Hospital Emergency Response System: Data Modeling and Integration

Summary Report

Contents

1. Executive Summary

This project report presents a thorough overhaul of the Hospital Emergency Response System (HERS), emphasizing significant improvements in emergency medical response through a comprehensive relational database approach. By implementing MySQL, the system effectively managed complex data interactions, which greatly enhanced rapid response capabilities and the accuracy of treatments provided. Additionally, exploratory efforts with MongoDB revealed its potential for handling unstructured data, though MySQL proved more effective for maintaining data integrity and operational efficiency within structured data environments.

Further analysis using Python provided detailed visualizations that revealed 'Medication' as the most commonly administered treatment. This insight points to critical areas for improving resource allocation and training for medical staff. The project also demonstrated the importance of a hybrid database approach, combining the data integrity features of MySQL with the scalability and flexibility of MongoDB. This approach is recommended to ensure the system remains robust and capable of adapting to the diverse demands of modern healthcare emergencies.

Through meticulous record-keeping and data integration—linking details from emergency calls through to final treatment outcomes—HERS has established a model that significantly reduces data silos and enhances the overall efficiency of medical emergency responses. This system not only ensures comprehensive and accessible data but also improves patient care outcomes, making it a valuable advancement in healthcare technology .

2. Introduction

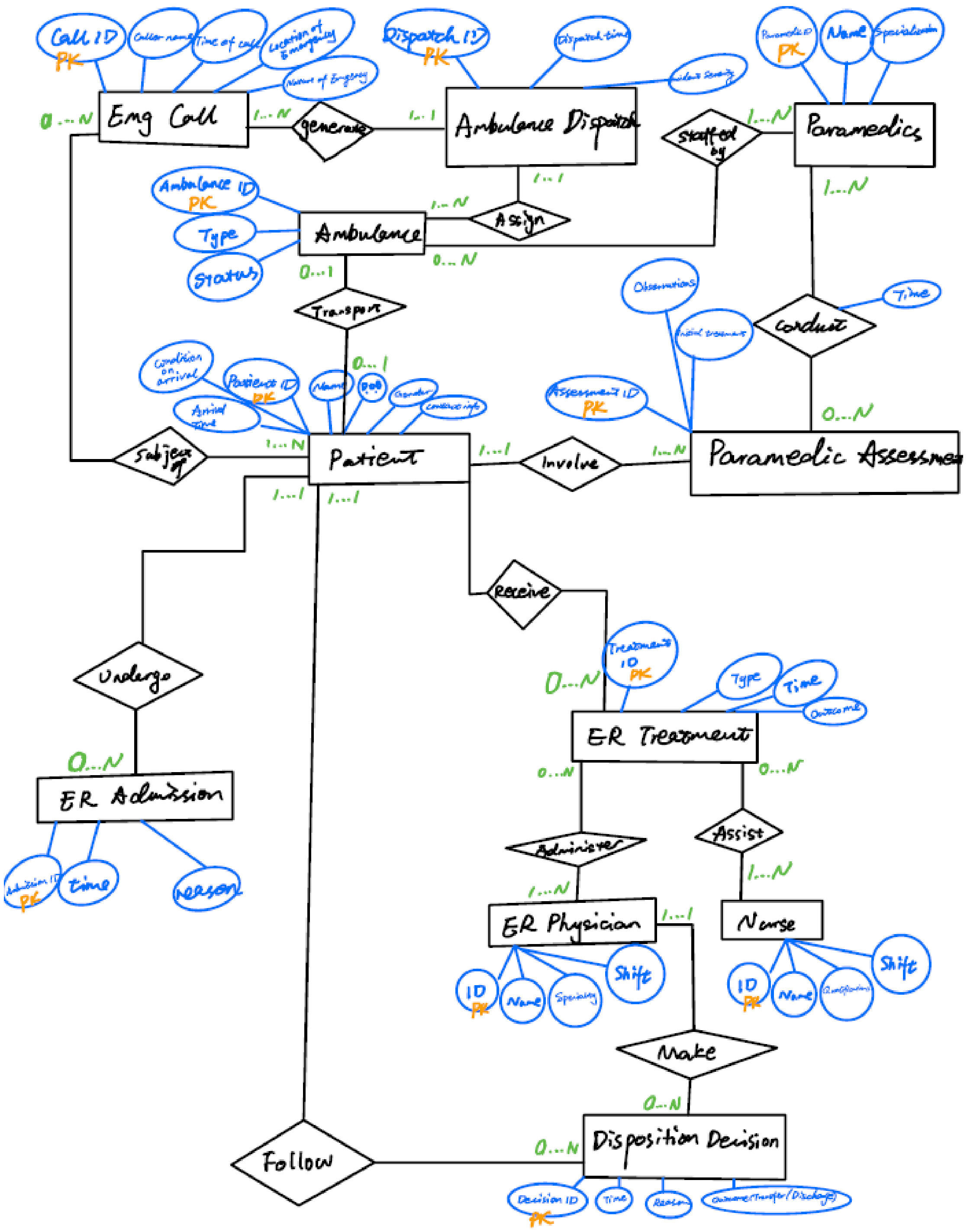
In the current healthcare environment, characterized by an array of unforeseen challenges ranging from natural disasters to public health crises, the need for a well-coordinated and efficient Hospital Emergency Response System (HERS) has never been more critical. Such systems are fundamental in ensuring the safety and well-being of patients, staff, and the community during emergencies.

At present, the HERS in many hospitals operates through a fragmented process that spans from the emergency call to the final disposition decision. Specifically, it involves multiple stages, including emergency dispatch, on-site assessment by paramedics, patient admission to the emergency room (ER), treatment, and final disposition decisions regarding patient care continuity. Each of these stages generates crucial data that, unfortunately, often remains siloed within specific operational segments. This fragmentation can lead to significant inefficiencies, as vital information about the patient's condition, treatment decisions, and outcomes is not seamlessly integrated across the HERS. Moreover, the current system is prone to other critical issues such as loss of records, difficulty in reconciling data across different stages, and redundancy in the information collected, further complicating the emergency management process.

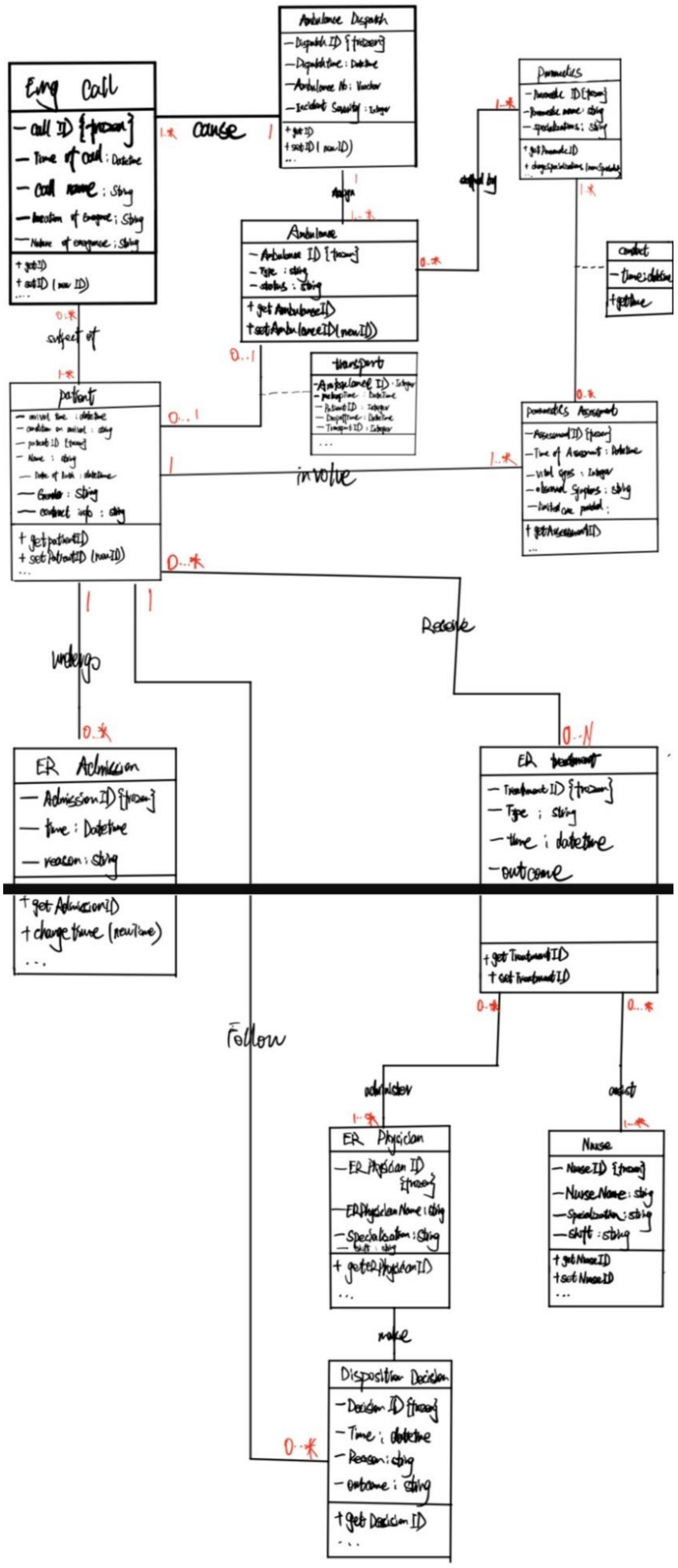
Our project proposes a comprehensive, interconnected record-keeping framework for the Hospital Emergency Response System (HERS) to address the existing inefficiencies and data gaps. From the moment an emergency call is logged, detailing the caller's information, time, location, and nature of the emergency, through the ambulance dispatch process where both incident severity and dispatch specifics are recorded, each stage is meticulously documented. This system further captures data on the ambulance details, including type and status, and patient information such as condition on arrival and demographics. Crucial interactions such as patient pickup and drop-off by the ambulance, assessments made by paramedics, treatments administered in the emergency room, and final disposition decisions are all linked via well-defined relational databases that ensure no critical information is missing or redundantly collected. This streamlined record-keeping ensures that data is not only comprehensive but also easily accessible, facilitating swift and accurate emergency responses, enhancing treatment accuracy, and improving overall patient care efficiency.

3. Conceptual modeling

1. EER Diagram



2. UML Diagram



4. Mapping Conceptual Model to Relational Model

Primary Key: Underlined **Foreign Key: Italic**

1. Ambulance Dispatch (**Dispatch ID**, Dispatch time, Incident severity)
2. Emergency Call (**Call-ID**, Caller Name, Time of Call, Location of Emergency, Nature of Emergency, *Dispatch ID*)

   Dispatch ID is the foreign key referring to the primary key in Ambulance Dispatch. Not Null.

3. Ambulance (**Ambulance ID,** Type, Status, *Dispatch ID, Patient ID*)
   - *i. Dispatch ID is the foreign key referring to the primary key in Ambulance Dispatch. Not Null.*
   - *ii. Patient ID is the foreign key referring to the primary key in Patient. Can be Null.*
4. Patient (**Patient ID**, Name, D.O.B, Gender, Contact info, ConditionOnArrival, Arrival time)
5. Transport (**Ambulance ID, Patient ID**, Pickup Time, Dropoff Time)
   - *i. Ambulance ID is the foreign key referring to the primary key in Ambulance. Not null*
   - *ii. Patient ID is the foreign key referring to the primary key in Patient. Not null.*
6. Subject_of (**Patient ID, Call-ID**)
   - *i. Patient ID is the foreign key referring to the primary key in Patient. Not null.*
   - *ii. Call-ID is the foreign key referring to the primary key in Emergency Call. Not null.*
7. Paramedics (**Paramedic ID**, Name, Specialization)
8. Staffed_by (**Ambulance ID**, **Paramedic ID**)
   - *i. Ambulance ID is the foreign key referring to the primary key in Ambulance. Not null*
   - *ii. Paramedic ID is the foreign key referring to the primary key in Paramedics. Not null*
9. Paramedics Assessment (**Assessment ID**, Observations, Initial Treatment, Time, *Patient ID*)

Patient ID is the foreign key referring to the primary key in Patient. Not null.

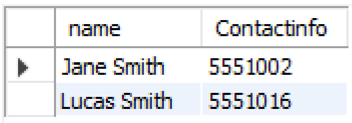
10. Conduct (**<u>Assessment ID, Paramedic ID</u>**, time**)**
    i. *Assessment ID is the foreign key referring to the primary key in Paramedics Assessment. Not null*
    ii. *Paramedic ID is the foreign key referring to the primary key in Paramedics. Not null*
11. ER Admission (**<u>Admission ID</u>**, Time, Reason, *Patient ID*)
    Patient ID is the foreign key referring to the primary key in Patient. Not null.
12. ER Treatment （**<u>Treatment ID</u>**, Type, Time, Outcome, *Patient ID*)
    Patient ID is the foreign key referring to the primary key in Patient. Not null.
13. ER Physician (**<u>ID</u>**, Name, Specialty, Shift)
14. Administer (**<u>ID</u>**, **<u>Treatment ID)</u>**
    i. *ID is the foreign key referring to the primary key in ER Physician. Not null.*
    ii. *Treatment ID is the foreign key referring to the primary key in ER Treatment. Not null.*
15. Nurse (**<u>ID</u>**, Name, Qualifications, Shift)
16. Assist (**<u>ID</u>**, **<u>Treatment ID)</u>**
    i. *ID is the foreign key referring to the primary key in Nurse. Not null.*
    ii. *Treatment ID is the foreign key referring to the primary key in ER Treatment. Not null.*
17. Disposition Decision (**<u>Decision ID</u>**, Time, Reason, Outcome, *ID, Patient ID*)
    i. *ID is the foreign key referring to the primary key in ER Physician. Not null.*
    ii. *Patient ID is the foreign key referring to the primary key in Patient. Not null.*

5. Implementation of Relational Model via MySQL

The "Hospital Emergency Response System" database (hereinafter "HERS") was established utilizing DDL (Data Definition Language) and DML (Data Manipulation Language). The subsequent queries were carried out:

Q1 (Simple Query): find the names and contact information of patients who have "Smith" as part of their names.

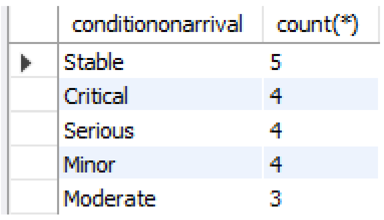
```sql
select name, Contactinfo
from Patient
where name like "%Smith%";
```



name	Contactinfo
Jane Smith	5551002
Lucas Smith	5551016

Q2 (Aggregate Query): counts the number of patients based on their condition upon arrival at the hospital.

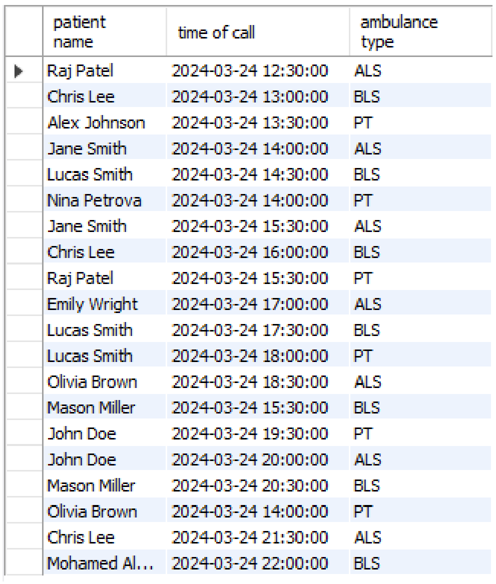
```sql
select conditiononarrival, count(*)
from patient
group by conditiononarrival;
```



conditiononarrival	count(*)
Stable	5
Critical	4
Serious	4
Minor	4
Moderate	3

Q3 (Inner Join Query): find patient names, times of their emergency calls, and the types of the ambulances dispatched to them.

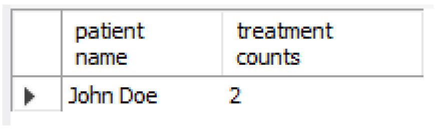
```sql
select p.name as "patient name", e.`time of call`, a.Type as "ambulance type"
from patient p
join ambulance a on p.patient_ID = a.patientID
join `emergency call` e on a.dispatchID = e.dispatchID;
```



patient name	time of call	ambulance type
Raj Patel	2024-03-24 12:30:00	ALS
Chris Lee	2024-03-24 13:00:00	BLS
Alex Johnson	2024-03-24 13:30:00	PT
Jane Smith	2024-03-24 14:00:00	ALS
Lucas Smith	2024-03-24 14:30:00	BLS
Nina Petrova	2024-03-24 14:00:00	PT
Jane Smith	2024-03-24 15:30:00	ALS
Chris Lee	2024-03-24 16:00:00	BLS
Raj Patel	2024-03-24 15:30:00	PT
Emily Wright	2024-03-24 17:00:00	ALS
Lucas Smith	2024-03-24 17:30:00	BLS
Lucas Smith	2024-03-24 18:00:00	PT
Olivia Brown	2024-03-24 18:30:00	ALS
Mason Miller	2024-03-24 15:30:00	BLS
John Doe	2024-03-24 19:30:00	PT
John Doe	2024-03-24 20:00:00	ALS
Mason Miller	2024-03-24 20:30:00	BLS
Olivia Brown	2024-03-24 14:00:00	PT
Chris Lee	2024-03-24 21:30:00	ALS
Mohamed Al...	2024-03-24 22:00:00	BLS

Q4 (Nested Query): Find the names of patients who have been treated more than the average number of treatments received by all patients.

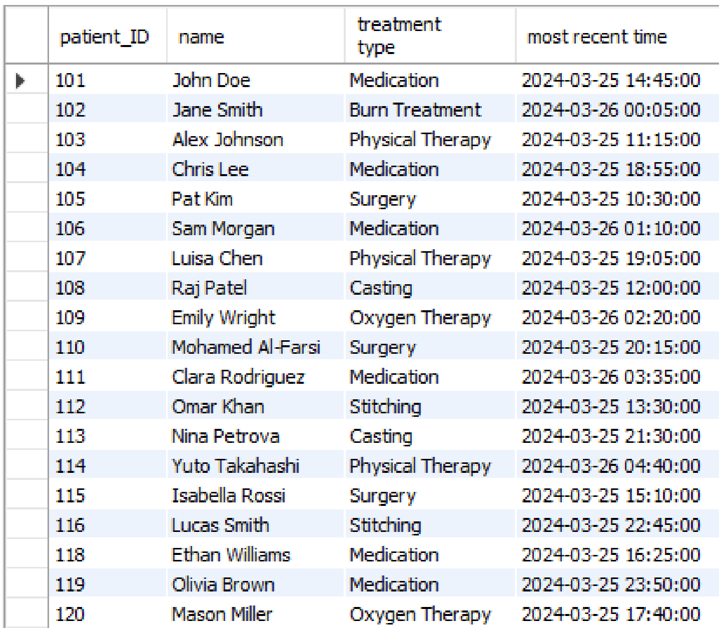
```
select p.name as "patient name", count(t.TreatmentID) as "treatment counts"
from patient p
join `er treatment` t on p.patient_ID = t.patientID
group by p.patient_ID, p.name
having count(t.TreatmentID) > (
    select avg(treatmentcount) from (
        select count(t1.treatmentID) as treatmentcount
        from `er treatment` t1
        group by patientID
        ) as subquery
    );
```



patient name	treatment counts
John Doe	2

Q5 (Correlated Query): find the type of the most recent treatment for each patient.

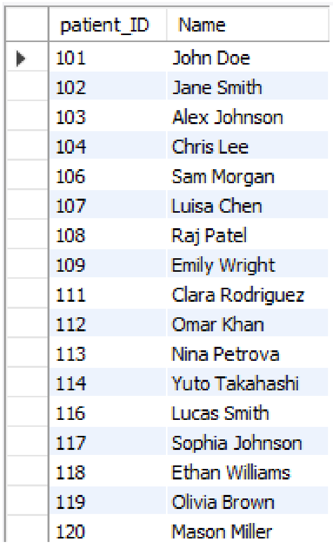
```
select p.patient_ID, p.name, t.type as "treatment type", t.time as "most recent time"
from patient p
join `er treatment` t on p.patient_ID = t.patientID
where t.time = (
    select max(t1.time)
    from  `er treatment` t1
    where t1.patientID = p.patient_ID
    )
order by p.patient_ID;
```



patient_ID	name	treatment type	most recent time
101	John Doe	Medication	2024-03-25 14:45:00
102	Jane Smith	Burn Treatment	2024-03-26 00:05:00
103	Alex Johnson	Physical Therapy	2024-03-25 11:15:00
104	Chris Lee	Medication	2024-03-25 18:55:00
105	Pat Kim	Surgery	2024-03-25 10:30:00
106	Sam Morgan	Medication	2024-03-26 01:10:00
107	Luisa Chen	Physical Therapy	2024-03-25 19:05:00
108	Raj Patel	Casting	2024-03-25 12:00:00
109	Emily Wright	Oxygen Therapy	2024-03-26 02:20:00
110	Mohamed Al-Farsi	Surgery	2024-03-25 20:15:00
111	Clara Rodriguez	Medication	2024-03-26 03:35:00
112	Omar Khan	Stitching	2024-03-25 13:30:00
113	Nina Petrova	Casting	2024-03-25 21:30:00
114	Yuto Takahashi	Physical Therapy	2024-03-26 04:40:00
115	Isabella Rossi	Surgery	2024-03-25 15:10:00
116	Lucas Smith	Stitching	2024-03-25 22:45:00
118	Ethan Williams	Medication	2024-03-25 16:25:00
119	Olivia Brown	Medication	2024-03-25 23:50:00
120	Mason Miller	Oxygen Therapy	2024-03-25 17:40:00

Q6 (not exists): Find all patients who have never undergone a surgical procedure.

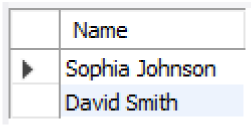
```
select p.patient_ID, p.Name
from patient p
where not exists (
    select 1
    from `er treatment` t
    where p.patient_ID = t.patientID and t.type = "Surgery"
    );
```



patient_ID	Name
101	John Doe
102	Jane Smith
103	Alex Johnson
104	Chris Lee
106	Sam Morgan
107	Luisa Chen
108	Raj Patel
109	Emily Wright
111	Clara Rodriguez
112	Omar Khan
113	Nina Petrova
114	Yuto Takahashi
116	Lucas Smith
117	Sophia Johnson
118	Ethan Williams
119	Olivia Brown
120	Mason Miller

Q7 (Set Operation): Find all unique individuals assessed with 'hypothermia' by paramedics or who reported 'hypothermia' during an emergency call.

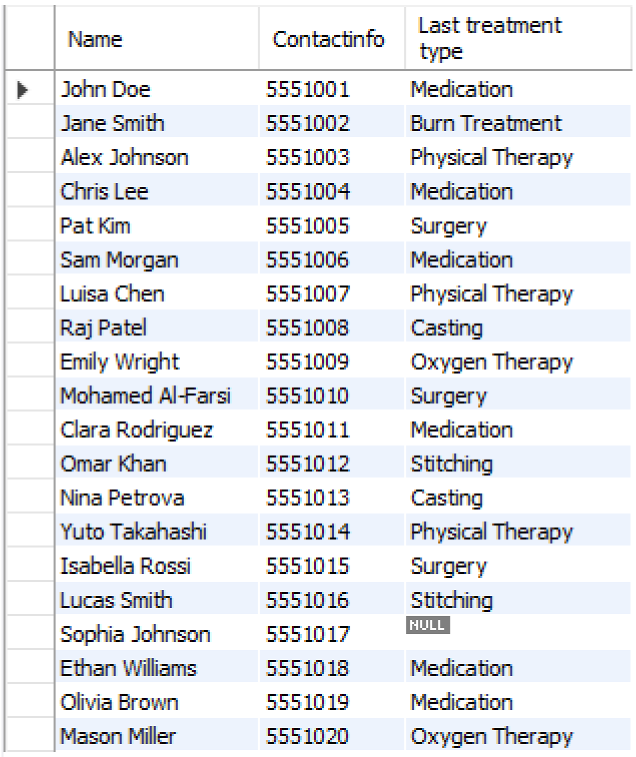
```
(select p.Name
from patient p
join `paramedics assessment` pa on p.patient_ID = pa.`patient ID`
where pa.observations = 'hypothermia')
union
(select `caller name`
from `emergency call`
where  `nature of emergency` = 'hypothermia');
```



Name
Sophia Johnson
David Smith

Q8 (Subqueries in Select): Find each patient's name, contact information, and the type of their most recent treatment (no matter they have or not).

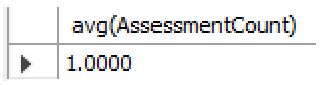
```
select p.Name, p.Contactinfo, (
    select t.type
    from `er treatment` t
    where p.patient_ID = t.patientID
    order by t.time desc
    limit 1) as "Last treatment type"
from patient p;
```



Name	Contactinfo	Last treatment type
John Doe	5551001	Medication
Jane Smith	5551002	Burn Treatment
Alex Johnson	5551003	Physical Therapy
Chris Lee	5551004	Medication
Pat Kim	5551005	Surgery
Sam Morgan	5551006	Medication
Luisa Chen	5551007	Physical Therapy
Raj Patel	5551008	Casting
Emily Wright	5551009	Oxygen Therapy
Mohamed Al-Farsi	5551010	Surgery
Clara Rodriguez	5551011	Medication
Omar Khan	5551012	Stitching
Nina Petrova	5551013	Casting
Yuto Takahashi	5551014	Physical Therapy
Isabella Rossi	5551015	Surgery
Lucas Smith	5551016	Stitching
Sophia Johnson	5551017	NULL
Ethan Williams	5551018	Medication
Olivia Brown	5551019	Medication
Mason Miller	5551020	Oxygen Therapy

Q9 (Subqueries in From): Find the Average number of paramedic assessments received by patients.

```
select avg(AssessmentCount)
from (
    select count(pa.assessmentID) as AssessmentCount
    from `paramedics assessment` pa
    group by pa.`patient ID`
    ) as sub;
```

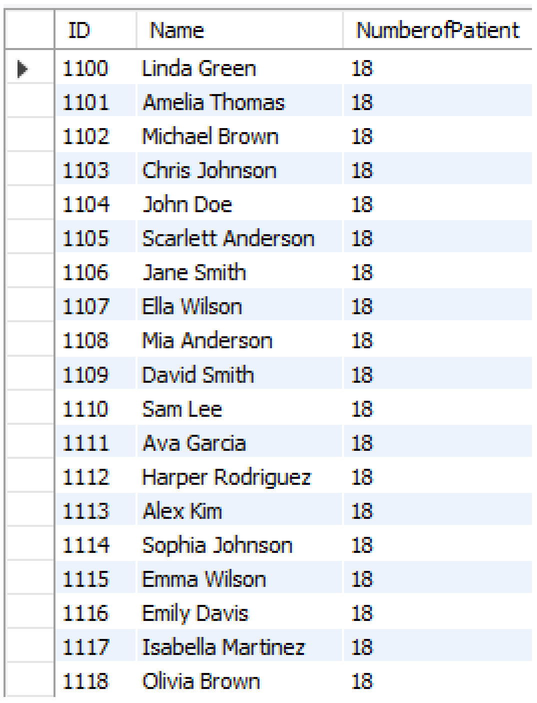


	avg(AssessmentCount)
▶	1.0000

Q10 (CTE + self-join): Find which doctors have treated the same number of patients as other doctors.

```
with DoctorTreatCounts as(
    select a.ID as DoctorID, count(distinct t.patientID) as NumberofPatient
    from `er treatment` t
    join administer a on a.treatmentID = t.TreatmentID
    group by a.ID)

select distinct phy.ID, phy.Name, d1.NumberofPatient
from `er physician` phy
join DoctorTreatCounts d1 on phy.ID = d1.doctorID
join DoctorTreatCounts d2 on d1.NumberofPatient = d2.NumberofPatient
where d1.doctorID < d2.doctorID;
```



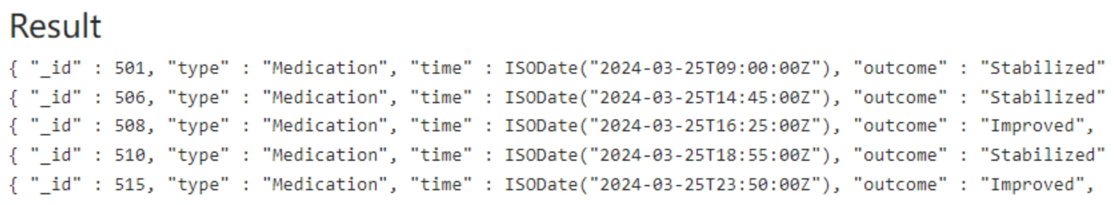
	ID	Name	NumberofPatient
▶	1100	Linda Green	18
	1101	Amelia Thomas	18
	1102	Michael Brown	18
	1103	Chris Johnson	18
	1104	John Doe	18
	1105	Scarlett Anderson	18
	1106	Jane Smith	18
	1107	Ella Wilson	18
	1108	Mia Anderson	18
	1109	David Smith	18
	1110	Sam Lee	18
	1111	Ava Garcia	18
	1112	Harper Rodriguez	18
	1113	Alex Kim	18
	1114	Sophia Johnson	18
	1115	Emma Wilson	18
	1116	Emily Davis	18
	1117	Isabella Martinez	18
	1118	Olivia Brown	18

6. Implementation of Relational Model via NoSQL

Three collections—ambulanceDispatch, nurse, and erTreatment—were set up in MongoDB Playground, where the following three queries were executed:

Query 1 (Simple Query): find the records of the first 5 Medication treatments.

```
db.erTreatment.find({"type":"Medication"}).sort({"time":1}).limit(5);
```

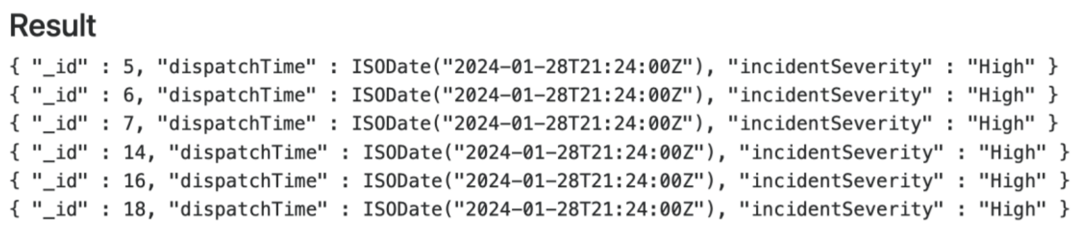


Result

```
{ "_id" : 501, "type" : "Medication", "time" : ISODate("2024-03-25T09:00:00Z"), "outcome" : "Stabilized"
{ "_id" : 506, "type" : "Medication", "time" : ISODate("2024-03-25T14:45:00Z"), "outcome" : "Stabilized"
{ "_id" : 508, "type" : "Medication", "time" : ISODate("2024-03-25T16:25:00Z"), "outcome" : "Improved",
{ "_id" : 510, "type" : "Medication", "time" : ISODate("2024-03-25T18:55:00Z"), "outcome" : "Stabilized"
{ "_id" : 515, "type" : "Medication", "time" : ISODate("2024-03-25T23:50:00Z"), "outcome" : "Improved",
```

Query 2 (Complex Query): Find high-severity incidents in the ambulance dispatch records that occurred on January 28th, 2024, at 9:24 PM.

```
db. ambulanceDispatch.find(
  {$and: [
    {"incidentSeverity": "High"},
    {"dispatchTime": ISODate("2024-01-28T21:24:00Z")}]
  });
```



Result

```
{ "_id" : 5, "dispatchTime" : ISODate("2024-01-28T21:24:00Z"), "incidentSeverity" : "High" }
{ "_id" : 6, "dispatchTime" : ISODate("2024-01-28T21:24:00Z"), "incidentSeverity" : "High" }
{ "_id" : 7, "dispatchTime" : ISODate("2024-01-28T21:24:00Z"), "incidentSeverity" : "High" }
{ "_id" : 14, "dispatchTime" : ISODate("2024-01-28T21:24:00Z"), "incidentSeverity" : "High" }
{ "_id" : 16, "dispatchTime" : ISODate("2024-01-28T21:24:00Z"), "incidentSeverity" : "High" }
{ "_id" : 18, "dispatchTime" : ISODate("2024-01-28T21:24:00Z"), "incidentSeverity" : "High" }
```

Query 3 (Aggregate Method): find the treatment types that occurred more than once.

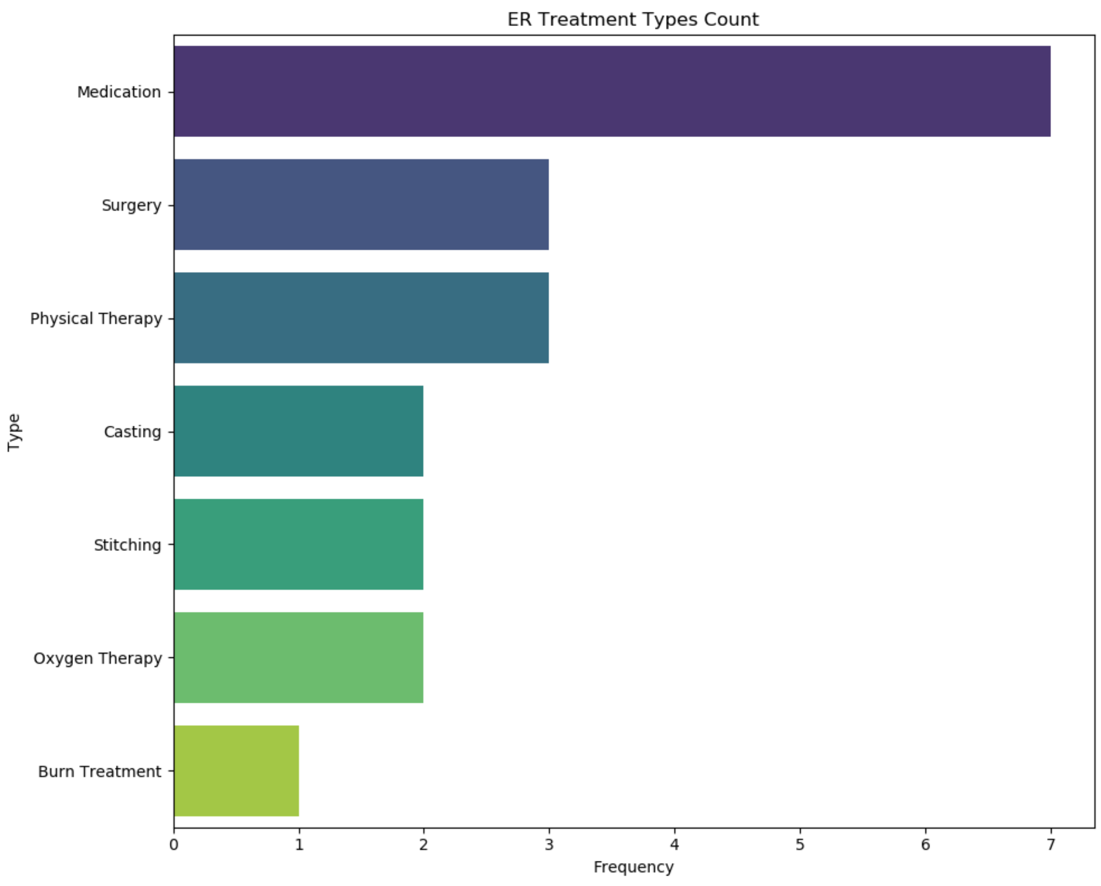
```
db.erTreatment.aggregate([
  {$group: {_id: "$type" , count: {$sum: 1}}}, // group by type and
count occurrences
  {$match: {count: {$gt: 1}}}, // filter types having more than one
occurrence
  {$sort: {count: -1}} // sort by count in descending order
  ]);
```

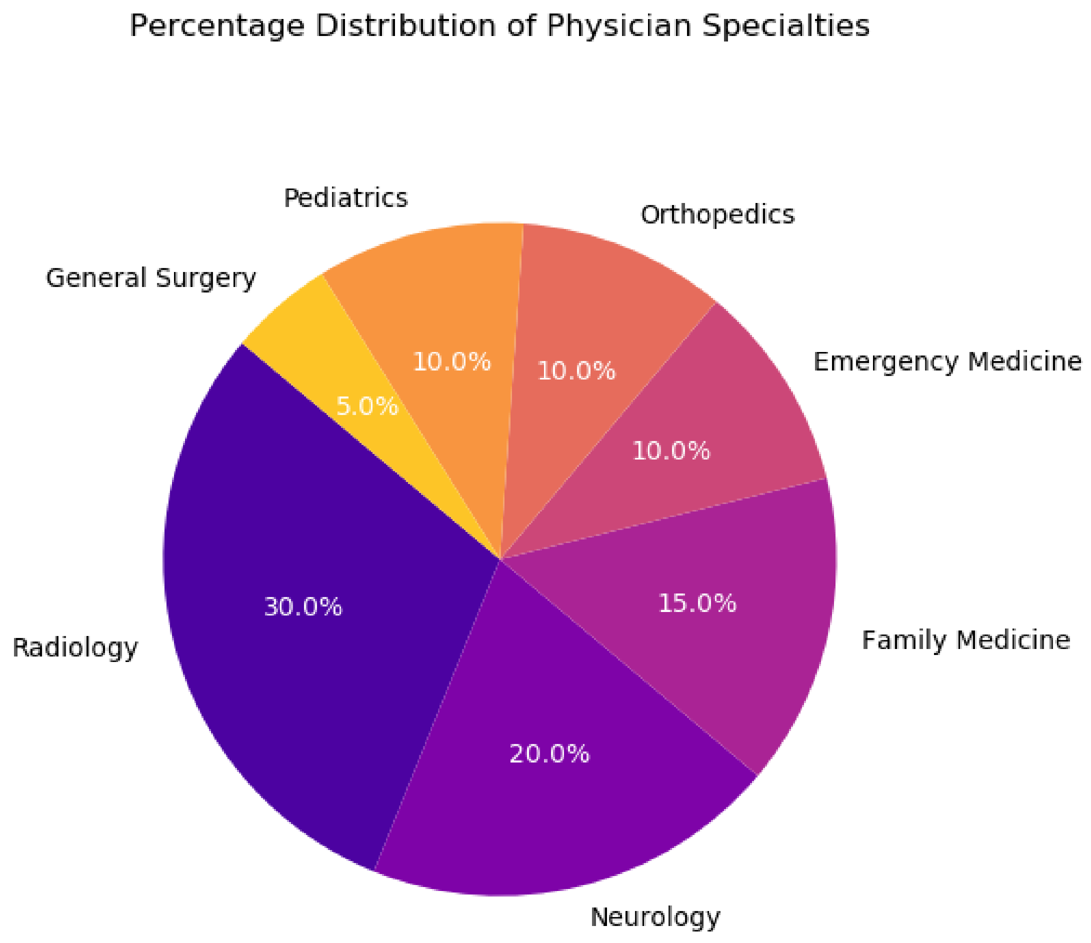
Result

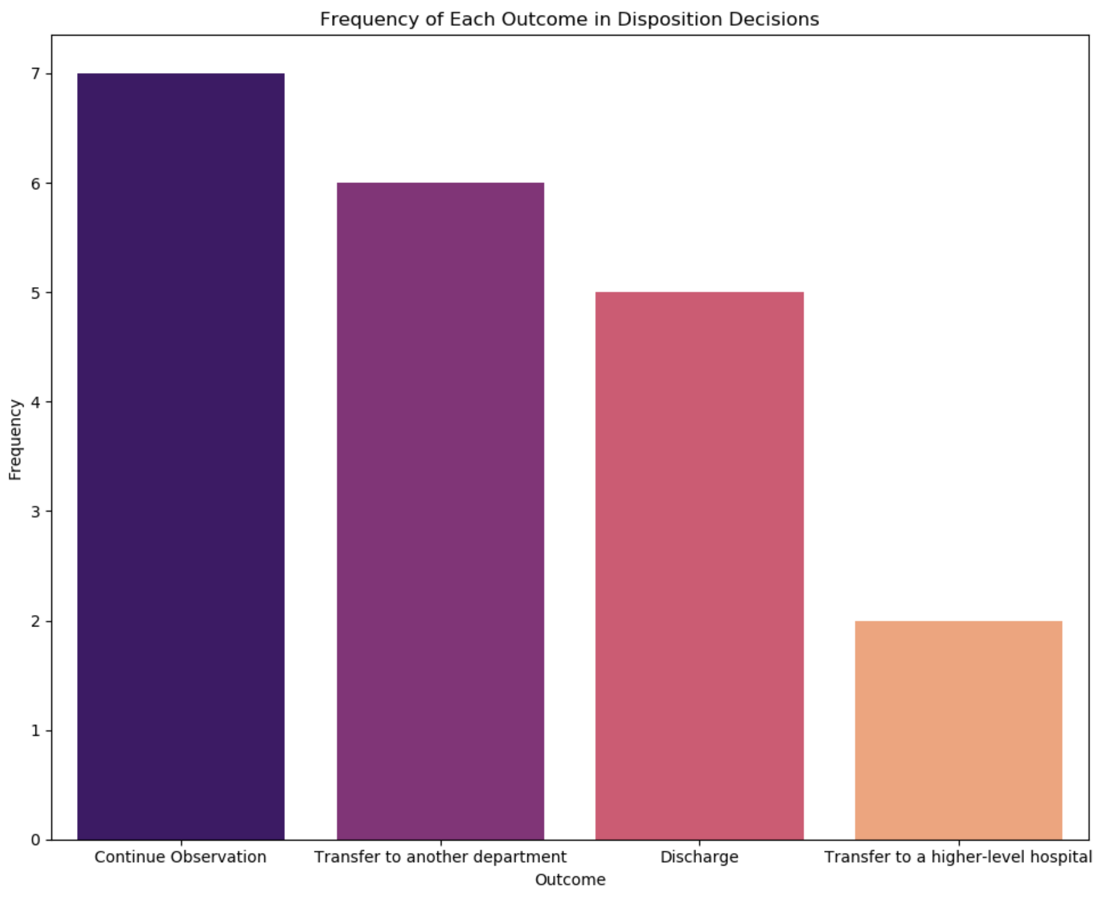
```
{ "_id" : "Medication", "count" : 7 }
{ "_id" : "Surgery", "count" : 3 }
{ "_id" : "Physical Therapy", "count" : 3 }
{ "_id" : "Oxygen Therapy", "count" : 2 }
{ "_id" : "Stitching", "count" : 2 }
{ "_id" : "Casting", "count" : 2 }
```

7. Database Access via Python

To visualize the data from “HERS” database, Python was used to access the database and perform the data analysis. A connection to MySQL was established using **mysql.connector**, and SQL queries specific to each data analysis case were run to retrieve the necessary information. Instead of using **cursor.execute** and **fetchall**, the data retrieval was directly converted into a pandas DataFrame using the **pd.read_sql** function, which streamlines the fetching and dataframe conversion process. For the visualization, Matplotlib, in conjunction with Seaborn, was utilized to plot the graphs that represent the analytics of ER physician specialties, ER treatment types, and disposition decisions.




Percentage Distribution of Physician Specialties
Pediatrics
Orthopedics
General Surgery
Emergency Medicine
Radiology
Family Medicine
Neurology
10.0%
10.0%
5.0%
10.0%
30.0%
15.0%
20.0%



The graph indicates that 'Medication' is the most frequent type of treatment, suggesting the supply chain for medications is robust. It might be beneficial to consider hiring or training more emergency medicine specialists, given the high demand for medication. The prevalent outcome of 'Continue Observation' in disposition decision could reflect a cautious approach to patient treatment or an abundance of cases that are not severe.

8. Conclusion and Recommendation

The report outlines the modeling of the Hospital Emergency Response System (HERS), significantly refining the process of emergency data management. By integrating a relational model, the system ensures seamless data connectivity from the initial emergency call to the patient's final care decision, preserving essential information and reducing redundancy. The implementation via MySQL highlighted the system's ability to handle complex queries efficiently, contributing to shorter response times and more accurate treatment delivery. Additionally, trials with MongoDB explored its capabilities with unstructured data and potential scalability. However, it was determined that the relational database model offers more substantial benefits for this application.

Further visualization analysis using Python, as in the report's graphical content, reveals that 'Medication' is the most frequently administered treatment, indicating areas where resource allocation and staff training could be optimized. Predictive analytics could also be used to better anticipate resource needs and improve preparedness for various emergency scenarios.

While the NoSQL approach, exemplified by databases like MongoDB, enhances flexibility and scalability within data management systems, it introduces significant limitations in terms of relational integrity and data traceability. This shift away from the structured, relational model leads to a reduction in data consistency. In a relational database setup, structured relationships enforced through foreign keys ensure robust data accuracy and traceability from one table to another, such as linking 'Patient' and 'Treatment' tables. However, the schema-less nature of NoSQL databases compromises these relational guarantees. Although data retrieval might be faster due to the absence of rigid schema constraints, maintaining data accuracy across documents becomes challenging, as the direct relationships necessary for traceability and consistency are diminished. This lack of enforced data relationships poses substantial challenges for ensuring the integrity and reliability of critical data within systems that require precise and dependable data management.

As a result, a hybrid system approach is recommended for effectively satisfying diverse data needs. This strategy involves using MySQL to handle structured data and maintain critical relational dependencies crucial to business logic, while employing MongoDB to manage flexible, high-volume unstructured data such as logs and real-time analytics.