UA Health Services

Enterprise Database Management

Group G – Bits Please

Aditya Singh
Affan Ahmed Kazim
Anmol Sabharwal
Keerthana Jagannatha
Niriksha Dalal
Viraj Singh

Table of Contents

Chapter 1: Requirements analysis document

Summary

Our database and application provide an in-house solution to manage healthcare at the University of Arizona. Over 50,000 students get easy access to good healthcare on-the-go, saving them time and effort. A well-integrated database that relates elements like the pharmacy and ambulance services helps UA health services accomplish its goal of providing quality health services for students. A database approach further simplifies managing the hospital and its operations.

Key users of this application are
1. University of Arizona students
2. Doctors and Administrator of UA Health.

Requirements and Motivation

UA Health is a big part of the University of Arizona. We all rely and depend on them for a healthy living and for medical care. UA Health has gone above and beyond to ensure quality care is provided to students and faculty members working at the University. UA Primary Campus Health is an inhouse application, connecting wildcats to doctors available at Campus Health. Our application is more diverse and offers a more personalized health care experience that cannot be matched by a third-party application.

UA Primary Campus Health is an online health service for students of the University of Arizona. This new initiative by UA Health allows students to reach a doctor for immediate primary care when visiting the hospital physically is a challenge. All students registered with the University of Arizona get free access to the online application. The application allows patients to chat with doctors and specialists and get medical prescriptions. Since some medical conditions are difficult to comprehend virtually, doctors and patients can request an in-person appointment after their discussion. The doctor may recommend the patient to visit a Lab to get medical tests done. UA Primary Campus Health is connected to the pharmacies on campus.

When the students first sign up, they are registered as Patients with a unique PatientID, a login and password details. After signing in, a patient-doctor chat session is initiated where students can discuss their ailments using text messages or pictures. Patient data is internal to UA Primary Campus Health, it is maintained by the department independent of the student data maintained by the University. The University student database contains students phone number, first and last name, email address, blood group and an address (zip, street and building number). Each student is uniquely identified by their studentID. Additional patient information is inherited from the UAccess Database of the student after registration.

Each time a doctor examines a patient, it is recorded as a unique interaction (referred to as a 'case') in a Case Details register. Each case has a unique caseID, a status attribute to check if the case is still open or closed. Case Details also contain the symptoms that the patient has reported.

All symptoms have a name and type, they are also uniquely identifiable. Doctors record their diagnosis with their ICD codes (International Statistical Classification of Diseases), description of diagnosis and ICD code version. The doctor records the symptoms of the patient, the symptoms contains a symptomID which uniquely defines the symptom name and type. The severity of the ailment is reported in the case details, which is useful when the doctor needs to escalate the case and call an ambulance. Each case has diagnosis details. Each diagnosis can have notes and comments further describing the case. The patient can book an appointment for an in-person examination. Appointments are tracked with a unique ID are made for a given time, for a fixed duration of 15 minutes or 30 minutes. There can be different types of appointments based on the type of consultation. A phone number is provided for confirming or rescheduling the appointment. A patient may never book an appointment or book an appointment more than once.

The case details are used to generate billing details which are directly sent to the specific insurance company for claims. Each bill has an ID, total cost for examination, any additional charges incurred, the date the bill was generated and a status to check if the bill has been cleared. UA Primary Campus Health stores information required to reach out to the insurance companies such as the company name, company id, address, insurance provider license number, email address and phone number. Each student is insured by one insurance company.

After examining the patient, the doctor may prescribe a prescription. Each prescription contains the date and prescriptionID. Prescriptions may contain all tests which are performed in labs. All the lab tests are stored in a common location
The doctor may prescribe lab tests as part of the prescription. Each prescription is unique and has a date. The type of prescription is also recorded for the purpose of regulation. If the prescription is for drugs, it contains the dosage of the medicine. If the prescription is for lab tests, the test name, test fee and test ID is recorded.

If the doctor feels the need for further diagnosis, a prescription for lab tests can be written. Each lab is differentiated by their lab id, tests available, phone number and address (building number, street and zip). Multiple tests can be performed at a single lab. Once the tests are performed, each lab generates medical reports for the patient. One report is generated per patient. A medical report has an ID, report name and date.

Pharmacies on campus have a unique ID, Address (with ZIP, Building_No and Street) and phone number. Each medicine stored in the pharmacy has a unique ID, commercial product name, brand, expiry date and type - additional information used to classify if the medicine is OTC, non-drug. A count of each medicine is also stored to check for availability. We know a medicine may be comprised of many drugs, the composition of each is tracked in ‘drugs’. The UA Pharmacy offers promotions and student discounts throughout the year, each discount has a proteinID, discount percentage and start date and end date during which the offer is valid. As a convenience option, drugs may be requested from the pharmacy. We store the frequency and composition of drugs that are prescribed by the doctor.

They are currently in the process of setting up a new student group who are employed as campus delivery associates. These associates will pick up medicines from the pharmacy and deliver them to the patients. These associates are hired on a need basis, i.e. they are hired only when a delivery has to be made. At a time there can be no associates or multiple associates at each pharmacy. Student associates are identified with a name, ID, their shift timings (start time and end time) and driving licence information. A student organization is in charge of employing and managing the delivery associates. The organization keeps a count of the number of students it currently employs. At any given time, the organization will have at least five associates. The organization has a phone number, email and office address for contact.

UA Health is supported by employees which comprises of doctors, pathologists, nurses, ward boys, pharmacists and drivers. Employees have an employee ID, name (first, middle and last), date of hire, date of birth, age, gender, SSN, phone number, email address and shift timings. Apart from this information, UA Health also tracks addition information for each employee category. Doctors have their legal registration number, highest degree earned and on call status.

Chapter 2: ER Diagram

We have created Entity-Relationship model by analysing the requirements for an on-the-go health service application. We have carefully designed the entities and relationships based on real world scenario.
Please find the ER attached below as an object. We'll also be adding it separately in the Dropbox.

UA_Onthego_ER_Diagram.vsdx

Data Dictionary

Schema Construct	Construct Description	Other Description
ACTUAL LAB TESTS	An entity to store the lab tests that were prescribed and actually perform.	
• actuallabtestID	To identify the lab tests that were performed	Identifying Attribute
• result	To store the results of the tests	
AMBULANCES	It is an aggregate entity	
• ambulanceID	A particular ID issued to each ambulance	Identifying Attribute
• vehiclenumber	Number plate of the vehicle	
• availability	To check if the particular ambulance is available	
APPOINTMENTS	An entity class to store appointment information	
• appointmentID	Appointment identification	Identifying Attribute
• starttime	Time for which appointment was booked	
• endtime	Time, if the appointment was rescheduled	

• type	Type of the appointment	
• appdate	Date for which the appointment is booked.	
Appoints	A relationship that models Pharmacy appoints a delivery Associate for Delivery	
BILLING DETAILS	An entity class to store billing details	
• totalcost	Cost of treatment	
• billID	Bill identification	Identifying Attribute
• additionalcharges	If any additional charges other than usual	
• billdate	Date on which bill was generated	
• billstatus	Status of the bill i.e if paid or not	
Books	A relationship that models Patient books an appointment	
Carry out	It is the constrained relationship	
Carried		
CASE DETAILS	An entity to store all the details of particular case	
• caseID	To identify case	Identifying Attribute
• status	To check if the case is closed or open	
• severityindex	Severity of the case	
• duration	Duration for which the particular case continued	
• datetime	Date and time when the case was created	
CHAT DETAILS	It is a weak entity to store chat details	
chatID	To identify chat	
chatactive	A Boolean to store if the chat is active or not	

initial_symptoms	To store the initial system that a patient reports	
Chatswith	A relationship that models chat details, patients and prescription	
Consist	A relationship that models the relationship between medicine prescribed and its dosage.	
Consist of	A relationship that models the different personnel who would board the ambulance	
Contains	A relationship that models the symptoms stored in case details	
Creates	A relationship that models the labs who create medical reports	
CREW		
• crewID	It stores the ID of the crew	Identifying Attribute
• crewname	It stores the name of the crew	
DELIVERY ASSOCIATES	An entity to store the information about delivery associate	
• name	Name of delivery associate	
• shifttime	Shift time of the associate	
○ shiftstarttime	Start time of the shift of the delivery associate	
○ shiftendtime	End time of the shift of the delivery associate	
• dlnumber	Driving License number	
• associateID	To identify the delivery associate	Identifying Attribute

DELIVERY ORGANIZATIONS	The organizations that provides delivery associates	
• organizationID	To identify the organization	Identifying Attribute
• organizationname	Name of the organization	
• noofemployees	Total delivery associates employed	
• Address		
◊ buildingno		
◊ street		
◊ zip		
• phoneno		
• emailaddress		Multi-valued attribute
DIAGNOSES	An entity to store the findings and recommendations of doctor	
• ICDcode	Medical codes for various illnesses	Identifying Attribute
• description	Description of the illness	
• version		
DIAGNOSIS DETAILS		
• notes	Captures the notes taken by doctors	
• comments		
• diagnosis_complete	Boolean to store the status of diagnosis	
Done By	A relationship that models the labs conducting tests	
DRUG DETAILS	To store the frequency and composition a particular drug	Weak Entity
frequency	It stores the frequency by which the medicine is supposed to be consumed	
composition	It stores the composition of the drugs in a medicine	

EMPLOYEE RATINGS	An entity to store the ratings for the employees	
• employeeratingid	Employee id to identify it	Identifying Attribute
• remarks	Description of the rating given	
• ratingdate		
EMPLOYEES	An entity to store the information of employees	
• employeeID	To identify the employee	Identifying Attribute
• name		Composite attribute
◊ firstname		
◊ middleinitial		
◊ lastname		
• hiredate	Date of hiring	
• dateofbirth		
• age		
• gender		
• ssn		
• phonenumber		
• emailaddresses		Multi valued Attribute
• shift time	Shift timing of the employee	
◊ starttime	Login time	
◊ endtime	Logout time	
• loyaltypoints	Points earned through the loyalty program run by the hospital	
• type	To classify various types of employees	
○ **DOCTORS**	A subclass of employee to store information about Doctors	
▪ **GENERAL PHYSICIANS**	A subclass of doctors to store information about physicians	
• certification	Certification name	
• certificationexpirydate		
• istrainee	Whether the physician is trainee or not	

▪ **SPECIALISTS**	Doctors from different departments	
• ispermanent	A visiting doctor or permanent	
• registrationnumber		
• highestdegree		
• username		
• password		
• oncall	Available on call or not	
○ **PATHOLOGISTS**	A subclass of employee to store information about Pathologists	
• pathologistregistrationno		
• certification		
○ **NURSES**	A subclass of employee to store information about nurses	
• nursinglicenseno		
• hourlybillingrate		
• type		
○ **EMT**	A subclass of employee to store information about Wardboys	
• level		
○ **PHARMACIST**	A subclass of employee to store information about pharmacists	
• pharmacistlicenseno		
○ **DRIVER**	A subclass of employee to store information about ambulance drivers	
• dlnumber	Driving license Number	
Employs	A relationship that models delivery organization employing delivery associates	
Examines	A relationship that models the doctor	

	examining the patients	
FEEDBACKS	An entity class to store the feedback received from users of the service	
• id	To identify a feedback	Identifying Attribute
• comments		
• datetime		
Fulfilled by	It models the relationship between the prescription and pharmacists.	
Generated For	A relationship that models the medical reports generated for patients	
Generates	It models the relationship between case details and billing details.	
Givenin	It models the relationship between actual lab tests, medical reports and case details.	
Have	A relationship that models the employees ratings having ratings	
INSURANCE COMPANIES	An entity to store information about various insurance companies available	
• inscompanyID	To identify the company	Identifying Attribute
• address		
• inscomlicenseno	License number of the insurance company	
• inscompanyname	Name of the insurance company	
• emailaddress		
• phonenumber		
Insured By	A relationship that models students being	

	insured by an insurance company	
LABS	An entity to store the information about the laboratories	
• labname		
• labID	To identify the lab	Identifying Attribute
• phoneno		
• address		Composite attribute
◊ zip		
◊ buildingno		
◊ street		
• emailaddress	Names of the test available in a lab	
LAB TESTS	To store information about the tests conducted by labs	
• testID	To identify lab test	Identifying Attribute
• testname	Name of the test	
• fee	Fee for the test	
MEDICAL REPORTS	An entity to store the medical reports created	
• reportID	To identify report	Identifying Attribute
• reportname		
• date		
MEDICINES	To store the information about the medicines	
• drugs		
• count	Number of medicines left in stock	
• medicineID	To identify medicine	Identifying Attribute
• productname		
• brand	To store the name of the brand to which the medicine belongs	
• expirydate		
• type		

Offers	A relationship that models the pharmacy offering promotions	
Onboard	It models the relationship between crew, ambulances and trip details.	
PATIENTS	An entity to store the information about patients	
• patientID	To identify patients	Identifying Attribute
• username		
• password		
PHARMACIES	An entity that stores the information about the pharmacies	
• pharmacyID	To identify pharmacy	Identifying Attribute
• phonenumber		
• emailaddress		
• address		Composite attribute
◊ zip		
◊ buildingnumber		
◊ street		
Prescribes	It models the relationship between doctor and prescription	
PRESCRIPTIONS	To store all the information about prescription	
• prescriptionID	To identify prescription	Identifying Attribute
• date		
• type		
Presentin	It models the relationship between prescription and case details	
PROMOTIONS	An entity to tract the promotions offered by the pharmacies	
• promotionID	To identify a particular offer	Identifying Attribute

• startdate		
• enddate		
• Discount		
Provides	A relationship that models the patients providing feedback for services	
RATINGS	An entity to store the scale used in rating the employee	
• ratingID	To identify a scale	Identifying Attribute
• description		
Receives	A relationship that models the patient receiving the prescription	
Records	A relationship that models the diagnosis recorded in case details	
Register As	A relationship that models the student registering as patient	
Requestedto	A relationship to model the drugs requested from pharmacy	
Sends	A relationship that models the billing details being sent to insurance company	
SPECIALIZATIONS		
• specializationID	To identify the specialization	Identifying Attribute
• description		
• specializationname	Name of the specialization of the doctor	
Specializesin	A relationship that model the specialist who specializes in a particular department	
Stocks	A relationship that models the pharmacy stocking the medicine	

STUDENTS	An entity to store the information about the students	
• address		Composite attribute
◊ dorm		
◊ street		
◊ zip		
◊ buildingnumber		
• studentID	To identify the student	Identifying Attribute
• phonenumber		
• name		
◊ firstname		
◊ middlename		
◊ lastname		
• email		
• bloodgroup	Blood group of the student	
SYMPTOMS	An entity containing a list of symptoms	
• symptomID	To identify a symptom	Identifying Attribute
• name		
• type	To store the type of symptom	
Triggers	It models the relationship between case details and trip details.	
TRIP DETAILS		
• tripID	To identify a particular trip	Identifying Attribute
• timeoftrip	It stores the date and time of the trip	
• address		Composite attribute
◊ zip		
◊ buildingnumber		
◊ street		
Works	A relationship that models a pathologists working in a lab	
Works In	A relationship that models a pharmacist working in a pharmacy	

Assumptions

1. The student delivery associate is a temporary employee, they might or might not be a part of the student delivery organisation.
2. All university of Arizona organizations should have a Unique organisation ID, so student delivery organisation too have an organisation ID
3. Student Delivery Organization should have 5 or more student delivery associates.
4. There is more than 1 pharmacy on campus, all pharmacies are open 24/7.
5. A pharmacy can have 1 or many pharmacists, a pharmacist can be associated to only 1 pharmacy.
6. One promotion should be associated with at least one pharmacy and can be associated with multiple pharmacies.
7. Our patients are only students. Each student who registers as a patient should have an insurance plan, which covers all on the go services.
8. A patient can give multiple feedback, group feedback involving more than 1 patient is not allowed.
9. There are multiple labs on the campus, each lab should do at least one type of test and can do multiple types of tests.
10. Each test is available in one of the labs on campus.
11. Every ambulance should have 1 nurse, 2 ward boys and 1 driver when it is sent out.
12. A case will have at least 1 symptom associated with it and must have 1 or multiple diagnosis.
13. An employee can be in the employee rating table only if they have a rating.
14. Specialists can be full time or visiting, physician can either permanent or trainee. If there is a specialization, our hospital has at least one doctor who specializes in it.
15. Each employee can receive multiple rating, one rating per quarter. The ratings they receive are predefined and saved in the rating table.
16. A lab can have multiple pathologists working with it, a pathologist can work with only 1 lab.
17. We have a list of lab tests available at UA. Each lab supports a set of tests, and every time a test is conducted the details of the performed lab test is updated in Actual Lab test.
18. A patient can book multiple appointments, each appointment is associated with only 1 patient.
19. An interaction between a doctor and a patient creates a Case, there can be multiple cases between a patient and doctor.
20. 1 case must generate at least 1 bill, 1 bill can only be associated with 1 case.
21. 1 bill can be sent to only 1 insurance comp, whereas each of the partner insurance companies might have 0 bills or multiple bills.
22. Every interaction creates a case and chat details is also saved for each case. For each case, doctor prescribes a prescription.
23. A prescription can have no or many lab tests prescribed and have no or many drugs prescribed.
24. A prescription will have the drugs and their dosage, there can be multiple medicines for the same drug. A medicine can have 1 or more drugs.
25. Prescription has drug details, with frequency and composition of the prescribed consumption.
26. A prescription is fulfilled by a pharmacist from one of the pharmacies and is delivered by a delivery person

27. A medical report is generated for a case. A report contains case details as well as the lab results.

Chapter 3: Normalized relational schema

We have normalized the tables to 4NF, for all our entities and relationships as per our business requirements.

1. ACTUAL_LAB_TESTS (actuallabtestID, result, labtestID, labID)
   FORIEGN KEY labtestID REFERENCES LAB_TESTS
   FORIEGN KEY (labID, labtestID) REFERENCES LABTESTS_DONEBY

2. AMBULANCES (ambulanceID, vehiclenumber, availability)

3. APPOINTMENT_TIME_SLOTS (slotID, startTime, endTime)

4. APPOINTMENTS (appointmentID,type, starttime, endtime, appdate, patientid, doctorID)
   FORIEGN KEY patientID REFERENCES PATIENTS (patientID)
   FORIEGN KEY doctorID REFERENCES DOCTORS (doctorID)

5. BILLING_DETAILS (billID, totalcost, additionalcharges, billdate, billstatus, inscompanyID, caseID)

   FOREIGN KEY inscompanyID REFERENCES INSURANCE_COMPANIES (inscompanyID)
   FOREIGN KEY caseID REFERENCES CASE_DETAILS (caseID)

6. CASE_DETAILS (caseID, status, datetime, severity, duration_min, doctorID,patientID)
   FOREIGN KEY doctorID REFERENCES DOCTORS (doctorID)
   FOREIGN KEY patientID REFERENCES PATIENTS (patientID)

7. CASE_REPORT_LAB (reportID, actuallabtestID, caseID)
   FOREIGN KEY reportID REFERENCES MEDICALREPORTS (reportID)
   FOREIGN KEY actuallabtestID REFERENCES ACTUAL_LAB_TESTS (actuallabtestID)
   FOREIGN KEY caseID REFERENCES CASE_DETAILS (caseID)

8. CASE_SYMPTOMS (caseID, symptomID)
   FOREIGN KEY caseID REFERENCES CASE_DETAILS (caseID)
   FOREIGN KEY symptomID REFERENCES SYMPTOMS (symptomID)

9. CHAT_DETAILS (chatID, chatActive, initial_symptoms, docID, pID)
   FOREIGN KEY docID REFERENCES DOCTORS (doctorID)
   FOREIGN KEY pID REFERENCES PATIENTS (patientID)

10. CREW_DRIVERS (crewID, driverID)
    FOREIGN KEY crewID REFERENCES CREWS (crewID)
    FOREIGN KEY driverID REFERENCES DRIVERS (driverID)

11. CREW_EMT (crewID, EMTID)
    FOREIGN KEY crewID REFERENCES CREWS (crewID)
    FOREIGN KEY emtID REFERENCES EMT (emtID)

12. CREW_NURSES (crewID, nurseID)
    FOREIGN KEY crewID REFERENCES CREWS (crewID)
    FOREIGN KEY nurseID REFERENCES NURSES (nurseID)

13. CREWS (crewID, crewname)

14. DELIVERY_ASSOCIATES (delassociateID, delassociatename, dlnumber, shiftstarttime, shiftendtime, delorgID)
    FOREIGN KEY delorgID REFERENCES DELIVERY_ORGANIZATIONS (delorgID)

15. DELIVERY_ORGANIZATIONS (delorgID, organizationname, noofemployees, buildingno, street, zip, phonenumber, emailaddress)
    Check Constraint (phoneno not like '%[^0-9]%')

16. DIAGNOSES (ICDCode, description, version)

17. DIAGNOSIS_DETAILS (caseID, ICDCode, notes, comments, diagnosis_complete)
    FOREIGN KEY caseID REFERENCES CASE_DETAILS (caseID)
    FOREIGN KEY ICDCode REFERENCES DIAGNOSES (ICDCode)
    Check Constraint (diagnosis_complete IN('Yes','No'))

18. DOCTORS (doctorID, registrationnumber, highestdegree, oncall)
    FOREIGN KEY doctorID REFERENCES EMPLOYEES (employeeID)

19. DRIVERS (driverID, dlno)
    FOREIGN KEY driverID REFERENCES EMPLOYEES (employeeID)

20. DRUG_DETAILS (prescriptionID, medicineID, frequency, composition)
    FOREIGN KEY medicineID REFERENCES MEDICINES
    FOREIGN KEY prescriptionID REFERENCES PRESCRIPTIONS (prescriptionID)

21. EMPLOYEE_RATINGS (employeeratingID, remarks, ratingdate, ratingID)
    FOREIGN KEY ratingID REFERENCES RATINGS (ratingID)

22. EMPLOYEE_RATINGS_RECIEVED (employeeID, employeeratingID)
    FOREIGN KEY employeeID REFERENCES EMPLOYEES (employeeID)
    FOREIGN KEY employeeratingID REFERENCES EMPLOYEE_RATINGS (employeeratingID)

23. EMPLOYEES (employeeID, firstname, middleinitial, lastname, hiredate, dateofbirth,bage, gender, SSN, phonenumber,loyaltypoints, emailaddress,shiftstarttime, shiftendtime, type)
    UNIQUE CONSTRAINT(SSN)

24. EMT (emtID, levels)
    FOREIGN KEY emtID REFERENCES EMPLOYEES (employeeID)

25. FEEDBACKS (feedbackID, comments, datetime, patientID)

FOREIGN KEY patientID REFERENCES PATIENTS (patientID)

26. GENERAL_PHYSICIANS (generalphysicianID, istrainee, certification, certexpdate, type)
    FOREIGN KEY generalphysicianID REFERENCES DOCTORS (doctorID)
    Check Constraint (isTrainee='YES' OR isTrainee='NO')

27. INSURANCE_COMPANIES (inscompanyID, inscompanyname, inscomlicenseno, address, email, phoneno)
    Check Constraint (phoneno not like '%[^0-9]%')

28. LABTESTS_DONEBY (labID, labtestID)
    FORIEGN KEY labID REFERENCES LABS (labID)
    FORIEGN KEY labtestID REFERENCES LAB_TESTS (labtestID)

29. LAB_TESTS (testID, testname,fees$)

30. LABS (labID, labname, phonenumber, zip, street, buildingno)
    Check Constraint (phoneno not like '%[^0-9]%')

31. MEDICALREPORTS (reportID, repDate, reportName, patientID, actualLabTestID)
    FORIEGN KEY patientID REFERENCES PATIENTS (patientID)
    FORIEGN KEY actualLabTestID REFERENCES ACTUAL_LAB_TESTS (actualLabTestID)

32. MEDICINE_DRUGS (medicineID, drugs)
    FOREIGN KEY medicineID REFERENCES MEDICINES (medicineID)

33. MEDICINES (medicineID, productname, brand, expirydate, type, count)

34. NURSES (nurseID, nursinglicenseno, type, hourlybillingrate)
    FOREIGN KEY nurseID REFERENCES EMPLOYEES (employeeID)

35. PATHOLOGISTS (pathologistID, certification, labID)
    FOREIGN KEY pathologistID REFERENCES EMPLOYEES (employeeID)
    FOREIGN KEY labID REFERENCES LABS (labID)

36. PATIENTS (patientID, username, password, studentID)
    FOREIGN KEY studentID REFERENCES STUDENTS (studentID)

37. PATIENTS_PRESCRIPTIONS (patientID, prescriptionID)
    FOREIGN KEY patientID REFERENCES PATIENTS (patientID)
    FOREIGN KEY prescriptionID REFERENCES PRESCRIPTIONS (prescriptionID)

38. PHAR_DELASSOC (delassociateID, pharmacyID)
    FOREIGN KEY delassociateID REFERENCES DELIVERY_ASSOCIATES (delassociateID)

39. PHAR_PROMO (pharmacyID, promotionID)
    FOREIGN KEY pharmacyID REFERENCES PHARMACIES (pharmacyID)
    FOREIGN KEY promotionID REFERENCES PROMOTIONS (promotionID)

40. PHARM_MEDICINES (medicineID, pharmacyID)
    FOREIGN KEY medicineID REFERENCES MEDICINES (medicineID)
    FOREIGN KEY pharmacyID REFERENCES PROMOTIONS (pharmacyID)

41. PHARM_PHARMACIST (pharmacistID, pharmacyID)
    FOREIGN KEY pharmacyID REFERENCES PHARMACIES (pharmacyID)
    FOREIGN KEY pharmacistID REFERENCES PHARMACISTS (pharmacistID)

42. PHARMACIES (pharmacyID, pharmacyName, zip, buildingno, street, phoneno, email)
    Check constraint on Phone Number, only numbers allowed

43. PHARMACISTS (pharmacistID, pharmacistlicenseno)
    FOREIGN KEY pharmacistID REFERENCES EMPLOYEES (employeeID)

44. PRESCRIPTIONS (prescriptionID, prescriptiondate, pharmacistID, caseID, doctorID, pharmacyID)
    FOREIGN KEY pharmacistID REFERENCES PHARMACISTS (pharmacistID)
    FOREIGN KEY caseID REFERENCES CASE_DETAILS (caseID)
    FOREIGN KEY doctorID REFERENCES DOCTORS (employeeID)
    FOREIGN KEY pharmacyID REFERENCES PHARMACIES (pharmacyID)

45. PRESCRIPTIONS_LABTESTS (testID, prescriptionID)
    FOREIGN KEY prescriptionID REFERENCES PRESCRIPTIONS (prescriptionID)
    FOREIGN KEY testID REFERENCES LAB_TESTS (testID)

46. PROMOTIONS (promotionID, discount, startdate, enddate)

47. RATINGS (ratingID, description)

48. SPECIALISTS (specialistID, specializationID, ispermanent)
    FOREIGN KEY specialistID REFERENCES DOCTORS (doctorID)
    Check Constraint (ISPERMANENT='1' OR ISPERMANENT='0')

49. SPECIALIZATIONS (specializationID, specializationName, description)
    FOREIGN KEY specializationID REFERENCES SPECIALIZATIONS (specializationID)

50. STUDENTS (studentID, firstname, lastname, bloodgroup, emailaddress, phonenumber, zip, street, buildingno, gender, inscompanyID, dateofbirth, age, city, state)
    FOREIGN KEY companyID REFERENCES INSURANCE_COMPANIES (companyID)
    Check Constraint (phoneno not like '%[^0-9]%')

51. SYMPTOMS (symptomID, name, type)

52. TRIP_DETAILS (tripID, timeoftrip, street, zip, buildingno, crewID, ambulanceID)
    FOREIGN KEY crewID REFERENCES CREW (crewID)
    FOREIGN KEY ambulanceID REFERENCES AMBULANCES (ambulanceID)

Chapter 4: Queries

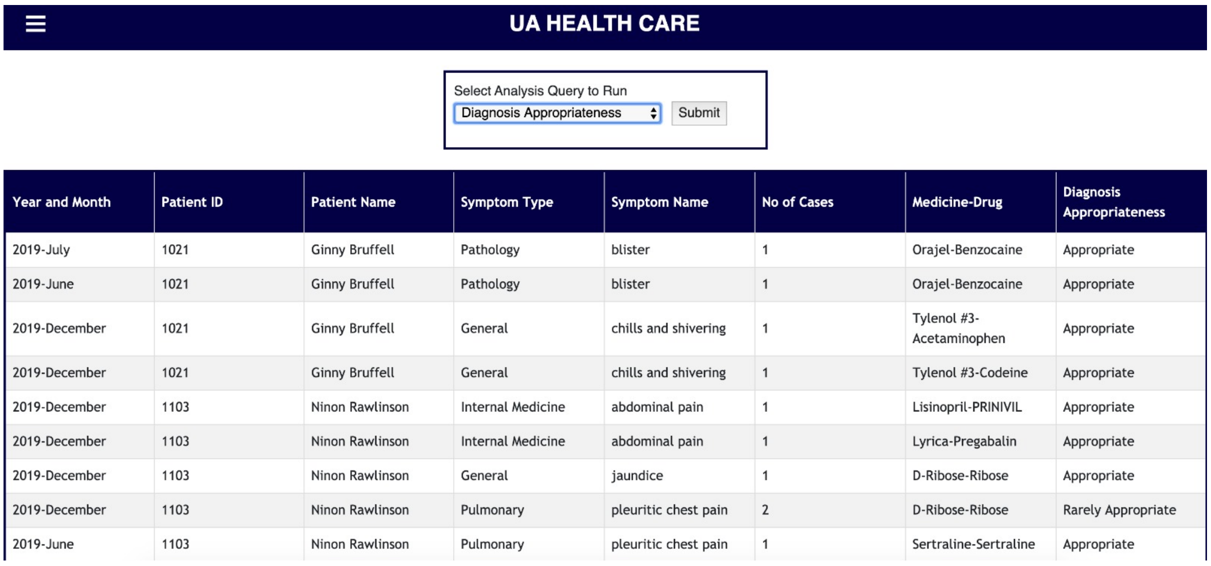
Complex Queries

Query 1 – Diagnosis Appropriateness

Calculate the appropriateness of the diagnosis doctor gives the patient.
For a Year and Month Combination (For Example: 2019-Decemeber) for a patient, show the patient ID, symptom type, symptom name and number of cases patient has registered during the given month and year and the Medicine-Drug combination that was prescribed for that patient.
Based on the No of cases registered for a given patient, symptom, medicine-drug combination, If the No of cases is equal to 2, the Diagnosis Appropriateness is "Rarely Appropriate", If the No of cases are greater than or equal to 3, the Diagnosis Appropriateness is "Not Appropriate", else it is "Appropriate".

```
WITH s AS
(SELECT * FROM (SELECT p.patientid pid, CONCAT(CONCAT(s.firstname,' '),s.lastname)
pname,s.type stype,s.name sname, COUNT(c.caseid) noofcase,
EXTRACT (year from   c.datetime) yd, to_char(c.datetime,'Month') md,
CONCAT(CONCAT(m.productname,'-'),md.drugs) medicine
FROM case_details c
   JOIN patients p ON p.patientid = c.patientid
   JOIN case_symptoms cs ON cs.caseid = c.caseid
   JOIN symptoms s ON s.symptomid = cs.symptomid
   JOIN prescriptions pr ON c.caseid = pr.caseid
   JOIN drug_details dd ON dd.prescriptiionid = pr.prescriptionid
   JOIN medicines m ON m.medicineid = dd.medicineid
   JOIN medicine_drugs md ON md.medicineid = m.medicineid
   JOIN students s ON s.studentid = p.studentid
WHERE (SELECT sysdate FROM dual) - (TO_DATE(to_char(c.datetime,'dd-MON-yyyy')))
<=365
AND s.type NOT IN ('Psychiatric')
GROUP BY p.patientid,s.name,s.type, extract(year from c.datetime),
to_char(c.datetime,'Month'), m.productname, md.drugs, s.firstname,  s.lastname
ORDER BY p.patientid))
SELECT CONCAT (CONCAT (yd,'-'), md) as "Year and Month",pid AS "Patient ID",
pname AS "Patient Name",   stype AS "Symptom Type",sname AS "Symptom Name",
noofcase AS "No of Cases",medicine AS "Medicine-Drug",
(CASE WHEN noofcase = 2 THEN 'Rarely Appropriate'
WHEN noofcase >= 3 THEN 'Not Appropriate'
ELSE 'Appropriate'
END) AS "Diagnosis Appropriateness"
FROM s;
```

Output:



Year and Month	Patient ID	Patient Name	Symptom Type	Symptom Name	No of Cases	Medicine-Drug	Diagnosis Appropriateness
2019-July	1021	Ginny Bruffell	Pathology	blister	1	Orajel-Benzocaine	Appropriate
2019-June	1021	Ginny Bruffell	Pathology	blister	1	Orajel-Benzocaine	Appropriate
2019-December	1021	Ginny Bruffell	General	chills and shivering	1	Tylenol #3-Acetaminophen	Appropriate
2019-December	1021	Ginny Bruffell	General	chills and shivering	1	Tylenol #3-Codeine	Appropriate
2019-December	1103	Ninon Rawlinson	Internal Medicine	abdominal pain	1	Lisinopril-PRINIVIL	Appropriate
2019-December	1103	Ninon Rawlinson	Internal Medicine	abdominal pain	1	Lyrica-Pregabalin	Appropriate
2019-December	1103	Ninon Rawlinson	General	jaundice	1	D-Ribose-Ribose	Appropriate
2019-December	1103	Ninon Rawlinson	Pulmonary	pleuritic chest pain	2	D-Ribose-Ribose	Rarely Appropriate
2019-June	1103	Ninon Rawlinson	Pulmonary	pleuritic chest pain	1	Sertraline-Sertraline	Appropriate

Figure 1: Diagnosis Appropriateness Output

Query 2 – Case Count Analysis

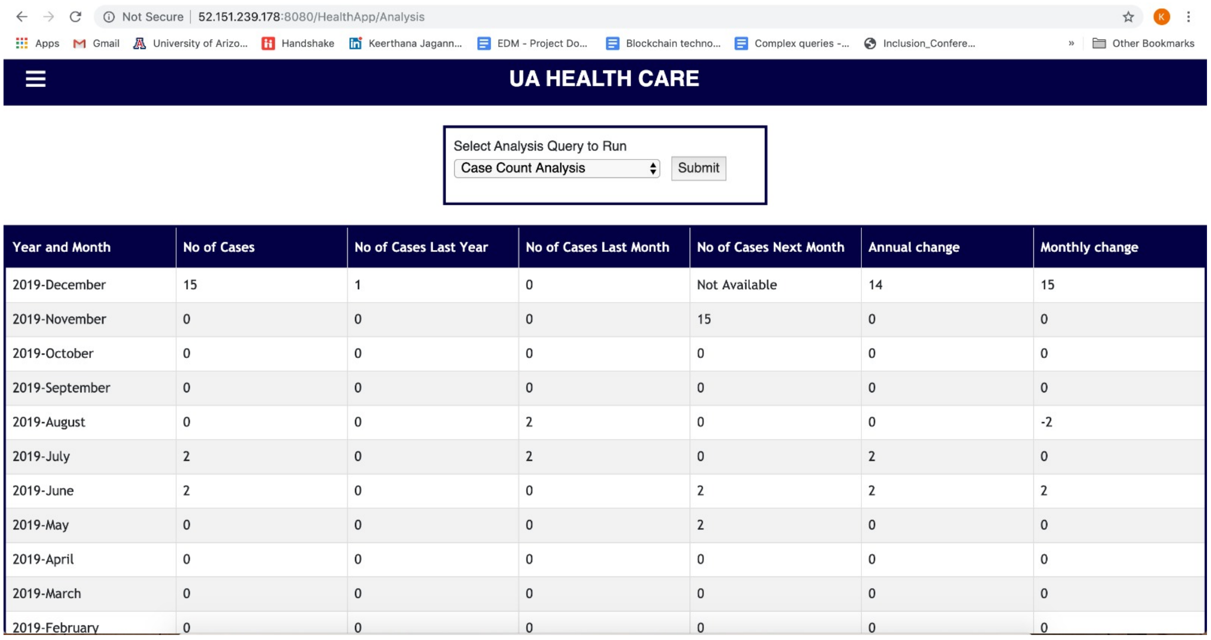
Display for the past 24 months the data about No of cases doctors have addressed in "current" month, number of cases from a month ago, number of cases in the same month a year, annual change in No of Cases and monthly change in No of cases.

-The "current" year and month (e.g., 2019-October 2019-September, etc.)
-The number of cases in that month
-The number of cases from a month ago (e.g., if we're in November 2019, we should get October 2019 case data). If we have no data for a month ago, show: Not Available
-The number of cases in the same month a year ago (e.g., if we're in December 2019, we should get December 2018 checkouts). If we have no data, show: N/A
-The number of cases in the following month
-The annual change in No of Cases, i.e., this month's No of cases – checkouts from a year ago, assuming data is available. If there is no data, show Not Available.
-The monthly change in No of cases (i.e., this month's No of cases – No of cases from a month ago) If there is no data or if there are null values, display 'Not available'

b) Sort the results by month so the latest month is on top

```
WITH mind as (
        SELECT add_months(sysdate, -24) as mindate
        FROMdual
),
listmonths (lmonth) as (
         SELECT mindate as lmonth
         FROM mind
         UNION ALL
         SELECT add_months(lmonth,1)
         FROM listmonths
         WHERE add_months(lmonth,1) <= sysdate+1
),
monyear as (
        SELECT extract (year from lmonth) as lyear, extract (month from lmonth) as lmonth
        FROM listmonths
),
borr as (
        SELECT c.caseid as Cases,extract(year from (cast(c.datetime as date))) as iyear,
      extract(month from (cast(c.datetime as date))) as imonth
        FROM case_details c
        JOIN doctors d ON c.doctorid = d.doctorid
),
monborr as (
        SELECT iyear, imonth, count (*) as NoofCases
        FROM borr
        GROUP BY iyear, imonth
),
rawstats as (
        SELECT lyear, lmonth, coalesce (NoofCases,0) ctissue
        FROM monyear my
        LEFT OUTER JOIN monborr mb on my.lyear = mb.iyear and my.lmonth =
      mb.imonth
),
monstats as (
        SELECT lyear, lmonth, ctissue,
        LEAD (ctissue,1) OVER (ORDER by lyear,lmonth) AS nmc,
        LAG (ctissue,1) OVER (ORDER by lyear,lmonth) AS lmc,
        LAG (ctissue,12) OVER (ORDER by lyear,lmonth) AS lyc
        FROM rawstats
)
SELECT lyear ||'-'|| to_char(to_date(lmonth,'MM'),'Month') as "Year and Month",
ctissue AS "No of Cases",
  coalesce(to_char(lyc),'Not available') as "No of Cases Last Year",
  coalesce(to_char(lmc),'Not available') as "No of Cases Last Month",
  coalesce(to_char(nmc),'Not available') as "No of Cases Next Month",
  coalesce(to_char(ctissue-lyc),'N/A') as "Annual change",
  coalesce(to_char(ctissue-lmc),'N/A') as "Monthly change"
FROM monstats
ORDER BY lyear desc, lmonth desc
FETCH FIRST 24 rows only;
```

Output:



Year and Month	No of Cases	No of Cases Last Year	No of Cases Last Month	No of Cases Next Month	Annual change	Monthly change
2019-December	15	1	0	Not Available	14	15
2019-November	0	0	0	15	0	0
2019-October	0	0	0	0	0	0
2019-September	0	0	0	0	0	0
2019-August	0	0	2	0	0	-2
2019-July	2	0	2	0	2	0
2019-June	2	0	0	2	2	2
2019-May	0	0	0	2	0	0
2019-April	0	0	0	0	0	0
2019-March	0	0	0	0	0	0
2019-February	0	0	0	0	0	0

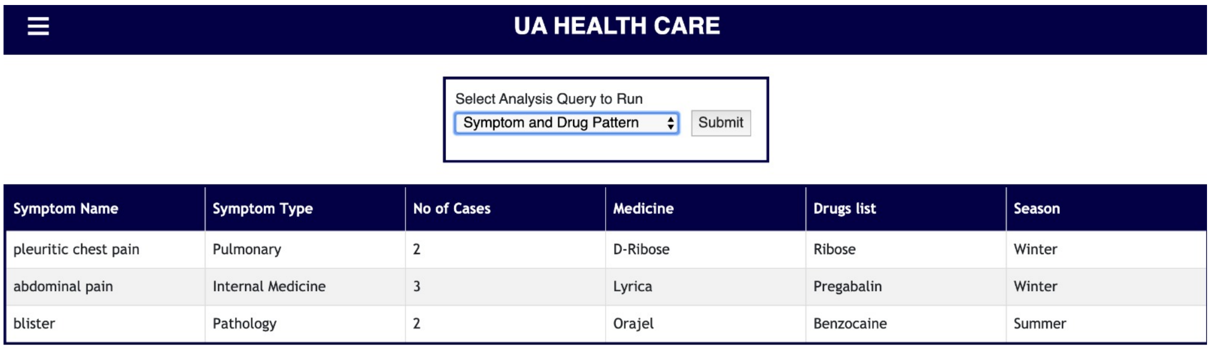
Figure 2: Case Count Analysis Output

Query 3 - Seasonal Symptom and Drug Pattern

A) For a given Symptom and Symptom type, calculate the number of cases registered, and the medicine and drugs prescribed for the respective symptom type. Based on the number of cases during a given month,
If the cases registered are in the month of March - May, then display Season as “Spring”,
If the cases registered are in the month of Jun – Aug, then display Season as “Summer”,
If the cases registered are in the month of Sep– Nov, then display Season as “Autumn”,
Else display the season as “Winter”
B) Display number of cases to >=2

```
WITH casenum AS
(SELECT * FROM (SELECT s.name sname, s.type stype, count(c.caseid) NoofCases,
extract (month from c.datetime) monthvalue, m.medicineid medicineid,m.productname
medicines, md.drugs drugnames
FROM symptoms s
   JOIN case_symptoms cs ON cs.symptomid = s.symptomid
   JOIN case_details c ON c.caseid = cs.caseid
   JOIN prescriptions p ON c.caseid = p.caseid
   JOIN drug_details dd ON p.prescriptionid = dd.prescriptiionid
   JOIN medicines m ON m.medicineid = dd.medicineid
   JOIN medicine_drugs md ON m.medicineid = md.medicineid
WHERE s.type NOT IN ('Psychiatric')
GROUP BY s.name, s.type, extract(month from c.datetime), m.productname, m.medicineid,
md.drugs)),
lagg AS (select * from (select distinct(md.medicineid) medid, listagg(drugs, '; ') within group
(order by drugs) over (partition by md.medicineid) as tlist
FROM medicine_drugs md))

SELECT d.sname as "Symptom Name", d.stype as "Symptom Type", d.NoofCases as "No of
Cases", d.medicines as "Medicine", l.tlist as "Drugs list",
(CASE
    WHEN d.monthvalue IN (3,4,5) THEN 'Spring'
    WHEN d.monthvalue IN (6,7,8) THEN 'Summer'
    WHEN d.monthvalue IN (9,10,11) THEN 'Autumn'
    ELSE 'Winter'
END) as "Season"
FROM lagg l
        JOIN casenum d ON l.medid = d.medicineid
WHERE d.NoofCases >= 2;
```

Output:



Symptom Name	Symptom Type	No of Cases	Medicine	Drugs list	Season
pleuritic chest pain	Pulmonary	2	D-Ribose	Ribose	Winter
abdominal pain	Internal Medicine	3	Lyrica	Pregabalin	Winter
blister	Pathology	2	Orajel	Benzocaine	Summer

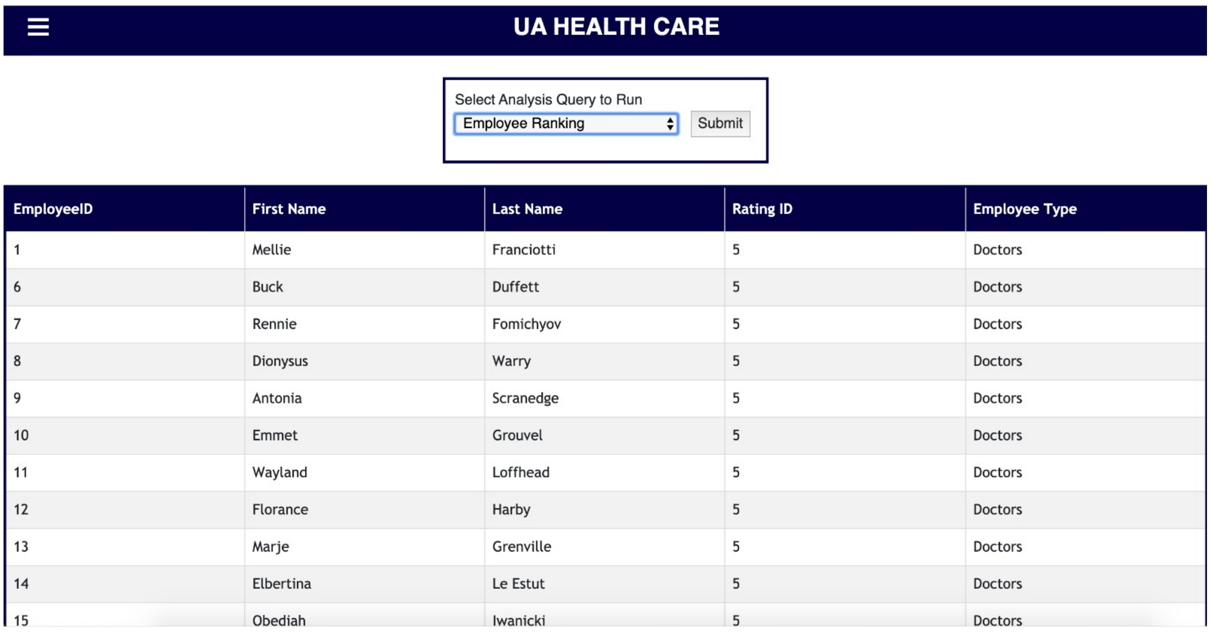
Figure 3: Seasonal Symptom and Drug Pattern Output

Query 4 - Best performing employees in each employee category

The employees receive a rating at the end of each quarter. The administrator wants to know the best performing employees working at the hospital for each quarter.
Write a query to rank the employees based on their employee ratings.
The administrator is concerned with rankings for the latest quarter only.
They employees are ranked based on their employee type - i.e doctors, nurses etc.
Display all the employees in each group who have received a rank of 1.

```
WITH latest_date AS (
   SELECT err.employeeid, max(er.ratingdate) as maxdate
   FROM employee_ratings er JOIN employee_ratings_received err
      ON er.employeeratingid = err.employeeratingid
   GROUP BY err.employeeid
),
rankingByGroup AS (
SELECT e.employeeid, e.firstname, e.lastname, er.ratingid, e.type,
DENSE_RANK () OVER (PARTITION BY e.type
ORDER BY er.ratingid desc) as DenseRank
FROM employee_ratings_received err
   JOIN employee_ratings er ON  er.employeeratingid = err.employeeratingid
   JOIN employees e ON e.employeeid = err.employeeid
   JOIN latest_date ON latest_date.employeeid = e.employeeid
   WHERE er.ratingdate = latest_date.maxdate
)
SELECT *
FROM rankingByGroup
WHERE DenseRank = 1;
```

Output:



EmployeeID	First Name	Last Name	Rating ID	Employee Type
1	Mellie	Franciotti	5	Doctors
6	Buck	Duffett	5	Doctors
7	Rennie	Fomichyov	5	Doctors
8	Dionysus	Warry	5	Doctors
9	Antonia	Scranedge	5	Doctors
10	Emmet	Grouvel	5	Doctors
11	Wayland	Loffhead	5	Doctors
12	Florance	Harby	5	Doctors
13	Marje	Grenville	5	Doctors
14	Elbertina	Le Estut	5	Doctors
15	Obediah	Iwanicki	5	Doctors

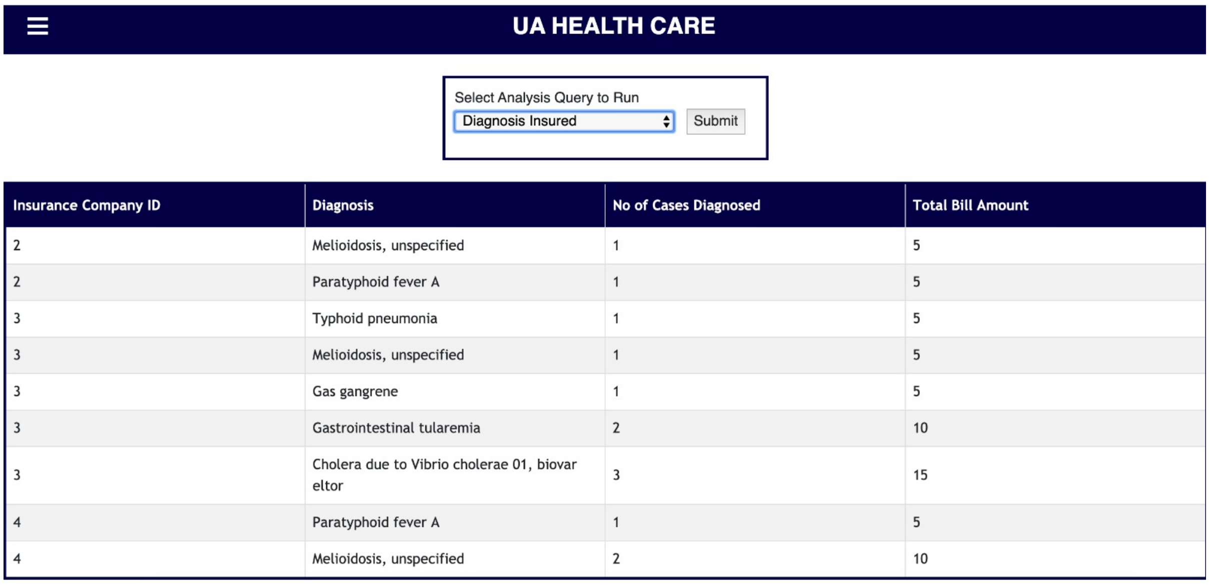
Figure 4: Best performing employees' output

Query 5 – Diagnosis Insured

For Every Patient Insured who has a case registered, display the Insurance Company ID, Diagnosis for which a patient has been claimed insurance,
No of cases registered for that particular diagnosis and the total bill amount

```
WITH t AS
  (SELECT * FROM (SELECT icdcode, b.inscompanyid icompanyid, sum ( coalesce(b.total_cost,0) + coalesce(b.additionalcharges,0)) billamount,count(icdcode) nooficdcode from billing_details b
      JOIN diagnosis_details dd on dd.caseid = b.caseid
      JOIN insurance_companies ic on ic.inscompanyid = b.inscompanyid
   GROUP BY icdcode,b.inscompanyid
))
SELECT t.icompanyid "Insurance Company ID",d.description as "Diagnosis",t.nooficdcode "No of Cases Diagnosed",t.billamount "Total Bill Amount"
FROM t
JOIN diagnoses d on d.icdcode = t.icdcode
ORDER BY t.icompanyid,t.nooficdcode;
```

Output:



Insurance Company ID	Diagnosis	No of Cases Diagnosed	Total Bill Amount
2	Melioidosis, unspecified	1	5
2	Paratyphoid fever A	1	5
3	Typhoid pneumonia	1	5
3	Melioidosis, unspecified	1	5
3	Gas gangrene	1	5
3	Gastrointestinal tularemia	2	10
3	Cholera due to Vibrio cholerae 01, biovar eltor	3	15
4	Paratyphoid fever A	1	5
4	Melioidosis, unspecified	2	10

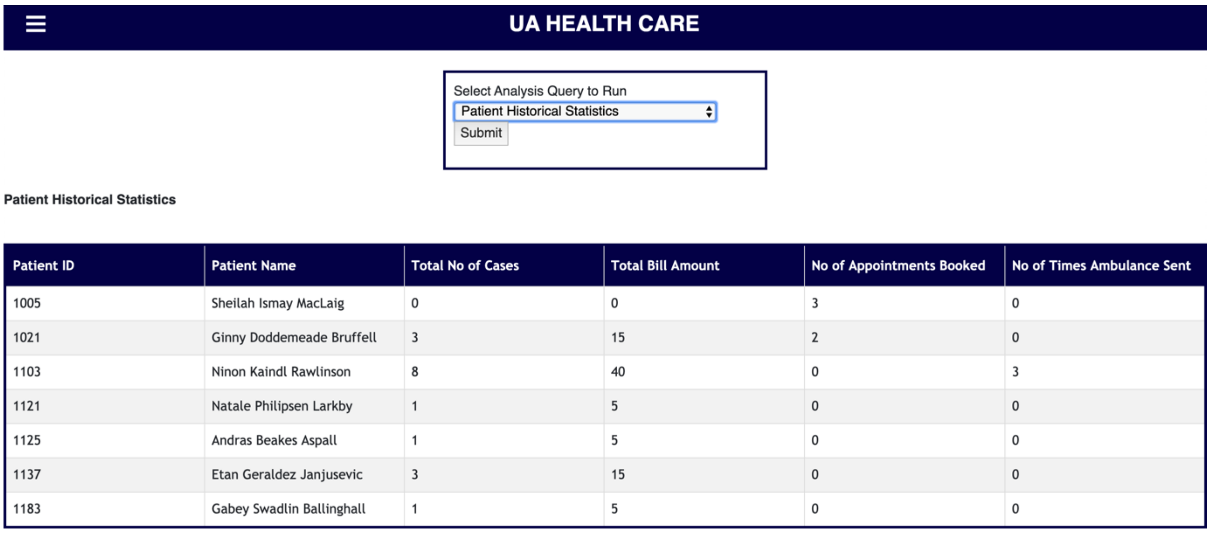
Figure 5: Diagnosis Insured Output

Query 6 – Patient Historical Statistics

Displaying the history of patient's history with the On-the-go services. This includes the total number of cases they have had, Sum of all the bills paid, No of appointments booked through the app and the total no of times an ambulance was sent for the patient. This lists out only those patients who have had at least 1 interaction with the application, i.e. at least 1 case or 1 appointment or 1 instance when an ambulance was sent.
Patient Name is an aggregate of their first, last and middle name.
Total no of cases is a count of number of cases for that patient
Total Bill Amount is a sum of all the bills generated for that customer
No of Appointments booked is a count of all the appointments booked by that patient through the application
No of Times ambulance sent is a count of the total no of times an ambulance is sent for a patient

```
WITH Student_Name AS
(
        SELECT unique(p.patientID) "Patient ID",
        s.Firstname || ' ' || s.middleinitial || ' ' || s.lastname as "Patient Name"
        FROM students S
        JOIN patients p on p.studentid = s.studentid
        LEFT OUTER JOIN case_details cd on p.patientid = cd.patientid
),

Case_count AS
(
        SELECT sum(bd.total_cost) "Total Cost", count(bd.caseid) "No of Cases", cd.patientid
        "Patid"
        FROM billing_details bd
        LEFT OUTER JOIN case_details cd on cd.caseid = bd.caseid
        GROUP BY cd.patientid
),

App_count AS
(
        SELECT count(appointmentid) "Appointment Count",
        patientid
        FROM appointments
        GROUP BY patientid
),

Amb_req AS
(
        SELECT count(td.caseid) "Ambulance Count", cd.patientid "Patid"
        FROM trip_details td
        JOIN case_details cd on cd.caseid = td.caseid
        GROUP BY cd.patientid
)

SELECT "Patient ID", "Patient Name", Coalesce ("No of Cases",0) "Total No of Cases",
Coalesce ("Total Cost",0) "Total Bill Amount",
Coalesce ("Appointment Count",0) "No of Appointments Booked",
Coalesce ("Ambulance Count",0) "No of Times Ambulance Sent"
from student_name
        LEFT OUTER JOIN Case_count ON student_name."Patient ID" = Case_count."Patid"
        LEFT OUTER JOIN App_count ON student_name."Patient ID" =
        App_count.patientid
        LEFT OUTER JOIN Amb_req ON student_name."Patient ID" = Amb_req."Patid"

WHERE Coalesce ("No of Cases",0) <> 0 or Coalesce ("Appointment Count",0) <> 0 or
Coalesce ("Ambulance Count",0) <> 0
ORDER BY "Patient ID";
```

Output:



Patient Historical Statistics

Patient ID	Patient Name	Total No of Cases	Total Bill Amount	No of Appointments Booked	No of Times Ambulance Sent
1005	Sheilah Ismay MacLaig	0	0	3	0
1021	Ginny Doddemeade Bruffell	3	15	2	0
1103	Ninon Kaindl Rawlinson	8	40	0	3
1121	Natale Philipsen Larkby	1	5	0	0
1125	Andras Beakes Aspall	1	5	0	0
1137	Etan Geraldez Janjusevic	3	15	0	0
1183	Gabey Swadlin Ballinghall	1	5	0	0

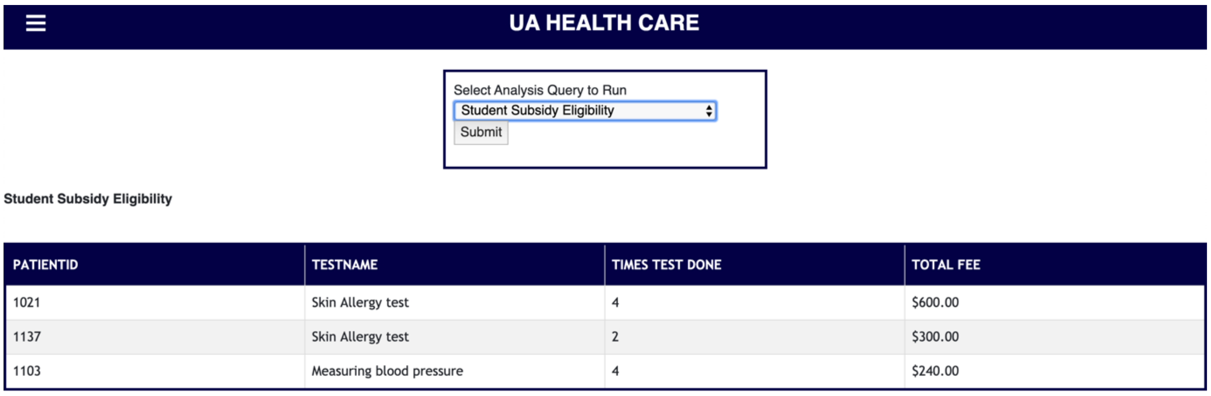
Figure 6: Patient Historical Statistics Output

Query 7 – Student Subsidy Eligibility

Campus health provides lab test fee waivers to students who have spent more than $250 on the same lab test.

```
SELECT mr.patientid, lt.testname, count(*) as "TIMES TEST DONE",
to_char(sum(lt.fees), '$99,999.99') as "TOTAL FEE"
FROM medicalreports mr
   JOIN actual_lab_tests alt ON alt.actuallabtestid = mr.actuallabtestid
   JOIN lab_tests lt ON lt.testID = alt.labtestid
GROUP BY mr.patientid, lt.testid, lt.testname
HAVING SUM(lt.fees) > 250
ORDER BY "TOTAL FEE" DESC
FETCH FIRST 5 ROWS ONLY;
```

Output:



Student Subsidy Eligibility

PATIENTID	TESTNAME	TIMES TEST DONE	TOTAL FEE
1021	Skin Allergy test	4	$600.00
1137	Skin Allergy test	2	$300.00
1103	Measuring blood pressure	4	$240.00

Figure 7: Student Subsidy Eligibility Output

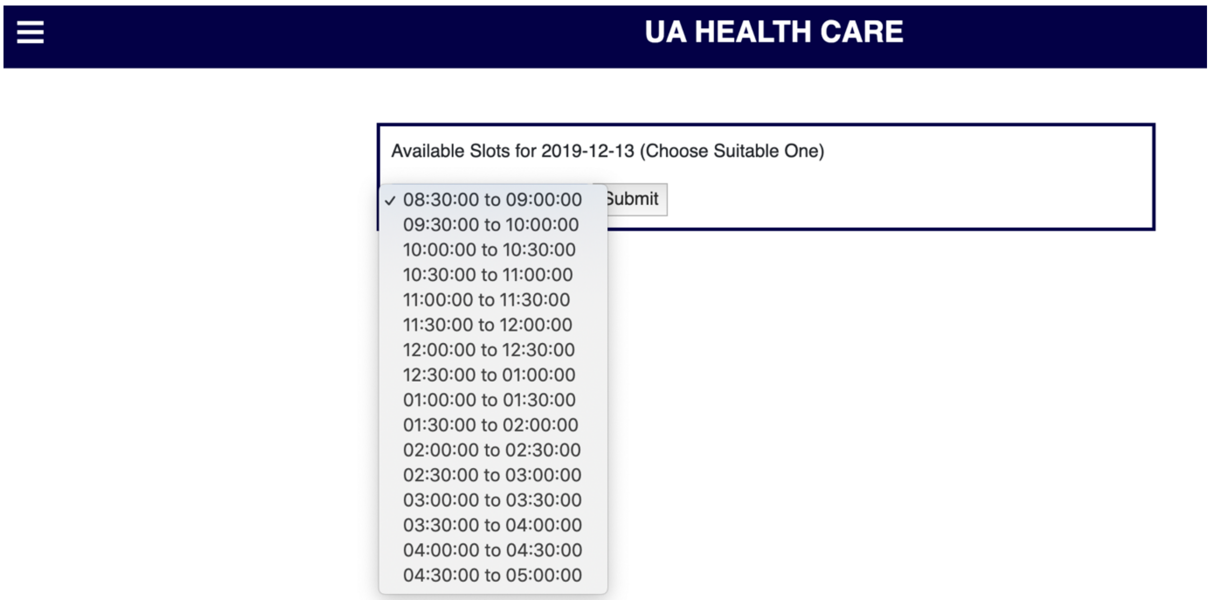
Query 8 – Available Time Slots for Appointments

For a given doctor and an appointment date, fetch available time slots to book an appointment

```
WITH s AS
(SELECT * FROM (SELECT starttime, endtime FROM appointment_slots_new))

SELECT to_char(cast(s.starttime as date),'hh12:mi:ss') as timeslotstarttime,
to_char(cast(s.endtime as date),'hh12:mi:ss') as timeslotendtime , to_char(cast(a.appdate as
date),'DD-MON-YY') ,a.doctorid
FROM s
   LEFT OUTER join appointments_new a on to_char(cast(a.starttime as date),'hh12:mi:ss') =
to_char(cast(s.starttime as date),'hh12:mi:ss')
WHERE (a.doctorid = 11
AND to_char(cast(a.appdate as date),'DD-MON-YY') != '12-DEC-19')
OR ( a.starttime is null AND a.endtime is null )
ORDER BY s.starttime,s.endtime;
```

Note: This query is used in the backend hence, Parameters for doctor id and appdate are inputted in our code. For testing purposes, doctor id = 11 and appdate = ‘12-Nov-19'

Output:


UA HEALTH CARE

Available Slots for 2019-12-13 (Choose Suitable One)

✓ 08:30:00 to 09:00:00
09:30:00 to 10:00:00
10:00:00 to 10:30:00
10:30:00 to 11:00:00
11:00:00 to 11:30:00
11:30:00 to 12:00:00
12:00:00 to 12:30:00
12:30:00 to 01:00:00
01:00:00 to 01:30:00
01:30:00 to 02:00:00
02:00:00 to 02:30:00
02:30:00 to 03:00:00
03:00:00 to 03:30:00
03:30:00 to 04:00:00
04:00:00 to 04:30:00
04:30:00 to 05:00:00

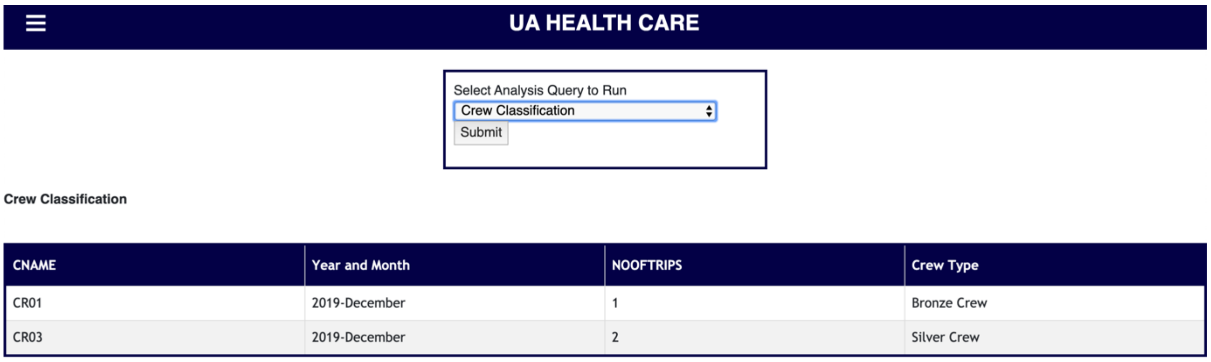
Figure 8: Appointments Output

Query 9 – Crew Classification

Classify the crews based on number of ambulance trips they have been a part of.
Display Crew Name, Year and Month of the trip, Number of trips and Crew Type.
If the number of trips is >= 4 display Crew Type as "Gold Crew",
If the number of trips is >= 2 display Crew Type as "Silver Crew",
Else the Crew Type is "Bronze Crew".

```
WITH t AS
(SELECT * FROM (SELECT cr.crewid cid, cr.crewname cname, extract (year from
(to_date(to_char(td.timeoftrip,'dd-MON-yyyy')))) as Yearv,
to_char((to_date(to_char(td.timeoftrip,'dd-MON-yyyy'))),'Month') as monthv,
count(td.tripid) as Nooftrips
FROM trip_details td
   JOIN crews cr ON cr.crewid = td.crewid
   JOIN case_details cd ON cd.caseid = td.caseid
group by cr.crewid, cr.crewname,(extract (year from (to_date(to_char(td.timeoftrip,'dd-MON-
yyyy'))))),
to_char((to_date(to_char(td.timeoftrip,'dd-MON-yyyy'))),'Month')))

SELECT cname "Crew Name",concat(concat(yearv,'-'),monthv) "Year and Month" ,Nooftrips
AS "No of Trips",
   (CASE
   WHEN Nooftrips >= 4 THEN 'Gold Crew'
   WHEN Nooftrips >=2 and Nooftrips <=3 THEN 'Silver Crew'
   ELSE 'Bronze Crew'
   END) AS "Crew Type"
FROM t
ORDER BY cid;
```

Output:



Crew Classification

CNAME	Year and Month	NOOFTRIPS	Crew Type
CR01	2019-December	1	Bronze Crew
CR03	2019-December	2	Silver Crew

Figure 9: Crew Classifications Output

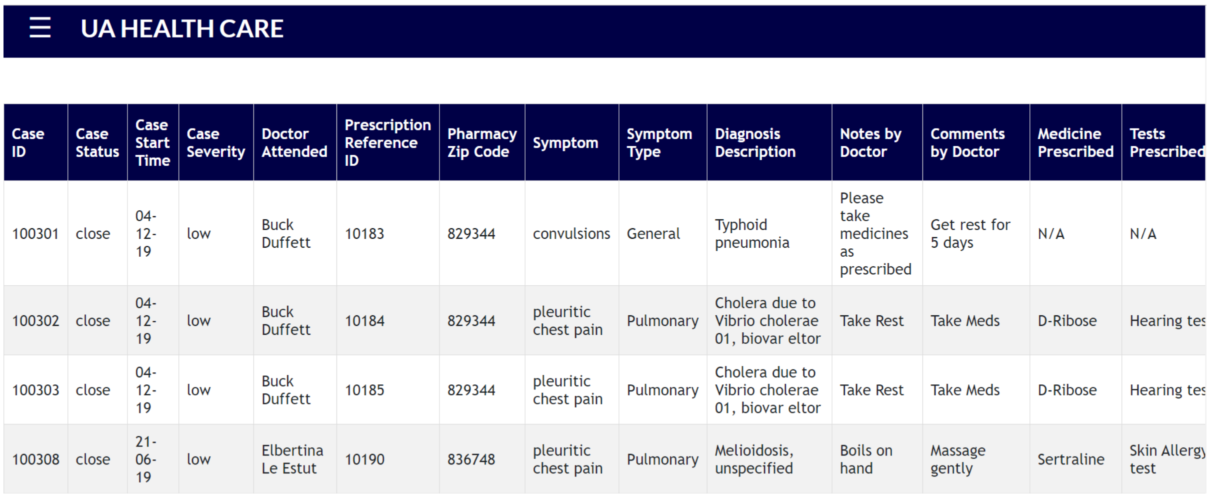
Query 10 – Patient Case History

When patient logs in, he can see his case history. This would allow him to go through his previous cases. This includes the caseid, case details, the doctor who attended the case and the medicines or tests prescribed by the doctor.

```
SELECT cd.caseid, cd.status, cd.datetime, severity, (e.firstname || ' ' ||e.lastname) as "Doctor
Name", p.prescriptionid,
ph.zip as "Pharmacy Zip", cs.symptomid, s.name, s.type, di.description, notes, comments,
td.street,td.buildingno, m.productname,lt.testname
FROM case_details cd
   LEFT OUTER JOIN employees e on e.employeeid = cd.doctorid
   LEFT OUTER JOIN prescriptions p on cd.caseid = p.caseid
   LEFT OUTER JOIN pharmacies ph on p.pharmacyid = ph.pharmacyid
   LEFT OUTER JOIN case_symptoms cs on cd.caseid = cs.caseid
   LEFT OUTER JOIN symptoms s on cs.symptomid = s.symptomid
   LEFT OUTER JOIN diagnosis_details dd on cd.caseid = dd.caseid
```

```
    LEFT OUTER JOIN diagnoses di on dd.icdcode = di.icdcode
    LEFT OUTER JOIN trip_details td on cd.caseid = td.caseid
    LEFT OUTER JOIN drug_details drd on p.prescriptionid = drd.prescriptiionid
    LEFT OUTER JOIN medicines m on drd.medicineid = m.medicineid
    LEFT OUTER JOIN prescriptions_labtests pl on p.prescriptionid= pl.prescriptionid
    LEFT OUTER JOIN lab_tests lt on pl.testid = lt.testid
WHERE cd.patientid = '1004'
ORDER BY cd.caseid;
```

Output:



☰ UA HEALTH CARE

Case ID	Case Status	Case Start Time	Case Severity	Doctor Attended	Prescription Reference ID	Pharmacy Zip Code	Symptom	Symptom Type	Diagnosis Description	Notes by Doctor	Comments by Doctor	Medicine Prescribed	Tests Prescribed
100301	close	04-12-19	low	Buck Duffett	10183	829344	convulsions	General	Typhoid pneumonia	Please take medicines as prescribed	Get rest for 5 days	N/A	N/A
100302	close	04-12-19	low	Buck Duffett	10184	829344	pleuritic chest pain	Pulmonary	Cholera due to Vibrio cholerae 01, biovar eltor	Take Rest	Take Meds	D-Ribose	Hearing tes
100303	close	04-12-19	low	Buck Duffett	10185	829344	pleuritic chest pain	Pulmonary	Cholera due to Vibrio cholerae 01, biovar eltor	Take Rest	Take Meds	D-Ribose	Hearing tes
100308	close	21-06-19	low	Elbertina Le Estut	10190	836748	pleuritic chest pain	Pulmonary	Melioidosis, unspecified	Boils on hand	Massage gently	Sertraline	Skin Allergy test

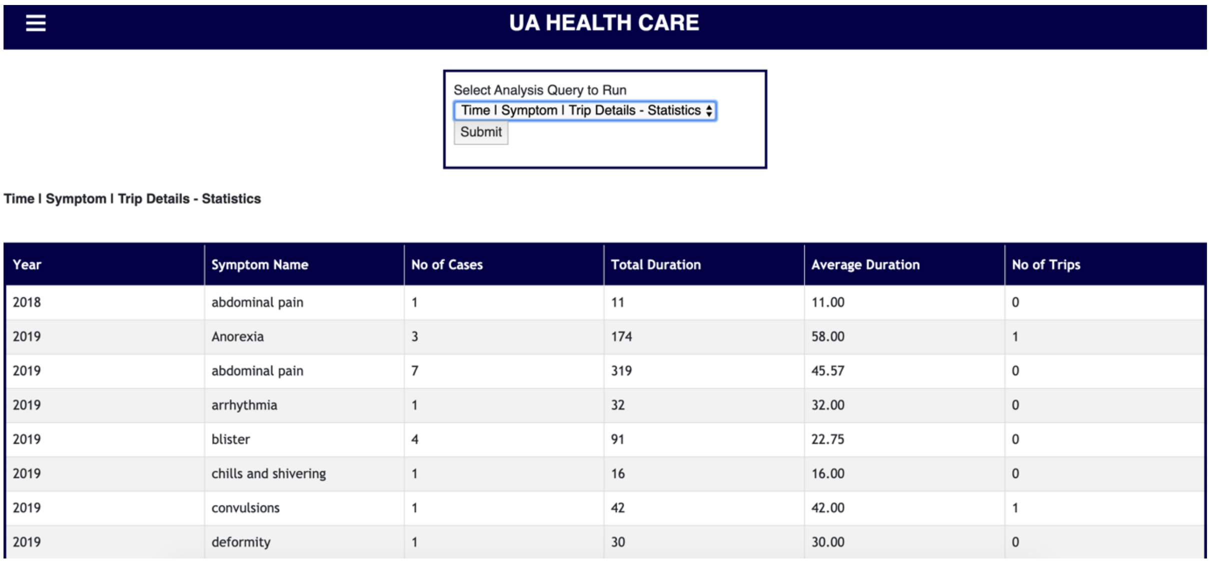
Figure 10: Patient Case History Output

Query 11 – Time and Trip Analysis based on a Symptom

In a year and for a given symptom, calculate the no of cases registered, total duration of the cases, average duration of the
cases and the total number of times ambulance has been sent for that symptom registered in a case

```
SELECT extract (year from (to_date(to_char(cd.datetime,'dd-MON-yyyy')))) as "Year",
s.name as "Symptom Name",
count(cd.caseid) as "No of Cases", sum(cd.duration_min) as "Total Duration",
to_char(Avg(cd.duration_min),'999.99') as "Average Duration", count(tripid) as "No of Trips"
FROM symptoms s
    JOIN case_symptoms cs ON s.symptomid = cs.symptomid
    JOIN case_details cd ON cd.caseid = cs.caseid
    LEFT OUTER JOIN trip_details td ON td.caseid = cd.caseid
GROUP BY s.name,extract (year from (to_date(to_char(cd.datetime,'dd-MON-yyyy'))))
ORDER BY extract (year from (to_date(to_char(cd.datetime,'dd-MON-yyyy'))))
```

Output:


UA HEALTH CARE
Select Analysis Query to Run
Time I Symptom I Trip Details - Statistics
Submit

Time I Symptom I Trip Details - Statistics

Year	Symptom Name	No of Cases	Total Duration	Average Duration	No of Trips
2018	abdominal pain	1	11	11.00	0
2019	Anorexia	3	174	58.00	1
2019	abdominal pain	7	319	45.57	0
2019	arrhythmia	1	32	32.00	0
2019	blister	4	91	22.75	0
2019	chills and shivering	1	16	16.00	0
2019	convulsions	1	42	42.00	1
2019	deformity	1	30	30.00	0

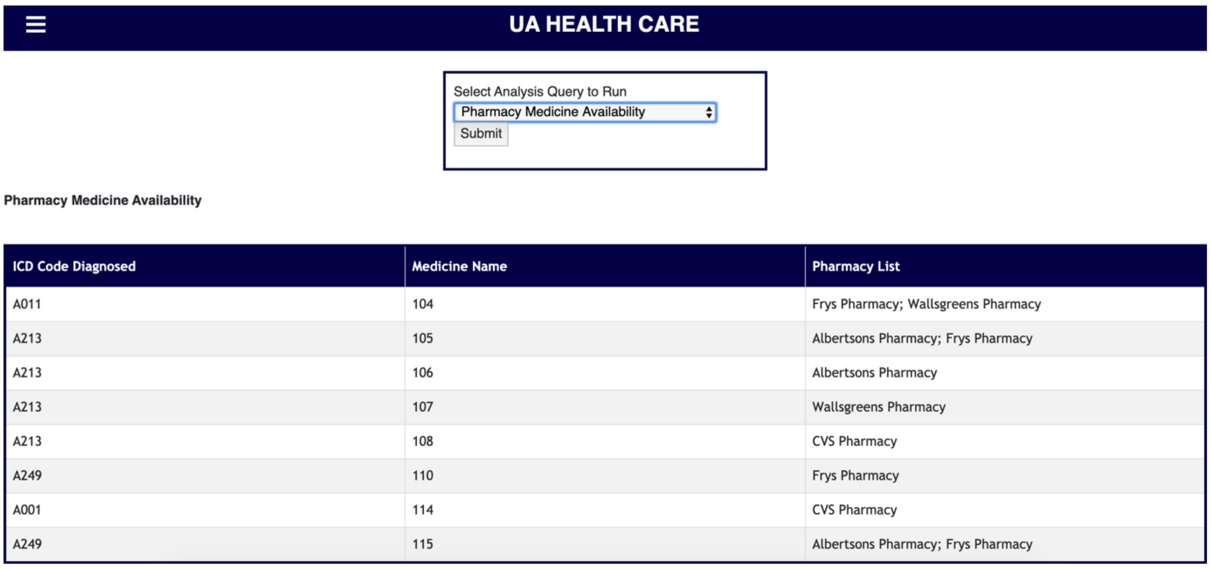
Figure 11: Time and Trip Analysis Output

Query 12 – Pharmacy Medicine Availability

For the ICD Codes diagnosed for a particular case, display the medicine prescribed and availability in the pharmacies

```
WITH x AS (
SELECT p.caseid cid, di.icdcode as "ICD CODE",dd.medicineid as medid,p.pharmacyid as pid, ph.pharmacyname as "Pharmacy Name"
FROM drug_details dd
  JOIN prescriptions p ON dd.prescriptiionid = p.prescriptionid
  JOIN pharmacies ph ON ph.pharmacyid = p.pharmacyid
  JOIN diagnosis_details di ON di.caseid = p.caseid
)
SELECT distinct ("ICD CODE") as "ICD Code Diagnosed", medid as "Medicine Name",
Listagg (p.pharmacyname, '; ') within group (order by p.pharmacyname) over (partition by x.medid) as "Pharmacy List"
FROM pharmacies p
  JOIN x ON x.pid = p.pharmacyid
GROUP BY "ICD CODE", medid, p.pharmacyname, x.medid
ORDER BY medid;
```

Output:


UA HEALTH CARE

Select Analysis Query to Run
Pharmacy Medicine Availability
Submit

Pharmacy Medicine Availability

ICD Code Diagnosed	Medicine Name	Pharmacy List
A011	104	Frys Pharmacy; Wallsgreens Pharmacy
A213	105	Albertsons Pharmacy; Frys Pharmacy
A213	106	Albertsons Pharmacy
A213	107	Wallsgreens Pharmacy
A213	108	CVS Pharmacy
A249	110	Frys Pharmacy
A001	114	CVS Pharmacy
A249	115	Albertsons Pharmacy; Frys Pharmacy

Figure 12: Pharmacy Medicine Availability Output

Chapter 5: Triggers and Procedures

Below are the triggers and procedures which we have used to complete our project requirements. Along with triggers and procedures, we have used encrypt and decrypt functions which encrypts patient credentials.

Triggers

Trigger 1 - trig_trip_details

```
CREATE OR REPLACE TRIGGER trig_trip_details
AFTER INSERT OR UPDATE ON case_details
FOR EACH ROW

DECLARE
check_severity case_details.severity%type;
new_trip_id trip_details.tripid%type;
patient_street students.street%type;
patient_building students.buildingno%type;
patient_zip students.zip%type;
current_crew crews.crewid%type;
current_ambulance ambulances.ambulanceid%type;
trip_time trip_details.timeoftrip%type;

BEGIN

check_severity:=:new.severity;

new_trip_id:= trip_id_seq.nextval;
trip_time :=sysdate;


select buildingno, s.street,zip into patient_building, patient_street, patient_zip
from students s
where rownum=1;

select crewid into current_crew
from
(SELECT crewid FROM crews
ORDER BY dbms_random.value )
WHERE rownum = 1;

select ambulanceid into current_ambulance
from
(SELECT ambulanceid FROM ambulances
where upper(availability)= 'TRUE'
ORDER BY dbms_random.value )
WHERE rownum=1;
```

```
if(upper(check_severity) = 'HIGH')
then
insert into trip_details (tripid,timeoftrip, buildingno, street, zip,crewid,ambulanceid, caseid)
values (new_trip_id,trip_time,
patient_building,patient_street,patient_zip,current_crew,current_ambulance, :new.caseid);
end if;

END;

/*
Trigger trig_trip_details first checks the severity of the case. This is dependent on "severity"
attribute of case.
If the severity is “High” then we trigger an ambulance. Here one record for trip details of
ambulance will be inserted.
For a trip the crew and the ambulance is randomly chosen.  The trip is recorded for the
patients address which is Available from student database. The trip id will be automatically
generated using sequence.
*/
```

Trigger 2 - generate_bill

```
create or replace TRIGGER BITSPLEASE.GENERATE_BILL
AFTER INSERT
ON diagnosis_details
FOR EACH ROW

DECLARE
diag_status diagnosis_details.diagnosis_complete%TYPE;
case_status case_details.status%TYPE;
current_case_id billing_details.caseid%TYPE;
current_ins_company billing_details.inscompanyid%TYPE;
new_bill_id billing_details.billid%TYPE;

BEGIN

diag_status := :new.diagnosis_complete;

SELECT inscompanyid INTO current_ins_company
FROM students s
JOIN patients p ON p.studentid = s.studentid
JOIN case_details cd ON p.patientid = cd.patientid
WHERE cd.caseid = :new.caseid;

new_bill_id := bill_id_seq1.nextval;

IF(UPPER(diag_status) = 'YES') THEN
  case_status:= 'close';
```

```
   INSERT INTO billing_details VALUES (new_bill_id,5,0,sysdate,
'Paid',current_ins_company, :new.caseid );
   UPDATE case_details SET status = case_status WHERE caseid= :new.caseid;
ELSIF(UPPER(diag_status) = 'NO') THEN
   case_status:= 'open';
   INSERT INTO billing_details VALUES (new_bill_id,5,0,sysdate,
'Paid',current_ins_company, :new.caseid );
   UPDATE case_details SET status = case_status WHERE caseid= :new.caseid;
END IF;

END;

/*
Trigger GENERATE_BILL is used to generate a bill for specific case. The bill is generated
as soon as the diagnosis is complete. There is fixed $5 consultation fees which is taken for
each case. There can be additional charges. sysdate is chosen to date the bill at the time of
insert. We add insurance company in billing details to send the bill to the insurance company
with which the student is registered. For that we join the insurance companies and student
tables. This trigger also marks the case as closed as soon as the diagnosis is complete. The
attribute "diagnosis_complete" in diagnosis details table is used check if the diagnosis is
complete or not. Bill is generated for both the cases, whether the diagnosis is complete or not.
The bill id is generated automatically using sequence.
*/
```

Trigger 3 - chat_details_doc

```
create or replace trigger chat_details_doc
before insert
on chat_details
for each row

declare
new_symptom chat_details.initial_symptoms%type;
new_doc chat_details.docid%type;
sym_type symptoms.type%type;

begin

:new.chatid := chat_details_id.nextval;

select type into sym_type
from symptoms
where symptomid = :new.initial_symptoms;

dbms_output.put_line(sym_type);
if(upper(sym_type)= 'GENERAL') then

select generalphysicianid into new_doc
from
(SELECT generalphysicianid FROM general_physicians
ORDER BY dbms_random.value )
WHERE rownum = 1;

else
select specialistid into new_doc
from specializations s
join specialists sp
on sp.specializationid = s.specializationid
where upper(specializationname) = upper(sym_type);

end if;

:new.docid := new_doc;


end;

/*
The trigger chat_details_doc is used to assign doctor to the patient according to the symptoms
that the patient gives.
The symptoms tables has types which is matched with the specialisation which a specialist
has. If the symptoms are "General", then a general physician is randomly assigned to the
patient. The if condition checks the symptoms type and assigns the doctor accordingly. The
doctor id is updated on the chat_details table. We also update the chatid using sequence
chat_details_id.
*/
```

Trigger 4 - age_calculation_emp

```
CREATE OR REPLACE TRIGGER age_calculation_emp
BEFORE INSERT OR UPDATE
ON EMPLOYEES
FOR EACH ROW
DECLARE
   agecalc EMPLOYEES.AGE%type;
   dateofbirth EMPLOYEES.DATEOFBIRTH%type;
   empid EMPLOYEES.EMPLOYEEID%type;
   eid EMPLOYEES.EMPLOYEEID%type;

BEGIN
with s as (select :new.dateofbirth dateofbirth, :new.employeeid empid from dual)
    select round(((select sysdate from dual) - s.dateofbirth)/365),s.empid into agecalc, eid
from s
    where s.empid = :new.employeeid;
    :new.age := agecalc;
END;

/*
Trigger age_calculation_emp is used to calculate derived attribute age of the employee. Age
is calculated using
```

```
the date of birth of the employee. We subtract the date of birth from the current date to obtain
recent age
*/
```

Trigger 5 - age_calculation_student

```
CREATE OR REPLACE TRIGGER age_calculation_student
BEFORE INSERT OR UPDATE
ON STUDENTS
FOR EACH ROW
DECLARE
   agecalc STUDENTS.AGE%type;
   dateofbirth STUDENTS.DATEOFBIRTH%type;
   studid students.studentid%type;
   sid students.studentid%type;

BEGIN
with s as (select :new.dateofbirth dateofbirth, :new.studentid studid from dual)
    select round(((select sysdate from dual) - s.dateofbirth)/365),s.studid into agecalc, sid
from s
    where s.studid = :new.studentid;
    :new.age := agecalc;
END;

/*
Trigger age_calculation_student is used to calculate the age of the student. We subtract date
of birth of the
student from current date to calculate the current age.
*/
```

Trigger 6 - appointmentendtime_slot

```
CREATE OR REPLACE TRIGGER appointmentendtime_slot
BEFORE INSERT OR UPDATE
ON appointments
FOR EACH ROW

DECLARE
appid appointments.appointmentid%type;
current_end_time appointments.endtime%type;
current_start_time appointments.starttime%type;

BEGIN

with s as (select :new.starttime current_start_time, :new.appointmentid appid from dual)
select  s.current_start_time into current_end_time
from s
where s.appid = :new.appointmentid;
```

```
:new.endtime := current_end_time + interval '30' minute ;

END;

/*
Trigger appointmentendtime_slot is used to calculate the end time of the appointment slot. A
slot is typically of 30 minutes. Hence end time of a slot would be calculated by adding 30
mins to the start time. As our data type is interval, the code interval '30' minute
would add 30 minutes to the start time.
*/
```

Trigger 7 - actuallabtestid_trigger

```
CREATE OR REPLACE TRIGGER actuallabtestid_trigger
BEFORE INSERT
ON actual_lab_tests
FOR EACH ROW
--DECLARE
--      temp_actuallabtestid actual_lab_tests.actuallabtestid%type;
BEGIN
--      SELECT actuallabtestid_seq.nextval INTO temp_actuallabtestid FROM dual;
      :new.actuallabtestid := actuallabtestid_seq.nextval;
END;

/*
The trigger populates the actuallabtestid using sequence actuallabtestid_seq
*/
```

Trigger 8 - appointment_id_trig

```
CREATE OR REPLACE TRIGGER appointment_id_trig
BEFORE INSERT OR UPDATE
ON appointments
FOR EACH ROW

DECLARE

BEGIN
DBMS_OUTPUT.ENABLE;
:new.appointmentid := appointment_id_seq.nextval;

END;

/*
Trigger appointment_id_trig updates the appointmentid for appointments using sequence
appointment_id_seq
*/
```

Trigger 9 - case_new_id

```
CREATE OR REPLACE TRIGGER case_new_id
BEFORE INSERT
ON CASE_DETAILS
FOR EACH ROW
BEGIN
  :new.caseid := case_id_seq.nextval;
END;

/*
Trigger case_new_id is used to update caseid using sequence case_id_seq
*/
```

Trigger 10 - feedback_id_trig

```
CREATE OR REPLACE TRIGGER feedback_id_trig
BEFORE INSERT
ON feedbacks
FOR EACH ROW

DECLARE


BEGIN
DBMS_OUTPUT.ENABLE;
:new.feedbackid := feedback_id_seq.nextval;

END;

/*
Trigger feedback_id_trig is used to update feedbackid using sequence feedback_id_seq
*/
```

Trigger 11 - lab_id_trig

```
CREATE OR REPLACE TRIGGER lab_id_trig
BEFORE INSERT
ON labs
FOR EACH ROW

DECLARE

BEGIN

DBMS_OUTPUT.ENABLE;
:new.labid := lab_id_seq.nextval;

END;

/*
Trigger lab_id_trig is used to update labid using sequence lab_id_seq
```

```
*/
```

Trigger 12 - medicine_id_trig

```
CREATE OR REPLACE TRIGGER  medicine_id_trig
BEFORE INSERT
ON medicines
FOR EACH ROW

DECLARE

BEGIN
DBMS_OUTPUT.ENABLE;
:new.medicineid := medicine_id_seq.nextval;

END;

/*
Trigger medicine_id_trig is used to update medicineid using sequence medicine_id_seq
*/
```

Trigger 13 - patient_trigger

```
CREATE OR REPLACE TRIGGER patient_trigger
BEFORE INSERT
ON PATIENTS
FOR EACH ROW

BEGIN
:new.PATIENTID := patient_seq.nextval;

END;

/*
Trigger patient_trigger is used to update patientid using sequence patient_seq
*/
```

Trigger 14 - pharmacy_id_trig

```
CREATE OR REPLACE pharmacy_id_trig
BEFORE INSERT
ON pharmacies
FOR EACH ROW

DECLARE



BEGIN
DBMS_OUTPUT.ENABLE;
```

```
:new.pharmacyid := pharmacy_id_seq.nextval;

END;

/*
Trigger pharmacy_id_trig is used to update patientid using sequence pharmacy_id_seq
*/
```

Trigger 15 - prescription_trigger

```
CREATE OR REPLACE TRIGGER prescription_trigger
BEFORE INSERT
ON Prescriptions
FOR EACH ROW

BEGIN
:new.prescriptionID := prescription_seq.nextval;
END;

/*
Trigger prescription_trigger is used to update prescriptionID using sequence prescription_seq
*/
```

Procedures

Procedure 1 - EMPLOYEE_LOYALTYPOINTS

```
CREATE OR REPLACE PROCEDURE EMPLOYEE_LOYALTYPOINTS
AS CURSOR Cursor1 IS
select err.employeeID employeeid,count(err.employeeratingid) employeeratingno,
sum(er.ratingID) ratingsum from employee_ratings_received err
join employee_ratings er on er.employeeratingid = err.employeeratingid
where ((select sysdate from dual) - er.ratingdate) <= 365
group by err.employeeID
order by err.employeeID;

points employees.loyaltypoints%type;

BEGIN
FOR emprating IN Cursor1 LOOP

points := 0;
IF (emprating.employeeratingno > 0)
THEN points := points + emprating.ratingsum;
END IF;

IF (emprating.employeeratingno = 4 and emprating.ratingsum = 20)
THEN points := points + 10;
END IF;
```

```
UPDATE Employees set loyaltypoints = points where employeeid = emprating.employeeid;
END LOOP;

END;

/*
Procedure EMPLOYEE_LOYALTYPOINTS is used to calculate loyalty points of the
employees. The loyalty points are updated
for each quarter on the basis of employee ratings.
*/
```

Procedure 2 - password_check

```
create or replace procedure password_check(sid patients.studentid%type , usernm
patients.username%type , oldpassword patients.password%type) as
var_sid number(3):= null;
begin

select studentid
into var_sid
from patients
where studentid=sid
and username=usernm
and password = (select encrypt(oldpassword,'keytestvalue') from dual);

exception
when NO_DATA_FOUND then
raise_application_error(-20001, 'Invalid Credentials');

end;

/*
The procedure password_check checks if the credentials of student are valid. This procedure
takes student id, username and password as input and compares the password from the
database. if the same credentials are not found for that studentid then we raise an application
error*/
```

Procedure 3 - signup_proc2

```
create or replace procedure signup_proc2 (uname VARCHAR2,stuid number) as
cursor ctest is
select username,studentid
from patients;

check_username patients.username%type;
check_stdid patients.studentid%type;
test_stdid students.studentid%type;
flag VARCHAR2(2);

BEGIN
```

```
select count(*) into test_stdid
from students
where studentid = stuid;

if (test_stdid > 0) then


   open ctest;
   loop
      FETCH ctest into check_username, check_stdid;
      EXIT WHEN ctest%NOTFOUND;

   if( uname = check_username) then
      raise_application_error
      (-20090, 'Username already exists');
   elsif( stuid = check_stdid)  then
      raise_application_error
      (-20091, 'Student ID already exists');
   end if;

   end loop;
   close ctest;


else
raise_application_error(-20092, 'Invalid StudentID');

end if;

end;

/*
The procedure signup_proc2 is used to check valid student when he/she signs up as patient. It
also ensures that username used by a patient during signup is unique. The if condition checks
if the patient is registered as a student. If no data is found of that student then we raise an
application error. If the student exists in the database, then we check if the username entered
is unique or not. If the username is duplicate, we raise an application error.
*/
```

Functions

Reference: https://jameshuangsj.wordpress.com/2019/05/09/data-encryption-and-decryption-in-oracle/

Encrypt

```
CREATE OR REPLACE FUNCTION encrypt (p_text IN VARCHAR2, p_key
VARCHAR2)
```

```
 RETURN RAW
 IS
 lc_text VARCHAR2(32767) := p_text;
 lr_key RAW(255) := UTL_RAW.cast_to_raw(p_key);
 lt_enc_text RAW(32767);
BEGIN
lc_text := RPAD( lc_text, (TRUNC(LENGTH(lc_text)/8)+1)*8, CHR(0) );
DBMS_OBFUSCATION_TOOLKIT.desencrypt(input => UTL_RAW.cast_to_raw(lc_text),
                                    key => lr_key,
                          encrypted_data => lt_enc_text);
RETURN lt_enc_text;
END;

/*
The function encrypt is used to store password in encrypted form in the database
*/
```

Decrypt

```
CREATE OR REPLACE FUNCTION decrypt (p_raw IN RAW, p_key VARCHAR2 )
RETURN VARCHAR2 IS
 lc_decrypted VARCHAR2(32767);
 lc_return_dec VARCHAR2(32767);
 lr_key RAW(255) := UTL_RAW.cast_to_raw(p_key);
 BEGIN
  DBMS_OBFUSCATION_TOOLKIT.desdecrypt(input => p_raw,
                                      key => lr_key,
                            decrypted_data => lc_decrypted);
   lc_return_dec := UTL_RAW.cast_to_varchar2(lc_decrypted);
  RETURN RTRIM(lc_return_dec, CHR(0) );
END;

/*
The decrypt function retrieves the encrypted password as a plain text.
*/
```

Chapter 6: Interface (UI) and Reports

Application URL : http://uahealthapp.eastus.cloudapp.azure.com:8080/HealthApp

PATIENT

Username: orlan23
Password: o123123

DOCTOR

Username: doc20
Password: mydoc20

ADMINISTRATOR

Username: admin
Password: pass

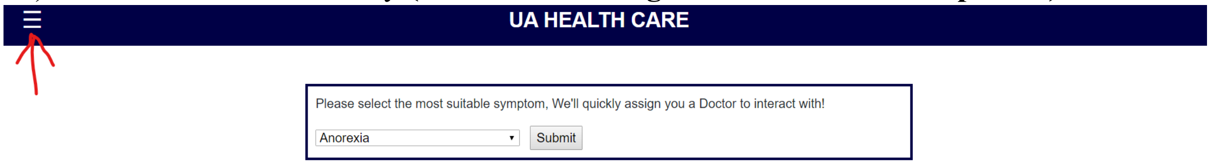
WEB APP WALKTHROUGH

DETAILED WALKTHROUGH VIDEO (DEMO WITH NARRATION) URL:
https://www.youtube.com/watch?v=R9R6UjR4-dU

STEPS IN DEMO

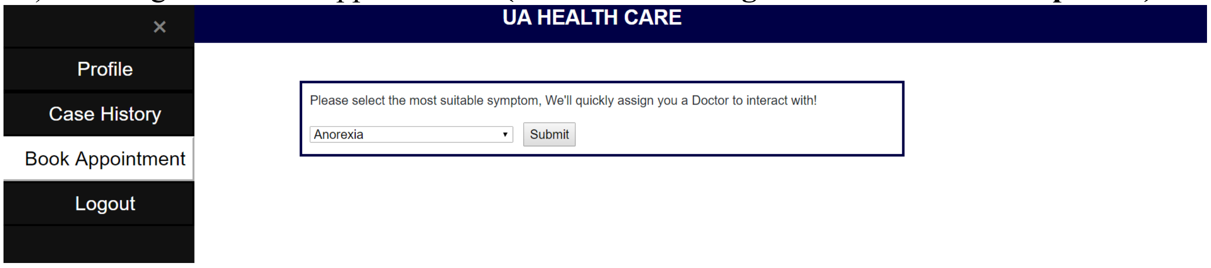
1) Signup as a patient **(Few available STUDENT ID for signup: 58, 61, 64, 67, 70, 73, 77)**
2) Test #1 - Student ID not present in database (Student not registered with University)
3) Test #2 - Student ID already present in patients table (Student already registered as patient)
4) Test #3 - username cannot be duplicate (needs to be said in voiceover - missed in video)
4) Enter correct credentials - Account created successfully
5) Change password
6) Test #4 - Enter Invalid Old Password
7) Enter correct old password and update the password
8) Test #5 - Sign in using old password - Error Thrown
9) Sign in using correct credentials **(You can use the ID created through sign up or use the patient credentials provided above)**
10) Select a symptom **(Please select "abdominal pain" as the doctor is assigned based on the specialization, we have provided the credentials for a doctor that will get associated with "abdominal pain")** and submit
11) In a new tab login as doctor (Doctor is assigned by matching the symptom type and the specialization – **Please use the Doctor login id and password provided above**
12) Initial symptom, name of patient and patient ID is auto populated
13) Patient starts the chat
14) Doctor fills the patient form
15) Doctor can add additional symptom if required. If the case severity is high, ambulance will be dispatched.
16) When the diagnosis is complete the case will be closed

17) Chat ends - Case created
18) Patient views Case History **(Access Left Navigation Pane for these options)**



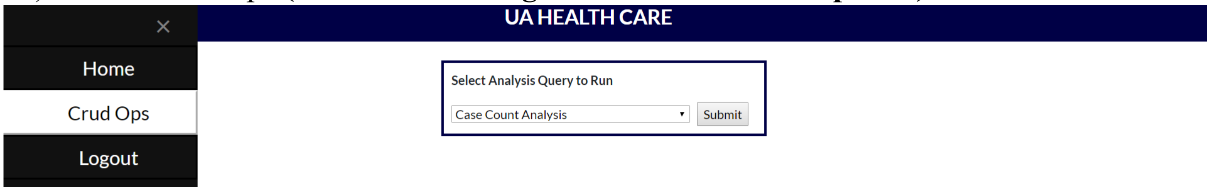
Please select the most suitable symptom, We'll quickly assign you a Doctor to interact with!

19) Patient views profile
20) Patient goes to book appointments **(Access Left Navigation Pane for these options)**

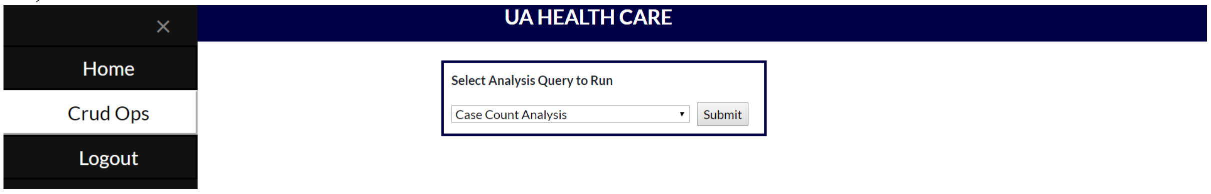


Please select the most suitable symptom, We'll quickly assign you a Doctor to interact with!

21) Select related Specialization and the available time slot - book appointment
22) Test #6 - Select same specialization to book another appointment, we will not be able to see previously booked slot (As it is already booked)
23) Login as admin **(Please use Administrator ID and Password provided above)** and run queries
24) Go to CRUD ops **(Access Left Navigation Pane for these options)**



25) Select Pharmacy - We insert new pharmacy
26) Delete a pharmacy
27) Similarly, we can perform CRUD operations on Labs, Ambulances and Delivery Organizations table in our database.
28)



A lot of CRUD operations are being already performed within in the application logic

The Maven Web Java Project code can be accessed in our GitHub Repository
GitHub Repo URL: https://github.com/virajsingh91/HealthApp

Chapter 7: Conclusions and implementation plan.

Lessons Learnt

1. Good communication was key to the way our team functioned.
2. Nothing is impossible if you start early, we achieved a lot by starting early.
3. Well defined scope helped develop a better product.
4. Importance of a good DB design and normalized tables.
5. Practice your presentation, it makes it easy to engage with your audience.
6. Conflicts led to better team chemistry and better software design.
7. Ask for help, ask your professor – it only makes your project better.
8. Never doubt your capabilities, if you have a great idea, believe in it and just go for it. The sense of achievement and happiness of witnessing the end results is worth all the efforts taken.
9. Learnt to listen and understand different perspectives and agree on a decision.
10. Most importantly, Trust! Trust your team members and their abilities.

Changes

Update the ER based on recommendations from

Steps to implement on a real-world database

This project document explains the scope of our project, the assumptions and conditions we considered while building the application. It also includes the entities, relationship diagram, normalized tables and its implementation in Oracle SQL.
To implement this application in the real world, the following steps will be helpful:

Prerequisites

1. Read the requirement document to understand our goal, this will give the Administrator an opportunity to expand/change the scope of the project.
2. Update the ER if there are changes in the relationships, constraints or entities.
3. Normalize the tables into their highest forms.

Application

1. Choose a database that best suits the requirements (we used Oracle)
2. Modify and run the create table scripts with the correct syntax of the chosen database.
3. If you choose to build the database on a cloud platform (DBaaS) follow instructions specific to that application. (like Amazon RDS, Azure SQL, Oracle DB, SAP Cloud etc.). However, we do not recommend this for our application as there is no significant benefit.

Web design and UI

1. The UI we developed is function over form. It was coded using JSP, Java, HTML and CSS. It can be improved using React, Angular or Spring framework.
2. Password encryption using DBMS_OBFUSCATION_TOOLKIT, can be improved using DES, AES etc.
3. SSL and TSL certificates to secure user data

4. Any additional functionality that came from the updated requirements must be incorporated as new features.
5. The chat-app was implemented on a Node.js, we recommend using third party software like HubSpot (https://www.hubspot.com/)

Cloud Hosting

1. Azure provides reasonable pricing for the annual year plan as can be seen from the table below.

Cost Breakup Assumptions

1. All employees hired are Graduate Assistants (GA's) from the University of Arizona which justifies the hourly rate of $15.00 per hour.
2. The developer and database administrator build the initial application in 3 months.
3. The application is maintained by one graduate assistant for approximately 8 months (giving a total cycle of one year).

Cost Breakup

	RATE ($)	**QTY**	**TOTAL COST ($)**
PEOPLE			
Front end developer	15/hr.	150 hrs.	2,250
Database Administrator	15/hr.	200 hrs.	3,000
Maintenance	15/hr.	300 hrs.	4,500
APPLICATION			
Oracle DB	-	-	-
Cloud Hosting*	149.88/mo.	12 mo.	1,798.51
TOTAL			**11,548.51**

***Microsoft Azure Estimate** (generated on Azure calculator)

Your Estimate

Service type	**Custom name**	**Region**	**Description**	**Estimated Cost**
Virtual Machines		West US	1 D2 v3 (2 vCPU(s), 8 GB RAM); Windows – (OS Only); 1 year reserved; 1 managed OS disks – E6, 9,999 transaction units	$149.88
Support			**Support**	$0.00
			Licensing Program	**Microsoft Online Services Agreement**
			Monthly Total	**$149.88**
			Annual Total	**$1,798.51**

Disclaimer

All prices shown are in US Dollar ($). This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/
This estimate was created at 12/12/2019 9:32:07 PM UTC.

https://azure.com/e/e0788c84714b451c82f41100c93c08b0 - Azure price calculator

APPENDIX – Create Table Scripts

1. **ACTUAL_LAB_TESTS**

```
CREATE TABLE BITSPLEASE.ACTUAL_LAB_TESTS
 (
        ACTUALLABTESTID NUMBER (38,0),
        RESULT VARCHAR2(100 BYTE),
        LABID NUMBER (38,0),
        LABTESTID NUMBER (38,0),
        CONSTRAINT ACTUAL_LAB_TEST_PK PRIMARY KEY
(ACTUALLABTESTID),
        CONSTRAINT ACTUAL_LAB_TEST_FK FOREIGN KEY (LABID,
LABTESTID)
        REFERENCES BITSPLEASE.LAB_TEST_DONE_BY (LABID,
LABTESTID)
 );
```

2. **AMBULANCES**

```
CREATE TABLE BITSPLEASE.AMBULANCES
  (
        AMBULANCEID NUMBER (38,0),
        VEHICLENUMBER VARCHAR2(20 BYTE),
        AVAILABILITY VARCHAR2(20 BYTE),
        CONSTRAINT AMULANCES_PK PRIMARY KEY (AMBULANCEID)
  );
```

3. **APPOINTMENT_TIME_SLOTS**

```
CREATE TABLE BITSPLEASE.APPOINTMENT_TIME_SLOTS
  (
        SLOTID NUMBER (38,0),
        STARTTIME INTERVAL DAY (0) TO SECOND (6),
        ENDTIME INTERVAL DAY (0) TO SECOND (6),
        CONSTRAINT APPOINTMENT_TIME_SLOTS_PK PRIMARY KEY
(SLOTID)
  );
```

4. **APPOINTMENTS**

```
CREATE TABLE BITSPLEASE.APPOINTMENTS
  (
        APPOINTMENTID NUMBER (38,0),
        TYPE VARCHAR2(20 BYTE),
        STARTTIME INTERVAL DAY (0) TO SECOND (6),
        ENDTIME INTERVAL DAY (0) TO SECOND (6),
        APPDATE DATE,
        PATIENTID NUMBER (38,0),
        DOCTORID NUMBER (38,0),
            CONSTRAINT APPOINTMENT_PK PRIMARY KEY
(APPOINTMENTID),
            CONSTRAINT APPOINTMENT_DOCTOR_FK REFERENCES
            BITSPLEASE.DOCTORS(DOCTORID),
            CONSTRAINT APPOINTMENT_PATIENTS_FK REFERENCES
            BITSPLEASE.PATIENTS(PATIENTID)
   );
```

5. BILLING_DETAILS

```
CREATE TABLE BITSPLEASE.BILLING_DETAILS
   (
            BILLID NUMBER (38,0),
            TOTAL_COST FLOAT (126),
            ADDITIONALCHARGES FLOAT (126),
            BILLDATE DATE,
            BILLSTATUS VARCHAR2(50 BYTE),
            INSCOMPANYID NUMBER (38,0),
            CASEID NUMBER (38,0),
            CONSTRAINT BILLING_DETAILS_PK PRIMARY KEY (BILLID),
            CONSTRAINT BILLING_INSURANCE_FK FOREIGN KEY
(INSCOMPANYID),
            REFERENCES BITSPLEASE.INSURANCE_COMPANIES
(INSCOMPANYID)
   );
```

6. CASE_DETAILS

```
CREATE TABLE BITSPLEASE.CASE_DETAILS
   (
            CASEID NUMBER (38,0),
            STATUS VARCHAR2(20 BYTE),
            DATETIME TIMESTAMP (6),
            SEVERITY VARCHAR2(20 BYTE),
            DURATION_MIN FLOAT (126),
            DOCTORID NUMBER (38,0),
            PATIENTID NUMBER (38,0),
            CONSTRAINT CASEDETAILS_PK PRIMARY KEY (CASEID),
            CONSTRAINT CASEDETAILS_DOCTORS_FK FOREIGN KEY
(DOCTORID)
            REFERENCES BITSPLEASE.DOCTORS (DOCTORID),
            CONSTRAINT CASEDETAILS_PATIENTS_FK FOREIGN KEY
(PATIENTID)
            REFERENCES BITSPLEASE.PATIENTS (PATIENTID)
   );
```

7. CASE_REPORT_LAB

```
CREATE TABLE BITSPLEASE.CASE_REPORT_LAB
(
      REPORTID NUMBER (38,0),
      ACTUALLABTESTID NUMBER (38,0),
      CASEID NUMBER (38,0),
      CONSTRAINT CASE_MEDICALREPORT_FK FOREIGN KEY (REPORTID)
      REFERENCES BITSPLEASE.MEDICAL_REPORTS (REPORTID),
      CONSTRAINT CASE_ACTUALLAB_FK FOREIGN KEY
(ACTUALLABTESTID)
      REFERENCES BITSPLEASE.ACTUAL_LAB_TESTS
(ACTUALLABTESTID),
      CONSTRAINT CASE_CASEDETAIL_FK FOREIGN KEY (CASEID)
      REFERENCES BITSPLEASE.CASE_DETAILS (CASEID)
);
```

8. CASE_SYMPTOMS

```
CREATE TABLE BITSPLEASE.CASE_SYMPTOMS
  (
         CASEID NUMBER (38,0),
         SYMPTOMID NUMBER (38,0),
         CONSTRAINT CASE_SYMPTOMS_PK PRIMARY KEY (CASEID,
SYMPTOMID),
         CONSTRAINT CASE_SYMPTOMS_DETAILS_FK FOREIGN KEY
(CASEID)
         REFERENCES BITSPLEASE.CASE_DETAILS (CASEID),
         CONSTRAINT CASE_SYMPTOMS_FK2 FOREIGN KEY (SYMPTOMID)
         REFERENCES BITSPLEASE.SYMPTOMS (SYMPTOMID)
  );
```

9. CHAT_DETAILS

```
CREATE TABLE BITSPLEASE.CHAT_DETAILS
  (
         CHATID NUMBER,
         DOCID NUMBER,
         PID NUMBER,
         CHAT_ACTIVE NUMBER,
         INITIAL_SYMPTOMS VARCHAR2(250 BYTE),
         CONSTRAINT CHAT_DETAILS_PK PRIMARY KEY (CHATID),
         CONSTRAINT CHAT_DETAILS_DOCTORS_FK1 FOREIGN KEY
(DOCID)
         REFERENCES BITSPLEASE.DOCTORS (DOCTORID),
         CONSTRAINT CHAT_DETAILS_FK1 FOREIGN KEY (PID)
         REFERENCES BITSPLEASE.PATIENTS (PATIENTID)
 );
```

10. CREW_DRIVERS

```
CREATE TABLE BITSPLEASE.CREW_DRIVERS
  (
            CREWID NUMBER (38,0),
            DRIVERID NUMBER (38,0),
            CONSTRAINT CREW_DRIVERS_PK PRIMARY KEY (CREWID,
DRIVERID),
            CONSTRAINT CREW_DRIVERS_CREWS_FK FOREIGN KEY
(CREWID)
            REFERENCES BITSPLEASE.CREWS (CREWID),
            CONSTRAINT CREW_DRIVERS_DRIVER_FK FOREIGN KEY
(DRIVERID)
            REFERENCES BITSPLEASE.DRIVERS (DRIVERID)
  );
```

11. CREW_EMT

```
CREATE TABLE BITSPLEASE.CREW_EMT
  (
            CREWID NUMBER (38,0),
            EMTID NUMBER (38,0),
            CONSTRAINT CREW_EMT_PK PRIMARY KEY (CREWID, EMTID),
            CONSTRAINT CREWS_EMT_FK FOREIGN KEY (EMTID)
            REFERENCES BITSPLEASE.EMT (EMTID),
            CONSTRAINT CREWS_FK FOREIGN KEY (CREWID)
            REFERENCES BITSPLEASE.CREWS (CREWID)
  );
```

12. CREW_NURSES

```
CREATE TABLE BITSPLEASE.CREW_NURSES
  (
            CREWID NUMBER (38,0),
            NURSEID NUMBER (38,0),
            CONSTRAINT CREW_NURSES_PK PRIMARY KEY (CREWID,
NURSEID),
            CONSTRAINT CREWS_NURSES_FK FOREIGN KEY (CREWID)
            REFERENCES BITSPLEASE.CREWS (CREWID),
            CONSTRAINT NURSES_FK FOREIGN KEY (NURSEID)
            REFERENCES BITSPLEASE.NURSES (NURSEID)
  );
```

13. **CREWS**

```
CREATE TABLE BITSPLEASE.CREWS
  (
        CREWID NUMBER (38,0),
        CREWNAME VARCHAR2(20 BYTE),
        CONSTRAINT CREWS_PK PRIMARY KEY (CREWID)
  );
```

14. **DELIVERY_ASSOCIATES**

```
CREATE TABLE BITSPLEASE.DELIVERY_ASSOCIATES
  (
        DELASSOCIATEID NUMBER (38,0),
        DELASSOCIATENAME VARCHAR2(100 BYTE),
        DLNUMBER VARCHAR2(20 BYTE),
        DELORGID NUMBER,
        SHIFTSTARTTIME INTERVAL DAY (0) TO SECOND (6),
        SHIFTENDTIME INTERVAL DAY (0) TO SECOND (6),
        CONSTRAINT DELIVERY_ASSOCIATES_PK PRIMARY KEY
(DELASSOCIATEID),
        CONSTRAINT DELIVERYASSOCIATES_ORG_FK FOREIGN KEY
(DELORGID)
        REFERENCES BITSPLEASE.DELIVERY_ORGANIZATIONS (DELORGID)
  );
```

15. **DELIVERY_ORGANIZATIONS**

```
CREATE TABLE BITSPLEASE.DELIVERY_ORGANIZATIONS
  (
        DELORGID NUMBER,
        ORGANIZATIONNAME VARCHAR2(50 BYTE),
        NOOFEMPLOYEES NUMBER (38,0),
        BUILDINGNUMBER VARCHAR2(100 BYTE),
        STREET VARCHAR2(100 BYTE),
        ZIP NUMBER (38,0),
        PHONENO VARCHAR2(20 BYTE),
        EMAILADDRESS VARCHAR2(50 BYTE),
        CONSTRAINT DELIVERYORGANIZATIONS_PK PRIMARY KEY
(DELORGID),
        CONSTRAINT CHK_PHONE_DELORG CHECK (phoneno not like '%[^0-9]%')
  );
```

16. **DIAGNOSIS**

```
CREATE TABLE BITSPLEASE.DIAGNOSESs
(
        ICDCODE VARCHAR2(20 BYTE),
        DESCRIPTION VARCHAR2(255 BYTE),
        VERSION VARCHAR2(20 BYTE),
        CONSTRAINT DIAGNOSIS_PK PRIMARY KEY (ICDCODE),
 );
```

17. **DIAGNOSIS_DETAILS**

```
CREATE TABLE BITSPLEASE.DIAGNOSIS_DETAILS
  (
        CASEID NUMBER (38,0),
        ICDCODE VARCHAR2(20 BYTE),
        NOTES VARCHAR2(255 BYTE),
        COMMENTS VARCHAR2(255 BYTE),
        DIAGNOSIS_COMPLETE VARCHAR2(20 BYTE),
        CONSTRAINT DIAGNOSIS_DETAILS PRIMARY KEY (CASEID, ICDCODE),
        CONSTRAINT DIAGNOSIS_STATUS CHECK (diagnosis_complete
IN('Yes','No')),
        CONSTRAINT DIAGNOSISDET_CASEDET_FK FOREIGN KEY (CASEID)
        REFERENCES BITSPLEASE.CASE_DETAILS (CASEID),
        CONSTRAINT DIAGNOSISDET_DIAGNOSES_FK FOREIGN KEY (ICDCODE)
        REFERENCES BITSPLEASE.DIAGNOSES (ICDCODE)
  );
```

18. **DOCTORS**

```
CREATE TABLE BITSPLEASE.DOCTORS
  (
        DOCTORID NUMBER (38,0),
        REGISTRATIONNO VARCHAR2(20 BYTE),
        HIGHESTDEGREE VARCHAR2(100 BYTE),
        ONCALL NUMBER (1,0),
        USERNAME VARCHAR2(50 BYTE),
        PASSWORD VARCHAR2(50 BYTE),
        CONSTRAINT DOCTORS_PK PRIMARY KEY (DOCTORID),
        CONSTRAINT DOCTOR_EMPLOYEE_FK FOREIGN KEY (DOCTORID)
        REFERENCES BITSPLEASE.EMPLOYEES (EMPLOYEEID)
  );
```

19. **DRIVERS**

```
  CREATE TABLE BITSPLEASE.DRIVERS
  (
        DRIVERID NUMBER (38,0),
        DLNO VARCHAR2(20 BYTE),
        CONSTRAINT DRIVERS_PK PRIMARY KEY (DRIVERID),
        CONSTRAINT EMPLOYEE_DRIVERS_FK FOREIGN KEY (DRIVERID)
  )
```

20. **DRUG_DETAILS**

```
CREATE TABLE BITSPLEASE.DRUG_DETAILS
  (
        PRESCRIPTIIONID NUMBER (38,0),
        MEDICINEID NUMBER (38,0),
        FREQUENCY VARCHAR2(255 BYTE),
        COMPOSITION VARCHAR2(255 BYTE),
        CONSTRAINT DRUG_DETAILS_PK PRIMARY KEY
        (PRESCRIPTIIONID, MEDICINEID),
        CONSTRAINT DRUG_DETAILS_MED_FK FOREIGN KEY (MEDICINEID)
        REFERENCES BITSPLEASE.MEDICINES (MEDICINEID),
        CONSTRAINT DRUG_DETAILS_PRESC_FK FOREIGN KEY
(PRESCRIPTIIONID)
        REFERENCES BITSPLEASE.PRESCRIPTIONS (PRESCRIPTIONID)
  )
```

21. **EMPLOYEE_RATINGS**

```
CREATE TABLE BITSPLEASE.EMPLOYEE_RATINGS
  (
        EMPLOYEERATINGID NUMBER (38,0),
        REMARKS VARCHAR2(255 BYTE),
        RATINGDATE DATE,
        RATINGID NUMBER (38,0),
        CONSTRAINT EMPLOYEERATING_PK PRIMARY KEY
(EMPLOYEERATINGID),
        CONSTRAINT EMPLOYEERATINGS_RATING_FK FOREIGN KEY
(RATINGID)
        REFERENCES BITSPLEASE.RATINGS (RATINGID)
  );
```

22. **EMPLOYEE_RATINGS_RECIEVED**

```
CREATE TABLE BITSPLEASE.EMPLOYEE_RATINGS_RECEIVED
  (
        EMPLOYEEID NUMBER (38,0),
        EMPLOYEERATINGID NUMBER (38,0),
        CONSTRAINT EMPID_EMPRATING_PK PRIMARY KEY (EMPLOYEEID,
        EMPLOYEERATINGID),
        CONSTRAINT EMPRATINGID_FK FOREIGN KEY (EMPLOYEERATINGID)
        REFERENCES BITSPLEASE.EMPLOYEE_RATINGS (EMPLOYEERATINGID),
        CONSTRAINT EMP_EMPRATING_FK FOREIGN KEY (EMPLOYEEID)
        REFERENCES BITSPLEASE.EMPLOYEES (EMPLOYEEID)
  );
```

23. **EMPLOYEES**

```
CREATE TABLE BITSPLEASE.EMPLOYEES
  (
        EMPLOYEEID NUMBER (38,0),
        FIRSTNAME VARCHAR2(20 BYTE),
        LASTNAME VARCHAR2(20 BYTE),
        MIDDLEINITIAL VARCHAR2(20 BYTE),
        HIREDATE DATE,
        DATEOFBIRTH DATE,
        AGE NUMBER (38,0),
        GENDER VARCHAR2(10 BYTE),
        SSN NUMBER (38,0),
        PHONENUMBER NUMBER (38,0),
        LOYALTYPOINTS NUMBER (38,0),
        EMAILADDRESS VARCHAR2(50 BYTE),
        SHIFTSTARTTIME INTERVAL DAY (0) TO SECOND (6),
        SHIFTENDTIME INTERVAL DAY (0) TO SECOND (6),
        TYPE VARCHAR2(20 BYTE),
        CONSTRAINT EMPLOYEES_PK PRIMARY KEY (EMPLOYEEID),
        CONSTRAINT EMPLOYEES_SSN_UK UNIQUE (SSN),
        CONSTRAINT CHK_PHONE_EMPLOYEE CHECK (phonenumber not like '%[^0-
9]%')
  ) ;
```

24. **EMT**

```
CREATE TABLE BITSPLEASE.EMT
  (
        EMTID NUMBER (38,0),
        LEVELS VARCHAR2(20 BYTE),
        CONSTRAINT EMT_PK PRIMARY KEY (EMTID),
        CONSTRAINT EMT_EMPLOYEE_FK FOREIGN KEY (EMTID)
        REFERENCES BITSPLEASE.EMPLOYEES (EMPLOYEEID)
  ) ;
```

25. **FEEDBACKS**

```
CREATE TABLE BITSPLEASE.FEEDBACKS
  (
        FEEDBACKID NUMBER (38,0),
        COMMENTS VARCHAR2(255 BYTE),
        DATETIME TIMESTAMP (6),
        PATIENTID NUMBER (38,0),
        CONSTRAINT FEEDBACKS_PK PRIMARY KEY (FEEDBACKID),
        CONSTRAINT FEEDBACKS_PATIENT_FK FOREIGN KEY (PATIENTID)
        REFERENCES BITSPLEASE.PATIENTS (PATIENTID)
  ) ;
```

26. **GENERAL_PHYSICIANS**

```
CREATE TABLE BITSPLEASE.GENERAL_PHYSICIANS
  (
        GENERALPHYSICIANID NUMBER (38,0),
        ISTRAINEE VARCHAR2(20 BYTE),
        CERTIFICATION VARCHAR2(150 BYTE),
        CERTEXPDATE DATE,
        TYPE VARCHAR2(20 BYTE),
        CONSTRAINT GENERAL_PHY_PK PRIMARY KEY
(GENERALPHYSICIANID),
        CONSTRAINT GENERAL_PHYSICIANS_DOCTOR_FK FOREIGN KEY
        (GENERALPHYSICIANID) REFERENCES BITSPLEASE.DOCTORS
(DOCTORID),
        CONSTRAINT GENERAL_PHY_CHECK CHECK (isTrainee='YES' OR
isTrainee='NO')
  ) ;
```

27. INSURANCE_COMPANIES

```
CREATE TABLE BITSPLEASE.INSURANCE_COMPANIES
  (       INSCOMPANYID  NUMBER NOT NULL ENABLE,
          INSCOMPANYNAME  VARCHAR2(100 BYTE),
          INSCOMLICENSENO  VARCHAR2(20 BYTE),
          ADDRESS  VARCHAR2(500 BYTE),
          EMAIL  VARCHAR2(100 BYTE),
          PHONENO  VARCHAR2(20 BYTE),
          CONSTRAINT  INSURANCE_COMPANIES_PK  PRIMARY KEY (
INSCOMPANYID )
CONSTRAINT  CHK_PHONE_INSCOMP  CHECK (phoneno not like '%[^0-9]%')
ENABLE
);
```

28. LAB_TEST_DONE_BY

```
CREATE TABLE BITSPLEASE.LAB_TEST_DONE_BY
  (       LABID NUMBER (38,0),
          LABTESTID NUMBER( 38,0),
          CONSTRAINT LAB_TEST_DONE_BY_PK PRIMARY KEY (LABID ,
LABTESTID )
CONSTRAINT LAB_TEST_DONE_BY_LAB_FK  FOREIGN KEY ( LABID )
           REFERENCES  BITSPLEASE.LABS  ( LABID ) ON DELETE CASCADE
ENABLE,
          CONSTRAINT  LAB_TEST_DONE_BY_LABTEST_FK  FOREIGN KEY (
LABTESTID )
           REFERENCES  BITSPLEASE.LAB_TESTS  ( TESTID ) ON DELETE CASCADE
ENABLE
  );
```

29. LAB_TESTS

```
CREATE TABLE  BITSPLEASE.LAB_TESTS
  (       TESTID  NUMBER( 38,0) NOT NULL ENABLE,
          TESTNAME  VARCHAR2(100 BYTE),
          FEES  FLOAT(126),
          CONSTRAINT  LAB_TESTS_PK  PRIMARY KEY ( TESTID )
);
```

30. LABS

```
CREATE TABLE  BITSPLEASE.LABS
  (       LABID  NUMBER( 38,0) NOT NULL ENABLE,
          LABNAME  VARCHAR2(20 BYTE),
          PHONENUMBER  VARCHAR2(20 BYTE),
          BUILDINGNO  VARCHAR2(50 BYTE),
          ZIP  VARCHAR2(50 BYTE),
          STREET  VARCHAR2(50 BYTE),
          EMAILADDRESS  VARCHAR2(50 BYTE),
          CONSTRAINT  LABS_PK  PRIMARY KEY ( LABID )
```

```
        CONSTRAINT  CHK_PHONE_LABS  CHECK (phonenumber not like '%[^0-
9]%') ENABLE
);
```

31. MEDICALREPORTS

```
CREATE TABLE  BITSPLEASE.MEDICALREPORTS
  (     REPORTID  NUMBER( 38,0) NOT NULL ENABLE,
        REPDATE  DATE,
        REPORTNAME  VARCHAR2(20 BYTE),
        PATIENTID  NUMBER( 38,0),
        ACTUALLABTESTID  NUMBER,
        PRIMARY KEY ( REPORTID )
      FOREIGN KEY ( PATIENTID )
         REFERENCES  BITSPLEASE.PATIENTS  ( PATIENTID ) ENABLE,
        FOREIGN KEY ( ACTUALLABTESTID )
         REFERENCES  BITSPLEASE.ACTUAL_LAB_TESTS  ( ACTUALLABTESTID )
ENABLE
  );
```

32. MEDICINE_DRUGS

```
CREATE TABLE  BITSPLEASE . MEDICINE_DRUGS
  (     MEDICINEID  NUMBER( 38,0),
        DRUGS  VARCHAR2(100 BYTE) NOT NULL ENABLE,
        CONSTRAINT  MEDICINE_DRUG_PK  PRIMARY KEY ( MEDICINEID ,
DRUGS )
      FOREIGN KEY ( MEDICINEID )
         REFERENCES  BITSPLEASE.MEDICINES  ( MEDICINEID ) ENABLE
  );
```

33. MEDICINES

```
CREATE TABLE  BITSPLEASE.MEDICINES
  (     MEDICINEID  NUMBER( 38,0),
        PRODUCTNAME  VARCHAR2(20 BYTE),
        BRAND  VARCHAR2(20 BYTE),
        EXPIRYDATE  DATE,
        TYPE  VARCHAR2(20 BYTE),
        COUNT  NUMBER( 38,0),
        COST_PU  VARCHAR2(20 BYTE),
        CONSTRAINT  MEDICINES  PRIMARY KEY ( MEDICINEID )
 );
```

34. NURSES

```
CREATE TABLE BITSPLEASE.NURSES
  (    NURSEID NUMBER(38,0) NOT NULL ENABLE,
       NURSINGLICENSENO VARCHAR2(20 BYTE),
          TYPE VARCHAR2(100 BYTE),
          HOURLYBILLINGRATE FLOAT(126),
           CONSTRAINT NURSES_PK PRIMARY KEY (NURSEID)
 CONSTRAINT NURSE_EMPLOYEE_FK FOREIGN KEY (NURSEID)
            REFERENCES BITSPLEASE.EMPLOYEES (EMPLOYEEID) ENABLE
)
```

35. PATHOLOGISTS

```
CREATE TABLE BITSPLEASE.PATHOLOGISTS
  (       PATHOLOGISTID NUMBER(38,0),
          CERTIFICATION VARCHAR2(100 BYTE),
          LABID NUMBER(38,0),
           CONSTRAINT PATHOLOGISTS_PK PRIMARY KEY (PATHOLOGISTID)
 CONSTRAINT PATHOLOGISTS_LABS_FK FOREIGN KEY (LABID)
            REFERENCES BITSPLEASE.LABS (LABID) ENABLE,
           CONSTRAINT PATHOLOGISTS_EMP_FK FOREIGN KEY (PATHOLOGISTID)
            REFERENCES BITSPLEASE.EMPLOYEES (EMPLOYEEID) ENABLE
  )
```

36. PATIENTS

```
CREATE TABLE BITSPLEASE.PATIENTS
  (       PATIENTID NUMBER(38,0) NOT NULL ENABLE,
          USERNAME VARCHAR2(20 BYTE) NOT NULL ENABLE,
          PASSWORD VARCHAR2(255 BYTE) NOT NULL ENABLE,
          STUDENTID NUMBER(38,0) NOT NULL ENABLE,
           CONSTRAINT PATIENTS_PK PRIMARY KEY (PATIENTID)
CONSTRAINT STUDENTID_UNIQUE UNIQUE (STUDENTID)
CONSTRAINT USERNAME_UNIQUE UNIQUE (USERNAME)
 CONSTRAINT STUDENTID_FK FOREIGN KEY (STUDENTID)
            REFERENCES BITSPLEASE.STUDENTS (STUDENTID) ENABLE
);
```

37. PRESCRIPTIONS

```
CREATE TABLE BITSPLEASE.PATIENTS_PRECRIPTIONS
  (       PATIENTID NUMBER(38,0) NOT NULL ENABLE,
          PRESCRIPTIONID NUMBER(38,0) NOT NULL ENABLE,
           CONSTRAINT PATIENTS_PRECRIPTIONS_PK PRIMARY KEY (PATIENTID,
PRESCRIPTIONID)
CONSTRAINT PATIENTS_PRESCRIPTIONS_FK FOREIGN KEY (PATIENTID)
            REFERENCES BITSPLEASE.PATIENTS (PATIENTID) ENABLE,
           CONSTRAINT PRESCRIPTIONS_PATIENTS_FK FOREIGN KEY
(PRESCRIPTIONID)
            REFERENCES BITSPLEASE.PRESCRIPTIONS (PRESCRIPTIONID) ENABLE
  );
```

38. PHAR_DELASSOC

```
CREATE TABLE BITSPLEASE.PHAR_DELASSOC
  (       DELASSOCIATEID NUMBER(38,0),
          PHARMACYID NUMBER(38,0),
           CONSTRAINT PHAR_DELASSOC_PK PRIMARY KEY (DELASSOCIATEID,
PHARMACYID)
 CONSTRAINT DELASSOCIATE_FK FOREIGN KEY (DELASSOCIATEID)
            REFERENCES BITSPLEASE.DELIVERY_ASSOCIATES (DELASSOCIATEID)
ENABLE,
           CONSTRAINT PHARMACY_FK FOREIGN KEY (PHARMACYID)
            REFERENCES BITSPLEASE.PHARMACIES (PHARMACYID) ENABLE
  );
```

39. PHAR_PROMO

```
CREATE TABLE BITSPLEASE.PHAR_PROMO
  (       PHARMACYID NUMBER(38,0),
          PROMOTIONID NUMBER(38,0),
           CONSTRAINT PHAR_PROMO_PK PRIMARY KEY (PHARMACYID,
PROMOTIONID)
 CONSTRAINT PHARMACY_PROMO_FK FOREIGN KEY (PHARMACYID)
            REFERENCES BITSPLEASE.PHARMACIES (PHARMACYID) ENABLE,
           CONSTRAINT PHARMACY_PROMO_FK2 FOREIGN KEY (PROMOTIONID)
            REFERENCES BITSPLEASE.PROMOTIONS (PROMOTIONID) ENABLE
  );
```

40. PHARM_MEDICINES

```
CREATE TABLE BITSPLEASE.PHARM_MEDICINES
  (       MEDICINEID NUMBER(38,0),
          PHARMACYID NUMBER(38,0),
           CONSTRAINT PHARM_MEDICINES PRIMARY KEY (MEDICINEID,
PHARMACYID)
CONSTRAINT PHARM_MEDICINEID_FK FOREIGN KEY (MEDICINEID)
            REFERENCES BITSPLEASE.MEDICINES (MEDICINEID) ENABLE,
           CONSTRAINT PHARM_PHARMACYID_FK FOREIGN KEY (PHARMACYID)
            REFERENCES BITSPLEASE.PHARMACIES (PHARMACYID) ENABLE
  );
```

41. PHARM_PHARMACIST

```
CREATE TABLE BITSPLEASE.PHARM_PHARMACIST
  (       PHARMACYID NUMBER(38,0),
          PHARMACISTID NUMBER(38,0),
           CONSTRAINT PHARM_PHARMACIST_PK PRIMARY KEY (PHARMACYID,
PHARMACISTID)
 CONSTRAINT PHARM_PHARMACY_FK1 FOREIGN KEY (PHARMACYID)
            REFERENCES BITSPLEASE.PHARMACIES (PHARMACYID) ENABLE,
           FOREIGN KEY (PHARMACISTID)
            REFERENCES BITSPLEASE.PHARMACISTS (PHARMACISTID) ENABLE
  );
```

42. PHARMACIES

```
CREATE TABLE BITSPLEASE.PHARMACIES
  (	PHARMACYID NUMBER(38,0) NOT NULL ENABLE,
	BULDINGNO VARCHAR2(100 BYTE),
	STREET VARCHAR2(100 BYTE),
	ZIP NUMBER(38,0),
	PHONENO NUMBER(38,0),
	EMAILADDRESS VARCHAR2(50 BYTE),
	PHARMACYNAME VARCHAR2(50 BYTE),
	 CONSTRAINT PHARMACY_PK PRIMARY KEY (PHARMACYID)
 CONSTRAINT CHK_PHONE_PHARMACIES CHECK (phoneno not like '%[^0-9]%')
ENABLE
  );
```

43. PHARMACISTS

```
CREATE TABLE BITSPLEASE.PHARMACISTS
  (	PHARMACISTID NUMBER(38,0) NOT NULL ENABLE,
	PHARMACISTICENSENO VARCHAR2(20 BYTE),
	 CONSTRAINT PHARMACIST_PK PRIMARY KEY (PHARMACISTID)
CONSTRAINT EMPLOYEE_PHARMACIST_FK FOREIGN KEY (PHARMACISTID)
	  REFERENCES BITSPLEASE.EMPLOYEES (EMPLOYEEID) ENABLE
  );
```

44. PRESCRIPTIONS

```
CREATE TABLE BITSPLEASE.PRESCRIPTIONS
  (	PRESCRIPTIONID NUMBER(38,0) NOT NULL ENABLE,
	PRESCRPTIONDATE TIMESTAMP (6),
	PHARMACISTID NUMBER(38,0),
	CASEID NUMBER(38,0),
	DOCTORID NUMBER(38,0),
	PHARMACYID NUMBER(38,0),
	 CONSTRAINT PRESCRIPTIONS_PK PRIMARY KEY (PRESCRIPTIONID)
CONSTRAINT PRES_PHARACY_FK FOREIGN KEY (PHARMACISTID)
	  REFERENCES BITSPLEASE.PHARMACISTS (PHARMACISTID) ENABLE,
	 CONSTRAINT PRES_CASEDETAILS_FK FOREIGN KEY (CASEID)
	  REFERENCES BITSPLEASE.CASE_DETAILS (CASEID) ENABLE,
	 CONSTRAINT PRES_DOCTORS_FK FOREIGN KEY (DOCTORID)
	  REFERENCES BITSPLEASE.DOCTORS (DOCTORID) ENABLE,
	 CONSTRAINT PRES_PHARMACIES_FK FOREIGN KEY (PHARMACYID)
	  REFERENCES BITSPLEASE.PHARMACIES (PHARMACYID) ENABLE
  );
```

45. PRESCRIPTIONS_LABTESTS

```
CREATE TABLE BITSPLEASE.PRESCRIPTIONS_LABTESTS
  (	TESTID NUMBER(38,0),
          PRESCRIPTIONID NUMBER(38,0),
           CONSTRAINT PRESCRIPTIONS_LABTESTS_PK PRIMARY KEY (TESTID,
PRESCRIPTIONID)
CONSTRAINT PRESC_LABTESTS_FK FOREIGN KEY (TESTID)
            REFERENCES BITSPLEASE.LAB_TESTS (TESTID) ENABLE,
           CONSTRAINT LABTESTS_PRES_FK FOREIGN KEY (PRESCRIPTIONID)
            REFERENCES BITSPLEASE.PRESCRIPTIONS (PRESCRIPTIONID) ENABLE
  );
```

46. PROMOTIONS

```
CREATE TABLE BITSPLEASE.PROMOTIONS
  (       PROMOTIONID NUMBER(38,0),
          DISCOUNT FLOAT(126),
          STARTDATE DATE,
          ENDDATE DATE,
           CONSTRAINT PROMOTIONS_PK PRIMARY KEY (PROMOTIONID)
);
```

47. RATINGS

```
CREATE TABLE BITSPLEASE.RATINGS
  (       RATINGID NUMBER(38,0) NOT NULL ENABLE,
          DESCRIPTION VARCHAR2(255 BYTE) NOT NULL ENABLE,
           PRIMARY KEY (RATINGID)
);
```

48. SPECIALISTS

```
CREATE TABLE BITSPLEASE.SPECIALISTS
  (       SPECIALISTID NUMBER(38,0),
          SPECIALIZATIONID NUMBER(38,0),
          ISPERMANENT NUMBER,
           CONSTRAINT SPECIALISTS_PK PRIMARY KEY (SPECIALISTID)
CONSTRAINT SPECIALISTS_CHK1 CHECK (ISPERMANENT='1' OR
ISPERMANENT='0') ENABLE,
           CONSTRAINT SPECIALISTS_DOCTOR_FK FOREIGN KEY (SPECIALISTID)
            REFERENCES BITSPLEASE.DOCTORS (DOCTORID) ENABLE,
           CONSTRAINT SPECIALIZATIONS_FK FOREIGN KEY (SPECIALIZATIONID)
            REFERENCES BITSPLEASE.SPECIALIZATIONS (SPECIALIZATIONID)
ENABLE
  )
```

49. SPECIALIZATIONS

```
CREATE TABLE BITSPLEASE.SPECIALIZATIONS
  (       SPECIALIZATIONID NUMBER NOT NULL ENABLE,
          SPECIALIZATIONNAME VARCHAR2(50 BYTE),
          DESCRIPTION VARCHAR2(255 BYTE),
           CONSTRAINT SPECIALIZATIONS_PK PRIMARY KEY (SPECIALIZATIONID)
```

);

50. STUDENTS

```
CREATE TABLE BITSPLEASE.STUDENTS
  (     STUDENTID NUMBER(38,0) NOT NULL ENABLE,
        FIRSTNAME VARCHAR2(20 BYTE),
        MIDDLEINITIAL VARCHAR2(20 BYTE),
        LASTNAME VARCHAR2(20 BYTE),
        BLOODGROUP VARCHAR2(20 BYTE),
        EMAILADDRESS VARCHAR2(50 BYTE),
        PHONENO NUMBER(10,0),
        BUILDINGNO VARCHAR2(20 BYTE),
        STREET VARCHAR2(20 BYTE),
        ZIP VARCHAR2(20 BYTE),
        GENDER VARCHAR2(20 BYTE),
        INSCOMPANYID NUMBER(38,0),
        DATEOFBIRTH DATE,
        AGE NUMBER,
        CITY VARCHAR2(20 BYTE),
        STATE VARCHAR2(20 BYTE),
         CONSTRAINT STUDENTS_PK PRIMARY KEY (STUDENTID)
 CONSTRAINT CHK_PHONE_STUDENT CHECK (phoneno not like '%[^0-9]%')
ENABLE,
         CONSTRAINT STUDENT_INSCOMP_FK FOREIGN KEY (INSCOMPANYID)
          REFERENCES BITSPLEASE.INSURANCE_COMPANIES (INSCOMPANYID)
ENABLE
  );
```

51. SYMPTOMS

```
CREATE TABLE BITSPLEASE.SYMPTOMS
  (     SYMPTOMID NUMBER(38,0),
        NAME VARCHAR2(50 BYTE),
        TYPE VARCHAR2(50 BYTE),
         CONSTRAINT SYMPTOMS_PK PRIMARY KEY (SYMPTOMID)
);
```

52. TRIP_DETAILS

```
CREATE TABLE BITSPLEASE.TRIP_DETAILS
  (     TRIPID NUMBER(38,0) NOT NULL ENABLE,
        TIMEOFTRIP TIMESTAMP (6),
        STREET VARCHAR2(100 BYTE),
        ZIP NUMBER,
        BUILDINGNO VARCHAR2(100 BYTE),
        CREWID NUMBER(38,0),
        AMBULANCEID NUMBER(38,0),
        CASEID NUMBER,
         CONSTRAINT TRIP_DETAILS_PK PRIMARY KEY (TRIPID)
CONSTRAINT TRIP_DETAILS_CREWS_FK FOREIGN KEY (CREWID)
```

```
            REFERENCES BITSPLEASE.CREWS (CREWID) ENABLE,
           CONSTRAINT TRIP_DETAILS_CASEID_FK FOREIGN KEY (CASEID)
            REFERENCES BITSPLEASE.CASE_DETAILS (CASEID) ENABLE,
           CONSTRAINT TRIP_DETAILS_AMBULANCES_FK FOREIGN KEY
(AMBULANCEID)
            REFERENCES BITSPLEASE.AMBULANCES (AMBULANCEID) ON DELETE
CASCADE ENABLE
   );
```

References

User Interface

- *https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css*
- *https://code.jquery.com/jquery-3.3.1.slim.min.js*
- integrity=*"sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo*
- *https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js*
- integrity=*"sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"*
- https://bootsnipp.com/snippets/dldxB
- Left Navigation Pane: https://www.w3schools.com/howto/tryit.asp?filename=tryhow_js_sidenav
- Chat app: https://github.com/www-leafie-io/chat

Function

- https://jameshuangsj.wordpress.com/2019/05/09/data-encryption-and-decryption-in-oracle/