Database System

24/25

QHE5701 – Coursework

Team Leader:

Name: Beibei Liu(刘蓓蓓)

QMUL id: 221155785

BUPT id: 2022213747

Class Number: 2022217702

QMUL email: jp2022213747@qmul.ac.uk

Team Member:

Name: Baodi Han(韩宝笛)

QMUL id: 221155969

BUPT id: 2022213764

Class Number: 2022217703

QMUL email: jp2022213764@qmul.ac.uk

Abstract

This report includes content generated or assisted by OpenAI's ChatGPT. Specifically, ChatGPT was utilized for drafting, description in Task 1, Task 3 with key information drafted by ourselves. Some detailed explanation for codes and functions are modified by ChatGPT, too. And produced data records in Task 7, as well as explanation for application in Task 9 and assisted to correct codes for the Web.

All the information provided by ChatGPT has been integrated with our own ideas and the designed structural framework. ChatGPT's role was solely to supplement and refine the details.

The entire sql queries are shown in SQL_Query.sql, queries in report are only examples.

Application

- Run the Flask in main.py before the htmls.
- All accounts should start from login page, or the route would not able to identify which page to direct to.
- Replace the URL with(or the webpage would not be able to run):

  http://127.0.0.1:5000

- Select an account you want (either is OK, just choose the way you want to focus on);

```
-- This will focus on the staff name

SELECT * FROM UserRole;

-- This will focus on the staff Role

SELECT * FROM UserRoleMapping;
```

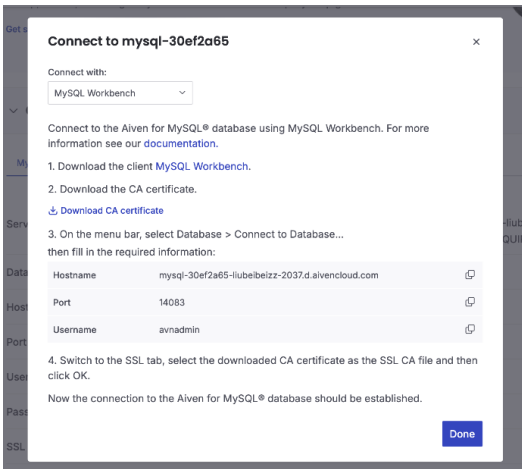
- Check the account is valid (There are some staffs were part-timed who has several roles, we are not going to interest in them now. Use codes below to know how many roles this person has involved.) and its password.

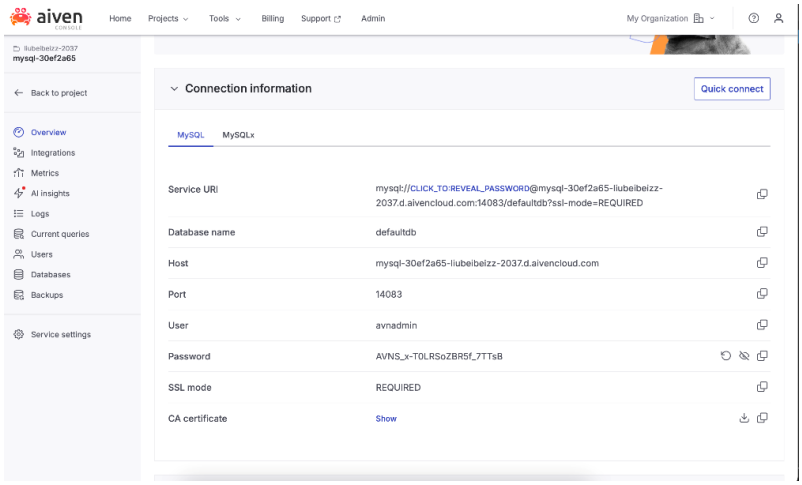
```
SELECT ur.UserID, ur.Password, urm.RoleID

FROM UserRole ur

JOIN UserRoleMapping urm ON ur.UserID = urm.UserID

WHERE ur.UserID = '6860';    -- Replace with the UserID you are interested in.
```

MySQL

When running the SQL_Query.sql, the first query for INSERT may go wrong (If you use select all and run). Just select it again and run the rest of the code, it would automatically fix the problem.

MySQL (Edit with created user account)





Please check it out in avnadmin account !

Server URL: mysql://avnadmin:AVNS_x-T0LRSoZBR5f_7TTsB@mysql-30ef2a65-liubeibeizz-2037.d.aivencloud.com:14083/defaultdb?ssl-mode=REQUIRED

Port: 14038

Username: avnadmin (You can change into other user accounts later)

Password: AVNS_x-T0LRSoZBR5f_7TTsB (the other accounts' password can be known by the UserRole table by the avnadmin account)

Change the CA certification path to ca.pem

- ✧ We can observe that account passwords that created by MySQL are invisible, accounts created on Web Page is readable.
- ✧ When encountered with any problems in created accounts （not avnadmin account）,please try the following code according to the Role checked in UserRoleMapping table:

```
SET ROLE Caregiver;

SET ROLE Nurse;

SET ROLE Admin;
```

Task 1 -Requirement Analysis:

(a) Background:

This project involves designing a database system for an elderly care facility to replace their traditional manual record-keeping method. The system will centralize the management of resident information, guardian information, care records, health records, and user roles, while ensuring the security and role-based access control of data access.

Manual record-keeping leads to scattered, redundant, and difficult-to-query data, which cannot meet the needs of expansion. The health data and personal information of residents are sensitive, and there is currently no effective security control in place. Different user roles (Caregivers, nurses, administrators) need to access different data, and the current method cannot clearly distinguish data permissions. As the institution expands, the current management method cannot support more users and data growth. Manual management is time-consuming and inefficient, with low input and query efficiency for data.

Resident information is the core entity, including personal basic information and health information. Guardian information is emergency contact information and financial payment-related information. CareLog created by caregivers to track the daily care activities of residents. HealthRecords are created by nurses to record the resident's health status and treatment history. User roles and permissions support different users' hierarchical access and management of data.

UserRole are associated with recorded data through UserIDs. Different UserRole have clear responsibilities and permissions (for example, Nurse can only manage the HeathRecord but can not manage CareLog). Database administrators need to adjust permissions dynamically to adapt to operations such as hiring or firing.

Resident information involves personal privacy (such as ID numbers, allergy histories) and health data. It must be stored in an encrypted manner and access by unauthorized users should be restricted.

(b) Assumptions:

1. **Each resident can have a maximum of two guardians, but having a guardian is optional.**

   Not all residents must have a designated guardian. If there is a guardian, there may be one or two people (such as parents, siblings or friends).

   This results in a one-to-many relationship between residents and guardians.

2. **Each resident can only be taken care of by one caregiver and one nurse at any given time, but a caregiver or a nurse can be responsible for multiple residents.**

   This means that the relationship between residents and caregivers or nurses is a many-to-one relationship. If a resident changes caregivers or nurses, the historical records still need to be retained.

3. **Each nursing record or health record corresponds to a single resident and is created by a caregiver or a nurse.**

   Each nursing record or health record corresponds to a single resident and is created by a caregiver or a nurse.

   This means that the relationship between residents and caregivers or nurses is a many-to-one relationship. If a resident changes caregivers or nurses, the historical records still need to be retained.

Task 2- Conceptual Design/ER Model:

Entities:

1.Residents{ResidentID,Name, Address, PhoneNumber, DateOfBirth, Gender, AllergyHistory, HealthNotes }

Primary key: ResidentID

2.Guardians{GuardianID, Name, Address, PhoneNumber, WeChatID}

Primary key: GuardianID

3.Residents_Guardians{Residents_ResidentID, Guardians_GuardianID, Relationship}

Primary key: Combination of Residents_ResidentID and Guardians_GuardianID

Foreign key: Residents_ResidentID references Residents(ResidentID)

Foreign key: Guardians_GuardianID references Guardians(GuardianID)

4.HealthRecord{HealthRecordID, Content, RecordTime, ResidentID, NurseID}

Primary key: HealthRecordID

Foreign key: ResidentID references Residents(ResidentID)

5.CareLog{CareLogID, Content, RecordTime, ResidentID, CaregiverID}

Primary key: CareLogID

Foreign key: ResidentID references Residents(ResidentID)

6.UserRole{UserID, Username, Password}

Primary key: UserID

7.Role{RoleID, RoleName}

Primary key: RoleID

8.UserRoleMapping{UserID, RoleID}

Primary key: Combination of UserID and RoleID

Foreign key: UserID references UserRole(UserID)

Foreign key: RoleID references Role(RoleID)

9.RolePermission{PermissionID, Permission, RoleID }

Primary key: PermissionID

Foreign key: RoleID references Role(RoleID)

Relationship:

1.Residents_Guardians (**many-to-many** relationship between **Residents** and **Guardians**)

Residents_Guardians{ResidentID, GuardianID}

Primary key: Combination of Residents_ResidentID and Guardians_GuardianID

Foreign key: Residents_ResidentID references Residents(ResidentID)

Foreign key: Guardians_GuardianID references Guardians(GuardianID)

2.HealthRecord (**one-to-many** relationship betwe**en Residents** and **HealthRecord**)

HealthRecord{HealthRecordID, ResidentID,NurseID}

Primary key: HealthRecordID

Foreign key: ResidentID references Residents(ResidentID)

3.CareLog (one-to-many relationship between **Residents** and **CareLog**)

CareLog{CareLogID, ResidentID, CaregiverID}

Primary key: CareLogID

Foreign key: ResidentID references Residents(ResidentID)

4.UserRoleMapping (many-to-many relationship between **UserRole** and **Role**)

UserRoleMapping{UserID, RoleID}

Primary key: Combination of UserID and RoleID

Foreign key: UserID references UserRole(UserID)

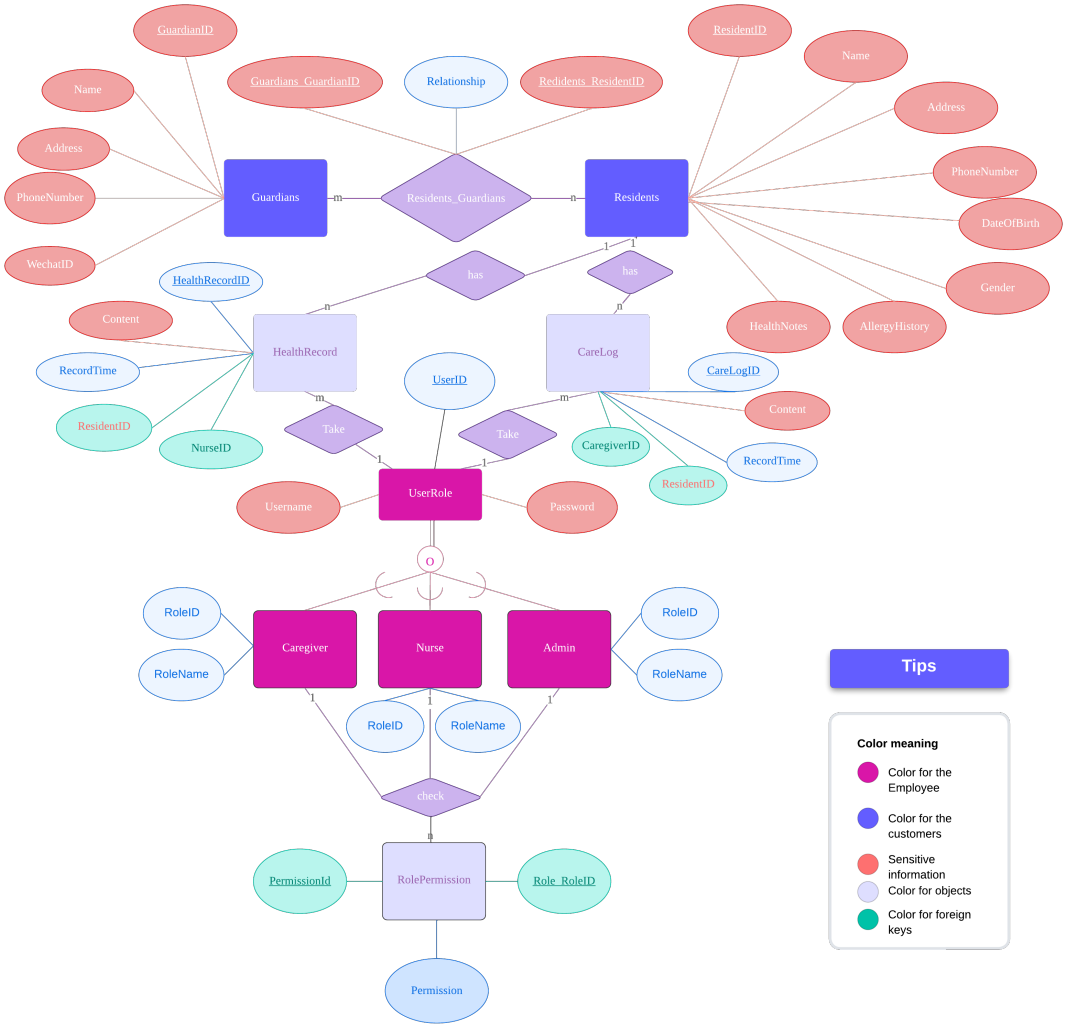
Foreign key: RoleID references Role(RoleID)

5.RolePermission (**one-to-many** relationship between **Role** and **RolePermission**)

RolePermission{PermissionID, RoleID}

- **Primary key:** PermissionID
- **Foreign key:** RoleID references Role(RoleID)

Task 3- ER Diagram



Cardinality and Participation

1. 'Residents-Guardians' Relationship

- **Structural constraints**:
  - Not every resident (min: 0) has a guardian, but a resident can have multiple **guardians** (max: M).
  - Each guardian (min: 1) must be associated with at least one resident but can be associated with multiple residents (max: N).
- **Cardinality ratio**: M:N
- **Relationship Attribute**: Relationship (e.g., parent, sibling).

- **Explanation**:
    - This is a binary **many-to-many** (M:N) relationship where each resident can optionally have none or multiple guardians, but a guardian must always be related to at least one resident.

Residents_Guardians {(ResidentID, GuardianID)} CR: M:N

---

2. 'Residents-CareLog' Relationship

- **Structural constraints**:
    - Each resident (min: 1) must have at least one care log entry (min: 1), but can have **many care logs** (max: N).
    - A care log (min: 1) must be associated with exactly **one resident** (max: 1).
- **Cardinality ratio**: 1:N
- **Relationship Attribute**: RecordTime, Content.
- **Explanation**:
    - This is a **one-to-many** (1:N) relationship where each care log belongs to one resident, but each resident can have multiple care logs.

CareLog {(ResidentID, CareLogID)} CR: 1:N

---

3. 'Residents-HealthRecord' Relationship

- **Structural constraints**:
    - Each resident (min: 1) must have at least one health record (min: 1), but can have **many health records** (max: N).
    - Each health record (min: 1) must be associated with exactly **one resident** (max: 1).
- **Cardinality ratio**: 1:N
- **Relationship Attribute**: RecordTime, Content.
- **Explanation**:
    - This is a **one-to-many** (1:N) relationship where each health record is linked to exactly one resident, but each resident can have multiple health records.

HealthRecord {(ResidentID, HealthRecordID)} CR: 1:N

---

4. 'UserRole-Role' Relationship

- **Structural constraints**:
    - Each user (min: 1) must have at least one role (min: 1) but can have multiple roles (max: N).
    - Each role (min: 1) must be assigned to at least one user (min: 1), but can be assigned to **many users** (max: N).
- **Cardinality ratio**: M:N
- **Relationship Attribute**: None.
- **Explanation**:
    - This is a **many-to-many** (M:N) relationship where users can have multiple roles, and roles can be assigned to multiple users.

UserRole {(UserID, RoleID)} CR: M:N

---

5. 'Role-RolePermission' Relationship

- **Structural constraints**:
    - Each role (min: 1) must have at least one permission (min: 1) but can have multiple permissions (max: N).
    - Each permission (min: 1) is assigned to at least one role (min: 1) but can belong to multiple roles (max: N).
- **Cardinality ratio**: M:N
- **Relationship Attribute**: None.
- **Explanation**:
    - This is a **many-to-many** (M:N) relationship where roles can be assigned multiple permissions, and permissions can apply to multiple roles.

RolePermission {(RoleID, PermissionID)} CR: M:N

---

6. 'Residents-Nurses' (via HealthRecord) Relationship

- **Structural constraints**:
    - Each health record (min: 1) must be created by one nurse (max: 1).
    - Each nurse (min: 0) can create multiple health records (max: N).
- **Cardinality ratio**: 1:N
- **Relationship Attribute**: None.
- **Explanation**:

- This is an **indirect one-to-many** (1:N) relationship where nurses create health records for residents.

HealthRecord {(NurseID, HealthRecordID)} CR: 1:N

---

7. 'Residents-Caregivers' (via CareLog) Relationship

- **Structural constraints**:
  - Each care log (min: 1) must be created by one caregiver (max: 1).
  - Each caregiver (min: 0) can create multiple care logs (max: N).
- **Cardinality ratio**: 1:N
- **Relationship Attribute**: None.
- **Explanation**:
  - This is an **indirect one-to-many** (1:N) relationship where caregivers log care activities for residents.

CareLog {(CaregiverID, CareLogID)} CR: 1:N

Relationship	Cardinality Ratio	Participation Constraints	Attributes
Residents-Guardians	M:N	Resident (min: 0, max: M), Guardian (min: 1)	Relationship
Residents-CareLog	1:N	Resident (min: 1), CareLog (min: 1)	RecordTime, Content
Residents-HealthRecord	1:N	Resident (min: 1), HealthRecord (min: 1)	RecordTime, Content
UserRole-Role	M:N	User (min: 1), Role (min: 1)	None
Role-RolePermission	M:N	Role (min: 1), Permission (min: 1)	None
Residents-Nurses	1:N	Nurse (min: 0), HealthRecord (min: 1)	None
Residents-Caregivers	1:N	Caregiver (min: 0), CareLog (min: 1)	None

Task4- Mapping ER to Logical Model:

Mapping Steps:

1. Entities to Tables:

Each entity becomes a table.

Attributes in the entity are columns in the corresponding table.

The primary key of the entity becomes the primary key of the table.

2. Relationships to Tables:

For one-to-many relationships, the foreign key of the "many" side is added to the table of the "one" side.

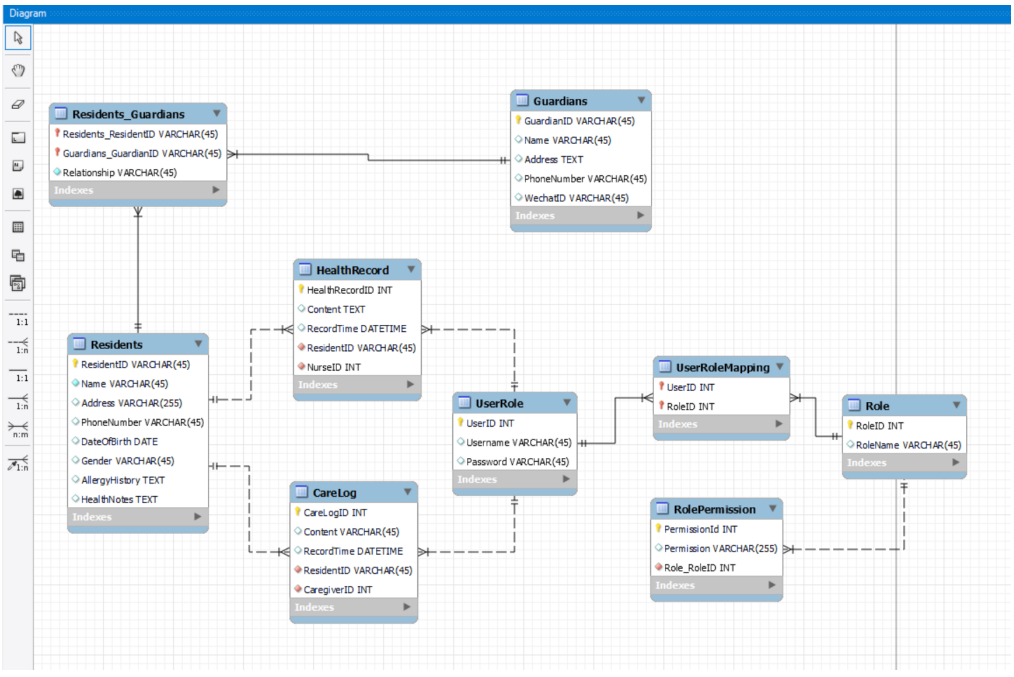
For many-to-many relationships, a separate associative table is created with foreign keys referencing the related entities' primary keys.

3. Primary and Foreign Keys:

Each primary key is explicitly defined.

Foreign keys are added to maintain referential integrity between related tables.

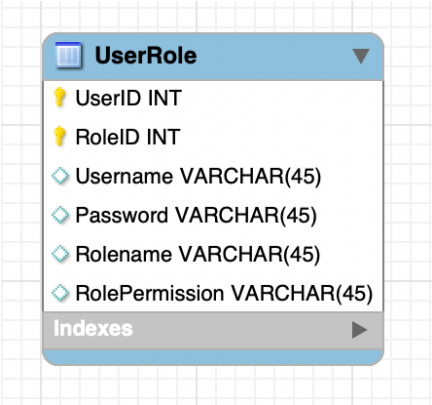
EER diagram of Database: hospital



Task 5- Normalisation-3NF

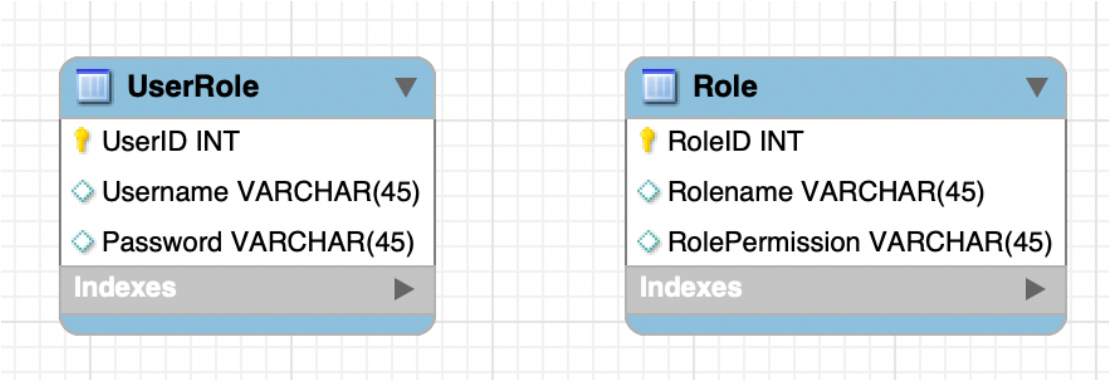
(1) Proof that the database is in 1

Each row is unique (there is a primary key), and there is no duplicated data. For example, in the **UserRole** table, every piece of data is independent, and each row in the table is unique, distinguished by the primary key. Therefore, this table satisfies the requirements of 1NF. We will further modify this table to meet the requirements of 2NF.



(2) Proof that the database is in 2NF

Partial dependencies have been eliminated, ensuring compliance with the Second Normal Form (2NF). Data redundancy is reduced. For example, if multiple users have the same role, Rolename and RolePermission are not stored repeatedly. This improves data consistency and integrity. By decomposing the table and storing user and role information in separate tables, the database structure satisfies the requirements of 2NF



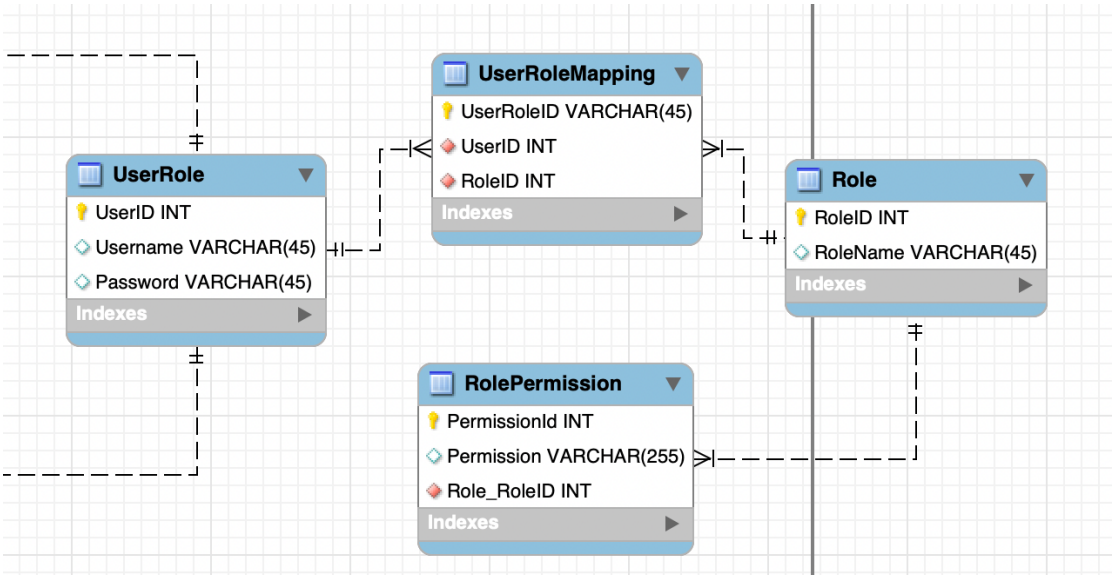
(3) Proof that the database is in 3NF

Elimination of partial dependencies. Username and Password have been separated into the User table and are fully dependent on UserID.Rolename and RolePermission have been

separated into the Role table and are fully dependent on RoleID.

Elimination of transitive dependencies. Rolename and RolePermission no longer indirectly depend on (UserID, RoleID) through RoleID. The mapping relationship between users and roles has been separated into a standalone table. The mapping of users and roles uses UserID and RoleID as a composite primary key.

By decomposing the tables into UserRole, Role, UserRoleMapping, and RolePermission, transitive dependencies and redundant data have been successfully eliminated. This ensures the database design meets the requirements of the Third Normal Form (3NF), while also improving data integrity and consistency. At this point, the normalization process is complete.



Task 6– Implement the Database

The database structure for hospital1 includes several core tables that manage critical information. Key tables include Residents, which stores residents' personal and health-related information; Guardians, which holds guardian details; and Residents_Guardians, which links residents to their guardians. Other essential tables include CareLog for tracking care activities, HealthRecord for storing health records, UserRole for managing user credentials, Role for defining system roles, RolePermission for assigning permissions, and UserRoleMapping for establishing many-to-many relationships between users and roles.

The database design emphasizes referential integrity through the use of foreign key constraints and sets ON DELETE and ON UPDATE`rules to NO ACTION. Indexes are created on foreign key fields to improve query performance, and enumeration fields are implemented dynamically using tables like Role instead of using the ENUM type. This ensures flexibility, scalability, and optimal performance in the database's design.

More Code is in file SQL query， this is an example：

```
CREATE TABLE IF NOT EXISTS `hospital1`.`Residents` (
  `ResidentID` VARCHAR(45) NOT NULL,
  `Name` VARCHAR(45) NOT NULL,
  `Address` VARCHAR(255) NULL,
  `PhoneNumber` VARCHAR(45) NULL,
  `DateOfBirth` DATE NULL,
  `Gender` VARCHAR(45) NULL,
  `AllergyHistory` TEXT NULL,
  `HealthNotes` TEXT NULL,
  PRIMARY KEY (`ResidentID`))
```

Task7– Populate the Tables with Data

These SQL code populates the hospital1 database with essential data for its key tables, ensuring that the structure supports the system's functionality. It inserts detailed data into tables such as Residents, Guardians, CareLog, HealthRecord, UserRole, Role, UserRoleMapping, and RolePermission, which are vital for managing healthcare-related information.

More Code is in file SQL query， this is an example：

```
use hospital1

-- Examples for Residents

INSERT INTO Residents (ResidentID, Name, Address, PhoneNumber, DateOfBirth, Gender,
AllergyHistory, HealthNotes)

VALUES (SHA2('224360184286272079', 256), 'Harry Hendricks', '9671 Gillespie Greens Suite 156,
Kelseychester, WV 18335', '18497938812', '1960-07-11', 'Male', 'Myself red will short.', 'Fly offer
let friend room.');

INSERT INTO Residents (ResidentID, Name, Address, PhoneNumber, DateOfBirth, Gender,
AllergyHistory, HealthNotes)

VALUES (SHA2('853501313039236018', 256), 'Donald Hudson', '13751 Brown Mission, North
Patrickbury, UT 56920', '17529323589',

-- Examples for Guardians

NSERT INTO Guardians (GuardianID, Name, Address, PhoneNumber, WechatID)

VALUES (SHA2('047407006853525560', 256), 'Kenneth Hernandez', '709 Patricia Locks, North
Heidi, ND 44262', '19501575689', 'murrayelizabeth');

INSERT INTO Guardians (GuardianID, Name, Address, PhoneNumber, WechatID)

VALUES (SHA2('817297387357544316', 256), 'Melissa Rodriguez', '22667 Ashley Highway Apt.
356, West Donna, TX 41232', '13195539962', 'alex79');
```

Task8– Query the Database

(a) Basic queries:

(1): Find all users involved in the CareLog (Caregiver role) and list their names and login usernames.

```
SELECT DISTINCT u.UserID, u.Username, r.RoleName
FROM UserRole u
JOIN UserRoleMapping urm ON u.UserID = urm.UserID
JOIN Role r ON urm.RoleID = r.RoleID
WHERE r.RoleName = 'Caregiver';
```

Result Grid | Filter Rows:

UserID	Username	RoleName
1428	Ryan	Caregiver
1461	Sheri	Caregiver
3320	Cole	Caregiver
3718	Daniel	Caregiver
4129	Peggy	Caregiver
5875	Jennifer	Caregiver
6035	Elizabeth	Caregiver
6955	Cynthia	Caregiver
9483	Michael	Caregiver
9822	Tracy	Caregiver

(2): Find all residents in the `Residents` table who are older than 50 years and have a health record in the `HealthRecord` table.

```
SELECT res.ResidentID, res.Name, res.DateOfBirth, hr.Content
FROM Residents res
JOIN HealthRecord hr ON res.ResidentID = hr.ResidentID
WHERE TIMESTAMPDIFF(YEAR, res.DateOfBirth, CURDATE()) > 50;
```

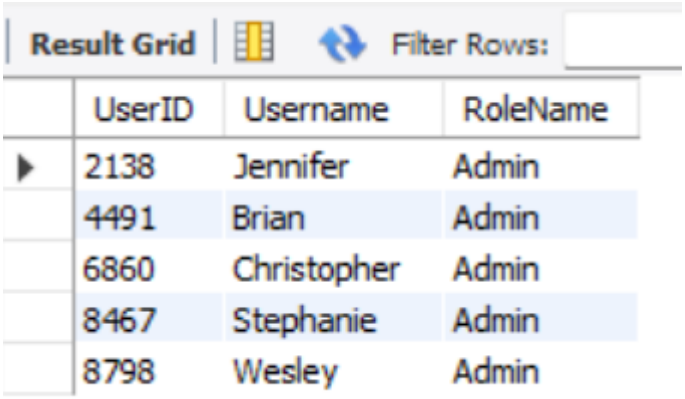
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ResidentID	Name	DateOfBirth	Content
970762394927732020	Theresa Collier	1962-05-23	Minor complications, addressed effectively.
231095816878478628	Roberta Anderson	1966-04-06	Patient is recovering well.
361595796296662468	Erik Miller	1961-09-04	Critical condition, requires monitoring.
353163091806114367	Kenneth Reid	1965-09-06	Stable but under observation.
224360184286272079	Harry Hendricks	1960-07-11	Condition remains stable.
490946937910629327	Keith Scott	1959-07-25	Significant improvement observed.
212750573394927405	Meghan Long	1973-10-11	Significant improvement observed.

(HealthRecord has 10 records, the Query can return 7 records, which proves the data selection is successful. If we want to know all residents aged more than 30, it would return 10 records with condition satisfied records.)

(3): Query the users who have access to all information (Admin role) and list their usernames and roles.

```
SELECT u.UserID, u.Username, r.RoleName
FROM UserRole u
JOIN UserRoleMapping urm ON u.UserID = urm.UserID
JOIN Role r ON urm.RoleID = r.RoleID
WHERE r.RoleName = 'Admin';
```



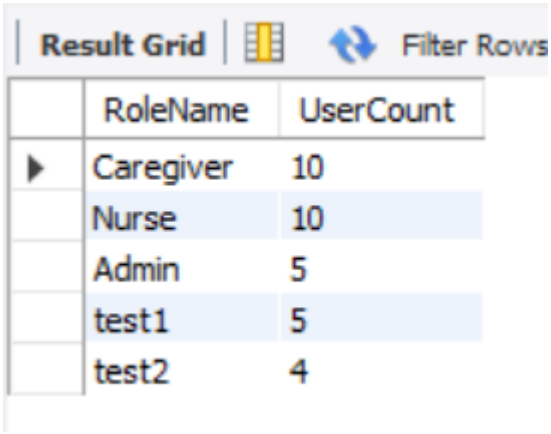
Result Grid | Filter Rows:

UserID	Username	RoleName
2138	Jennifer	Admin
4491	Brian	Admin
6860	Christopher	Admin
8467	Stephanie	Admin
8798	Wesley	Admin

(b)Medium queries:

(1)： Query the number of users for each role (`RoleName`), and only display roles with more than 1 user.

```
SELECT r.RoleName, COUNT(urm.UserID) AS UserCount
FROM Role r
LEFT JOIN UserRoleMapping urm ON r.RoleID = urm.RoleID
GROUP BY r.RoleName
HAVING COUNT(urm.UserID) > 1;
```



Result Grid | Filter Rows

RoleName	UserCount
Caregiver	10
Nurse	10
Admin	5
test1	5
test2	4

(2)： Find the names of residents with health records (`HealthRecord`) and the details of their associated care logs (`CareLog`), if care logs exist.

```
SELECT res.Name AS ResidentName, hr.Content AS HealthRecord, cl.Content AS
CareLog
FROM Residents res
JOIN HealthRecord hr ON res.ResidentID = hr.ResidentID
LEFT JOIN CareLog cl ON res.ResidentID = cl.ResidentID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ResidentName	HealthRecord	CareLog
Theresa Collier	Minor complications, addressed effectively.	Checked patient's respiratory condition.
Roberta Anderson	Patient is recovering well.	NULL
Donald Hudson	Stable but under observation.	Conducted routine health check-up.
Erik Miller	Critical condition, requires monitoring.	NULL
Tyrone Walker	Condition remains stable.	NULL
Kenneth Reid	Stable but under observation.	NULL
Harry Hendricks	Condition remains stable.	NULL
Keith Scott	Significant improvement observed.	NULL
Meghan Long	Significant improvement observed.	NULL
Dale Mccann	Stable but under observation.	NULL

(Note: There are a total of 20 residents, 10 of whom have records inserted into the `HealthRecord` table, and 10 into the `CareLog` table. However, the residents who have data inserted are selected randomly, so the assigned nurses and caregivers are also random.)

(c) Advanced queries:

(1) : Check if the number of residents with health records (`HealthRecord`) is less than 60% of the total number of residents. If so, display the detailed information of these residents.

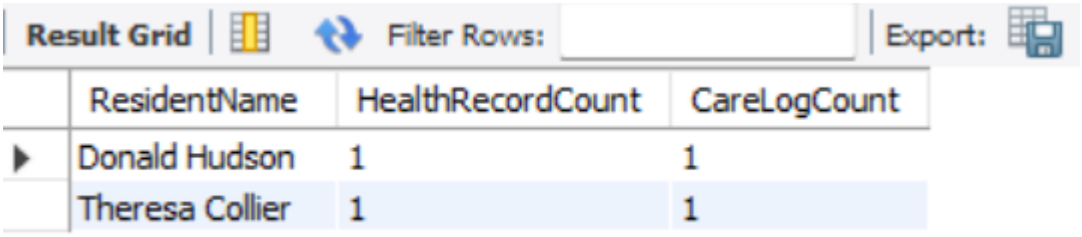
```
SELECT *
FROM Residents
WHERE ResidentID IN (
    SELECT ResidentID
    FROM HealthRecord
    GROUP BY ResidentID
    HAVING COUNT(HealthRecordID) > 0
) AND (
    SELECT COUNT(DISTINCT ResidentID)
    FROM HealthRecord
) < (
    SELECT COUNT(*) * 0.6
    FROM Residents
);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ResidentID	Name	Address	PhoneNumber	DateOfBirth	Gender	AllergyHistory	HealthNotes
212750573394927405	Meghan Long	5937 Amber Valley, Amyfurt, AZ 96942	15634306312	1973-10-11	Male	Spring house if.	Gun main cover prove father.
224360184286272079	Harry Hendricks	9671 Gillespie Greens Suite 156, Kelseychester,...	18497938812	1960-07-11	Male	Myself red will short.	Fly offer let friend room.
231095816878478628	Roberta Anderson	780 Garrett Road Apt. 452, Ronniemouth, AL 2...	18708606725	1966-04-06	Male	Back give quite take call democratic.	Nor side college within song relationship.
3531630918061143 67	Kenneth Reid	85323 Martin Centers, Lake Aliciachester, LA 86...	12686782852	1965-09-06	Male	Share during represent reveal factor.	Lose because bit hope.
3615957962966624 68	Erik Miller	757 Johnson Groves Apt. 851, Leahborough, I...	14347147139	1961-09-04	Male	Five top car car far.	Meet social once chance information.
490946937910629327	Keith Scott	440 Leach Junctions Suite 906, Laurenfurt, NH ...	11710175702	1959-07-25	Female	Couple wind wide.	Front whether candidate.
5492393795080316 42	Dale Mccann	342 James Common, Pageport, MT 58210	15481071444	1974-07-13	Female	Other current on law.	Including dog task task.
6369100123089142 41	Tyrone Walker	207 Karen Squares, East Nathan, IA 65792	19282758434	1980-05-23	Male	Contain system art forget born.	Difference politics under glass generation.
8535013130392360 18	Donald Hudson	13751 Brown Mission, North Patrickbury, UT 56...	17529323589	1991-04-26	Female	Kitchen heart course.	Section story marriage whatever event not.
970762394927732020	Theresa Collier	USS Vasquez, FPO AA 49729	14873888913	1962-05-23	Female	Exactly stock PM simply.	Cut become meeting effect.
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

（2）： Find the names of residents who have both health records (`HealthRecord`) and care logs (`CareLog`), and count their total associated records.

```
SELECT res.Name AS ResidentName,
       COUNT(DISTINCT hr.HealthRecordID) AS HealthRecordCount,
       COUNT(DISTINCT cl.CareLogID) AS CareLogCount
FROM Residents res
JOIN HealthRecord hr ON res.ResidentID = hr.ResidentID
JOIN CareLog cl ON res.ResidentID = cl.ResidentID
GROUP BY res.ResidentID
HAVING COUNT(hr.HealthRecordID) > 0 AND COUNT(cl.CareLogID) > 0;
```



ResidentName	HealthRecordCount	CareLogCount
Donald Hudson	1	1
Theresa Collier	1	1

Task 9 - Database Application

Login

1. Purpose of the Login Page

The login page is designed to:

- Authenticate users based on their credentials (UserID and Password).
- Identify their role (RoleID) in the system (e.g., Caregiver, Nurse, or Admin).
- Redirect them to their respective dashboard based on their role after successful authentication.
- Animated background allows user to interact with.

---

2. Workflow

Frontend Logic (login.html):

1. **Form Inputs:**
    - A form with fields for:
        - UserID: To enter the unique identifier of the user.
        - Password: To enter the corresponding password.
    - A submit button that sends the credentials to the server.
2. **Form Submission:**
    - The form's method is set to POST.
    - The action attribute is set to /authenticate to send the input data to the authenticate route in the Flask application.
3. **Error Handling:**
    - If authentication fails, the page will display an error message prompting the user to try again.

3. Key Functionalities

1. **Input Validation:**
    - Ensures that both UserID and Password fields are filled before

submission.

- Backend checks if the UserID exists in the database and if the password matches.

2. **Role-Based Redirection:**
   - Depending on the user's RoleID:
     - **RoleID = 1 (Caregiver)**: Redirects to /dashboard_care.
     - **RoleID = 2 (Nurse)**: Redirects to /dashboard_nurse.
     - **RoleID = 3 (Admin)**: Redirects to /dashboard_admin.
3. **Session Management:**
   - Once authenticated, the user's UserID and RoleID are stored in the session, allowing the application to manage the user's session state across different pages.
4. **Error Handling:**
   - If authentication fails (e.g., invalid UserID or Password), the login page reloads with an error message.
   - If the user's RoleID is unrecognized, it returns a 403 Forbidden response.
5. **How It Handles Scenarios**

Scenario	Outcome
Valid UserID and Password	Redirects to the respective dashboard based on RoleID.
Invalid UserID or Password	Reloads login page with the error message: **"Invalid credentials. Please try again."**
User without RoleID (misconfigured user)	Displays **"Role not found."** with a 403 Forbidden status.
User tries accessing a restricted page	Redirects them back to the login page or shows **"Access denied."** if the session is invalid.

main.py

```python
from flask import Flask, render_template, request, redirect, session, url_for
import pymysql

# Flask app 初始化
app = Flask(__name__)
app.secret_key = '114514'

# 数据库连接设置
def get_connection():
    timeout = 10
    connection = pymysql.connect(
        charset="utf8mb4",
        connect_timeout=timeout,
        cursorclass=pymysql.cursors.DictCursor,
        db="hospital1",  # 使用你的实际数据库名
        host="mysql-30ef2a65-liubeibeizz-2037.d.aivencloud.com",
        password="AVNS_x-T0LRSoZBR5f_7TTsB",
        read_timeout=timeout,
        port=14083,
        user="avnadmin",
        write_timeout=timeout,
    )
    return connection

@app.route('/')
def login():
    return render_template('login.html')

@app.route('/authenticate', methods=['POST'])
```

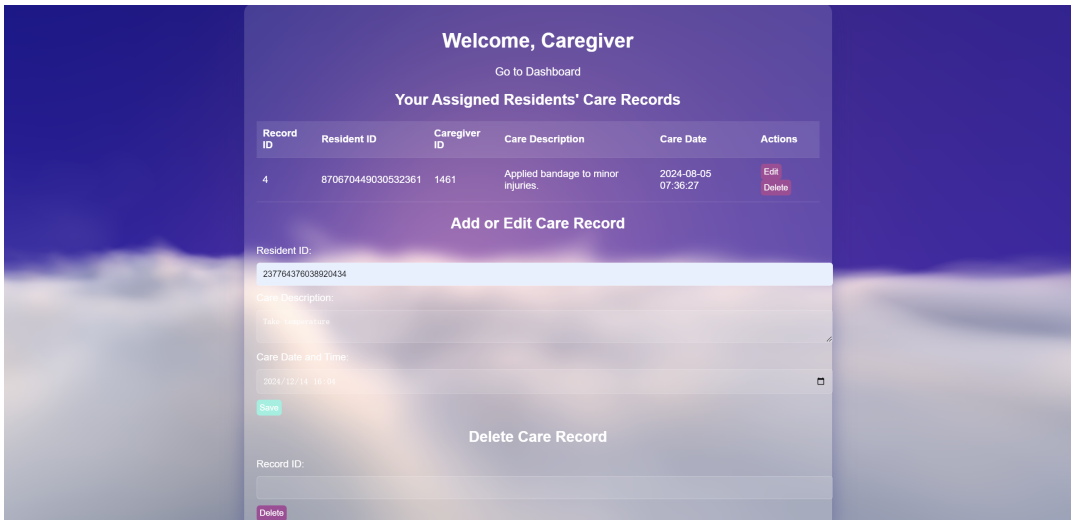
```
def authenticate():
    userID = request.form.get('UserID')
    password = request.form.get('password')

    if not userID or not password:
        return render_template('login.html', error_message="Please enter both UserID and Password.")

    connection = get_connection()
    try:
        with connection.cursor() as cursor:
            # 查询用户信息
            cursor.execute(
                "SELECT ur.UserID, ur.Password, urm.RoleID "
                "FROM UserRole ur JOIN UserRoleMapping urm ON ur.UserID = urm.UserID "
                "WHERE ur.UserID = %s",
                (userID,)
            )
            user = cursor.fetchone()

            if user:
                # 检查密码
                if user['Password'] == password:  # 如果密码加密，请替换为相应的哈希比较
                    session['UserID'] = user['UserID']
                    session['RoleID'] = user['RoleID']

                    # 跳转到对应角色页面
                    if user['RoleID'] == 1:
                        return redirect('/caregiver')
                    elif user['RoleID'] == 2:
                        return redirect('/nurse')
                    elif user['RoleID'] == 3:
                        return redirect('/admin')
                    else:
                        return "Role not found.", 403
                else:
                    return render_template('login.html', error_message="Invalid credentials. Please try again.")
            else:
                return render_template('login.html', error_message="Invalid credentials. Please try again.")
    finally:
        connection.close()
```

Caregiver

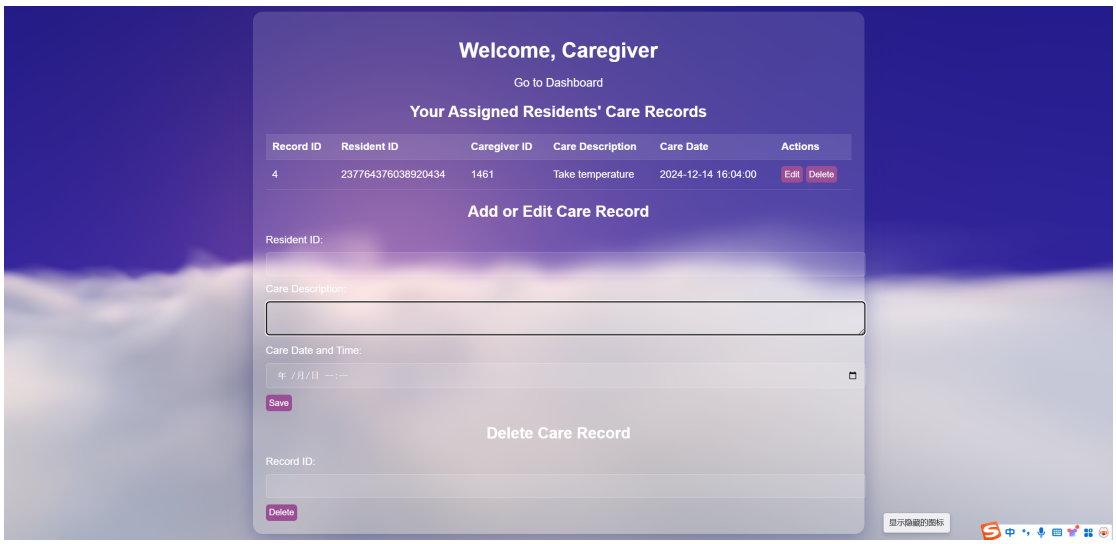
This is a Caregiver account, whose dashboard page will be used soon.

The Caregiver pages allow users with RoleID = 1 to:

1. View care records (CareLog) assigned to them.
2. Add new care records.
3. Edit existing care records.
4. Delete care records.

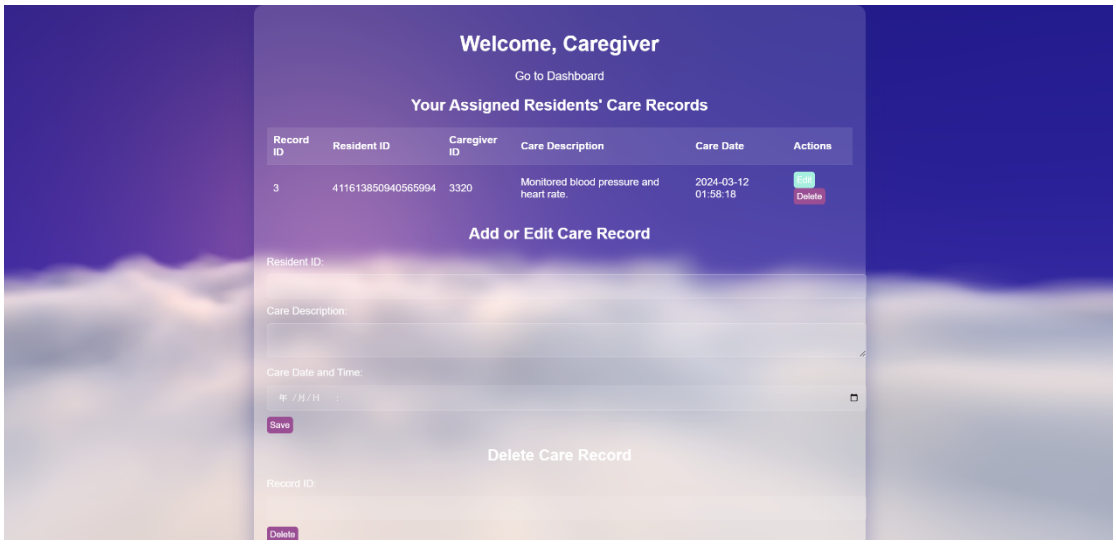


1. We can type in the new CareLog in the boxes.
2. We can control the background to move to the direction we want with our mouse speed.
3. Update the CareLogs.



4. Different caregivers only see his own CareLogs.

Eg. This is the dashboard of another caregiver, he can't see Sheri's (UserID= 1461) CareLogs.



This is managed through three main routes in the backend:

- /caregiver (to display caregiver information and care logs).
- /caregiver/save (to save new or updated care records).
- /caregiver/delete (to delete care records).

main.py

```
@app.route('/caregiver')
def caregiver():
if session.get('RoleID') != 1:
return "Access denied."

caregiver_id = session['UserID']
connection = get_connection()
try:
with connection.cursor() as cursor:
# 获取分配给护理人员的所有护理记录
query = """
SELECT cl.CareLogID, cl.ResidentID, cl.CaregiverID, cl.Content AS CareDescription, cl.RecordTime AS CareDate
FROM CareLog cl
WHERE cl.CaregiverID = %s
"""
cursor.execute(query, (caregiver_id,))
care_logs = cursor.fetchall()
```

```
    finally:
    connection.close()

    return render_template('caregiver.html', care_logs=care_logs)


@app.route('/caregiver/save', methods=['POST'])
def save_care_record():
    if session.get('RoleID') != 1:
        return "Access denied."

    record_id = request.form.get('record_id')
    resident_id = request.form['resident_id']
    care_description = request.form['care_description']
    care_date = request.form['care_date']  # 从前端获取 datetime-local 数据
    caregiver_id = session['UserID']

    # 转换日期为 datetime 格式
    from datetime import datetime
    try:
        care_date = datetime.strptime(care_date, '%Y-%m-%dT%H:%M')  # 解析 datetime-local 格式
    except ValueError:
        return "Invalid date format.", 400

    connection = get_connection()
    try:
        with connection.cursor() as cursor:
            if record_id:  # 更新记录
                query = """
                UPDATE CareLog
                SET ResidentID = %s, Content = %s, RecordTime = %s
                WHERE CareLogID = %s AND CaregiverID = %s
                """
                cursor.execute(query, (resident_id, care_description, care_date, record_id, caregiver_id))
            else:  # 插入新记录
                query = """
                INSERT INTO CareLog (ResidentID, CaregiverID, Content, RecordTime)
                VALUES (%s, %s, %s, %s)
                """
                cursor.execute(query, (resident_id, caregiver_id, care_description, care_date))
        connection.commit()
    finally:
        connection.close()

    # 修改后跳转到 Caregiver 页面
```

```
59.  return redirect('/caregiver')


60.  @app.route('/caregiver/delete', methods=['POST'])
61.  def delete_care_record():
62.  if session.get('RoleID') != 1:
63.  return "Access denied."

64.  record_id = request.form['record_id']
65.  caregiver_id = session['UserID']

66.  connection = get_connection()
67.  try:
68.  with connection.cursor() as cursor:
69.  query = "DELETE FROM CareLog WHERE CareLogID = %s AND CaregiverID = %s"
70.  cursor.execute(query, (record_id, caregiver_id))
71.  connection.commit()
72.  finally:
73.  connection.close()

74.  return redirect('/caregiver')
75.  #dashboards
76.  from flask import render_template, request, session, redirect
77.  import mysql.connector
78.
79.  @app.route('/dashboard_care')
80.  def dashboard_care():
81.      if session.get('RoleID') != 1:
82.          return "Access denied."
83.
84.      caregiver_id = session['UserID']
85.      connection = get_connection()
86.
87.      try:
88.          with connection.cursor() as cursor:
89.              # 查询 Caregiver 的所有护理记录
90.              query_care_logs = """
91.                  SELECT cl.CareLogID, cl.ResidentID, cl.Content AS CareDescription, cl.RecordTime
    AS CareDate
92.                  FROM CareLog cl
93.                  WHERE cl.CaregiverID = %s
94.                  ORDER BY cl.RecordTime DESC
95.              """
96.              cursor.execute(query_care_logs, (caregiver_id,))
97.              care_logs = cursor.fetchall()
```

```
98.         finally:
99.             connection.close()
100.
101.        return render_template('dashboard_care.html', care_logs=care_logs)
```

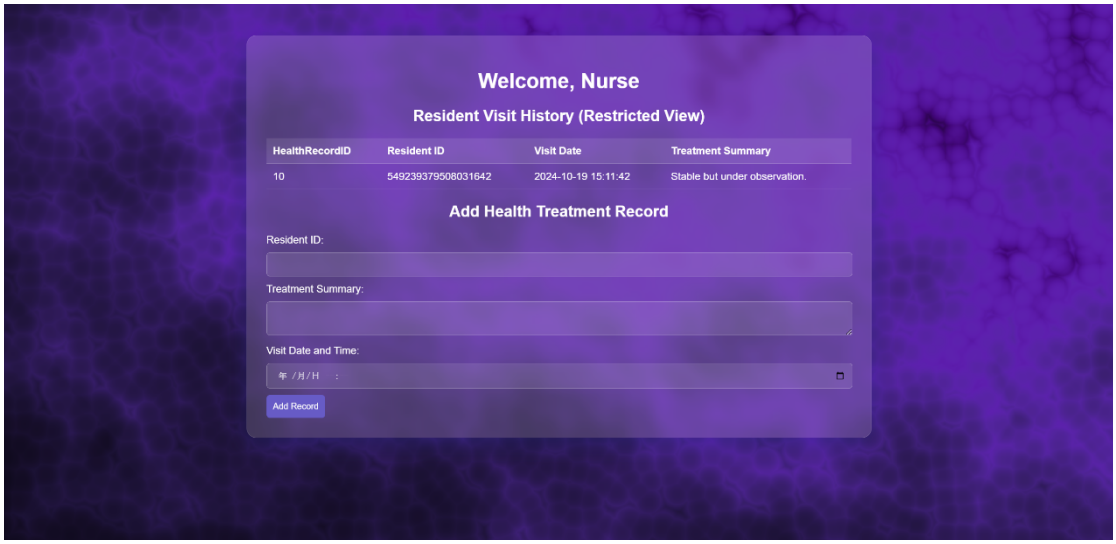
Nurse

The **Nurse** pages allow users with RoleID = 2 to:

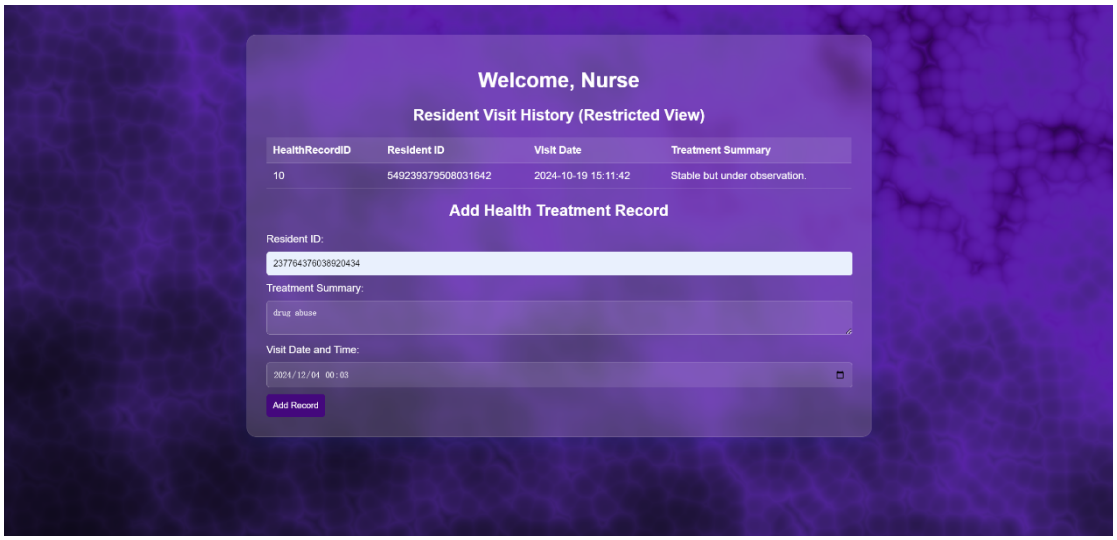
1. **View Health Records:**
    - Nurses can view health records (HealthRecord) assigned to them.
    - The /nurse route fetches and displays all health records where the NurseID matches the logged-in nurse's UserID.
2. **Add Health Records:**
    - Nurses can add new health records for residents via the /nurse/add_health_record route.
    - The submitted data (resident ID, treatment summary, and visit date) is saved to the HealthRecord table.
3. **Search Functionality:**
    - Nurses can filter health records by ResidentID or VisitDate on the /nurse page.

Key Features:

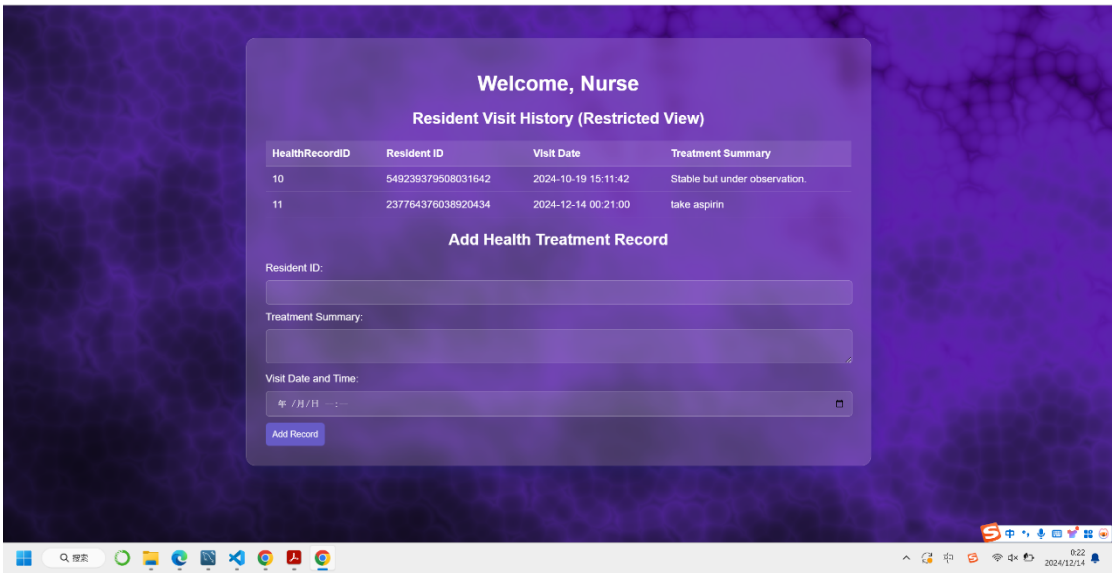
- Nurses can only view and manage records assigned to them.
- Any new or updated records are displayed after saving.



View the nurse dashboard, includes all healthRecords related to this nurse.



Insert a new HealthRecord, press "add record"



Refresh and you can see it. The database also receive this record.

HealthRecordID	Content	RecordTime	ResidentID	NurseID
7	Condition remains stable.	2024-05-11 06:47:22	224360184286272079	5969
8	Significant improvement observed.	2024-03-21 07:40:09	490946937910629327	8655
9	Significant improvement observed.	2024-02-11 07:22:34	212750573394927405	3718
10	Stable but under observation.	2024-10-19 15:11:42	549239379508031642	1112
11	take aspirin	2024-12-14 00:21:00	237764376038920434	1112
12	dance	2024-12-14 14:43:00	237764376038920434	4052
13	Stable but under observation.	2024-08-30 13:00:40	237764376038920434	3154
NULL	NULL	NULL	NULL	NULL

main.py

```
@app.route('/nurse')
def nurse():
    if session.get('RoleID') != 2:
```

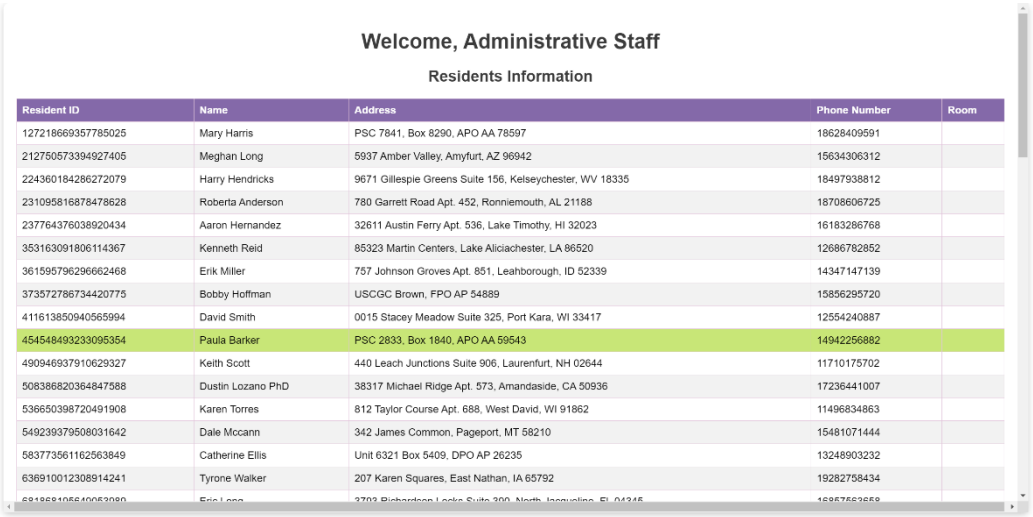
```
        return "Access denied."

    nurse_id = session.get('UserID')
    connection = get_connection()
    try:
        with connection.cursor() as cursor:
            # 修正查询字段名，匹配数据库表结构
            query = """
                SELECT hr.HealthRecordID, hr.ResidentID, hr.RecordTime, hr.Content
                FROM HealthRecord hr
                WHERE hr.NurseID = %s
            """
            cursor.execute(query, (nurse_id,))
            visit_history = cursor.fetchall()
    finally:
        connection.close()

    return render_template('nurse.html', visit_history=visit_history)


from datetime import datetime

@app.route('/nurse/add_health_record', methods=['POST'])
def add_health_record():
    if session.get('RoleID') != 2:
        return "Access denied."

    resident_id = request.form['resident_id']
    treatment_summary = request.form['treatment_summary']
    visit_date = request.form['visit_date']
    nurse_id = session.get('UserID')

    # 转换 `visit_date` 为 `datetime`
    try:
        visit_date = datetime.strptime(visit_date, '%Y-%m-%dT%H:%M')  # 解析 `datetime-local` 数
据
    except ValueError:
        return "Invalid date format.", 400

    connection = get_connection()
    try:
        with connection.cursor() as cursor:
            # 插入新记录
            query = """
                INSERT INTO HealthRecord (ResidentID, NurseID, Content, RecordTime)
```

```
                VALUES (%s, %s, %s, %s)
            """
            cursor.execute(query, (resident_id, nurse_id, treatment_summary, visit_date))
        connection.commit()
    finally:
        connection.close()

    # 添加记录后跳转到 dashboard_nurse 页面
    return redirect('/dashboard_nurse')
```

Admin

The **Admin** pages allow users with RoleID = 3 to:

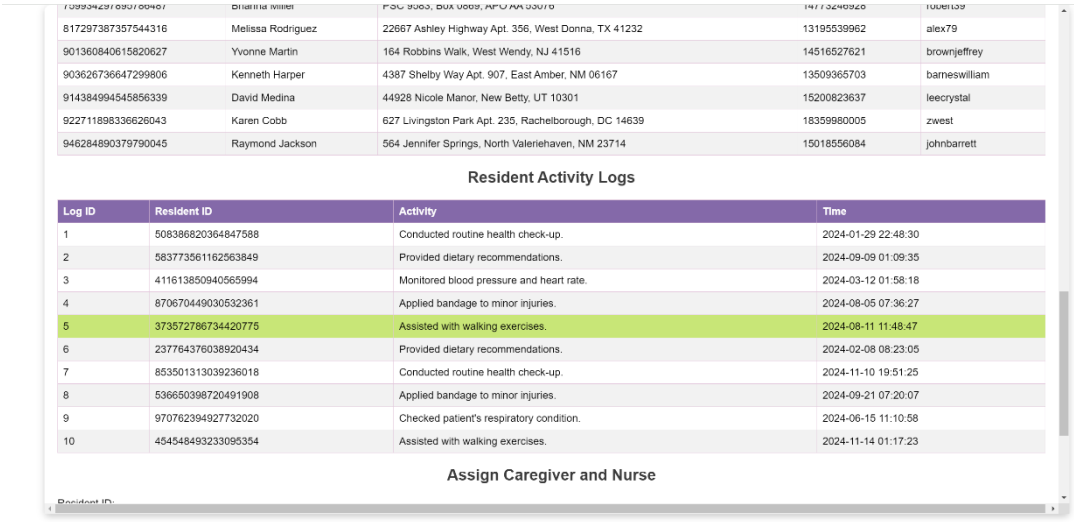
1. **View Residents and Logs:**
    - The /admin route fetches and displays:
        - All residents' information (Residents table).
        - Activity logs (CareLog table) for all residents.



Welcome, Administrative Staff

Residents Information

Resident ID	Name	Address	Phone Number	Room
127218669357785025	Mary Harris	PSC 7841, Box 8290, APO AA 78597	18628409591	
212750573394927405	Meghan Long	5937 Amber Valley, Amyfurt, AZ 96942	15634306312	
224360184286272079	Harry Hendricks	9671 Gillespie Greens Suite 156, Kelseychester, WV 18335	18497938812	
231095816878478628	Roberta Anderson	780 Garrett Road Apt. 452, Ronniemouth, AL 21188	18708606725	
237764376038920434	Aaron Hernandez	32611 Austin Ferry Apt. 536, Lake Timothy, HI 32023	16183286768	
353163091806114367	Kenneth Reid	85323 Martin Centers, Lake Aliciachester, LA 86520	12686782852	
361595796296662468	Erik Miller	757 Johnson Groves Apt. 851, Leahborough, ID 52339	14347147139	
373572786734420775	Bobby Hoffman	USCGC Brown, FPO AP 54889	15856295720	
411613850940565994	David Smith	0015 Stacey Meadow Suite 325, Port Kara, WI 33417	12554240887	
454548493233095354	Paula Barker	PSC 2833, Box 1840, APO AA 59543	14942256882	
490946937910629327	Keith Scott	440 Leach Junctions Suite 906, Laurenfurt, NH 02644	11710175702	
508386820364847588	Dustin Lozano PhD	38317 Michael Ridge Apt. 573, Amandaside, CA 50936	17236441007	
536650398720491908	Karen Torres	812 Taylor Course Apt. 688, West David, WI 91862	11496834863	
549239379508031642	Dale Mccann	342 James Common, Pageport, MT 58210	15481071444	
583773561162563849	Catherine Ellis	Unit 6321 Box 5409, DPO AP 26235	13248903232	
636910012308914241	Tyrone Walker	207 Karen Squares, East Nathan, IA 65792	19282758434	



817297387357544316	Melissa Rodriguez	22667 Ashley Highway Apt. 356, West Donna, TX 41232	13195539962	alex79
901360840615820627	Yvonne Martin	164 Robbins Walk, West Wendy, NJ 41516	14516527621	brownjeffrey
903626736647299806	Kenneth Harper	4387 Shelby Way Apt. 907, East Amber, NM 06167	13509365703	barneswilliam
914384994545856339	David Medina	44928 Nicole Manor, New Betty, UT 10301	15200823637	leecrystal
922711898336626043	Karen Cobb	627 Livingston Park Apt. 235, Rachelborough, DC 14639	18359980005	zwest
946284890379790045	Raymond Jackson	564 Jennifer Springs, North Valeriehaven, NM 23714	15018556084	johnbarrett

Resident Activity Logs

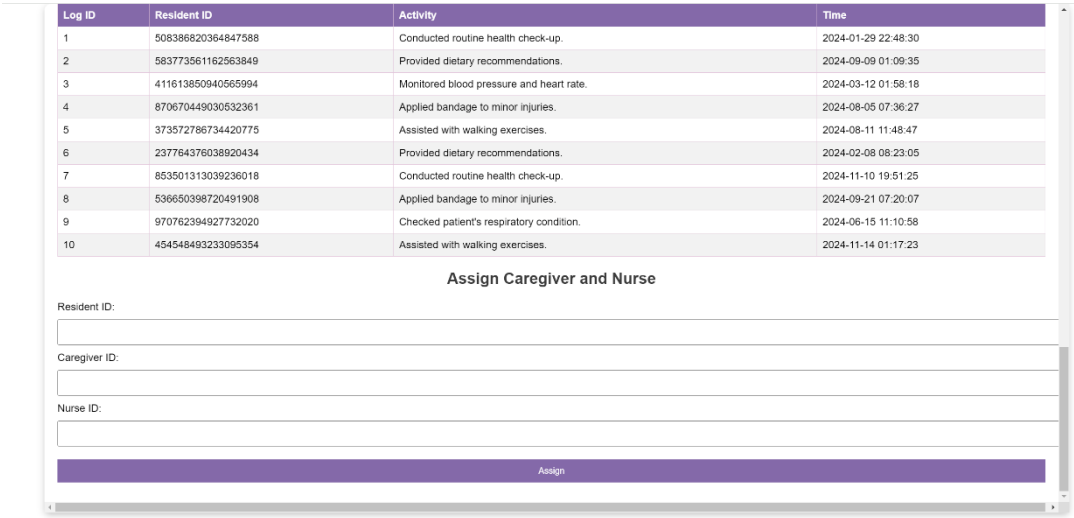
Log ID	Resident ID	Activity	Time
1	508386820364847588	Conducted routine health check-up.	2024-01-29 22:48:30
2	583773561162563849	Provided dietary recommendations.	2024-09-09 01:09:35
3	411613850940565994	Monitored blood pressure and heart rate.	2024-03-12 01:58:18
4	8706704490305323 61	Applied bandage to minor injuries.	2024-08-05 07:36:27
5	373572786734420775	Assisted with walking exercises.	2024-08-11 11:48:47
6	237764376038920434	Provided dietary recommendations.	2024-02-08 08:23:05
7	853501313039236018	Conducted routine health check-up.	2024-11-10 19:51:25
8	536650398720491908	Applied bandage to minor injuries.	2024-09-21 07:20:07
9	970762394927732020	Checked patient's respiratory condition.	2024-06-15 11:10:58
10	454548493233095354	Assisted with walking exercises.	2024-11-14 01:17:23

Assign Caregiver and Nurse

Resident ID:

2. **Assign Staff:**

   - Admins can assign caregivers and nurses to specific residents via the /admin/assign_staff route.
   - The data is saved to a table (e.g., ResidentStaffAssignments).



Log ID	Resident ID	Activity	Time
1	508386820364847588	Conducted routine health check-up.	2024-01-29 22:48:30
2	583773561162563849	Provided dietary recommendations.	2024-09-09 01:09:35
3	411613850940565994	Monitored blood pressure and heart rate.	2024-03-12 01:58:18
4	8706704490305323 61	Applied bandage to minor injuries.	2024-08-05 07:36:27
5	373572786734420775	Assisted with walking exercises.	2024-08-11 11:48:47
6	237764376038920434	Provided dietary recommendations.	2024-02-08 08:23:05
7	853501313039236018	Conducted routine health check-up.	2024-11-10 19:51:25
8	536650398720491908	Applied bandage to minor injuries.	2024-09-21 07:20:07
9	970762394927732020	Checked patient's respiratory condition.	2024-06-15 11:10:58
10	454548493233095354	Assisted with walking exercises.	2024-11-14 01:17:23

Assign Caregiver and Nurse

Resident ID:

Caregiver ID:

Nurse ID:

Assign

Assign Caregiver and Nurse

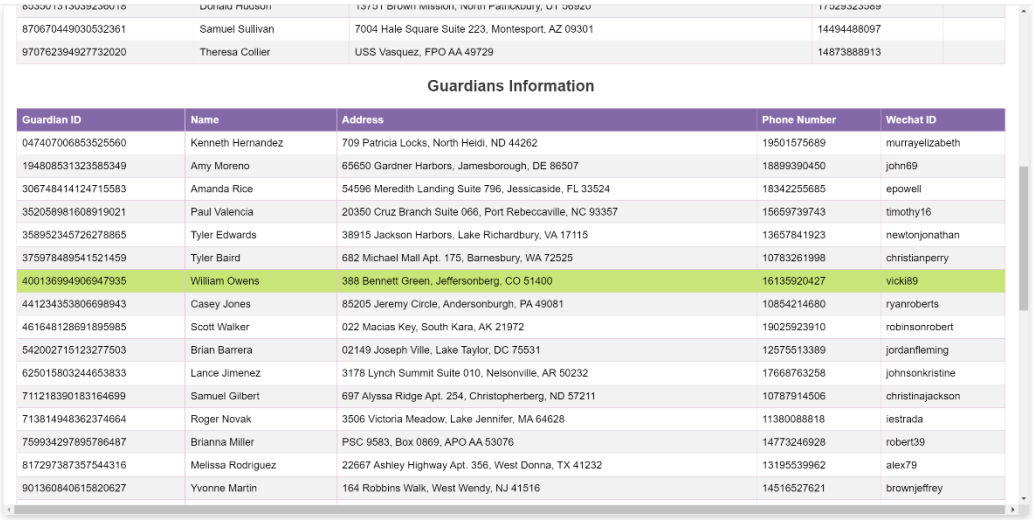
Resident ID:

Caregiver ID:

Nurse ID:

3. **View Guardians:**

   - Admins can see guardian details for each resident (linked through the Guardians table).



8706704490305323 61	Samuel Sullivan	7004 Hale Square Suite 223, Montesport, AZ 09301	14494488097
970762394927732020	Theresa Collier	USS Vasquez, FPO AA 49729	14873888913

Guardians Information

Guardian ID	Name	Address	Phone Number	Wechat ID
047407006853525560	Kenneth Hernandez	709 Patricia Locks, North Heidi, ND 44262	19501575689	murrayelizabeth
194808531323585349	Amy Moreno	65650 Gardner Harbors, Jamesborough, DE 86507	18899390450	john69
306748414124715583	Amanda Rice	54596 Meredith Landing Suite 796, Jessicaside, FL 33524	18342255685	epowell
352058981608919021	Paul Valencia	20350 Cruz Branch Suite 066, Port Rebeccaville, NC 93357	15659739743	timothy16
358952345726278865	Tyler Edwards	38915 Jackson Harbors, Lake Richardbury, VA 17115	13657841923	newtonjonathan
375978489541521459	Tyler Baird	682 Michael Mall Apt. 175, Barnesbury, WA 72525	10783261998	christianperry
4001369949069479 35	William Owens	388 Bennett Green, Jeffersonberg, CO 51400	16135920427	vicki89
4412343538066989 43	Casey Jones	85205 Jeremy Circle, Andersonburgh, PA 49081	10854214680	ryanroberts
461648128691895985	Scott Walker	022 Macias Key, South Kara, AK 21972	19025923910	robinsonrobert
542002715123277503	Brian Barrera	02149 Joseph Ville, Lake Taylor, DC 75531	12575513389	jordanfleming
625015803244653833	Lance Jimenez	3178 Lynch Summit Suite 010, Nelsonville, AR 50232	17668763258	johnsonkristine
711218390183164699	Samuel Gilbert	697 Alyssa Ridge Apt. 254, Christopherberg, ND 57211	10787914506	christinajackson
713814948362374664	Roger Novak	3506 Victoria Meadow, Lake Jennifer, MA 64628	11380088818	iestrada
759934297895786487	Brianna Miller	PSC 9583, Box 0869, APO AA 53076	14773246928	robert39
817297387357544316	Melissa Rodriguez	22667 Ashley Highway Apt. 356, West Donna, TX 41232	13195539962	alex79
901360840615820627	Yvonne Martin	164 Robbins Walk, West Wendy, NJ 41516	14516527621	brownjeffrey

Key Features:

- Admins have unrestricted access to all resident, caregiver, and nurse data.
- They can manage staff assignments for residents.

main.py

```
@app.route('/admin')
def admin():
    if session.get('RoleID') != 3:
        return "Access denied."

    connection = get_connection()
    try:
        with connection.cursor() as cursor:
            # 查询居民信息
            cursor.execute("SELECT * FROM Residents")
            residents = cursor.fetchall()

            # 查询活动日志
            cursor.execute("SELECT * FROM CareLog")
            activity_logs = cursor.fetchall()

            # 查询监护人信息
            cursor.execute("SELECT * FROM Guardians")
            guardians = cursor.fetchall()
    finally:
        connection.close()

    return render_template(
```

```
        'admin.html',
        residents=residents,
        activity_logs=activity_logs,
        guardians=guardians
    )


@app.route('/admin/assign_staff', methods=['POST'])
def assign_staff():
    if session.get('RoleID') != 3:
        return "Access denied."

    resident_id = request.form['resident_id']
    caregiver_id = request.form['caregiver_id']
    nurse_id = request.form['nurse_id']

    connection = get_connection()
    try:
        with connection.cursor() as cursor:
            # 确保居民已存在
            cursor.execute("SELECT ResidentID FROM Residents WHERE ResidentID = %s", (resident_id,))
            resident = cursor.fetchone()
            if not resident:
                return "Resident not found."

            # 更新居民的看护人员和护士
            # 这里可以根据业务需求设计更新操作
            # 如果要添加具体的关联数据，例如更新表格，请根据业务需求进行操作
            # 此处简单地返回操作完成的消息
            # 假设你有一个表来记录居民与护理人员的分配
            cursor.execute("""
                INSERT INTO ResidentStaffAssignments (ResidentID, CaregiverID, NurseID)
                VALUES (%s, %s, %s)
            """, (resident_id, caregiver_id, nurse_id))
        connection.commit()
    finally:
        connection.close()

    return redirect('/admin')
```

Task 10 – Considerations on Privacy and Security

1. Identification of Sensitive Data

The following tables and columns contain sensitive personal data that require special protection:

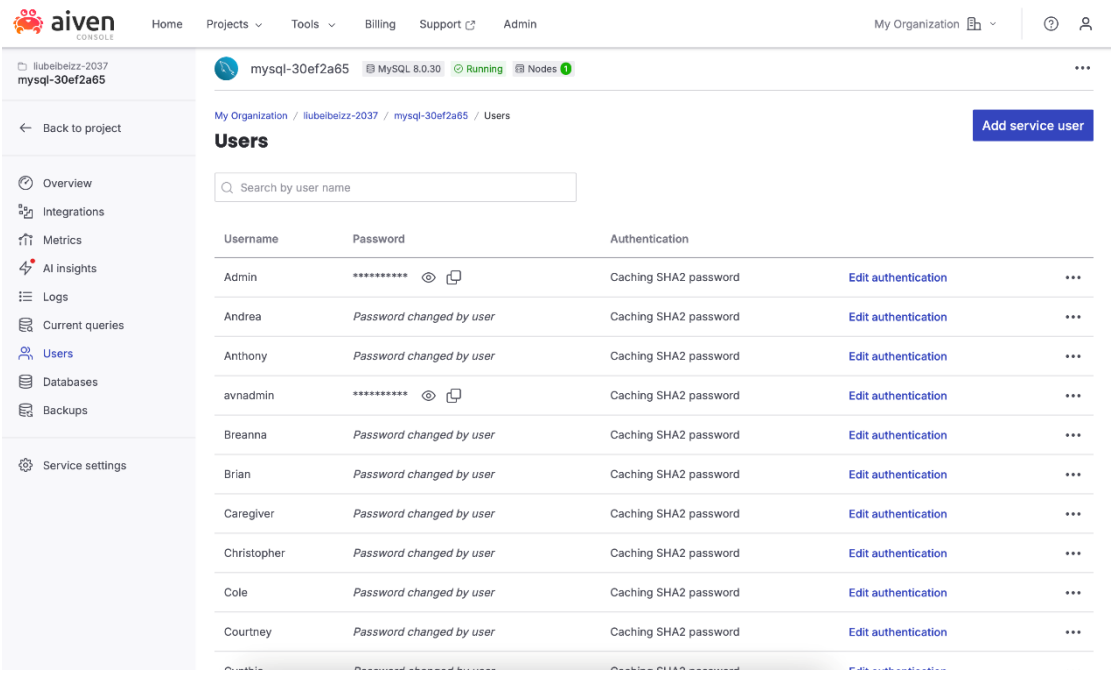
- **Table: CareLog**
  - Content: May contain health-related notes or activities.
- **Table: Guardians**
  - Name, Address, PhoneNumber, WechatID: PII that can uniquely identify a guardian.
- **Table: HealthRecord**
  - Content: Contains detailed medical information, a highly sensitive category under GDPR.
- **Table: Residents**
  - Name, Address, PhoneNumber, DateOfBirth, Gender, AllergyHistory, HealthNotes: All PII, including sensitive health data.
- **Table: UserRole**
  - Username & Password: Authentication credentials, requiring encryption and secure handling.

- Data protection measures that you would add to the design.

2. Data Protection Measures

To protect the above sensitive data, the following measures should be incorporated:

1. **Encryption**:
   - Store sensitive fields (Password, Content, Address, PhoneNumber) using strong encryption methods like AES-256.
   - Encrypt data at rest and in transit using protocols like TLS 1.3.



All the staff users have their own account and the password areencrypted by SHA-2

- Hashing with SHA-2:
    - Caching SHA2 Password utilizes the SHA-256 hashing algorithm, which is part of the SHA-2 family of cryptographic hash functions. This ensures that:
        - Passwords are not stored in plaintext.
        - Even if the database is compromised, the actual passwords cannot be easily recovered because hash functions are one-way (irreversible).
- Salting:
    - The SHA-256 hash includes a random salt value (a unique piece of data added to each password before hashing). This prevents rainbow table attacks, where precomputed hash values are used to crack passwords.
- Key Stretching:
    - Caching SHA2 Password includes iterations in the hashing process, making brute-force attacks computationally expensive.

Merits:

- **Irreversible**: SHA-2 is a cryptographic hashing algorithm. Once data is hashed, it cannot be reversed to its original form without brute-forcing all possible inputs, which is computationally infeasible for strong algorithms like SHA-256.

- **Consistency**: The same input always produces the same hash, ensuring the hashed IDs can be used as unique identifiers across database tables without revealing the original values.
- **No Plaintext Storage**: The original IDs are not stored in the database, reducing the risk of sensitive information exposure.

  Codes:

  Enforce Strong Password Policies

  - Add a strong password policy to ensure that users create secure passwords. This can include requirements for minimum length, special characters, and complexity.

  ```
  -- Enforce minimum password strength in MySQL (8.0+)

  SET GLOBAL validate_password.policy = 'STRONG';

  SET GLOBAL validate_password.length = 12; -- Minimum length of 12 characters

  SET GLOBAL validate_password.mixed_case_count = 1; -- Require both uppercase and lowercase

  SET GLOBAL validate_password.number_count = 1; -- Require at least one number

  SET GLOBAL validate_password.special_char_count = 1; -- Require at least one special character
  ```

  Lock Accounts After Failed Login Attempts

  - Prevent brute-force attacks by locking accounts after a certain number of failed login attempts.

  ```
  -- Enable account locking

  ALTER USER 'username'@'%' FAILED_LOGIN_ATTEMPTS 5 PASSWORD_LOCK_TIME 1;



  -- Explanation:

  -- FAILED_LOGIN_ATTEMPTS 5: Lock the account after 5 failed login attempts.
  ```

-- PASSWORD_LOCK_TIME 1: Automatically unlock the account after 1 day (or a specified duration).

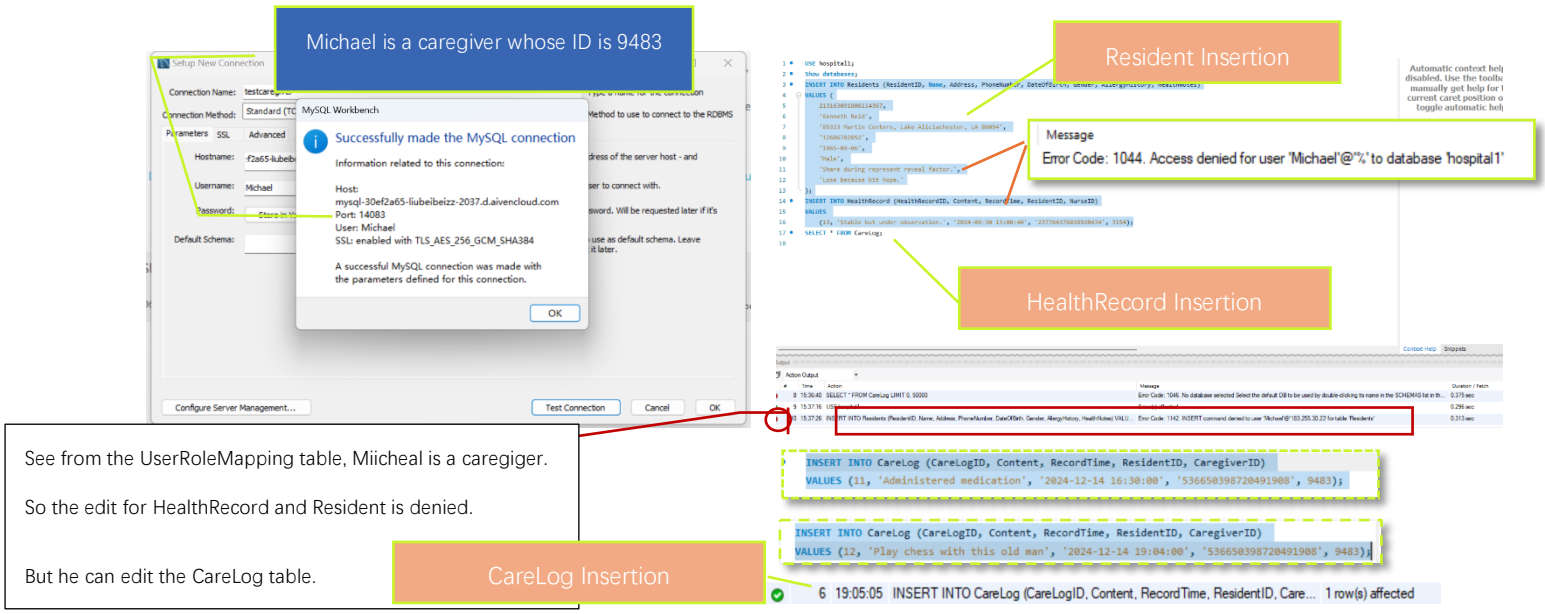
2. **Access Control**:
   - Implement role-based access control (RBAC) through the Role and UserRoleMapping tables to ensure users can only access data relevant to their role (e.g., Nurses access HealthRecord, not administrative data).

```
SHOW GRANTS FOR 'Michael'@'%';
```

Result Grid	Filter Rows:	Expo
	Grants for Michael@%	
▶	GRANT USAGE ON *.* TO 'Michael'@'%'	
	GRANT 'Caregiver'@'%','Nurse'@'%' TO 'Mic...	

3.

   - Limit access to sensitive columns like Password and HealthNotes.



4. **Audit Logging**:
   - Record access and modification events on sensitive data, including the user and timestamp, to monitor for unauthorized access.

5. **Data Masking**:
    - Mask sensitive information (e.g., PhoneNumber, Address) in user interfaces unless explicitly required for authorized tasks.
    - The ResidentID and GuardianID are the Chinese National ID number, which is very sensitive. Hence, the **SHA2 function** is used to generate a secure, one-way hash of the original ResidentID and GuardianID. This means the original ID is transformed into a fixed-length, unique, and irreversible string.
    - Run the Task7_query.sql, which will give you a masked Chinese National ID number.

      Code Examples for masking:

      INSERT INTO Residents (ResidentID, Name, Address, PhoneNumber, DateOfBirth, Gender, AllergyHistory, HealthNotes)

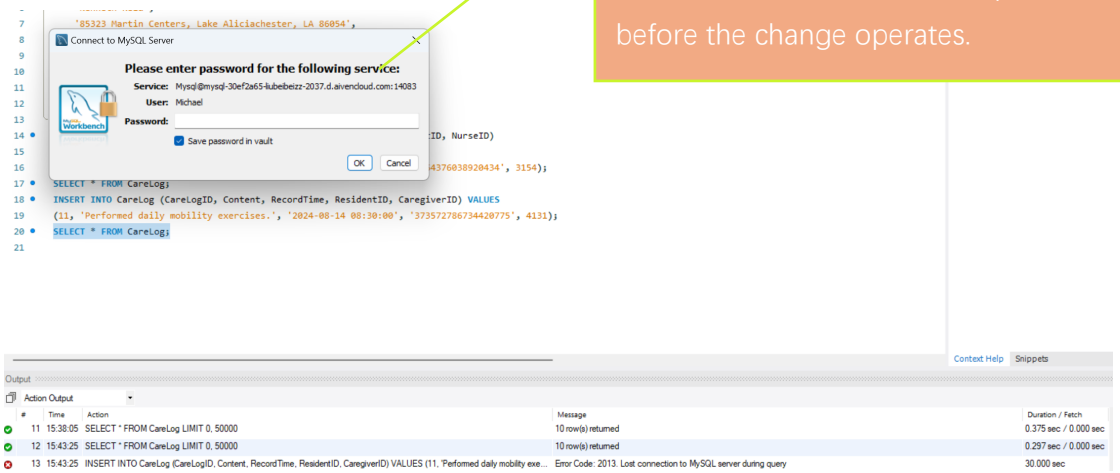
      VALUES (SHA2('2243601842862720 79', 256), 'Harry Hendricks', '9671 Gillespie Greens Suite 156, Kelseychester, WV 18335', '18497938812', '1960-07-11', 'Male', 'Myself red will short.', 'Fly offer let friend room.');

      INSERT INTO Guardians (GuardianID, Name, Address, PhoneNumber, WechatID)

      VALUES (SHA2('0474070068535255 60', 256), 'Kenneth Hernandez', '709 Patricia Locks, North Heidi, ND 44262', '19501575689', 'murrayelizabeth');

6. **Password Security**:
    - Store passwords using a hashing algorithm like I sufficient salt to prevent brute force attacks.

To operate the tables in allowed range, one should enter the account password before the change operates.



Enter password before any edit.

7. **Anonymization and Pseudonymization**:
    - Anonymize ResidentID, CaregiverID, and similar identifiers for analytics purposes.
    - Pseudonymize health-related data in HealthRecord and CareLog for research or secondary purposes.

- GDPR compliance.

To ensure GDPR compliance, the following principles must be adhered to:

1. **Data Minimization**:
    - Collect and store only the necessary data. For example, avoid storing WeChat IDs if not critical to system operations.
2. **Purpose Limitation**:
    - Clearly define the purpose of data processing in system documentation. Use sensitive data like HealthNotes solely for providing care, not for secondary purposes without explicit consent.
3. **Storage Limitation**:
    - Automatically delete or anonymize data after it is no longer needed. For example, delete records from CareLog after a defined retention period (e.g., 5 years).
4. **Data Subject Rights**:
    - Implement features that allow residents, guardians, and caregivers to:
        - Access their data (e.g., via a secure web portal).
        - Rectify incorrect information.
        - Request deletion of personal data where legally permissible.
5. **Lawful Processing**:
    - Obtain explicit consent for processing health-related data.
    - Provide clear privacy notices explaining how data is used.

Recommendations

1. **Security by Design**:
    - Integrate privacy and security into every stage of development, including database design, APIs, and user interfaces.
2. **Regular Penetration Testing**:
    - Conduct regular security audits and penetration testing to identify and fix vulnerabilities.
3. **Data Breach Policy**:
    - Create and test a data breach response plan to comply with GDPR's 72-hour notification rule.
4. **Training and Awareness**:
    - Train staff on GDPR principles and best practices for handling sensitive data.

By implementing these measures, the system will ensure strong privacy and security, protect sensitive data, and maintain compliance with GDPR requirements.

Task 11 – Critical Evaluation

This project effectively demonstrated our ability to design and implement a secure, scalable, and efficient relational database system for a nursing facility. By analyzing requirements, we developed a robust ER model and implemented it using MySQL, ensuring normalization and adherence to GDPR standards for data privacy. The use of SHA2 encryption for sensitive data, role-based access control (RBAC), and deployment on a cloud server highlighted our focus on security, scalability, and accessibility.

The cloud-hosted database enabled seamless remote access and real-time testing, while Flask-based application development ensured user-friendly interactions for caregivers, nurses, and admins.

For Liu Beibei:

Liu Beibei's work on **Tasks 3, 9, and 10** and deploying the database on a **cloud server** contributed significantly to enhancing practical skills in database security, cloud technologies, and relational database design. The value gained includes:

1. **Cloud Database Deployment**
2. **Database Application Development (Task 9)**:
    - The role-specific dashboards implemented role-based access control (RBAC), showcasing practical ways to manage data visibility and functionality.
3. **Data Privacy and Security (Task 10)**:
    - Implementing **SHA2 hashing** for passwords and sensitive identifiers (e.g., ResidentID and GuardianID) deepened knowledge of cryptographic hashing techniques.

For Han Baodi:

Han Baodi's work on **Tasks 1, 2, 4–8, and 11** strengthened understanding and implementation of relational database systems through logical design, normalization, and querying. Key value includes:

1. **Requirement Analysis and ER Modeling (Tasks 1 & 2):**
2. **Normalization and Logical Mapping (Tasks 4 & 5):**
3. **Database Implementation and Queries (Tasks 6–8):**
4. **Security and Access Control**:
    - Adding constraints like **ON DELETE NO ACTION** ensured data consistency and security when handling foreign keys.
5. **Cloud Database Interaction**:
    - Testing and connecting the **cloud-hosted database** provided hands-on experience with remote database management and tools for handling real-world scalability.