID	Status	Resource_ID
54228	In Progress	12340
54227	Completed	12340
54226	In Progress	12340
54225	In Progress	12340
54224	In Progress	12340

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.