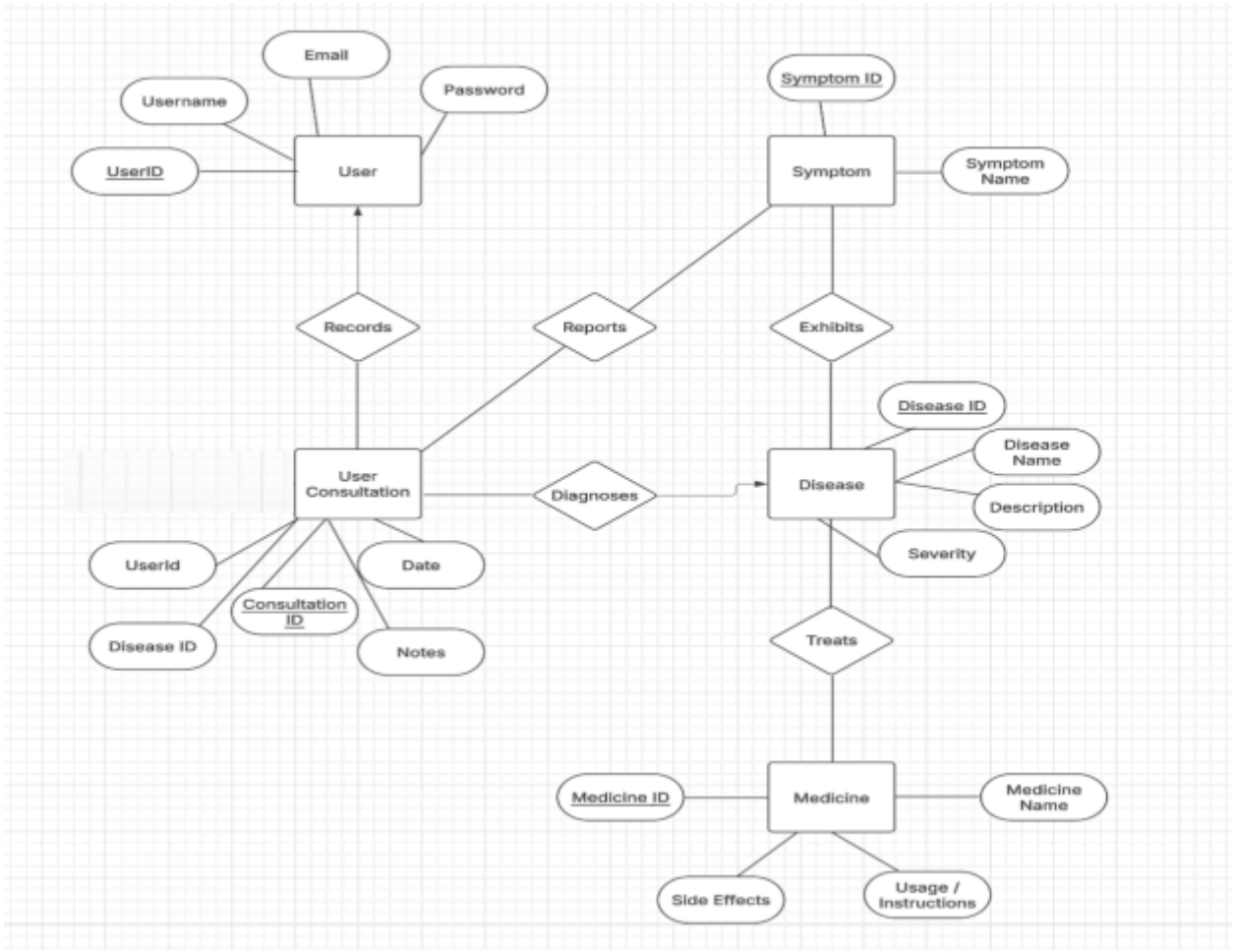
ER Diagram:



5 entities:

1. User
2. User Consultation
3. Symptom
4. Disease
5. Medicine

2 types of relationships:

- 1–to-many
- many-to-many

Discussion of ER Diagram:

Our diagram has five entities: User, User Consultation, Symptom, Disease and Medicine. We chose to make User an entity as there will be multiple Users, with each User having distinct personal information (userid, email, username, etc.) and they need to be tracked. Each User therefore is a distinct object that can log into our system, with UserId being a unique identifier. A User Consultation is a single interaction between a doctor and a User, and will have a description, Symptom Id, etc. to describe that consultation. Since multiple Users can have multiple different consultations all at different times, we chose to make them an entity to log all of them properly as they are distinct entities that can be tracked, edited, etc.. Symptoms are signs/effects felt by User a result of a disease. We chose to make symptoms an entity because

symptoms are unique identifiers/descriptions that can be used to explain diseases and can be associated with multiple different diseases. This same justification goes for the Diseases table, which have scientific names, descriptions and levels of severity. Medicines similarly are entities because they are unique, can be prescribed for different diseases, and therefore can not just be an attribute of a disease. Medicines are prescribed by healthcare personnel in response to a disease.

Our diagram has 5 relationships: User - User Consultation (Records) (1-to-Many), User Consultation - Symptom (Reports) (Many-to-Many), User Consultation - Disease (Diagnoses) (1-to-Many), Symptom - Disease (Exhibits) (Many-to-Many), Disease - Medicine (Treats) (Many-to-Many). The first relationship is (1-to-Many) because a User can have many consultations, but only one User will ever be associated with a single consultation, as per our application requirements that the User must follow. The second relationship is Many to Many because a consultation can involve multiple symptoms, and the same symptom can appear in multiple consultations, as per a logical understanding of how real world consultations with a doctor go. The third relationship is User Consultation - Disease (Diagnoses) which is 1-to-Many because one consultation will only yield one disease diagnosis (as our application will require that if somebody, for example, is diagnosed with Cancer and the Flu to record those as two different consultation records) but one disease can be associated with many consultations. The fourth relationship is Many-to-Many because one disease can have many symptoms and one symptom can be linked to many diseases, as per a logical understanding of how diseases and symptoms are related in real life. Lastly, the fifth relationship is also Many-to-Many as a single disease can be treated by many different medicines and one medicine can treat multiple diseases, again due to how diseases and medicines are related in real life.

Relational Schema:

User(UserID:INT [PK], Username:VARCHAR(50), Email:VARCHAR(100), Password:VARCHAR(255))

Symptom(SymptomID:INT [PK], SymptomName:VARCHAR(100))

Disease(DiseaseID:INT [PK], DiseaseName:VARCHAR(100), Description:TEXT, Severity:ENUM('Mild', 'Moderate', 'Severe')

Medicine(MedicineID:INT [PK], MedicineName:VARCHAR(100), SideEffects:TEXT, UsageInstruction:TEXT)

UserConsultation(ConsultationID:INT [PK], UserID:INT [FK to User.UserID], DiseaseID:INT [FK to Disease.DiseaseID], Date:DATE, Notes:TEXT)

Disease_Symptom(DiseaseID:INT [PK, FK to Disease.DiseaseID], SymptomID:INT [PK, FK to Symptom.SymptomID])

Disease_Medicine(DiseaseID:INT [PK, FK to Disease.DiseaseID], MedicineID:INT [PK, FK to Medicine.MedicineID])

UserConsultation_Symptom(ConsultationID:INT [PK, FK to UserConsultation.ConsultationID], SymptomID:INT [PK, FK to Symptom.SymptomID])

The last 3 tables (Disease_Symptom, Disease_Medicine, and UserConsultation_Symptom) serve as junction tables for our many-to-many relationships in our ER diagram and also for our table to remain normalized.

Database Schema Normalization Process:

For a relation to be in 3NF: All non-prime attributes will be fully dependent on the primary key, and the non-prime attributes should not depend on another non-prime attribute. This is the standard for which we will evaluate and prove each of our tables follows the normalization standard of 3NF.

USER TABLE:

User(UserID:INT [PK], Username:VARCHAR(50), Email:VARCHAR(100), Password:VARCHAR(255))

- **Primary Key:** UserID - uniquely determines all attributes
- Since there is only 1 attribute in primary key, no partial dependencies exist
- No transitive dependencies exist because… (all attributes depend directly on UserID)
    - Having the same Username doesn't imply having the same Email or Password (different users can share names)
    - Having the same Email doesn't imply having the same Username or Password (a user could change their username while keeping the same email)
    - Having the same Password doesn't imply having the same Username or Email (many different users could use similar passwords)
- Username, Email, and Password are all properties of the user entity

SYMPTOM TABLE:

Symptom(SymptomID:INT [PK], SymptomName:VARCHAR(100))

- **Primary Key**: SymptomID - uniquely determines SymptomName
- Since there is only 1 attribute in primary key, no partial dependencies exist
    - This table has only one non-key attribute (SymptomName). Since there are no other non-key attributes for SymptomName to functionally determine, transitive dependencies cannot exist in this table by definition.
- No transitive dependencies exist because…

DISEASE TABLE:

Disease(DiseaseID:INT [PK], DiseaseName:VARCHAR(100), Description:TEXT, Severity:ENUM('Mild', 'Moderate', 'Severe')

- **Primary Key:** DiseaseID - uniquely determines all attributes

- Since there is only 1 attribute in primary key, no partial dependencies exist
- No transitive dependencies because… (all attributes directly depend on DiseaseID)
    - Different diseases might share similar names but have different descriptions or severity levels
    - Similar descriptions could apply to diseases with different names or severity levels
    - Many diseases could share the same severity level while having different names and descriptions
- DiseaseName, Description, and Severity are all properties of the disease entity

MEDICINE TABLE:

Medicine(MedicineID:INT [PK], MedicineName:VARCHAR(100), SideEffects:TEXT, UsageInstruction:TEXT)

- **Primary Key:** MedicineID - uniquely determines all attributes
- Since there is only 1 attribute in primary key, no partial dependencies exist
- No transitive dependencies because… (all attributes depend directly on MedicineID)
    - Medicines with the same name (e.g., generic vs. brand) might have different side effects or usage instructions
    - Medicines with similar side effects often have different names and usage instructions
    - Similar usage instructions could apply to medicines with different names and side effects
    - Therefore, no non-key attribute functionally determines any other, eliminating the possibility of transitive dependencies.
- MedicineName, SideEffects, and UsageInstruction are all properties of the medicine entity

USERCONSULTATION TABLE:

UserConsultation(ConsultationID:INT [PK], UserID:INT [FK to User.UserID], DiseaseID:INT [FK to Disease.DiseaseID], Date:DATE, Notes:TEXT)

- **Primary Key:** ConsultationID - uniquely determines all attributes
- **Foreign Keys:** UserID, DiseaseID
- No transitive dependencies because…
    - The same user can have consultations for different diseases on different dates with different notes
    - The same disease can be diagnosed in different users on different dates with different notes
    - Consultations on the same date can involve different users, diseases, and notes
    - Similar notes could be recorded for consultations involving different users, diseases, and dates
- No partial dependencies (only one attribute in Primary Key)

For all 3 of our join tables:

- These junction tables contain only key attributes (composite primary keys consisting entirely of foreign keys) and no non-key attributes. Since transitive dependencies require non-key attributes functionally determining other non-key attributes, and these tables have no non-key attributes, transitive dependencies cannot exist in these tables by definition. Therefore, these tables automatically satisfy 3NF.

DISEASE_SYMPTOM TABLE:

- Join table for many-to-many relationship between Disease and Symptom tables, integral to our table retaining 3NF normalization.
- Disease_Symptom(DiseaseID:INT [PK, FK to Disease.DiseaseID], SymptomID:INT [PK, FK to Symptom.SymptomID])
    - **Composite Primary Key:** (DiseaseID, SymptomID)
    - No non-key attributes -> no partial or transitive dependencies

DISEASE MEDICINE TABLE:

- Join table for many-to-many relationship between Disease and Medicine tables, integral to our table retaining 3NF normalization.
- Disease_Medicine(DiseaseID:INT [PK, FK to Disease.DiseaseID], MedicineID:INT [PK, FK to Medicine.MedicineID])
    - Composite Primary Key: (DiseaseID, MedicineID)
    - No non-key attributes -> no partial or transitive dependencies

USERCONSULTATION_SYMPTOM TABLE:

- Join table for the many-to-many relationship between UserConsultation and Symptom tables, integral to our table retaining 3NF normalization.
- UserConsultation_Symptom(ConsultationID:INT [PK, FK to UserConsultation.ConsultationID], SymptomID:INT [PK, FK to Symptom.SymptomID])
    - Composite Primary Key: (ConsultationID, SymptomID)
    - No non-key attributes => no partial or transitive dependencies

As such, we have proven with all the above analysis that ***our database relational schema is already normalized according to 3NF***, showing our process and reasoning of checking for normalization for each step, thus fulfilling this requirement.