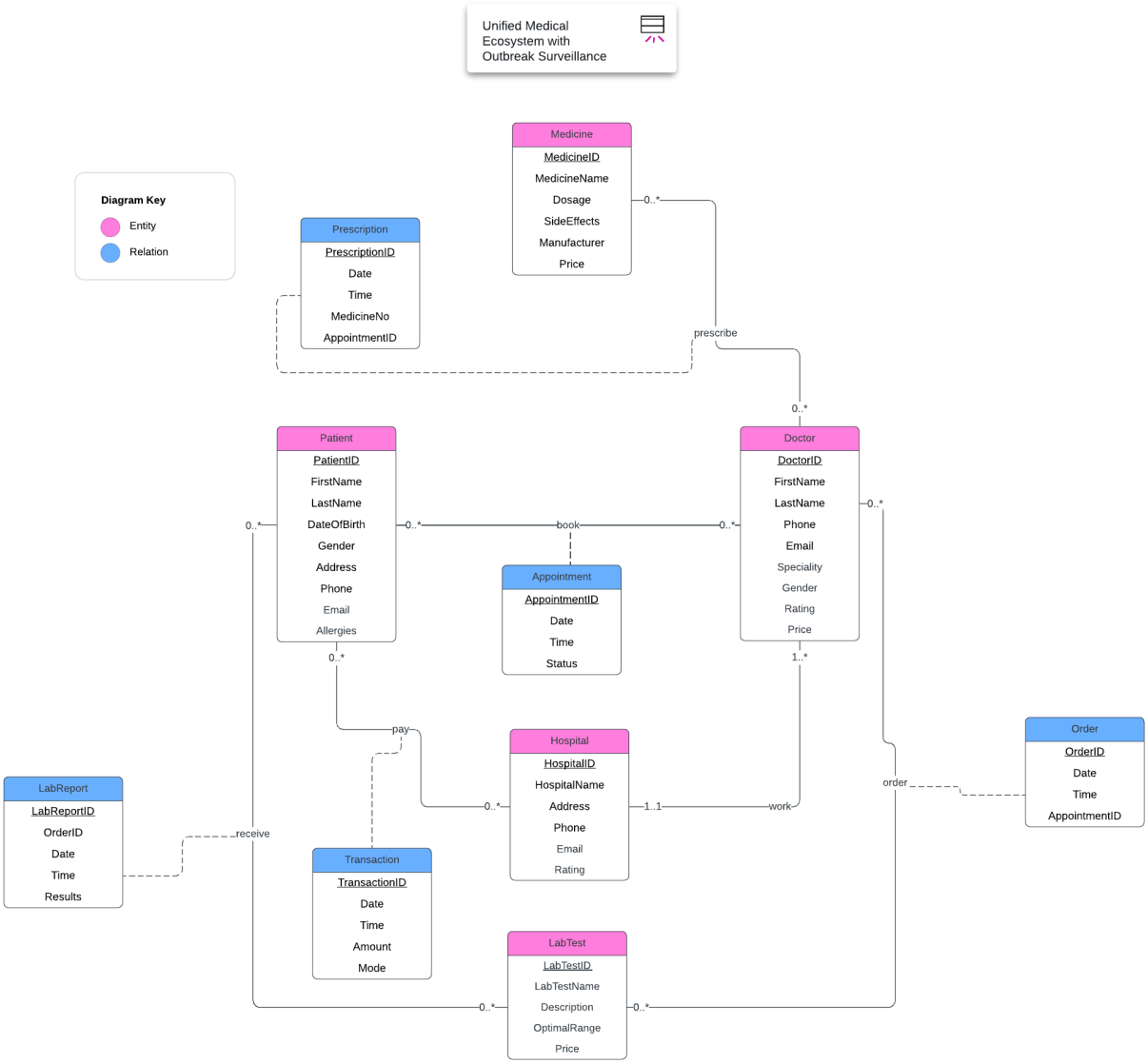
Database Design

UML diagram:



Entities and Attributes:

1. **Hospital**

As per our design, we will have a number of hospitals to be managed using a single application. Relationships with other entities are explained in later sections.

a. **HospitalID** - Unique Identifier allocated to each hospital

b. **HospitalName** - Name of the hospital - This name may not be unique. However, it will be unique when combined with the address.
c. **Address** - location of the hospital. Includes Street Address, City, State, PIN
d. **Phone** - Phone number of the hospital. Follows international number format.
e. **Email** - Email ID of the hospital. Follows email ID format (Eg., example@abc.com)
f. **Rating** - Rating of the hospital. It could be a real value (from 0.0 to 5.0). This value would be randomly populated. However, in the future, the application could be extended to include patients' hospital and doctor ratings after their visit. Something like a survey could be sent to the patients.

2. **Doctor**

As per our design, each hospital could manage a number of doctors. Each doctor could work in only one hospital. Relationships with other entities are explained in later sections.

a. **DoctorID** - Unique identifier is allocated to each doctor. This is unique across the entire application.
b. **FirstName** - First name of the doctor
c. **LastName** - Last name of the doctor
d. **Phone** - Phone number of the doctor. Follows international number format (111-222-3333)
e. **Email** - Email ID of the doctor. Follows email ID format (Eg., example@abc.com)
f. **Speciality** - Speciality of the doctor. Something like Cardiologist, Physiologist, etc.
g. **Gender** - Gender of the doctor. Possible values include ('Male', 'Female', 'Other')
h. **Rating** - Rating of the doctor. It could be a real value (from 0.0 to 5.0). This value would be randomly populated. Similar to hospital ratings.
i. **Price** - Price of each doctor's appointment. This would be the cost added to the patient's bill.

3. **Patient**

As per our design, the patient is not associated with any hospital. They are an independent entity who could go to any doctor in any hospital.

a. **PatientID** - Unique identifier is allocated to each patient. This is unique across the entire application.
b. **FirstName** - First name of the patient
c. **LastName** - Last name of the patient
d. **DateOfBirth** - Birth date of the patient. Follows YYYY-MM-DD format.
e. **Gender** - Gender of the patient. Possible values include ('Male', 'Female', 'Other')
f. **Address** - Address of the patient. Includes Street Address, City, State, PIN
g. **Phone** - Phone number of the doctor. Follows international number format (111-222-3333)
h. **Email** - Email ID of the doctor. Follows email ID format (Eg., example@abc.com)
i. **Allergies** - List of allergies that the patient may have. Comma-separated values.

4. **LabTest**

As per our design, patients would be booked for a number of lab tests by the doctor during their appointment.

a. **LabTestID** - A unique identifier is allocated to each lab test. This is unique across the entire application.
b. **LabTestName** - Name of the test, such as a blood sugar level test.

c. **Description** - Description of the test.
d. **OptimalRange** - Optimal range for the test. For example, blood sugar levels have an optimal range of 70 to 99 mg/dL (3.9 to 5.5 mmol/L). This could be included in the report so that the patients can understand.
e. **Price** - Price of each lab test appointment. This would be the cost added to the patient's bill.

5. **Medicine**
   As per our design, patients would be prescribed a number of medicines along with the doctor's dosage during their appointment.
   a. **MedicineID** - A unique identifier is allocated to each lab test. This is unique across the entire application.
   b. **MedicineName** - Name of the medicine.
   c. **Dosage** - Dosage of the medicine. It could be 500 mg, 10 ml, etc.
   d. **SideEffects** - Side effects of the medicine.
   e. **Manufacturer** - Manufacturer of the drug
   f. **Price** - Price of each dosage of the medicine.

Relations and attributes:

1. Patient-**book**-Doctor : **Appointment**
   As per our design, each patient would be able to book any number of doctor appointments as long as the times didn't clash with each other. Appointment relation lies between the patients and the doctors. It is also possible that there is a patient with no appointments with a doctor. We initially planned to have this as an entity but later decided to have this as a relation.
   a. **AppointmentID** - A unique identifier is allocated to each appointment. This is unique across the entire application. There are two designs possible. Doctors may create empty appointment slots; the status field will reflect the booked/available status. There is also one more possibility we are considering where doctor entity could have another field called working hours. The patients would be able to book an appointment with a doctor as long as it doesn't clash with other appointments and falls within the doctor's working hours.
   b. **Date** - Date of the appointment
   c. **Time** - Time of the appointment
   d. **Status** - Status of the appointment - (booked/available, booked/canceled/rescheduled)
2. Doctor-**prescribe**-Medicine: **Order**
   As per our design, doctors could book 0 or more lab orders for a patient during their appointment. Order relation lies between the doctor, labs, and indirectly with the patient via appointment. We could fit this into the appointment. However, having this separate would help us organize patients' lab orders and their billing.
   a. **OrderID** - A unique identifier is allocated to each order.
   b. **Date** - Date of the order

c. **Time** - Time of the order
d. **AppointmentID** - Appointment ID of the patient during which the doctor made this order.

3. **Prescription**
   During each appointment, a doctor could give zero or one prescription with one or more medicines. We could promote prescription to its own entity, but we decided not to do that for now. Instead, we created an attribute (MedicineNo) to identify the medicines within a prescription. That is why the cardinality is (0..*), indicating a patient could also be prescribed no medicines.
   a. **PrescriptionID** - A unique identifier is allocated to each order.
   b. **Date** - Date of the prescription
   c. **Time** - Time of the prescription
   d. **MedicineNo** - Identifier to identify the medicines within a single prescription. Starting with 1.
   e. **AppointmentID** - Appointment ID of the patient during which the doctor made this prescription.
4. Patient-**receive**-Report: **LabReport**
   For each lab order, there could be only one patient. However, each patient could have zero or more lab reports. This report could also be added as an attribute to the order.
   a. **LabReportID** - A unique identifier is allocated to each lab report.
   b. **Date** - Date of the lab report.
   c. **Time** - Time of the lab report.
   d. **Results** - Results of the lab report include the values of the lab test that was conducted.
5. Patient-**pay**-Hospital: **Transaction**
   Transaction is the relation between the patient and the hospital. A patient may or may not have a transaction (bill payment) with the hospital. Hospitals may or may not have transactions with patients. In an ideal scenario, this shouldn't happen, as patients would be paying their hospitals. But we left it this way to allow a scenario where hospitals to not accept any cash, for example, hospitals giving free treatments or newly started hospitals without any transaction to report. The amount is calculated based on the customer's appointments and lab orders.
   a. **TransactionID** - A unique identifier is allocated to each patient transaction.
   b. **Date** - Date of the transaction
   c. **Time** - Time of the transaction
   d. **Amount** - Amount which the patient paid to the hospital.
   e. **Mode** - Mode of transaction. Possible values could be ('credit card', 'debit card', 'cash' etc.)
6. Doctor-**work**-Hospital: As per our design, each doctor works at only one hospital, and each hospital has one or many doctors.

Normalization

These are the initial entities and relations.

Entities:
Doctor(DoctorId, FirstName, LastName, Phone, Email, Gender, Rating, Price, HospitalId, Password)
Patient(PatientId, FirstName, LastName, DateOfBirth, Gender, Address, Phone, Email, Allergies, Password)
Medicine(MedicineId, MedicineName, Dosage, SideEffects, Manufacturer, Price)
Hospital(HospitalId, HospitalName, Address, Phone,Email, Rating)
LabTest(LabTestId, LabTestName, Description, OptimalRange, Price)

Relations:
Prescription(PrescriptionId, Date, Time, AppoinmentId, DoctorId, MedicineId)
Appointment(AppointmentId, Date, Time, Status, PatentId, DoctorId)
Order(OrderId, Date, Time, AppoinmentId, DoctorId, LabTestId)
Transaction(TransactionId, Date, Time, Amount, Mode, HospitalId, PatientId)
LabReport(LabReportId, Date, Time, Result, LabTestId, PatientId)

After 1NF
Entities:
Doctor(DoctorId, FirstName, LastName, Gender, Phone, Email, Rating, Price, HospitalId, Password)
Doctors should have one phone number and email. To maintain atomicity (a single value per field),

Patient(PatientId, FirstName, LastName, DateOfBirth, Gender, Phone, Email, Address, Password)
PatientAllergy(id, PatientId, Allergy)
A patient may have multiple allergies. These were split into separate relations to ensure atomic fields, adhering to the 1NF requirement. We will not be using this id field, but according to the rules of MySql, if the table does not have a primary key, it will generate it automatically. So we are generating the id field and keeping it as primary auto increment.

Medicine(MedicineId, MedicineName, Dosage, Manufacturer, Price)
MedicineSideEffect(id, MedicineId, SideEffect)
Medicines can have multiple side effects, so this field is moved to a separate relation to avoid violating 1NF. We will not be using this id field, but according to the rules of MySql, if the table does not have a primary key, it will generate it automatically. So we are generating the id field and keeping it as primary auto increment.

Hospital(HospitalId, HospitalName, Address, Rating, Phone, Email)

Hospitals should have one phone number and email. To maintain atomicity (a single value per field),

LabTest(LabTestId, LabTestName, Description, OptimalRange, Price) (All are atomic)

Relations:
Prescription(PrescriptionId, Timestamp, AppointmentId, DoctorId, MedicineId)
The Date and Time fields are merged into a Timestamp field to avoid redundancy and to make the field atomic.

Appointment(AppointmentId, Timestamp, Status, PatientId, DoctorId)
The Date and Time fields are merged into a single Timestamp to ensure atomicity.

Order(OrderId, Timestamp, AppointmentId, DoctorId, LabTestId)
Similar to the above, the Date and Time fields are combined into a Timestamp for atomicity.

Transaction(TransactionId, Timestamp, Amount, Mode, HospitalId, PatientId)
The Date and Time fields are replaced with a Timestamp field to simplify and maintain atomicity.

LabReport(LabReportId, Timestamp, Result, LabTestId, PatientId)
The Date and Time fields are merged into a Timestamp, following the atomicity requirement of 1NF.

After 2NF:
Entities:
Doctor(DoctorId, FirstName, LastName, Gender, Rating, Price, Phone, Email, HospitalId, Password)
Patient(PatientId, FirstName, LastName, DateOfBirth, Gender, Address, Phone, Email, Password)
PatientAllergy(id,PatientId, Allergy)
Medicine(MedicineId, MedicineName, Dosage, Manufacturer, Price)
MedicineSideEffect(id,MedicineId, SideEffect)
Hospital(HospitalId, HospitalName, Address, Rating, Phone, Email)
LabTest(LabTestId, LabTestName, Description, OptimalRange, Price)
Do not have composite keys, so there are no partial dependencies within the entities. They remain the same as they are already in 2NF.

Relations:
PrescriptionDetails(PrescriptionId, AppointmentId, DoctorId, MedicineId)
PrescriptionTimestamp(PrescriptionId, Timestamp)
The Timestamp depends only on AppointmentId, not on DoctorId or MedicineId. Decomposing resolves the partial dependency.

AppointmentDetails(AppointmentId, PatientId, DoctorId)

AppointmentStatus(AppointmentId, Timestamp, Status)
Timestamp and Status are only dependent on AppointmentId, not on PatientId or DoctorId, so decomposition resolves this partial dependency.

OrderDetails(OrderId, DoctorId, LabTestId)
OrderTimestamp(OrderId, AppointmentId, Timestamp)
Timestamp depends only on AppointmentId, not on DoctorId or LabTestId, so splitting the table eliminates the partial dependency.

Transaction (TransactionId, Timestamp, Amount, Mode, HospitalId, PatientId)
No compound key

LabReport (LabReportId, Timestamp, Result, LabTestId, PatientId)
The key (LabTestId, PatientId) determines each other thing, so no partial dependency.

After 3NF
Doctor(DoctorId, FirstName, LastName, Gender, Rating, Price, Phone, Email, HospitalId, Password)
FDs:
DoctorId → FirstName, LastName, Gender, Rating, Price, Phone, Email, Password, HospitalId
Minimal Basis:
The FD DoctorId → FirstName, LastName, Gender, Rating, Price, Phone, Email, HospitalId, Password is already minimal.
Superkey Check:
DoctorId is a superkey because it uniquely identifies all attributes.
The relation is already in 3NF, as all non-prime attributes are fully dependent on a superkey.

Patient(PatientId, FirstName, LastName, DateOfBirth, Gender, Address, Phone, Email, Password)

FDs:
PatientId → FirstName, LastName, DateOfBirth, Gender, Address, Phone, Email, Password
Minimal Basis:
PatientId → FirstName, LastName, DateOfBirth, Gender, Address, Phone, Email, Password is minimal.
Superkey Check:
PatientId is a superkey.
The relation is already in 3NF.

PatientAllergy(id, PatientId, Allergy)
FDs:

Id -→ PatientId, Allergy
Minimal Basis:
Id -→ PatientId, Allergy is minimal.
Superkey Check:
Id is a superkey.
The relation is already in 3NF.

Medicine(MedicineId, MedicineName, Dosage, Manufacturer, Price)
FDs:
MedicineId → MedicineName, Dosage, Manufacturer, Price
Minimal Basis:
MedicineId → MedicineName, Dosage, Manufacturer, Price is minimal.
Superkey Check:
MedicineId is a superkey.
The relation is already in 3NF.

MedicineSideEffect(id, MedicineId, SideEffect)
FDs:
Id -→ MedicineId , SideEffect
Minimal Basis:
Id -→ MedicineId , SideEffect is minimal.
Superkey Check:
MedicineId is a superkey.
The relation is already in 3NF.

Hospital(HospitalId, HospitalName, Address, Rating, Phone, Email)
FDs:
HospitalId → HospitalName, Address, Rating, Phone, Email
Minimal Basis:
HospitalId → HospitalName, Address, Rating, Phone, Email is minimal.
Superkey Check:
HospitalId is a superkey.
Conclusion: The relation is already in 3NF.

LabTest(LabTestId, LabTestName, Description, OptimalRange, Price)
FDs:
LabTestId → LabTestName, Description, OptimalRange, Price
Minimal Basis:
LabTestId → LabTestName, Description, OptimalRange, Price is minimal.
Superkey Check:
LabTestId is a superkey.
The relation is already in 3NF.

PrescriptionDetails(PrescriptionId, AppointmentId, DoctorId, MedicineId)
FDs:
PrescriptionId → AppointmentId, DoctorId, MedicineId
Minimal Basis:
PrescriptionId → AppointmentId, DoctorId, MedicineId is minimal.
Superkey Check:
PrescriptionId is a superkey.
The relation is already in 3NF.

PrescriptionTimestamp(PrescriptionId, Timestamp)
FDs:
PrescriptionId → Timestamp
Minimal Basis:
PrescriptionId → Timestamp is minimal.
Superkey Check:
PrescriptionId is a superkey.
The relation is already in 3NF.

AppointmentDetails(AppointmentId, PatientId, DoctorId)
FDs:
AppointmentId → PatientId, DoctorId
Minimal Basis:
AppointmentId → PatientId, DoctorId is minimal.
Superkey Check:
AppointmentId is a superkey.
Conclusion: The relation is already in 3NF.

AppointmentStatus(AppointmentId, Timestamp, Status)
FDs:
AppointmentId → Timestamp, Status
Minimal Basis:
AppointmentId → Timestamp, Status is minimal.
Superkey Check:
AppointmentId is a superkey.
Conclusion: The relation is already in 3NF.

OrderDetails(OrderId, DoctorId, LabTestId)
FDs:
OrderId → DoctorId, LabTestId
Minimal Basis:
OrderId → DoctorId, LabTestId is minimal.
Superkey Check:
OrderId is a superkey.

Conclusion: The relation is already in 3NF.

OrderTimestamp(OrderId, AppointmentId, Timestamp)
FDs:
OrderId → AppointmentId, Timestamp
Minimal Basis:
OrderId → AppointmentId, Timestamp is minimal.
Superkey Check:
OrderId is a superkey.
Conclusion: The relation is already in 3NF.

Transaction(TransactionId, Timestamp, Amount, Mode, HospitalId, PatientId)
FDs:
TransactionId → Timestamp, Amount, Mode, HospitalId, PatientId
Minimal Basis:
TransactionId → Timestamp, Amount, Mode, HospitalId, PatientId is minimal.
Superkey Check:
TransactionId is a superkey.
Conclusion: The relation is already in 3NF.

LabReport(LabReportId, Timestamp, Result, LabTestId, PatientId)
FDs:
LabReportId → Timestamp, Result, LabTestId, PatientId
Minimal Basis:
LabReportId → Timestamp, Result, LabTestId, PatientId is minimal.
Superkey Check:
LabReportId is a superkey.
Conclusion: The relation is already in 3NF.

Each functional dependency has a superkey on the left-hand side. All relations are free of transitive dependencies. The minimal basis for each relation is already applied, and no further decomposition is required.

The provided schema is in 3NF.

Final Schema

Thus, the final schema is
1. Doctor(DoctorId: INT [PK], FirstName: VARCHAR(50), LastName: VARCHAR(50), Gender: CHAR(1), Rating: DECIMAL(3,2), Price: DECIMAL(10,2), Phone: VARCHAR(15), Email: VARCHAR(100), HospitalId: INT [FK to Hospital.HospitalId], Password: CHAR(32))

Constraints:

- Gender must be one of ['M', 'F', 'O'].
- Rating should be between 0 and 5.
- Price should be a non-negative value.
- The MD5 hash produces a 128-bit value, which is typically represented as a 32-character hexadecimal string.
- The maximum length of international phone numbers is typically 15 digits as per the ITU-T E.164 standard, which defines the format for international phone numbers.
- This includes the country code (up to 3 digits) and the subscriber number (up to 12 digits).
- Doctor can have only one email and phone number.

2. Patient(PatientId: INT [PK], FirstName: VARCHAR(50), LastName: VARCHAR(50), DateOfBirth: DATE, Gender: CHAR(1), Address: VARCHAR(255), Phone: VARCHAR(15), Email: VARCHAR(100), Password: CHAR(32))

Constraints:

- Gender must be one of ['M', 'F', 'O'].
- DateOfBirth should be a valid past date (patients cannot be born in the future).
- Patients can have only one phone and email.

3. PatientAllergy(id: INT[PK], PatientId: INT [FK to Patient.PatientId], Allergy: VARCHAR(100))

Constraints:

- A patient can have multiple allergies.

4. Medicine(MedicineId: INT [PK], MedicineName: VARCHAR(100), Dosage: VARCHAR(50), Manufacturer: VARCHAR(100), Price: DECIMAL(10,2))

Constraints:

- Price should be non-negative.

5. MedicineSideEffect(id: INT[PK], MedicineId: INT [FK to Medicine.MedicineId], SideEffect: VARCHAR(255))

Constraints:

- A medicine can have multiple side effects.

6. Hospital(HospitalId: INT [PK], HospitalName: VARCHAR(100), Address: VARCHAR(255), Rating: DECIMAL(3,2), , Phone: VARCHAR(15), Email: VARCHAR(100),)

Constraints:

- Rating should be between 0 and 5.
- Hospitals should have unique phone and email.

7. LabTest(LabTestId: INT [PK], LabTestName: VARCHAR(100), Description: VARCHAR(255), OptimalRange: VARCHAR(50), Price: DECIMAL(10,2))

Constraints:

- Price should be non-negative.

8. PrescriptionDetails(PrescriptionId: INT [PK], AppointmentId: INT [FK to AppointmentDetails.AppointmentId], DoctorId: INT [FK to Doctor.DoctorId], MedicineId: INT [FK to Medicine.MedicineId])

Constraints:

- Each prescription must reference an appointment, a doctor, and a medicine.

9. PrescriptionTimestamp(PrescriptionId: INT [FK to PrescriptionDetails.PrescriptionId], Timestamp: TIMESTAMP)

Constraints:

- Timestamp should be the valid time of prescription.

10. AppointmentDetails(AppointmentId: INT [PK], PatientId: INT [FK to Patient.PatientId], DoctorId: INT [FK to Doctor.DoctorId])

Constraints:

- A valid appointment should reference both a patient and a doctor.

11. AppointmentStatus(AppointmentId: INT [FK to AppointmentDetails.AppointmentId], Timestamp: TIMESTAMP, Status: VARCHAR(50))

Constraints:

- Status can be ["Scheduled", "Completed", "Cancelled"].

12. OrderDetails(OrderId: INT [PK], DoctorId: INT [FK to Doctor.DoctorId], LabTestId: INT [FK to LabTest.LabTestId])

Constraints:

- Each order references a doctor and a lab test.

13. OrderTimestamp(OrderId: INT [FK to OrderDetails.OrderId], AppointmentId: INT [FK to AppointmentDetails.AppointmentId], Timestamp: TIMESTAMP)

Constraints:

- Timestamp should be the time when the order was placed.

14. Transaction(TransactionId: INT [PK], Timestamp: TIMESTAMP, Amount: DECIMAL(10,2), Mode: VARCHAR(50), HospitalId: INT [FK to Hospital.HospitalId], PatientId: INT [FK to Patient.PatientId])

Constraints:

- Mode can be ["Cash", "Credit Card", "Insurance"].
- Amount should be non-negative.

15. LabReport(LabReportId: INT [PK], Timestamp: TIMESTAMP, Result: VARCHAR(255), LabTestId: INT [FK to LabTest.LabTestId], PatientId: INT [FK to Patient.PatientId])

Constraints:

- Each lab report must reference a valid lab test and a patient.

16. AdminLogin(Email: VARCHAR(100) [PK], Password: CHAR(32))

Constraints:

- Email is the primary key

Real-World Conditions Considered:

Gender and Rating Validation: Constraints ensure valid input for gender (restricted to 'M', 'F', 'O') and rating values (0-5 scale).

Appointment Status: Appointment status is tracked to ensure clarity on scheduled, completed, or canceled appointments, which is critical for real-world hospital management.

Transaction Modes: Transactions can only happen through specific modes (Cash, Credit Card, Insurance), making it easier to handle real-world billing scenarios.

Time-Sensitive Data: Timestamps are crucial for tracking the exact time of appointments, prescriptions, orders, and transactions, as they are critical in healthcare systems.

Data Consistency with FKs: Foreign keys ensure that relationships between entities like doctors, patients, appointments, and hospitals are always valid and enforce integrity in the system. For instance, a prescription must always be linked to a valid doctor, patient, and medicine.

Additional datasets:

1. Drugs, Side Effects and Medical Condition
2. US Hospital Locations
3. USA Hospitals
4.